TiDB Documentation

PingCAP Inc.

20250625

Table of Contents

1 Docs Home **32**

2 About TiDB **32**

2.1 TiDB Introduction · · · · · 32

2.1.1 Key features · · · · · 32

2.1.2 Use cases · · · · · 33

2.1.3 See also · · · · · 34

2.2 TiDB 6.1.0 Release Notes · · · · · 34

2.2.1 New Features · · · · · 35

2.2.2 Compatibility changes · · · · · 41

2.2.3 Improvements · · · · · 82

2.2.4 Bug fixes · · · · · 83

2.3 TiDB Features · · · · · 86

2.3.1 Data types, functions, and operators · · · · · 86

2.3.2 Indexing and constraints · · · · · 86

2.3.3 SQL statements · · · · · 87

2.3.4 Advanced SQL features · · · · · 87

2.3.5 Data definition language (DDL) · · · · · 88

2.3.6 Transactions · · · · · 88

2.3.7 Partitioning · · · · · 88

2.3.8 Statistics · · · · · 89

2.3.9 Security · · · · · 89
2.3.10 Data import and export · · · · · 89
2.3.11 Management, observability, and tools · · · · · 90
2.4 TiDB Experimental Features · · · · · 90
2.4.1 Performance · · · · · 90
2.4.2 Stability · · · · · 91
2.4.3 Scheduling · · · · · 91
2.4.4 SQL · · · · · 91
2.4.5 Storage · · · · · 91
2.4.6 Backup and restoration · · · · · 92
2.4.7 Data migration · · · · · 92
2.4.8 Garbage collection · · · · · 92
2.4.9 Diagnostics · · · · · 92
2.5 Benchmarks · · · · · 92
2.5.1 TiDB Sysbench Performance Test Report – v6.1.0 vs. v6.0.0 · · · · · 92
2.5.2 TiDB TPC-C Performance Test Report – v6.1.0 vs. v6.0.0 · · · · · 99
2.5.3 Performance Comparison between TiFlash and Greenplum/Spark · · · · · 102
2.6 MySQL Compatibility · · · · · 103
2.6.1 Unsupported features · · · · · 103
2.6.2 Features that are different from MySQL · · · · · 104
2.7 TiDB Limitations · · · · · 109
2.7.1 Limitations on identifier length · · · · · 109
2.7.2 Limitations on the total number of databases, tables, views, and connections · · · · · 110
2.7.3 Limitations on a single database · · · · · 110
2.7.4 Limitations on a single table · · · · · 110
2.7.5 Limitation on a single row · · · · · 110
2.7.6 Limitations on data types · · · · · 110
2.7.7 Limitations on SQL statements · · · · · 111
2.8 Credits · · · · · 111
2.8.1 TiDB developers · · · · · 111
2.8.2 Writers and translators for TiDB documentation · · · · · 112

3 Quick Start **112**

3.1 Quick Start Guide for the TiDB Database Platform 112

3.1.1 Deploy a local test cluster 113

3.1.2 Simulate production deployment on a single machine 118

3.1.3 What's next 124

3.2 Quick Start Guide for TiDB HTAP 124

3.2.1 Basic concepts 125

3.2.2 Steps 125

3.2.3 What's next 129

3.3 Explore SQL with TiDB 129

3.3.1 Category 130

3.3.2 Show, create and drop a database 130

3.3.3 Create, show, and drop a table 131

3.3.4 Create, show, and drop an index 131

3.3.5 Insert, update, and delete data 132

3.3.6 Query data 132

3.3.7 Create, authorize, and delete a user 133

3.4 Explore HTAP 133

3.4.1 Use cases 134

3.4.2 Architecture 134

3.4.3 Environment preparation 135

3.4.4 Data preparation 135

3.4.5 Data processing 136

3.4.6 Performance monitoring 136

3.4.7 Troubleshooting 137

3.4.8 What's next 137

3.5 Import Example Database 137

3.5.1 Download all data files 137

3.5.2 Load data into TiDB 137

4 Develop **139**

4.1 Developer Guide Overview · · · · · 139
4.1.1 TiDB basics · · · · · 139
4.1.2 TiDB transaction mechanisms · · · · · 139
4.1.3 The way applications interact with TiDB · · · · · 140
4.1.4 Read More · · · · · 140
4.2 Quick Start · · · · · 140
4.2.1 Build a TiDB Serverless Cluster · · · · · 140
4.2.2 CRUD SQL in TiDB · · · · · 143
4.3 Example Applications · · · · · 145
4.3.1 Build a Simple CRUD App with TiDB and Golang · · · · · 145
4.3.2 Build a TiDB Application Using Spring Boot · · · · · 163
4.3.3 Build a Simple CRUD App with TiDB and Java · · · · · 195
4.3.4 Build a Simple CRUD App with TiDB and Python · · · · · 234
4.4 Connect to TiDB · · · · · 257
4.4.1 Choose Driver or ORM · · · · · 257
4.4.2 Connect to TiDB · · · · · 263
4.4.3 Connection Pools and Connection Parameters · · · · · 266
4.5 Design Database Schema · · · · · 274
4.5.1 TiDB Database Schema Design Overview · · · · · 274
4.5.2 Create a Database · · · · · 277
4.5.3 Create a Table · · · · · 279
4.5.4 Create a Secondary Index · · · · · 291
4.6 Write Data · · · · · 295
4.6.1 Insert Data · · · · · 295
4.6.2 Update Data · · · · · 303
4.6.3 Delete Data · · · · · 313
4.6.4 Prepared Statements · · · · · 322
4.7 Read Data · · · · · 327
4.7.1 Query Data from a Single Table · · · · · 327
4.7.2 Multi-table Join Queries · · · · · 333
4.7.3 Subquery · · · · · 341
4.7.4 Paginate Results · · · · · 344

4.7.5 Views · · · · · 351
4.7.6 Temporary Tables · · · · · 353
4.7.7 Common Table Expression · · · · · 359
4.7.8 Read Replica Data · · · · · 363
4.7.9 HTAP Queries · · · · · 377
4.8 Transaction · · · · · 383
4.8.1 Transaction overview · · · · · 383
4.8.2 Optimistic Transactions and Pessimistic Transactions · · · · · 387
4.8.3 Transaction Restraints · · · · · 418
4.8.4 Handle Transaction Errors · · · · · 437
4.9 Optimize · · · · · 441
4.9.1 Overview of Optimizing SQL Performance · · · · · 441
4.9.2 SQL Performance Tuning · · · · · 441
4.9.3 Performance Tuning Best Practices · · · · · 448
4.9.4 Best Practices for Indexing · · · · · 451
4.9.5 Other Optimization Methods · · · · · 454
4.10 Troubleshoot · · · · · 458
4.10.1 SQL or Transaction Issues · · · · · 458
4.10.2 Unstable Result Set · · · · · 459
4.10.3 Timeouts in TiDB · · · · · 464
4.11 Reference · · · · · 466
4.11.1 Bookshop Example Application · · · · · 466
4.11.2 Guidelines · · · · · 473
4.11.3 Legacy Docs · · · · · 475
4.12 Cloud Native Development Environment · · · · · 481
4.12.1 Gitpod · · · · · 481
4.13 Third-Party Support · · · · · 487
4.13.1 Third-Party Tools Supported by TiDB · · · · · 487
4.13.2 Known Incompatibility Issues with Third-Party Tools · · · · · 492
4.13.3 Integrate TiDB with ProxySQL · · · · · 497
4.13.4 Integrate TiDB with Amazon AppFlow · · · · · 517

5 Deploy **535**

5.1 Software and Hardware Recommendations · · · · · 535
5.1.1 OS and platform requirements · · · · · 535
5.1.2 Software recommendations · · · · · 538
5.1.3 Server recommendations · · · · · 539
5.1.4 Network requirements · · · · · 541
5.1.5 Disk space requirements · · · · · 548
5.1.6 Web browser requirements · · · · · 551
5.2 TiDB Environment and System Configuration Check · · · · · 551
5.2.1 Mount the data disk ext4 filesystem with options on the target machines that deploy TiKV · · · · · 551
5.2.2 Check and disable system swap · · · · · 553
5.2.3 Check and stop the firewall service of target machines · · · · · 554
5.2.4 Check and install the NTP service · · · · · 554
5.2.5 Check and configure the optimal parameters of the operating system · · 557
5.2.6 Manually configure the SSH mutual trust and sudo without password · 563
5.2.7 Install the `numactl` tool · · · · · 564
5.3 Plan Cluster Topology · · · · · 565
5.3.1 Minimal Deployment Topology · · · · · 565
5.3.2 TiFlash Deployment Topology · · · · · 567
5.3.3 TiCDC Deployment Topology · · · · · 570
5.3.4 TiDB Binlog Deployment Topology · · · · · 572
5.3.5 TiSpark Deployment Topology · · · · · 577
5.3.6 Geo-Distributed Deployment Topology · · · · · 580
5.3.7 Hybrid Deployment Topology · · · · · 584
5.4 Install and Start · · · · · 589
5.4.1 Deploy a TiDB Cluster Using TiUP · · · · · 589
5.4.2 Deploy a TiDB Cluster on Kubernetes · · · · · 605
5.5 Check Cluster Status · · · · · 606
5.5.1 Check the TiDB cluster status · · · · · 606
5.5.2 Log in to the database and perform simple operations · · · · · 609

5.6 Test Cluster Performance · · · · · 611
5.6.1 How to Test TiDB Using Sysbench · · · · · 611
5.6.2 How to Run TPC-C Test on TiDB · · · · · 616
5.6.3 How to Run CH-benCHmark Test on TiDB · · · · · 618

6 Migrate **623**
6.1 Data Migration Overview · · · · · 623
6.1.1 Migrate data from Aurora MySQL to TiDB · · · · · 623
6.1.2 Migrate data from MySQL to TiDB · · · · · 624
6.1.3 Migrate and merge MySQL shards into TiDB · · · · · 624
6.1.4 Migrate data from files to TiDB · · · · · 624
6.1.5 More advanced migration solutions · · · · · 624
6.2 TiDB Migration Tools Overview · · · · · 625
6.2.1 TiDB Data Migration (DM) · · · · · 625
6.2.2 TiDB Lightning · · · · · 625
6.2.3 Dumpling · · · · · 626
6.2.4 TiCDC · · · · · 626
6.2.5 Backup & Restore (BR) · · · · · 631
6.2.6 sync-diff-inspector · · · · · 632
6.2.7 Install tools using TiUP · · · · · 633
6.2.8 See also · · · · · 634
6.3 Migration Scenarios · · · · · 634
6.3.1 Migrate Data from Amazon Aurora to TiDB · · · · · 634
6.3.2 Migrate MySQL of Small Datasets to TiDB · · · · · 648
6.3.3 Migrate MySQL of Large Datasets to TiDB · · · · · 653
6.3.4 Migrate and Merge MySQL Shards of Small Datasets to TiDB · · · · · 666
6.3.5 Migrate and Merge MySQL Shards of Large Datasets to TiDB · · · · · 674
6.3.6 Migrate Data from CSV Files to TiDB · · · · · 691
6.3.7 Migrate Data from SQL Files to TiDB · · · · · 696
6.3.8 Migrate from One TiDB Cluster to Another TiDB Cluster · · · · · 699
6.3.9 Migrate Data from TiDB to MySQL-compatible Databases · · · · · 706

6.4 Advanced Migration · · · · · 711
6.4.1 Continuous Replication from Databases that Use gh-ost or pt-osc · · · · · 711
6.4.2 Migrate Data to a Downstream TiDB Table with More Columns · · · · · 713
6.4.3 Filter Binlog Events · · · · · 720
6.4.4 Filter DML Events Using SQL Expressions · · · · · 723

7 Integrate **725**
7.1 Data Integration Overview · · · · · 725
7.1.1 Integrate with Confluent Cloud and Snowflake · · · · · 726
7.1.2 Integrate with Apache Kafka and Apache Flink · · · · · 726
7.2 Integration Scenarios · · · · · 726
7.2.1 Integrate Data with Confluent Cloud and Snowflake · · · · · 726
7.2.2 Integrate Data with Apache Kafka and Apache Flink · · · · · 741

8 Maintain **746**
8.1 Security · · · · · 746
8.1.1 Best Practices for TiDB Security Configuration · · · · · 746
8.1.2 Enable TLS between TiDB Clients and Servers · · · · · 752
8.1.3 Enable TLS Between TiDB Components · · · · · 757
8.1.4 Generate Self-Signed Certificates · · · · · 761
8.1.5 Encryption at Rest · · · · · 764
8.1.6 Enable Encryption for Disk Spill · · · · · 771
8.1.7 Log Redaction · · · · · 771
8.2 Upgrade · · · · · 772
8.2.1 Upgrade TiDB Using TiUP · · · · · 772
8.2.2 Use TiDB Operator · · · · · 779
8.3 Scale · · · · · 779
8.3.1 Scale a TiDB Cluster Using TiUP · · · · · 779
8.3.2 Use TiDB Operator · · · · · 791
8.4 Backup and Restore · · · · · 791
8.4.1 Use BR to Back Up Cluster Data · · · · · 791
8.4.2 Use BR to Restore Cluster Data · · · · · 798
8.4.3 BR Use Cases · · · · · 804

8.5 Time Zone Support · · · · · 823
8.6 Daily Check · · · · · 825
8.6.1 Key indicators of TiDB Dashboard · · · · · 825
8.7 Maintain a TiFlash Cluster · · · · · 831
8.7.1 Check the TiFlash version · · · · · 831
8.7.2 TiFlash critical logs · · · · · 831
8.7.3 TiFlash system table · · · · · 832
8.8 TiUP Common Operations · · · · · 833
8.8.1 View the cluster list · · · · · 833
8.8.2 Start the cluster · · · · · 833
8.8.3 View the cluster status · · · · · 834
8.8.4 Modify the configuration · · · · · 834
8.8.5 Replace with a hotfix package · · · · · 835
8.8.6 Rename the cluster · · · · · 836
8.8.7 Stop the cluster · · · · · 837
8.8.8 Clean up cluster data · · · · · 837
8.8.9 Destroy the cluster · · · · · 838
8.9 Modify Configuration Dynamically · · · · · 838
8.9.1 Common Operations · · · · · 838
8.10 Online Unsafe Recovery · · · · · 905
8.10.1 Feature description · · · · · 905
8.10.2 User scenarios · · · · · 906
8.10.3 Usage · · · · · 906
8.11 Replicate Data Between Primary and Secondary Clusters · · · · · 911
8.11.1 Step 1. Set up the environment · · · · · 911
8.11.2 Step 2. Migrate full data · · · · · 913
8.11.3 Step 3. Migrate incremental data · · · · · 916
8.11.4 Step 4. Simulate a disaster in the upstream cluster · · · · · 917
8.11.5 Step 5. Use redo log to ensure data consistency · · · · · 917
8.11.6 Step 6. Recover the primary cluster and its services · · · · · 918

9 Monitor and Alert **918**

9.1 TiDB Monitoring Framework Overview · · · · · 918
9.1.1 About Prometheus in TiDB · · · · · 918
9.1.2 About Grafana in TiDB · · · · · 919
9.2 TiDB Monitoring API · · · · · 921
9.2.1 Use the status interface · · · · · 922
9.2.2 Use the metrics interface · · · · · 924
9.3 Deploy Monitoring Services for the TiDB Cluster · · · · · 924
9.3.1 Deploy Prometheus and Grafana · · · · · 924
9.3.2 Configure Grafana · · · · · 928
9.3.3 View component metrics · · · · · 929
9.4 Upgrade TiDB Cluster Monitoring Services · · · · · 930
9.4.1 Upgrade Prometheus · · · · · 931
9.4.2 Upgrade Grafana · · · · · 932
9.4.3 Upgrade Alertmanager · · · · · 934
9.5 Export Grafana Snapshots · · · · · 934
9.5.1 Usage · · · · · 935
9.5.2 FAQs · · · · · 936
9.6 TiDB Cluster Alert Rules · · · · · 937
9.6.1 TiDB alert rules · · · · · 939
9.6.2 PD alert rules · · · · · 943
9.6.3 TiKV alert rules · · · · · 949
9.6.4 TiFlash alert rules · · · · · 957
9.6.5 TiDB Binlog alert rules · · · · · 957
9.6.6 TiCDC Alert rules · · · · · 957
9.6.7 Node_exporter host alert rules · · · · · 958
9.6.8 Blackbox_exporter TCP, ICMP, and HTTP alert rules · · · · · 961
9.7 TiFlash Alert Rules · · · · · 966
9.7.1 `TiFlash_schema_error` · · · · · 966
9.7.2 `TiFlash_schema_apply_duration` · · · · · 966
9.7.3 `TiFlash_raft_read_index_duration` · · · · · 967
9.7.4 `TiFlash_raft_wait_index_duration` · · · · · 967

9.8 Customize Configurations of Monitoring Servers · · · · · 968
9.8.1 Customize Prometheus configurations · · · · · 968
9.8.2 Customize Grafana configurations · · · · · 969
9.8.3 Customize Alertmanager configurations · · · · · 971

10 Troubleshoot **971**
10.1 TiDB Troubleshooting Map · · · · · 971
10.1.1 1. Service Unavailable · · · · · 971
10.1.2 2. Latency increases significantly · · · · · 972
10.1.3 3. TiDB issues · · · · · 973
10.1.4 4. TiKV issues · · · · · 977
10.1.5 5. PD issues · · · · · 981
10.1.6 6. Ecosystem tools · · · · · 983
10.1.7 7. Common log analysis · · · · · 989
10.2 Identify Slow Queries · · · · · 991
10.2.1 Usage example · · · · · 991
10.2.2 Fields description · · · · · 992
10.2.3 Related system variables · · · · · 995
10.2.4 Memory mapping in slow log · · · · · 997
10.2.5 `SLOW_QUERY` / `CLUSTER_SLOW_QUERY` usage examples · · · · · 998
10.2.6 Query slow queries with pseudo `stats` · · · · · 1000
10.2.7 Identify problematic SQL statements · · · · · 1005
10.3 Analyze Slow Queries · · · · · 1010
10.3.1 Identify the performance bottleneck of the query · · · · · 1010
10.3.2 Analyze system issues · · · · · 1011
10.3.3 Analyze optimizer issues · · · · · 1018
10.4 SQL Diagnostics · · · · · 1018
10.4.1 Overview · · · · · 1019
10.4.2 Cluster information tables · · · · · 1019
10.4.3 Cluster monitoring tables · · · · · 1020
10.4.4 Automatic diagnostics · · · · · 1021

10.5 TiDB Dashboard Top SQL Page · · · · · 1021
10.5.1 Recommended scenarios · · · · · 1021
10.5.2 Access the page · · · · · 1022
10.5.3 Enable Top SQL · · · · · 1023
10.5.4 Use Top SQL · · · · · 1024
10.5.5 Disable Top SQL · · · · · 1028
10.5.6 Frequently asked questions · · · · · 1028
10.6 Identify Expensive Queries · · · · · 1029
10.6.1 Expensive query log example · · · · · 1029
10.6.2 Fields description · · · · · 1029
10.7 Statement Summary Tables · · · · · 1031
10.7.1 `statements_summary` · · · · · 1031
10.7.2 `statements_summary_history` · · · · · 1033
10.7.3 `statements_summary_evicted` · · · · · 1033
10.7.4 The `cluster` tables for statement summary · · · · · 1033
10.7.5 Parameter configuration · · · · · 1034
10.7.6 Limitation · · · · · 1036
10.7.7 Troubleshooting examples · · · · · 1036
10.7.8 Fields description · · · · · 1037
10.8 Troubleshoot Hotspot Issues · · · · · 1040
10.8.1 Common hotspots · · · · · 1041
10.8.2 Identify hotspot issues · · · · · 1042
10.8.3 Use `SHARD_ROW_ID_BITS` to process hotspots · · · · · 1046
10.8.4 Handle auto-increment primary key hotspot tables using `AUTO_RANDOM` 1049
10.8.5 Optimization of small table hotspots · · · · · 1052
10.9 Troubleshoot Increased Read and Write Latency · · · · · 1052
10.9.1 Common causes · · · · · 1052
10.9.2 Other causes · · · · · 1056
10.10 Use PLAN REPLAYER to Save and Restore the On-Site Information of a Cluster · · · · · 1057
10.10.1 Use `PLAN REPLAYER` to export cluster information · · · · · 1057
10.10.2 Use `PLAN REPLAYER` to import cluster information · · · · · 1059

10.11 Troubleshoot TiDB OOM Issues · · · · · 1060
10.11.1 Typical OOM phenomena · · · · · 1061
10.11.2 Overall troubleshooting process · · · · · 1061
10.11.3 Typical causes and solutions · · · · · 1062
10.11.4 Diagnostic information to be collected to troubleshoot OOM issues · · · 1065
10.11.5 See also · · · · · 1066
10.12 TiDB Cluster Troubleshooting Guide · · · · · 1066
10.12.1 Cannot connect to the database · · · · · 1067
10.12.2 Cannot start `tidb-server` · · · · · 1067
10.12.3 Cannot start `tikv-server` · · · · · 1068
10.12.4 Cannot start `pd-server` · · · · · 1068
10.12.5 The TiDB/TiKV/PD process aborts unexpectedly · · · · · 1068
10.12.6 TiDB panic · · · · · 1068
10.12.7 The connection is rejected · · · · · 1069
10.12.8 Open too many files · · · · · 1069
10.12.9 Database access times out and the system load is too high · · · · · 1069
10.13 Troubleshoot High Disk I/O Usage in TiDB · · · · · 1069
10.13.1 Check the current I/O metrics · · · · · 1070
10.13.2 Handle I/O issues · · · · · 1073
10.14 Troubleshoot Lock Conflicts · · · · · 1073
10.14.1 Use Lock View to troubleshoot lock issues · · · · · 1073
10.14.2 Troubleshoot optimistic lock conflicts · · · · · 1078
10.14.3 Troubleshoot pessimistic lock conflicts · · · · · 1083
10.15 Troubleshoot a TiFlash Cluster · · · · · 1085
10.15.1 TiFlash fails to start · · · · · 1085
10.15.2 TiFlash replica is always unavailable · · · · · 1086
10.15.3 Some queries return the `Region Unavailable` error · · · · · 1087
10.15.4 Data file corruption · · · · · 1087
10.15.5 TiFlash analysis is slow · · · · · 1087
10.15.6 Data is not replicated to TiFlash · · · · · 1088
10.15.7 Data replication gets stuck · · · · · 1090
10.15.8 Data replication is slow · · · · · 1091

10.16 Troubleshoot Write Conflicts in Optimistic Transactions · · · · · 1091
10.16.1 The reason of write conflicts · · · · · 1091
10.16.2 Detect write conflicts · · · · · 1092
10.16.3 Resolve write conflicts · · · · · 1093
10.17 Troubleshoot Inconsistency Between Data and Indexes · · · · · 1095
10.17.1 Error explanation · · · · · 1095
10.17.2 Solutions · · · · · 1097
10.18 Support Resources · · · · · 1098

11 Performance Tuning **1098**
11.1 Tuning Guide · · · · · 1098
11.1.1 TiDB Performance Tuning Overview · · · · · 1098
11.1.2 Performance Analysis and Tuning · · · · · 1103
11.1.3 Performance Tuning Practices for OLTP Scenarios · · · · · 1125
11.2 Configuration Tuning · · · · · 1149
11.2.1 Tune Operating System Performance · · · · · 1149
11.2.2 TiDB Memory Control · · · · · 1154
11.2.3 Tune TiKV Thread Pool Performance · · · · · 1161
11.2.4 Tune TiKV Memory Parameter Performance · · · · · 1165
11.2.5 Follower Read · · · · · 1173
11.2.6 Tune Region Performance · · · · · 1174
11.2.7 Tune TiFlash Performance · · · · · 1176
11.2.8 Coprocessor Cache · · · · · 1177
11.2.9 Garbage Collection (GC) · · · · · 1179
11.3 SQL Tuning · · · · · 1183
11.3.1 SQL Tuning Overview · · · · · 1183
11.3.2 Understanding the Query Execution Plan · · · · · 1183
11.3.3 SQL Optimization Process · · · · · 1249
11.3.4 Control Execution Plans · · · · · 1335

12 Tutorials **1383**

12.1 Multiple Availability Zones in One Region Deployment · · · · · 1383
12.1.1 Raft protocol · · · · · 1383
12.1.2 Three AZs in one region deployment · · · · · 1384
12.2 Three Availability Zones in Two Regions Deployment · · · · · 1390
12.2.1 Overview · · · · · 1390
12.2.2 Deployment architecture · · · · · 1391
12.2.3 Configuration · · · · · 1394
12.3 Two Availability Zones in One Region Deployment · · · · · 1398
12.3.1 Introduction · · · · · 1398
12.3.2 Deployment architecture · · · · · 1398
12.3.3 Configuration · · · · · 1400
12.4 Read Historical Data · · · · · 1406
12.4.1 Use Stale Read (Recommended) · · · · · 1406
12.4.2 Read Historical Data Using the System Variable `tidb_snapshot` · · · · · 1416
12.5 Best Practices · · · · · 1421
12.5.1 TiDB Best Practices · · · · · 1421
12.5.2 Best Practices for Developing Java Applications with TiDB · · · · · 1428
12.5.3 Best Practices for Using HAProxy in TiDB · · · · · 1441
12.5.4 Highly Concurrent Write Best Practices · · · · · 1455
12.5.5 Best Practices for Monitoring TiDB Using Grafana · · · · · 1467
12.5.6 PD Scheduling Best Practices · · · · · 1478
12.5.7 Best Practices for TiKV Performance Tuning with Massive Regions · · · 1487
12.5.8 Best Practices for Three-Node Hybrid Deployment · · · · · 1494
12.5.9 Local Read under Three Data Centers Deployment · · · · · 1500
12.5.10 UUID Best Practices · · · · · 1501
12.6 Placement Rules · · · · · 1503
12.6.1 Rule system · · · · · 1503
12.6.2 Configure rules · · · · · 1505
12.6.3 Typical usage scenarios · · · · · 1510
12.7 Load Base Split · · · · · 1513
12.7.1 Scenarios · · · · · 1514
12.7.2 Implementation principles · · · · · 1514
12.7.3 Usage · · · · · 1514

12.8 Store Limit · · · · · 1515
12.8.1 Implementation principles · · · · · 1515
12.8.2 Usage · · · · · 1516

13 TiDB Tools **1517**
13.1 TiDB Tools Overview · · · · · 1517
13.1.1 Deployment and operation Tools · · · · · 1517
13.1.2 Data management tools · · · · · 1518
13.1.3 OLAP Query tool - TiSpark · · · · · 1521
13.2 TiDB Tools Use Cases · · · · · 1521
13.2.1 Deploy and operate TiDB on physical or virtual machines · · · · · 1521
13.2.2 Deploy and operate TiDB on Kubernetes · · · · · 1521
13.2.3 Import data from CSV to TiDB · · · · · 1522
13.2.4 Import full data from MySQL/Aurora · · · · · 1522
13.2.5 Migrate data from MySQL/Aurora · · · · · 1522
13.2.6 Back up and restore TiDB cluster · · · · · 1522
13.2.7 Migrate data to TiDB · · · · · 1522
13.2.8 TiDB incremental data subscription · · · · · 1522
13.3 Download TiDB Tools · · · · · 1522
13.3.1 Environment requirements · · · · · 1523
13.3.2 Download link · · · · · 1523
13.3.3 TiDB Toolkit description · · · · · 1523
13.4 TiUP · · · · · 1526
13.4.1 TiUP Documentation Map · · · · · 1526
13.4.2 TiUP Overview · · · · · 1527
13.4.3 TiUP Terminology and Concepts · · · · · 1530
13.4.4 Manage TiUP Components with TiUP Commands · · · · · 1531
13.4.5 TiUP FAQs · · · · · 1535
13.4.6 TiUP Troubleshooting Guide · · · · · 1537
13.4.7 Command Reference · · · · · 1538
13.4.8 Topology Configuration File for TiDB Deployment Using TiUP · · · · · 1640
13.4.9 Topology Configuration File for DM Cluster Deployment Using TiUP · 1663
13.4.10 TiUP Mirror Reference Guide · · · · · 1674
13.4.11 TiUP Components · · · · · 1684

13.5 TiDB Operator · · · · · 1713
13.6 TiDB Data Migration · · · · · 1714
13.6.1 TiDB Data Migration Overview · · · · · 1714
13.6.2 Data Migration Architecture · · · · · 1716
13.6.3 Quick Start Guide for TiDB Data Migration · · · · · 1719
13.6.4 Deploy a DM cluster · · · · · 1722
13.6.5 Tutorials · · · · · 1745
13.6.6 Advanced Tutorials · · · · · 1799
13.6.7 Maintain · · · · · 1843
13.6.8 Reference · · · · · 1905
13.7 TiDB Lightning · · · · · 2061
13.7.1 TiDB Lightning Overview · · · · · 2061
13.7.2 Quick Start for TiDB Lightning · · · · · 2063
13.7.3 Deploy TiDB Lightning · · · · · 2066
13.7.4 TiDB Lightning Requirements for the Target Database · · · · · 2067
13.7.5 Data Sources · · · · · 2069
13.7.6 Physical Import Mode · · · · · 2078
13.7.7 Logical Import Mode · · · · · 2086
13.7.8 TiDB Lightning Prechecks · · · · · 2088
13.7.9 Table Filter · · · · · 2097
13.7.10 TiDB Lightning Checkpoints · · · · · 2102
13.7.11 Use TiDB Lightning to Import Data in Parallel · · · · · 2104
13.7.12 TiDB Lightning Error Resolution · · · · · 2112
13.7.13 Troubleshoot TiDB Lightning · · · · · 2119
13.8 Use Dumpling to Export Data · · · · · 2191
13.8.1 Export data from TiDB or MySQL · · · · · 2192
13.8.2 Option list of Dumpling · · · · · 2199
13.9 TiCDC · · · · · 2204
13.9.1 TiCDC Overview · · · · · 2204
13.9.2 Deploy TiCDC · · · · · 2214
13.9.3 Manage TiCDC Cluster and Replication Tasks · · · · · 2217
13.9.4 Monitor and Alert · · · · · 2243

13.9.5 Troubleshoot TiCDC · · · 2253
13.9.6 TiCDC FAQs · · · 2302
13.9.7 TiCDC Glossary · · · 2315
13.10 Backup & Restore (BR) · · · 2316
13.10.1 BR Overview · · · 2316
13.10.2 Deploy and Use BR · · · 2329
13.10.3 Use BR to Back Up Cluster Data · · · 2330
13.10.4 Use BR to Restore Cluster Data · · · 2337
13.10.5 BR Use Cases · · · 2343
13.10.6 BR Features · · · 2362
13.10.7 References · · · 2367
13.11 TiDB Binlog · · · 2403
13.11.1 TiDB Binlog Cluster Overview · · · 2403
13.11.2 TiDB Binlog Tutorial · · · 2406
13.11.3 TiDB Binlog Cluster Deployment · · · 2416
13.11.4 TiDB Binlog Cluster Operations · · · 2428
13.11.5 TiDB Binlog Configuration File · · · 2431
13.11.6 Upgrade TiDB Binlog · · · 2450
13.11.7 TiDB Binlog Monitoring · · · 2451
13.11.8 Reparo User Guide · · · 2468
13.11.9 binlogctl · · · 2472
13.11.10 Binlog Consumer Client User Guide · · · 2475
13.11.11 TiDB Binlog Relay Log · · · 2478
13.11.12 Bidirectional Replication between TiDB Clusters · · · 2480
13.11.13 TiDB Binlog Glossary · · · 2485
13.11.14 Troubleshoot · · · 2485
13.11.15 TiDB Binlog FAQs · · · 2487
13.12 PingCAP Clinic Diagnostic Service · · · 2496
13.12.1 PingCAP Clinic Overview · · · 2496
13.12.2 Quick Start Guide for PingCAP Clinic · · · 2498
13.12.3 Troubleshoot Clusters Using PingCAP Clinic · · · 2503
13.12.4 PingCAP Clinic Diagnostic Data · · · 2512

13.13 TiSpark 2523
13.13.1 TiSpark User Guide 2523
13.14 sync-diff-inspector 2534
13.14.1 sync-diff-inspector User Guide 2534
13.14.2 Data Check for Tables with Different Schema or Table Names 2542
13.14.3 Data Check in the Sharding Scenario 2543
13.14.4 Data Check for TiDB Upstream and Downstream Clusters 2547
13.14.5 Data Check in the DM Replication Scenario 2549

14 Reference **2550**

14.1 Cluster Architecture 2550
14.1.1 TiDB Architecture 2550
14.1.2 TiDB Storage 2552
14.1.3 TiDB Computing 2557
14.1.4 TiDB Scheduling 2564
14.2 Storage Engine - TiKV 2569
14.2.1 TiKV Overview 2569
14.2.2 RocksDB Overview 2571
14.2.3 Titan Overview 2574
14.2.4 Titan Configuration 2582
14.3 Storage Engine - TiFlash 2586
14.3.1 TiFlash Overview 2586
14.3.2 Create TiFlash Replicas 2590
14.3.3 Use TiDB to Read TiFlash Replicas 2596
14.3.4 Use TiSpark to Read TiFlash Replicas 2600
14.3.5 Use TiFlash MPP Mode 2600
14.3.6 Push-down Calculations Supported by TiFlash 2607
14.3.7 TiFlash Data validation 2609
14.3.8 TiFlash MinTSO Scheduler 2611
14.3.9 TiFlash Compatibility Notes 2617
14.4 System Variables 2618
14.4.1 Variable Reference 2619

14.5 Configuration File Parameters 2707
14.5.1 TiDB Configuration File 2707
14.5.2 TiKV Configuration File 2733
14.5.3 Configure TiFlash 2786
14.5.4 PD Configuration File 2796
14.6 CLI 2808
14.6.1 TiKV Control User Guide 2808
14.6.2 PD Control User Guide 2823
14.6.3 TiDB Control User Guide 2851
14.6.4 PD Recover User Guide 2858
14.7 Command Line Flags 2862
14.7.1 Configuration Options 2862
14.7.2 TiKV Configuration Flags 2868
14.7.3 TiFlash Command-Line Flags 2870
14.7.4 PD Configuration Flags 2872
14.8 Key Monitoring Metrics 2876
14.8.1 Key Metrics 2876
14.8.2 Key Metrics on Performance Overview 2881
14.8.3 TiDB Monitoring Metrics 2886
14.8.4 Key Monitoring Metrics of PD 2892
14.8.5 Key Monitoring Metrics of TiKV 2903
14.8.6 Monitor the TiFlash Cluster 2925
14.8.7 Key Monitoring Metrics of TiCDC 2929
14.9 Privileges 2935
14.9.1 Security Compatibility with MySQL 2935
14.9.2 Privilege Management 2935
14.9.3 TiDB User Account Management 2946
14.9.4 Role-Based Access Control 2951
14.9.5 Certificate-Based Authentication for Login 2958

14.10 SQL · · · · · 2968
14.10.1 SQL Language Structure and Syntax · · · · · 2968
14.10.2 SQL Statements · · · · · 3010
14.10.3 Data Types · · · · · 3349
14.10.4 Functions and Operators · · · · · 3367
14.10.5 Clustered Indexes · · · · · 3418
14.10.6 Constraints · · · · · 3425
14.10.7 Generated Columns · · · · · 3430
14.10.8 SQL Mode · · · · · 3434
14.10.9 Table Attributes · · · · · 3461
14.10.10 Transactions · · · · · 3464
14.10.11 Views · · · · · 3509
14.10.12 Partitioning · · · · · 3514
14.10.13 Temporary Tables · · · · · 3546
14.10.14 Cached Tables · · · · · 3554
14.10.15 Character Set and Collation · · · · · 3559
14.10.16 Placement Rules in SQL · · · · · 3574
14.10.17 System Tables · · · · · 3583
14.11 UI · · · · · 3708
14.11.1 TiDB Dashboard · · · · · 3708
14.12 Telemetry · · · · · 3855
14.12.1 What is shared? · · · · · 3856
14.12.2 Disable telemetry · · · · · 3858
14.12.3 Check telemetry status · · · · · 3860
14.12.4 Compliance · · · · · 3860
14.13 Error Codes and Troubleshooting · · · · · 3861
14.13.1 Error codes · · · · · 3861
14.13.2 Troubleshooting · · · · · 3872
14.14 Table Filter · · · · · 3872
14.14.1 Usage · · · · · 3872
14.14.2 Syntax · · · · · 3873
14.14.3 Multiple rules · · · · · 3876

14.15 Schedule Replicas by Topology Labels · · · · · 3877
14.15.1 Configure `labels` based on the cluster topology · · · · · 3877
14.15.2 PD schedules based on topology label · · · · · 3882

15 FAQs **3883**

15.1 TiDB FAQ Summary · · · · · 3883
15.2 TiDB Architecture FAQs · · · · · 3884
15.2.1 TiDB introduction and architecture · · · · · 3884
15.2.2 TiDB techniques · · · · · 3887
15.3 SQL FAQs · · · · · 3887
15.3.1 Does TiDB support the secondary key? · · · · · 3888
15.3.2 How does TiDB perform when executing DDL operations on a large table? · · · · · 3888
15.3.3 How to choose the right query plan? Do I need to use hints? Or can I use hints? · · · · · 3888
15.3.4 How to prevent the execution of a particular SQL statement? · · · · · 3888
15.3.5 What are the MySQL variables that TiDB is compatible with? · · · · · 3889
15.3.6 The order of results is different from MySQL when `ORDER BY` is omitted 3889
15.3.7 Does TiDB support `SELECT FOR UPDATE`? · · · · · 3891
15.3.8 Can the codec of TiDB guarantee that the UTF-8 string is memcomparable? Is there any coding suggestion if our key needs to support UTF-8? · · · · · 3891
15.3.9 What is the maximum number of statements in a transaction? · · · · · 3891
15.3.10 Why does the auto-increment ID of the later inserted data is smaller than that of the earlier inserted data in TiDB? · · · · · 3891
15.3.11 How do I modify the `sql_mode` in TiDB? · · · · · 3891
15.3.12 Error: `java.sql.BatchUpdateExecption:statement count 5001 exceeds the transaction limitation` while using Sqoop to write data into TiDB in batches · · · · · 3892
15.3.13 Does TiDB have a function like the Flashback Query in Oracle? Does it support DDL? · · · · · 3892
15.3.14 Does TiDB release space immediately after deleting data? · · · · · 3892
15.3.15 Why does the query speed get slow after data is deleted? · · · · · 3892
15.3.16 What should I do if it is slow to reclaim storage space after deleting data? · · · · · 3893

15.3.17 Does `SHOW PROCESSLIST` display the system process ID? 3893
15.3.18 How to control or change the execution priority of SQL commits? 3893
15.3.19 What's the trigger strategy for `auto analyze` in TiDB? 3894
15.3.20 Can I use optimizer hints to override the optimizer behavior? 3894
15.3.21 Why the `Information schema is changed` error is reported? 3894
15.3.22 What are the causes of the "Information schema is out of date" error? 3895
15.3.23 Error is reported when executing DDL statements under high concurrency? 3896
15.3.24 SQL optimization 3896
15.3.25 Database optimization 3898
15.4 TiDB Deployment FAQs 3898
15.4.1 Software and hardware requirements 3899
15.4.2 Installation and deployment 3899
15.4.3 What public cloud vendors are currently supported by TiDB? 3907
15.5 Migration FAQs 3907
15.5.1 Full data export and import 3908
15.5.2 Migrate the data online 3911
15.5.3 Migrate the traffic 3911
15.6 Upgrade and After Upgrade FAQs 3913
15.6.1 Upgrade FAQs 3913
15.6.2 After upgrade FAQs 3914
15.7 TiDB Monitoring FAQs 3919
15.7.1 Is there a better way of monitoring the key metrics? 3919
15.7.2 The Prometheus monitoring data is deleted every 15 days by default. Could I set it to two months or delete the monitoring data manually? 3919
15.7.3 Region Health monitor 3920
15.7.4 What is the meaning of `selectsimplefull` in Statement Count monitor? 3920
15.7.5 What is the difference between `QPS` and `Statement OPS` in the monitor? 3920
15.8 TiDB Cluster Management FAQs 3920
15.8.1 Daily management 3920
15.8.2 PD management 3924
15.8.3 TiDB server management 3926

15.8.4 TiKV server management · · · · · 3929
15.8.5 TiDB testing · · · · · 3934
15.8.6 Backup and restoration · · · · · 3935
15.9 High Availability FAQs · · · · · 3936
15.9.1 How is TiDB strongly consistent? · · · · · 3936
15.9.2 What's the recommended solution for the deployment of three geo-distributed data centers? · · · · · 3936
15.10 High Reliability FAQs · · · · · 3936
15.10.1 Does TiDB support data encryption? · · · · · 3936
15.10.2 Does TiDB support modifying the MySQL version string of the server to a specific one that is required by the security vulnerability scanning tool? · · · · · 3936
15.10.3 What authentication protocols does TiDB support? What's the process? · · · · · 3937
15.10.4 How to modify the user password and privilege? · · · · · 3937

16 Release Notes 3937

16.1 TiDB Release Notes · · · · · 3937
16.1.1 6.1 · · · · · 3937
16.1.2 6.0 · · · · · 3938
16.1.3 5.4 · · · · · 3938
16.1.4 5.3 · · · · · 3938
16.1.5 5.2 · · · · · 3938
16.1.6 5.1 · · · · · 3938
16.1.7 5.0 · · · · · 3938
16.1.8 4.0 · · · · · 3939
16.1.9 3.1 · · · · · 3939
16.1.10 3.0 · · · · · 3939
16.1.11 2.1 · · · · · 3940
16.1.12 2.0 · · · · · 3941
16.1.13 1.0 · · · · · 3941

16.2 TiDB Release Timeline · · · · · 3942
16.3 TiDB Versioning · · · · · 3945
16.3.1 Release versioning · · · · · 3946
16.3.2 Long-Term Support releases · · · · · 3946
16.3.3 Development Milestone Releases · · · · · 3946
16.3.4 Versioning of TiDB ecosystem tools · · · · · 3947
16.3.5 Historical versioning (deprecated) · · · · · 3947
16.4 TiDB Installation Packages · · · · · 3948
16.4.1 See also · · · · · 3949
16.5 v6.1 · · · · · 3949
16.5.1 TiDB 6.1.7 Release Notes · · · · · 3949
16.5.2 TiDB 6.1.6 Release Notes · · · · · 3953
16.5.3 TiDB 6.1.5 Release Notes · · · · · 3956
16.5.4 TiDB 6.1.4 Release Notes · · · · · 3958
16.5.5 TiDB 6.1.3 Release Notes · · · · · 3961
16.5.6 TiDB 6.1.2 Release Notes · · · · · 3963
16.5.7 TiDB 6.1.1 Release Notes · · · · · 3966
16.5.8 TiDB 6.1.0 Release Notes · · · · · 3971
16.6 v6.0 · · · · · 4023
16.6.1 TiDB 6.0.0 Release Notes · · · · · 4023
16.7 v5.4 · · · · · 4052
16.7.1 TiDB 5.4.3 Release Notes · · · · · 4052
16.7.2 TiDB 5.4.2 Release Notes · · · · · 4055
16.7.3 TiDB 5.4.1 Release Notes · · · · · 4058
16.7.4 TiDB 5.4 Release Notes · · · · · 4064
16.8 v5.3 · · · · · 4092
16.8.1 TiDB 5.3.4 Release Notes · · · · · 4092
16.8.2 TiDB 5.3.3 Release Note · · · · · 4094
16.8.3 TiDB 5.3.2 Release Notes · · · · · 4094
16.8.4 TiDB 5.3.1 Release Notes · · · · · 4099
16.8.5 TiDB 5.3 Release Notes · · · · · 4104

16.9 v5.2 4130
16.9.1 TiDB 5.2.4 Release Notes 4130
16.9.2 TiDB 5.2.3 Release Note 4136
16.9.3 TiDB 5.2.2 Release Notes 4136
16.9.4 TiDB 5.2.1 Release Notes 4140
16.9.5 TiDB 5.2 Release Notes 4141
16.10 v5.1 4157
16.10.1 TiDB 5.1.5 Release Notes 4157
16.10.2 TiDB 5.1.4 Release Notes 4160
16.10.3 TiDB 5.1.3 Release Note 4166
16.10.4 TiDB 5.1.2 Release Notes 4166
16.10.5 TiDB 5.1.1 Release Notes 4171
16.10.6 TiDB 5.1 Release Notes 4175
16.11 v5.0 4194
16.11.1 TiDB 5.0.6 Release Notes 4194
16.11.2 TiDB 5.0.5 Release Note 4199
16.11.3 TiDB 5.0.4 Release Notes 4199
16.11.4 TiDB 5.0.3 Release Notes 4206
16.11.5 TiDB 5.0.2 Release Notes 4209
16.11.6 TiDB 5.0.1 Release Notes 4213
16.11.7 What's New in TiDB 5.0 4215
16.11.8 TiDB 5.0 RC Release Notes 4232
16.12 v4.0 4239
16.12.1 TiDB 4.0.16 Release Notes 4239
16.12.2 TiDB 4.0.15 Release Notes 4242
16.12.3 TiDB 4.0.14 Release Notes 4246
16.12.4 TiDB 4.0.13 Release Notes 4251
16.12.5 TiDB 4.0.12 Release Notes 4255
16.12.6 TiDB 4.0.11 Release Notes 4259
16.12.7 TiDB 4.0.10 Release Notes 4264
16.12.8 TiDB 4.0.9 Release Notes 4266
16.12.9 TiDB 4.0.8 Release Notes 4273

16.12.10 TiDB 4.0.7 Release Notes 4277
16.12.11 TiDB 4.0.6 Release Notes 4279
16.12.12 TiDB 4.0.5 Release Notes 4284
16.12.13 TiDB 4.0.4 Release Notes 4289
16.12.14 TiDB 4.0.3 Release Notes 4289
16.12.15 TiDB 4.0.2 Release Notes 4294
16.12.16 TiDB 4.0.1 Release Notes 4299
16.12.17 TiDB 4.0 GA Release Notes 4300
16.12.18 TiDB 4.0 RC.2 Release Notes 4304
16.12.19 TiDB 4.0 RC.1 Release Notes 4310
16.12.20 TiDB 4.0 RC Release Notes 4314
16.12.21 TiDB 4.0.0 Beta.2 Release Notes 4317
16.12.22 TiDB 4.0.0 Beta.1 Release Notes 4319
16.12.23 TiDB 4.0 Beta Release Notes 4322
16.13 v3.1 4326
16.13.1 TiDB 3.1.2 Release Notes 4326
16.13.2 TiDB 3.1.1 Release Notes 4326
16.13.3 TiDB 3.1.0 GA Release Notes 4328
16.13.4 TiDB 3.1 RC Release Notes 4330
16.13.5 TiDB 3.1 Beta.2 Release Notes 4333
16.13.6 TiDB 3.1 Beta.1 Release Notes 4335
16.13.7 TiDB 3.1 Beta Release Notes 4336
16.14 v3.0 4337
16.14.1 TiDB 3.0.20 Release Notes 4337
16.14.2 TiDB 3.0.19 Release Notes 4339
16.14.3 TiDB 3.0.18 Release Notes 4340
16.14.4 TiDB 3.0.17 Release Notes 4341
16.14.5 TiDB 3.0.16 Release Notes 4342
16.14.6 TiDB 3.0.15 Release Notes 4343
16.14.7 TiDB 3.0.14 Release Notes 4344
16.14.8 TiDB 3.0.13 Release Notes 4348
16.14.9 TiDB 3.0.12 Release Notes 4349

16.14.10TiDB 3.0.11 Release Notes · · · · · 4350
16.14.11TiDB 3.0.10 Release Notes · · · · · 4352
16.14.12TiDB 3.0.9 Release Notes · · · · · 4354
16.14.13TiDB 3.0.8 Release Notes · · · · · 4356
16.14.14TiDB 3.0.7 Release Notes · · · · · 4361
16.14.15TiDB 3.0.6 Release Notes · · · · · 4361
16.14.16TiDB 3.0.5 Release Notes · · · · · 4365
16.14.17TiDB 3.0.4 Release Notes · · · · · 4368
16.14.18TiDB 3.0.3 Release Notes · · · · · 4373
16.14.19TiDB 3.0.2 Release Notes · · · · · 4376
16.14.20TiDB 3.0.1 Release Notes · · · · · 4381
16.14.21TiDB 3.0 GA Release Notes · · · · · 4385
16.14.22TiDB 3.0.0-rc.3 Release Notes · · · · · 4392
16.14.23TiDB 3.0.0-rc.2 Release Notes · · · · · 4396
16.14.24TiDB 3.0.0-rc.1 Release Notes · · · · · 4399
16.14.25TiDB 3.0.0 Beta.1 Release Notes · · · · · 4404
16.14.26TiDB 3.0 Beta Release Notes · · · · · 4407
16.15 v2.1 · · · · · 4411
16.15.1 TiDB 2.1.19 Release Notes · · · · · 4411
16.15.2 TiDB 2.1.18 Release Notes · · · · · 4414
16.15.3 TiDB 2.1.17 Release Notes · · · · · 4417
16.15.4 TiDB 2.1.16 Release Notes · · · · · 4420
16.15.5 TiDB 2.1.15 Release Notes · · · · · 4422
16.15.6 TiDB 2.1.14 Release Notes · · · · · 4424
16.15.7 TiDB 2.1.13 Release Notes · · · · · 4426
16.15.8 TiDB 2.1.12 Release Notes · · · · · 4427
16.15.9 TiDB 2.1.11 Release Notes · · · · · 4428
16.15.10TiDB 2.1.10 Release Notes · · · · · 4429
16.15.11TiDB 2.1.9 Release Notes · · · · · 4430
16.15.12TiDB 2.1.8 Release Notes · · · · · 4432
16.15.13TiDB 2.1.7 Release Notes · · · · · 4434
16.15.14TiDB 2.1.6 Release Notes · · · · · 4435

16.15.15TiDB 2.1.5 Release Notes · · · · · 4436
16.15.16TiDB 2.1.4 Release Notes · · · · · 4438
16.15.17TiDB 2.1.3 Release Notes · · · · · 4439
16.15.18TiDB 2.1.2 Release Notes · · · · · 4441
16.15.19TiDB 2.1.1 Release Notes · · · · · 4442
16.15.20TiDB 2.1 GA Release Notes · · · · · 4443
16.15.21TiDB 2.1 RC5 Release Notes · · · · · 4449
16.15.22TiDB 2.1 RC4 Release Notes · · · · · 4451
16.15.23TiDB 2.1 RC3 Release Notes · · · · · 4452
16.15.24TiDB 2.1 RC2 Release Notes · · · · · 4454
16.15.25TiDB 2.1 RC1 Release Notes · · · · · 4458
16.15.26TiDB 2.1 Beta Release Notes · · · · · 4463
16.16 v2.0 · · · · · 4465
16.16.1 TiDB 2.0.11 Release Notes · · · · · 4465
16.16.2 TiDB 2.0.10 Release Notes · · · · · 4466
16.16.3 TiDB 2.0.9 Release Notes · · · · · 4467
16.16.4 TiDB 2.0.8 Release Notes · · · · · 4468
16.16.5 TiDB 2.0.7 Release Notes · · · · · 4469
16.16.6 TiDB 2.0.6 Release Notes · · · · · 4470
16.16.7 TiDB 2.0.5 Release Notes · · · · · 4472
16.16.8 TiDB 2.0.4 Release Notes · · · · · 4473
16.16.9 TiDB 2.0.3 Release Notes · · · · · 4474
16.16.10TiDB 2.0.2 Release Notes · · · · · 4475
16.16.11TiDB 2.0.1 Release Notes · · · · · 4475
16.16.12TiDB 2.0 Release Notes · · · · · 4476
16.16.13TiDB 2.0 RC5 Release Notes · · · · · 4481
16.16.14TiDB 2.0 RC4 Release Notes · · · · · 4482
16.16.15TiDB 2.0 RC3 Release Notes · · · · · 4483
16.16.16TiDB 2.0 RC1 Release Notes · · · · · 4484
16.16.17TiDB 1.1 Beta Release Notes · · · · · 4485
16.16.18TiDB 1.1 Alpha Release Notes · · · · · 4487

16.17 v1.0 4488
16.17.1 TiDB 1.0.8 Release Notes 4488
16.17.2 TiDB 1.0.7 Release Notes 4489
16.17.3 TiDB 1.0.6 Release Notes 4490
16.17.4 TiDB 1.0.5 Release Notes 4490
16.17.5 TiDB 1.0.4 Release Notes 4491
16.17.6 TiDB 1.0.3 Release Notes 4491
16.17.7 TiDB 1.0.2 Release Notes 4492
16.17.8 TiDB 1.0.1 Release Notes 4493
16.17.9 TiDB 1.0 Release Notes 4493
16.17.10 Pre-GA Release Notes 4499
16.17.11 TiDB RC4 Release Notes 4500
16.17.12 TiDB RC3 Release Notes 4501
16.17.13 TiDB RC2 Release Notes 4503
16.17.14 TiDB RC1 Release Notes 4504

17 Glossary **4506**
17.1 A 4506
17.1.1 ACID 4506
17.2 B 4506
17.2.1 Batch Create Table 4506
17.2.2 Baseline Capturing 4506
17.2.3 Bucket 4506
17.3 C 4507
17.3.1 Cached Table 4507
17.3.2 Continuous Profiling 4507
17.4 D 4507
17.4.1 Dynamic Pruning 4507
17.5 I 4507
17.5.1 Index Merge 4507
17.5.2 In-Memory Pessimistic Lock 4507
17.6 L 4508
17.6.1 leader/follower/learner 4508

17.7 M 4508
17.7.1 Multi-version concurrency control (MVCC) 4508
17.8 O 4508
17.8.1 Old value 4508
17.8.2 Operator 4508
17.8.3 Operator step 4508
17.9 P 4509
17.9.1 pending/down 4509
17.9.2 Predicate columns 4509
17.10 Q 4509
17.10.1 Quota Limiter 4509
17.11 R 4509
17.11.1 Raft Engine 4509
17.11.2 Region/peer/Raft group 4509
17.11.3 Region split 4510
17.11.4 restore 4510
17.12 S 4510
17.12.1 scheduler 4510
17.12.2 Store 4510
17.13 T 4510
17.13.1 Top SQL 4510
17.13.2 TSO 4510

1 Docs Home

2 About TiDB

2.1 TiDB Introduction

TiDB (/'ta di bi:/, "Ti" stands for Titanium) is an open-source distributed SQL database that supports Hybrid Transactional and Analytical Processing (HTAP) workloads. It is MySQL compatible and features horizontal scalability, strong consistency, and high availability. The goal of TiDB is to provide users with a one-stop database solution that covers OLTP (Online Transactional Processing), OLAP (Online Analytical Processing), and HTAP services. TiDB is suitable for various use cases that require high availability and strong consistency with large-scale data.

The following video introduces key features of TiDB.

2.1.1 Key features

- **Horizontally scaling out or scaling in easily**

  The TiDB architecture design of separating computing from storage enables you to separately scale out or scale in the computing or storage capacity online as needed. The scaling process is transparent to application operations and maintenance staff.

- **Financial-grade high availability**

  The data is stored in multiple replicas. Data replicas obtain the transaction log using the Multi-Raft protocol. A transaction can be committed only when data has been successfully written into the majority of replicas. This can guarantee strong consistency, and availability when a minority of replicas go down. To meet the requirements of different disaster tolerance levels, you can configure the geographic location and number of replicas as needed.

- **Real-time HTAP**

  TiDB provides two storage engines: TiKV, a row-based storage engine, and TiFlash, a columnar storage engine. TiFlash uses the Multi-Raft Learner protocol to replicate data from TiKV in real time, ensuring that the data between the TiKV row-based storage engine and the TiFlash columnar storage engine are consistent. TiKV and TiFlash can be deployed on different machines as needed to solve the problem of HTAP resource isolation.

- **Cloud-native distributed database**

  TiDB is a distributed database designed for the cloud, providing flexible scalability, reliability and security on the cloud platform. Users can elastically scale TiDB to meet the requirements of their changing workloads. In TiDB, each piece of data has 3 replicas at least, which can be scheduled in different cloud availability zones to tolerate

the outage of a whole data center. TiDB Operator helps manage TiDB on Kubernetes and automates tasks related to operating the TiDB cluster, which makes TiDB easier to deploy on any cloud that provides managed Kubernetes. TiDB Cloud, the fully-managed TiDB service, is the easiest, most economical, and most resilient way to unlock the full power of TiDB in the cloud, allowing you to deploy and run TiDB clusters with just a few clicks.

- **Compatible with the MySQL 5.7 protocol and MySQL ecosystem**

  TiDB is compatible with the MySQL 5.7 protocol, common features of MySQL, and the MySQL ecosystem. To migrate your applications to TiDB, you do not need to change a single line of code in many cases or only need to modify a small amount of code. In addition, TiDB provides a series of data migration tools to help easily migrate application data into TiDB.

2.1.2 Use cases

- **Financial industry scenarios with high requirements for data consistency, reliability, availability, scalability, and disaster tolerance**

  As we all know, the financial industry has high requirements for data consistency, reliability, availability, scalability, and disaster tolerance. The traditional solution is to provide services in two data centers in the same city, and provide data disaster recovery but no services in a third data center located in another city. This solution has the disadvantages of low resource utilization, high maintenance cost, and the fact that RTO (Recovery Time Objective) and RPO (Recovery Point Objective) cannot meet expectations. TiDB uses multiple replicas and the Multi-Raft protocol to schedule data to different data centers, racks, and machines. When some machines fail, the system can automatically switch to ensure that the system RTO 30 seconds and RPO = 0.

- **Massive data and high concurrency scenarios with high requirements for storage capacity, scalability, and concurrency**

  As applications grow rapidly, the data surges. Traditional standalone databases cannot meet the data capacity requirements. The solution is to use sharding middleware or a distributed SQL database (like TiDB), and the latter is more cost-effective. TiDB adopts a separate computing and storage architecture, which enables you to scale out or scale in the computing or storage capacity separately. The computing layer supports a maximum of 512 nodes, each node supports a maximum of 1,000 concurrencies, and the maximum cluster capacity is at the PB (petabytes) level.

- **Real-time HTAP scenarios**

  With the fast growth of 5G, Internet of Things, and artificial intelligence, the data generated by a company keeps increasing tremendously, reaching a scale of hundreds of TB (terabytes) or even the PB level. The traditional solution is to process online transactional applications using an OLTP database and use an ETL (Extract, Transform, Load) tool to replicate the data into an OLAP database for data analysis. This solution

has multiple disadvantages such as high storage costs and poor real-time performance. TiDB introduces the TiFlash columnar storage engine in v4.0, which combines with the TiKV row-based storage engine to build TiDB as a true HTAP database. With a small amount of extra storage cost, you can handle both online transactional processing and real-time data analysis in the same system, which greatly saves the cost.

- **Data aggregation and secondary processing scenarios**

  The application data of most companies are scattered in different systems. As the application grows, the decision-making leaders need to understand the business status of the entire company to make decisions in time. In this case, the company needs to aggregate the scattered data into the same system and execute secondary processing to generate a T+0 or T+1 report. The traditional solution is to use ETL and Hadoop, but the Hadoop system is complicated, with high operations and maintenance cost and storage cost. Compared with Hadoop, TiDB is much simpler. You can replicate data into TiDB using ETL tools or data migration tools provided by TiDB. Reports can be directly generated using SQL statements.

2.1.3 See also

- TiDB Architecture
- TiDB Storage
- TiDB Computing
- TiDB Scheduling

2.2 TiDB 6.1.0 Release Notes

Release date: June 13, 2022

TiDB version: 6.1.0

Quick access: Quick start | Production deployment

In 6.1.0, the key new features or improvements are as follows:

- List partitioning and list COLUMNS partitioning become GA, compatible with MySQL 5.7
- TiFlash partitioned table (dynamic pruning) becomes GA
- Support user-level lock management, compatible with MySQL
- Support non-transactional DML statements (only support `DELETE`)
- TiFlash supports on-demand data compaction
- MPP introduces the window function framework
- TiCDC supports replicating changelogs to Kafka via Avro
- TiCDC supports splitting large transactions during replication, which significantly reduces replication latency caused by large transactions
- The optimistic mode for merging and migrating sharded tables becomes GA

2.2.1 New Features

2.2.1.1 SQL

- List partitioning and list COLUMNS partitioning become GA. Both are compatible with MySQL 5.7.

  User documents: List partitioning, List COLUMNS partitioning

- TiFlash supports initiating a compact command. (experimental)

  TiFlash v6.1.0 introduces the `ALTER TABLE ... COMPACT` statement, which provides a manual way to compact physical data based on the existing background compaction mechanism. With this statement, you can update data in earlier formats and improve read/write performance any time as appropriate. It is recommended that you execute this statement to compact data after upgrading your cluster to v6.1.0. This statement is an extension of the standard SQL syntax and therefore is compatible with MySQL clients. For scenarios other than TiFlash upgrade, usually there is no need to use this statement.

  User document, #4145

- TiFlash implements the window function framework and supports the following window functions:

  - `RANK()`
  - `DENSE_RANK()`
  - `ROW_NUMBER()`

  User document, #33072

2.2.1.2 Observability

- Continuous Profiling supports the ARM architecture and TiFlash.

  User document

- Grafana adds a Performance Overview dashboard to provide a system-level entry for overall performance diagnosis.

  As a new dashboard in the TiDB visualized monitoring component Grafana, Performance Overview provides a system-level entry for overall performance diagnosis. According to the top-down performance analysis methodology, the Performance Overview dashboard reorganizes TiDB performance metrics based on database time breakdown and displays these metrics in different colors. By checking these colors, you can identify performance bottlenecks of the entire system at the first glance, which significantly reduces performance diagnosis time and simplifies performance analysis and diagnosis.

  User document

2.2.1.3 Performance

- Support customized Region size

  Starting from v6.1.0, you can configure `coprocessor.region-split-size` to set Regions to a larger size. This can effectively reduce the number of Regions, make Regions easier to manage, and improve the cluster performance and stability.

  User document, #11515

- Support using buckets to increase concurrency (experimental)

  To help you further improve the query concurrency after setting Regions to a larger size, TiDB introduces the concept of bucket, which is a smaller range within a Region. Using buckets as the query unit can optimize concurrent query performance when Regions are set to a larger size. Using buckets as the query unit can also dynamically adjust the sizes of hotspot Regions to ensure the scheduling efficiency and load balance. This feature is currently experimental. It is not recommended to use it in production environments.

  User document, #11515

- Use Raft Engine as the default log storage engine

  Since v6.1.0, TiDB uses Raft Engine as the default storage engine for logs. Compared with RocksDB, Raft Engine can reduce TiKV I/O write traffic by up to 40% and CPU usage by 10%, while improving foreground throughput by about 5% and reducing tail latency by 20% under certain loads.

  User document, #95

- Support the join order hint syntax

  - The `LEADING` hint reminds the optimizer to use the specified order as the prefix of join operations. A good prefix of join can quickly reduce the amount of data at the early phase of join and improve the query performance.
  - The `STRAIGHT_JOIN` hint reminds the optimizer to join tables in an order that is consistent with the order of tables in the `FROM` clause.

  This provides a method for you to fix the order of table joins. A proper use of the hints can effectively enhance the SQL performance and cluster stability.

  User document: `LEADING`, `STRAIGHT_JOIN`, #29932

- TiFlash supports four more functions:

  - `FROM_DAYS`
  - `TO_DAYS`
  - `TO_SECONDS`
  - `WEEKOFYEAR`

User document, #4679, #4678, #4677

- TiFlash supports partitioned tables in dynamic pruning mode.

  To enhance performance in OLAP scenarios, dynamic pruning mode is supported for partitioned tables. If your TiDB is upgraded from versions earlier than v6.0.0, it is recommended that you manually update statistics of existing partitioned tables, so as to maximize the performance (not required for new installations or new partitions created after upgrade to v6.1.0).

  User documents: Access partitioned tables in the MPP mode, Dynamic pruning mode, #3873

2.2.1.4 Stability

- Automatic recovery from SST corruption

  When RocksDB detects a damaged SST file in the background, TiKV will try to schedule the affected Peer and recover its data using other replicas. You can set the maximum allowable time for the recovery using the `background-error-recovery-window` parameter. If the recovery operation is not completed within the time window, TiKV will panic. This feature automatically detects and recovers recoverable damaged storage, thus improving the cluster stability.

  User document, #10578

- Support non-transactional DML statement

  In the scenarios of large data processing, a single SQL statement with a large transaction might have a negative impact on the cluster stability and performance. Since v6.1.0, TiDB supports providing a syntax in which a `DELETE` statement is split into multiple statements for batch processing. The split statements compromise transactional atomicity and isolation but greatly improve the cluster stability. For detailed syntax, see `BATCH`.

  User document

- TiDB supports configuring the maximum GC wait time

  The transaction of TiDB adopts the Multi-Version Concurrency Control (MVCC) mechanism. When the newly written data overwrites the old data, the old data is not replaced, and both versions of data are stored. The old data is cleaned up by the Garbage Collection (GC) task periodically, which helps reclaim storage space to improve the performance and stability of the cluster. GC is triggered every 10 minutes by default. To ensure that long-running transactions can access the corresponding historical data, when there are transactions in execution, the GC task is delayed. To ensure that the GC task is not delayed indefinitely, TiDB introduces the system variable `tidb_gc_max_wait_time` to control the maximum delay time of the GC task. If the maximum delay time is exceeded, the GC will be forcibly executed. The default value of the variable is 24 hours. This feature enables you to control the relationship

between the GC waiting time and the long-running transaction, which improves the stability of the cluster.

User document

- TiDB supports configuring the maximum execution time for automatic statistics collection tasks

  Databases can effectively understand the distribution of data by collecting statistics, which helps generate reasonable execution plans and improve the efficiency of SQL execution. TiDB regularly collects statistics on frequently changed data objects in the background. However, collecting statistics takes up cluster resources and might affect the stable operation of the business during business peaks.

  Starting from v6.1.0, TiDB introduces `tidb_max_auto_analyze_time` to control the maximum execution time for background statistics collection, which is 12 hours by default. When the application does not encounter a resource bottleneck, it is recommended not to modify this variable so that TiDB can timely collect statistics.

  User document

2.2.1.5 Ease of use

- Support a one-stop online data recovery when multiple replicas are lost

  Before TiDB v6.1.0, when multiple Region replicas are lost because of machine failure, users have to stop all TiKV servers and use TiKV Control to recover TiKV one by one. Since TiDB v6.1.0, the recovery process is fully automated, does not require to stop TiKV, and does not affect other applications online. The recovery process can be triggered using PD Control and provides a more user-friendly summary information.

  User document, #10483

- Support viewing history statistics collection tasks

  You can use the `SHOW ANALYZE STATUS` statement to show cluster-level statistics collection tasks. Before TiDB v6.1.0, the `SHOW ANALYZE STATUS` statement shows instance-level tasks only, and history task records are cleared after a TiDB restart. Therefore, you cannot view history statistics collection time and details. Starting from TiDB v6.1.0, history records of statistics collection tasks are persisted and can be queried after a cluster restart, which provides a reference for troubleshooting query performance issues caused by statistics anomalies.

  User document

- Support modifying TiDB, TiKV, and TiFlash configurations dynamically

In earlier TiDB versions, after modifying a configuration item, you must restart the cluster to make the modification effective. This might interrupt online services. To address this issue, TiDB v6.1.0 introduces the dynamic configuration feature, which allows you to validate a parameter change without restarting the cluster. The specific optimizations are as follows:

```
* Transform some TiDB configuration items to system variables, so that they
    ↪ can be modified dynamically and persisted. Note that the original
    ↪ configuration items are deprecated after transformation. For a
    ↪ detailed list of the transformed configuration items, see [
    ↪ Configuration file parameters](#configuration-file-parameters).
* Support configuring some TiKV parameters online. For a detailed list of
    ↪ the parameters, see [Others](#others).
* Transform the TiFlash configuration item `max_threads` to a system
    ↪ variable `tidb_max_tiflash_threads`, so that the configuration can be
    ↪  modified dynamically and persisted. Note that the original
    ↪ configuration item remains after transformation.
```

For v6.1.0 clusters upgraded (including online and offline upgrades) from earlier versions, note that:

```
* If the configuration items specified in the configuration file before the
    ↪ upgrade already exist, TiDB will automatically update the values of
    ↪ the configured items to those of the corresponding system variables
    ↪ during the upgrade process. In this way, after the upgrade, the
    ↪ system behavior is not affected by parameter optimization.
* The automatic update mentioned above occurs only once during the upgrade.
    ↪ After the upgrade, the deprecated configuration items are no longer
    ↪ effective.
```

This feature allows you to modify parameters dynamically, and validate and persist them, instead of restarting the system and interrupting services. This makes your daily maintenance easier.

User document

- Support killing queries or connections globally

  You can control the Global Kill feature using the `enable-global-kill` configuration (enabled by default).

  Before TiDB v6.1.0, when an operation consumes a lot of resources and causes cluster stability issues, you have to connect to the target TiDB instance and then run the `KILL TIDB ${id};` command to terminate the target connection and operation. In the case of many TiDB instances, this method is not easy to use and prone to wrong operations. Starting from v6.1.0, the `enable-global-kill` configuration is introduced and enabled by default. You can run the kill command in any TiDB instance to terminate a specified connection and operation, without worrying about incorrectly terminating other queries or sessions by mistake when there is a proxy between the client and TiDB. Currently, TiDB does not support using Ctrl+C to terminate queries or sessions.

  User document, #8854

- TiKV API V2 (experimental)

  Before v6.1.0, when TiKV is used as Raw Key Value storage, TiKV only provides basic Key Value read and write capability because it only stores the raw data passed in by the client.

  TiKV API V2 provides a new Raw Key Value storage format and access interface, including:

  - The data is stored in MVCC and the change timestamp of the data is recorded. This feature will lay the foundation for implementing Change Data Capture and incremental backup and restore.
  - Data is scoped according to different usage and supports co-existence of a single TiDB cluster, Transactional KV, RawKV applications.

Due to significant changes in the underlying storage format, after enabling API V2, you cannot roll back a TiKV cluster to a version earlier than v6.1.0. Downgrading TiKV might result in data corruption.

```
[User document](#api-version-new-in-v610), [#11745](https://github.com/tikv/
    ↪ tikv/issues/11745)
```

2.2.1.6 MySQL compatibility

- Support compatibility with user-level lock management with MySQL

  User-level locks are a user-named lock management system provided by MySQL through built-in functions. The locking functions can provide lock blocking, waiting, and other lock management capabilities. User-level locks are also widely used in ORM frameworks, such as Rails, Elixir, and Ecto. Since v6.1.0, TiDB has supported MySQL-compatible user-level lock management, and supports `GET_LOCK`, `RELEASE_LOCK`, and `RELEASE_ALL_LOCKS` functions.

  User document, #14994

2.2.1.7 Data migration

- The optimistic mode for merging and migrating sharded tables becomes GA

  DM adds a large number of scenario tests for tasks that merge and migrate data from sharded tables in the optimistic mode, which covers 90% of the daily use scenarios. Compared with the pessimistic mode, the optimistic mode is simpler and more efficient to use. It is recommended to use the optimistic mode preferably after you are familiar with the usage notes.

  User document

- DM WebUI supports starting a task according to the specified parameters

  When starting a migration task, you can specify a start time and a safe mode duration. This is especially useful when you create an incremental migration task with lots of sources, eliminating the need to specify the binlog start position specifically for each source.

  User document, #5442

2.2.1.8 TiDB data share subscription

- TiDB supports data sharing with various third-party data ecosystems
  - TiCDC supports sending TiDB incremental data to Kafka in the Avro format, allowing data sharing with third-parties, such as KSQL and Snowflake via Confluent.

    User document, #5338
  - TiCDC supports dispatching incremental data from TiDB to different Kafka topics by table, which, combined with the Canal-json format, allows sharing data directly with Flink.

    User document, #4423
  - TiCDC supports SASL GSSAPI authentication types and adds SASL authentication examples using Kafka.

    User document, #4423
- TiCDC supports replicating `charset=GBK` tables.

  User document, #4806

2.2.2 Compatibility changes

2.2.2.1 System variables

Variable name	Change type	Description
`tidb_enable_clustered_index`	Modified	The default value is changed from `OFF` to `ON`.

Variable name	Change type	Description
tidb_mem_quota_query	Modified	This variable adds the GLOBAL scope, and the variable value persists to the cluster.

Variable name	Change type	Description
tidb_query_log_max_len	Modified	The variable scope is changed from INSTANCE to GLOBAL. The variable value persists to the cluster, and the value range is changed to [0, 1073741824].

Variable name	Change type	Description
`require_secure_transport`	Newly added	This setting was previously a `tidb.toml` option (`security.require-secure-transport`), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_committer_concurrency	Newly added	This setting was previously a `tidb.toml` option (`performance.committer-concurrency`), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_enable_auto_analyze	Newly added	This setting was previously a `tidb.toml` option (`run-auto-analyze`), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_enable_only_full_group_by_check ↪	Newly added	This variable controls the behavior when TiDB performs the ONLY_FULL_GROUP_BY ↪ check.

Variable name	Change type	Description
tidb_enable_outer_join_reorder ↪	Newly added	Since v6.1.0, the Join Reorder algorithm of TiDB supports Outer Join. This variable controls the support behavior, and the default value is ON.

Variable name	Change type	Description
tidb_enable_prepared_plan_cache	Newly added	This setting was previously a tidb. toml option (prepared - plan - cache . enabled), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_gc_max_wait_time ↪	Newly added	This variable is used to set the maximum time of GC safe point blocked by uncommitted transactions.

Variable name	Change type	Description
tidb_max_auto_analyze_time	Newly added	This variable is used to specify the maximum execution time of auto analyze.
tidb_max_tiflash_threads ↪	Newly added	This variable is used to set the maximum concurrency for TiFlash to execute a request.

Variable name	Change type	Description
tidb_memory_oom_action ↪	Newly added	This setting was previously a tidb ↪ . ↪ toml ↪ option (oom ↪ - ↪ action ↪), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_mem_quota_analyze ↪	Newly added	This variable controls the maximum memory usage when TiDB updates statistics, including manually executed ANALYZE ↪ ↪ TABLE ↪ by users and automatic analyze tasks in the TiDB background.

Variable name	Change type	Description
tidb_non_transactional_ignore_error	Newly added	This variable specifies whether to return error immediately when an error occurs in a non-transactional DML statement.

Variable name	Change type	Description
tidb_prepared_plan_cache_memory_guard_ratio	Newly added	This setting was previously a tidb. toml option (prepared - plan - cache . memory - guard - ratio), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_prepared_plan_cache_size	Newly added	This setting was previously a tidb. toml option (prepared - plan - cache . capacity), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_stats_cache_mem_quota ↪	Newly added	This variable sets the memory quota for the TiDB statistics cache.

2.2.2.2 Configuration file parameters

Configuration file	Configuration	Change type	Description
TiDB	`committer-concurrency`	Deleted	Replaced by the system variable `tidb_committer_concurrency`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`lower_case_table_names`	Deleted	Currently TiDB only supports `lower_case_table_name =2`. If another value is set, after the cluster is upgraded to v6.1.0, the value is lost.

Configuration file	Configuration	Change type	Description
TiDB	`mem-quota-query`	Deleted	Replaced by the system variable `tidb_mem_quota_query`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`oom-action`	Deleted	Replaced by the system variable `tidb_mem_oom_action`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`prepared-plan-cache`	Deleted	Replaced by the system variable `tidb_prepared_plan_cache_size`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`prepared-plan-cache.enabled`	Deleted	Replaced by the system variable `tidb_enable_prepared_plan_cache`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration item	Change type	Description
TiDB	`query-log-max-len`	Deleted	Replaced by the system variable `tidb_query_log_max_len`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration item	Change type	Description
TiDB	`require`	Deleted	Replaced by the system variable `require_secure_transport`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.
	`-`		
	`secure`		
	`-`		
	`transport`		

Configuration file	Configuration item	Change type	Description
TiDB	`run-auto-analyze`	Deleted	Replaced by the system variable `tidb_enable_auto_analyze`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`enable-global-kill`	Newly added	Controls whether to enable the Global Kill (terminating queries or connections across instances) feature. When the value is `true`, both `KILL` and `KILL TIDB` statements can terminate queries or connections

Configuration file	Configuration	Change type	Description
TiDB	`enable-stats-cache`	Newly added	Controls whether to enable the memory quota for the statistics cache.
TiKV	`raft.engine.enable`	Modified	The default value is changed from `FALSE` to `TRUE`.

Configuration file	Configuration	Change type	Description
TiKV	`region`	Modified	The default value is changed from 1440000 to `region` ↪ `-` ↪ `split` ↪ `-` ↪ `keys` ↪ ↪ `/` ↪ ↪ `2` ↪ ↪ `*` ↪ ↪ `3` ↪ `.`

Configuration file	Configuration	Change type	Description
TiKV	`region-max-size`	Modified	The default value is changed from 144 MB to `region-split-size / 2 * 3`.
TiKV	`coprocessor.enable-region-bucket`	Newly added	Determines whether to divide a Region into smaller ranges called buckets.

Configuration file	Configuration	Change type	Description
TiKV	coprocessor.region-bucket-size	Newly added	The size of a bucket when enable is true.
TiKV	causal-ts.renew-batch-min-size	Newly added	The minimum number of locally cached timestamps.
TiKV	causal-ts.renew-interval	Newly added	The interval at which the locally cached timestamps are refreshed.

Configuration file	Configuration	Change type	Description
TiKV	`max-snapshot-file-raw-size`	Newly added	The snapshot file will split to multiple files when the snapshot file size exceeds this value.
TiKV	`raft-engine.memory-limit`	Newly added	Specifies the limit on the memory usage of Raft Engine.

Configuration file	Configuration	Change type	Description
TiKV	`storage.background-error-recovery-window`	Newly added	The maximum recovery time is allowed after RocksDB detects a recoverable background error.

Configuration file	Configuration item	Change type	Description
TiKV	`storage.api-version`	Newly added	The storage format and interface version used by TiKV when TiKV serves as the raw key-value store.
PD	`scheduler.max-store-preparing-time`	Newly added	Controls the maximum waiting time for the store to go online.

Configuration file	Configuration	Change type	Description
TiCDC	`enable-tls`	Newly added	Whether to use TLS to connect to the downstream Kafka instance.

Configuration file	Configuration item	Change type	Description
TiCDC	`sasl-gssapi-service-name`	Newly added	Used to support SASL/GSSAPI authentication for Kafka. For details, see Configure sink URI with `kafka` .
	`sasl-gssapi-realm`		
	`sasl-gssapi-key`		
	`sasl-gssapi-keytab-path`		
	`sasl-gssapi-kerberos-config-path`		

Configuration file	Configuration	Change type	Description
TiCDC	`avro-decimal-handling-mode`	Newly added	Determines the output details of Avro format.
TiCDC	`dispatchers.topic`	Newly added	Controls how TiCDC dispatches incremental data to different Kafka topics.

Configuration file	Configuration	Change type	Description
TiCDC	`dispatchers.partition`	Newly added	`dispatchers.partition` is an alias for `dispatchers.dispatcher`. Controls how TiCDC dispatches incremental data to Kafka partitions.
TiCDC	`schema-registry`	Newly added	Specifies the schema registry endpoint that stores Avro schema.

Configuration file	Configuration	Change type	Description
DM	`worker`	Deleted	This parameter in the `dmctl` start - relay command
DM	`relay` - `dir`	Deleted	Replaced by the same configuration item in the worker configuration file.

Configuration file	Configuration item	Change type	Description
DM	`is-sharding` in the task configuration file	Deleted	Replaced by the `shard-mode` configuration item.
DM	`auto-fix-gtid` in the task configuration file	Deleted	Deprecated in v5.x and officially deleted in v6.1.0.

Configuration file	Configuration	Change type	Description
DM	`meta` ↪ `-` ↪ `dir` ↪ and `charset` ↪ in the source configuration file	Deleted	Deprecated in v5.x and officially deleted in v6.1.0.

2.2.2.3 Others

- Enable Prepared Plan Cache by default

  Prepared Plan Cache is enabled by default in new clusters to cache the execution plans for `Prepare` / `Execute` requests. In the subsequent execution, query plan optimization can be skipped and thus leads to a performance boost. Upgraded clusters inherit the configuration from the configuration file. New clusters use the new default values, which means Prepared Plan Cache is enabled by default and each session can cache 100 plans at most (`capacity=100`). For the memory consumption of this feature, see memory management of Prepared Plan Cache.

- Prior to TiDB v6.1.0, `SHOW ANALYZE STATUS` shows instance-level tasks and the task records are cleared after TiDB restarts. Since TiDB v6.1.0, `SHOW ANALYZE STATUS` ↪ shows cluster-level tasks, and the task records persist after the restart. When `tidb_analyze_version = 2`, the `Job_info` column adds the `analyze option` information.

- Damaged SST files in TiKV might cause the TiKV process to panic. Before TiDB v6.1.0, damaged SST files caused TiKV to panic immediately. Since TiDB v6.1.0, the TiKV process will panic 1 hour after SST files are damaged.

- The following TiKV configuration items support modifying values dynamically:

  - `raftstore.raft-entry-max-size`
  - `quota.foreground-cpu-time`

- `quota.foreground-write-bandwidth`
- `quota.foreground-read-bandwidth`
- `quota.max-delay-duration`
- `server.grpc-memory-pool-quota`
- `server.max-grpc-send-msg-len`
- `server.raft-msg-max-batch-size`

- In v6.1.0, some configuration file parameters are converted to system variables. For v6.1.0 clusters upgraded (including online and offline upgrades) from earlier versions, note that:

    - If the configuration items specified in the configuration file before the upgrade already exist, TiDB will automatically update the values of the configured items to those of the corresponding system variables during the upgrade process. In this way, after the upgrade, the system behavior does not change thanks to parameter optimization.
    - The automatic update mentioned above occurs only once during the upgrade. After the upgrade, the deprecated configuration items are no longer effective.

- The Dashboard page is removed from DM WebUI.

- When `dispatchers.topic` and `dispatchers.partition` are enabled, TiCDC cannot be downgraded to versions earlier than v6.1.0.

- TiCDC Changefeed using the Avro protocol cannot be downgraded to versions earlier than v6.1.0.

2.2.3 Improvements

- TiDB

    - Improve the performance of the `UnionScanRead` operator #32433
    - Improve the display of task types in the output of `EXPLAIN` (add the MPP task type) #33332
    - Support using `rand()` as the default value of a column #10377
    - Support using `uuid()` as the default value of a column #33870
    - Support modifying the character set of columns from `latin1` to `utf8`/`utf8mb4` #34008

- TiKV

    - Improve the old value hit rate of CDC when using in-memory pessimistic lock #12279
    - Improve the health check to detect unavailable Raftstore, so that the TiKV client can update Region Cache in time #12398
    - Support setting memory limit on Raft Engine #12255

- TiKV automatically detects and deletes the damaged SST files to improve the product availability #10578
- CDC supports RawKV #11965
- Support splitting a large snapshot file into multiple files #11595
- Move the snapshot garbage collection from Raftstore to background thread to prevent snapshot GC from blocking Raftstore message loops #11966
- Support dynamic setting of the the maximum message length (`max-grpc-send-msg-len`) and the maximum batch size of gPRC messages (`raft-msg-max-batch-size`) #12334
- Support executing online unsafe recovery plan through Raft #10483

- PD

  - Support time-to-live (TTL) for region labels #4694
  - Support Region Buckets #4668
  - Disable compiling swagger server by default #4932

- TiFlash

  - Optimize memory calculation for an aggregate operator so that a more efficient algorithm is used in the merge phase #4451

- Tools

  - Backup & Restore (BR)
    - Support backing up and restoring empty databases #33866
  - TiDB Lightning
    - Optimize Scatter Region to batch mode to improve the stability of the Scatter Region process #33618
  - TiCDC
    - TiCDC supports splitting large transactions during replication, which significantly reduces replication latency caused by large transactions #5280

2.2.4 Bug fixes

- TiDB

  - Fix the issue of possible panic that might occur when the `in` function processes the `bit` type data #33070
  - Fix the issue of wrong query result because the `UnionScan` operator cannot maintain the order #33175
  - Fix the issue that the Merge Join operator gets wrong results in certain cases #33042

- Fix the issue that the `index join` result might be wrong in the dynamic pruning mode #33231
- Fix the issue that data might not be garbage-collected when some partitions of a partitioned table is dropped #33620
- Fix the issue that some DDL statements might be stuck for a period after the PD node of a cluster is replaced #33908
- Fix the issue that the TiDB server might run out of memory when the `INFORMATION_SCHEMA.CLUSTER_SLOW_QUERY` table is queried. This issue can be triggered when you check slow queries on the Grafana dashboard #33893
- Fix the issue that the system variable `max_allowed_packet` does not take effect #31422
- Fix the issue of memory leak in the TopSQL module #34525 #34502
- Fix the issue that the Plan Cache might be wrong on the PointGet plan #32371
- Fix the issue that query result might be wrong when Plan Cache is started in the RC isolation level #34447

- TiKV

  - Fix the issue that the Raft log lag is increasing when a TiKV instance is taken offline #12161
  - Fix the issue that TiKV panics and destroys peers unexpectedly because the target Region to be merged is invalid #12232
  - Fix the issue that TiKV reports the `failed to load_latest_options` error when upgrading from v5.3.1 or v5.4.0 to v6.0.0 or later versions #12269
  - Fix the issue of OOM caused by appending Raft logs when the memory resource is insufficient #11379
  - Fix the issue of TiKV panic caused by the race between destroying peers and batch splitting Regions #12368
  - Fix the issue of TiKV memory usage spike in a short time after `stats_monitor` falls into a dead loop #12416
  - Fix the issue that TiKV reports the `invalid store ID 0` error when using Follower Read #12478

- PD

  - Fix the wrong status code of `not leader` #4797
  - Fix a bug of TSO fallback in some corner cases #4884
  - Fix the issue that a removed tombstone store appears again after the PD leader transfer #4941
  - Fix the issue that scheduling cannot start immediately after the PD leader transfer #4769

- TiDB Dashboard

  - Fix a bug that Top SQL cannot collect the CPU overhead of the SQL statements that were running before the Top SQL feature is enabled #33859

- TiFlash
    - Fix potential data inconsistency after a lot of INSERT and DELETE operations #4956
- Tools
    - TiCDC
        - Fix excessive memory usage by optimizing the way DDL schemas are buffered #1386
        - Fix data loss that occurs in special incremental scanning scenarios #5468
    - TiDB Data Migration (DM)
        - Fix the `start-time` time zone issue and change DM behavior from using the downstream time zone to using the upstream time zone #5271
        - Fix the issue that DM occupies more disk space after the task automatically resumes #3734 #5344
        - Fix the problem that checkpoint flush may cause the data of failed rows to be skipped #5279
        - Fix the issue that in some cases manually executing the filtered DDL in the downstream might cause task resumption failure #5272
        - Fix an issue that the uppercase table cannot be replicated when `case-sensitive: true` is not set #5255
        - Fix the DM worker panic issue that occurs when the primary key is not first in the index returned by the `SHOW CREATE TABLE` statement #5159
        - Fix the issue that CPU usage may increase and a large amount of log is printed when GTID is enabled or when the task is automatically resumed #5063
        - Fix the offline option and other usage issues in DM WebUI #4993
        - Fix the issue that incremental tasks fail to start when GTID is empty in the upstream #3731
        - Fix the issue that empty configurations may cause dm-master to panic #3732
    - TiDB Lightning
        - Fix the issue that the precheck does not check local disk resources and cluster availability #34213
        - Fix the issue of incorrect routing for schemas #33381
        - Fix the issue that the PD configuration is not restored correctly when TiDB Lightning panics #31733
        - Fix the issue of Local-backend import failure caused by out-of-bounds data in the `auto_increment` column #29737
        - Fix the issue of local backend import failure when the `auto_random` or `auto_increment` column is null #34208

2.3 TiDB Features

This document lists the features supported in each TiDB version. Note that supports for experimental features might change before the final release.

2.3.1 Data types, functions, and operators

Data types, functions, and operators	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Numeric types	Y	Y	Y	Y	Y	Y	Y	Y
Date and time types	Y	Y	Y	Y	Y	Y	Y	Y
String types	Y	Y	Y	Y	Y	Y	Y	Y
JSON type	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
Control flow functions	Y	Y	Y	Y	Y	Y	Y	Y
String functions	Y	Y	Y	Y	Y	Y	Y	Y
Numeric functions and operators	Y	Y	Y	Y	Y	Y	Y	Y
Date and time functions	Y	Y	Y	Y	Y	Y	Y	Y
Bit functions and operators	Y	Y	Y	Y	Y	Y	Y	Y
Cast functions and operators	Y	Y	Y	Y	Y	Y	Y	Y
Encryption and compression functions	Y	Y	Y	Y	Y	Y	Y	Y
Information functions	Y	Y	Y	Y	Y	Y	Y	Y
JSON functions	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
Aggregation functions	Y	Y	Y	Y	Y	Y	Y	Y
Window functions	Y	Y	Y	Y	Y	Y	Y	Y
Miscellaneous functions	Y	Y	Y	Y	Y	Y	Y	Y
Operators	Y	Y	Y	Y	Y	Y	Y	Y
Character sets and collations [1]	Y	Y	Y	Y	Y	Y	Y	Y
User-level lock	Y	N	N	N	N	N	N	N

2.3.2 Indexing and constraints

Indexing and constraints	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Expression indexes	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
Columnar storage (TiFlash)	Y	Y	Y	Y	Y	Y	Y	Y
RocksDB engine	Y	Y	Y	Y	Y	Y	Y	Y
Titan plugin	Y	Y	Y	Y	Y	Y	Y	Y
Invisible indexes	Y	Y	Y	Y	Y	Y	Y	N

[1] TiDB incorrectly treats latin1 as a subset of utf8. See TiDB #18955 for more details.

Indexing and constraints	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Composite `PRIMARY KEY`	Y	Y	Y	Y	Y	Y	Y	Y
Unique indexes	Y	Y	Y	Y	Y	Y	Y	Y
Clustered index on integer `PRIMARY KEY`	Y	Y	Y	Y	Y	Y	Y	Y
Clustered index on composite or non-integer key	Y	Y	Y	Y	Y	Y	Y	N

2.3.3 SQL statements

SQL statements [2]	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Basic `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `REPLACE`	Y	Y	Y	Y	Y	Y	Y	Y
`INSERT ON DUPLICATE KEY UPDATE`	Y	Y	Y	Y	Y	Y	Y	Y
`LOAD DATA INFILE`	Y	Y	Y	Y	Y	Y	Y	Y
`SELECT INTO OUTFILE`	Y	Y	Y	Y	Y	Y	Y	Y
`INNER JOIN`, LEFT|RIGHT [OUTER] JOIN	Y	Y	Y	Y	Y	Y	Y	Y
`UNION`, `UNION ALL`	Y	Y	Y	Y	Y	Y	Y	Y
`EXCEPT` and `INTERSECT` operators	Y	Y	Y	Y	Y	Y	Y	N
`GROUP BY`, `ORDER BY`	Y	Y	Y	Y	Y	Y	Y	Y
Window Functions	Y	Y	Y	Y	Y	Y	Y	Y
Common Table Expressions (CTE)	Y	Y	Y	Y	Y	Y	N	N
`START TRANSACTION`, `COMMIT`, `ROLLBACK`	Y	Y	Y	Y	Y	Y	Y	Y
`EXPLAIN`	Y	Y	Y	Y	Y	Y	Y	Y
`EXPLAIN ANALYZE`	Y	Y	Y	Y	Y	Y	Y	Y
User-defined variables	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
`BATCH [ON COLUMN] LIMIT INTEGER DELETE`	Y	N	N	N	N	N	N	N
Table Lock	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental

2.3.4 Advanced SQL features

Advanced SQL features	6.1	6.0	5.4	5.3	5.2	5.1
Prepared statement cache	Y	Y	Y	Y	Experimental	Experim
SQL plan management (SPM)	Y	Y	Y	Y	Y	Y

Advanced SQL features	6.1	6.0	5.4	5.3	5.2	5.1
Coprocessor cache	Y	Y	Y	Y	Y	Y
Stale Read	Y	Y	Y	Y	Y	Y
Follower reads	Y	Y	Y	Y	Y	Y
Read historical data (tidb_snapshot)	Y	Y	Y	Y	Y	Y
Optimizer hints	Y	Y	Y	Y	Y	Y
MPP Execution Engine	Y	Y	Y	Y	Y	Y
Index Merge	Y	Y	Y	Experimental	Experimental	Experim
Placement Rules in SQL	Y	Y	Experimental	Experimental	N	N

2.3.5 Data definition language (DDL)

Data definition language (DDL)	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Basic `CREATE`, `DROP`, `ALTER`, `RENAME`, `TRUNCATE`	Y	Y	Y	Y	Y	Y	Y	Y
Generated columns	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
Views	Y	Y	Y	Y	Y	Y	Y	Y
Sequences	Y	Y	Y	Y	Y	Y	Y	Y
Auto increment	Y	Y	Y	Y	Y	Y	Y	Y
Auto random	Y	Y	Y	Y	Y	Y	Y	Y
DDL algorithm assertions	Y	Y	Y	Y	Y	Y	Y	Y
Multi-schema change: add columns	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
Change column type	Y	Y	Y	Y	Y	Y	N	N
Temporary tables	Y	Y	Y	Y	N	N	N	N

2.3.6 Transactions

Transactions	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Async commit	Y	Y	Y	Y	Y	Y	Y	N
1PC	Y	Y	Y	Y	Y	Y	Y	N
Large transactions (10GB)	Y	Y	Y	Y	Y	Y	Y	Y
Pessimistic transactions	Y	Y	Y	Y	Y	Y	Y	Y
Optimistic transactions	Y	Y	Y	Y	Y	Y	Y	Y
Repeatable-read isolation (snapshot isolation)	Y	Y	Y	Y	Y	Y	Y	Y
Read-committed isolation	Y	Y	Y	Y	Y	Y	Y	Y

2.3.7 Partitioning

Partitioning	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Range partitioning	Y	Y	Y	Y	Y	Y	Y	Y
Hash partitioning	Y	Y	Y	Y	Y	Y	Y	Y
List partitioning	Y	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	N
List COLUMNS partitioning	Y	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	N
`EXCHANGE PARTITION`	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	N
Dynamic Pruning	Y	Experimental	Experimental	Experimental	Experimental	Experimental	N	N

2.3.8 Statistics

Statistics	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
CMSketch	Disabled by default	Disabled by default	Disabled by default	Disabled by default	Y	Y	Y	Y
Histograms	Y	Y	Y	Y	Y	Y	Y	Y
Extended statistics	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	N
Statistics feedback	Deprecated	Deprecated	Deprecated	Experimental	Experimental	Experimental	Experimental	Experimental
Automatically update statistics	Y	Y	Y	Y	Y	Y	Y	Y
Dynamic pruning	Y	Experimental	Experimental	Experimental	Experimental	Experimental	N	N

2.3.9 Security

Security	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Transparent layer security (TLS)	Y	Y	Y	Y	Y	Y	Y	Y
Encryption at rest (TDE)	Y	Y	Y	Y	Y	Y	Y	Y
Role-based authentication (RBAC)	Y	Y	Y	Y	Y	Y	Y	Y
Certificate-based authentication	Y	Y	Y	Y	Y	Y	Y	Y
`caching_sha2_password` authentication	Y	Y	Y	Y	Y	N	N	N
MySQL compatible `GRANT` system	Y	Y	Y	Y	Y	Y	Y	Y
Dynamic Privileges	Y	Y	Y	Y	Y	Y	N	N
Security Enhanced Mode	Y	Y	Y	Y	Y	Y	N	N
Redacted Log Files	Y	Y	Y	Y	Y	Y	Y	N

2.3.10 Data import and export

Data import and export	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Fast Importer (TiDB Lightning)	Y	Y	Y	Y	Y	Y	Y	Y
mydumper logical dumper	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated

Data import and export	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
Dumpling logical dumper	Y	Y	Y	Y	Y	Y	Y	Y
Transactional `LOAD DATA`	Y	Y	Y	Y	Y	Y	Y	N [3]
Database migration toolkit (DM)	Y	Y	Y	Y	Y	Y	Y	Y
TiDB Binlog	Y	Y	Y	Y	Y	Y	Y	Y
Change data capture (CDC)	Y	Y	Y	Y	Y	Y	Y	Y

2.3.11 Management, observability, and tools

Management, observability, and tools	6.1	6.0	5.4	5.3	5.2	5.1	5.0	4.0
TiDB Dashboard UI	Y	Y	Y	Y	Y	Y	Y	Y
TiDB Dashboard Continuous Profiling	Y	Y	Experimental	Experimental	N	N	N	N
TiDB Dashboard Top SQL	Y	Y	Experimental	N	N	N	N	N
TiDB Dashboard SQL Diagnostics	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
Information schema	Y	Y	Y	Y	Y	Y	Y	Y
Metrics schema	Y	Y	Y	Y	Y	Y	Y	Y
Statements summary tables	Y	Y	Y	Y	Y	Y	Y	Y
Slow query log	Y	Y	Y	Y	Y	Y	Y	Y
TiUP deployment	Y	Y	Y	Y	Y	Y	Y	Y
Ansible deployment	N	N	N	N	N	N	N	Deprecated
Kubernetes operator	Y	Y	Y	Y	Y	Y	Y	Y
Built-in physical backup	Y	Y	Y	Y	Y	Y	Y	Y
Global Kill	Y	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
Lock View	Y	Y	Y	Y	Y	Experimental	Experimental	Experimental
`SHOW CONFIG`	Y	Y	Y	Y	Y	Y	Y	Y
`SET CONFIG`	Y	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental	Experimental
DM WebUI	Experimental	Experimental	N	N	N	N	N	N

2.4 TiDB Experimental Features

This document introduces the experimental features of TiDB in different versions. It is **NOT** recommended to use these features in the production environment.

2.4.1 Performance

- Support collecting statistics for `PREDICATE COLUMNS` (Introduced in v5.4)
- Support synchronously loading statistics. (Introduced in v5.4)

[3] For TiDB v4.0, the `LOAD DATA` transaction does not guarantee atomicity.

- Control the memory quota for collecting statistics. (Introduced in v6.1.0)
- Extended statistics. (Introduced in v5.0.0)
- Randomly sample about 10000 rows of data to quickly build statistics (Introduced in v3.0)

2.4.2 Stability

- Improve the stability of the optimizer's choice of indexes: extend the statistics feature by collecting the multi-column order dependency information (Introduced in v5.0).
- When TiKV is deployed with limited resources, if the foreground of TiKV processes too many read and write requests, the CPU resources used by the background are occupied to help process such requests, which affects the performance stability of TiKV. To avoid this situation, you can use the Quota Limiter to limit the CPU resources to be used by the foreground. (Introduced in v6.0)

2.4.3 Scheduling

Elastic scheduling feature. It enables the TiDB cluster to dynamically scale out and in on Kubernetes based on real-time workloads, which effectively reduces the stress during your application's peak hours and saves overheads. See Enable TidbCluster Auto-scaling for details. (Introduced in v4.0)

2.4.4 SQL

- The expression index feature. The expression index is also called the function-based index. When you create an index, the index fields do not have to be a specific column but can be an expression calculated from one or more columns. This feature is useful for quickly accessing the calculation-based tables. See Expression index for details. (Introduced in v4.0)
- Generated Columns (Introduced in v2.1)
- User-Defined Variables (Introduced in v2.1)
- JSON data type and JSON functions (Introduced in v2.1)
- Cascades Planner: a cascades framework-based top-down query optimizer (Introduced in v3.0)
- Table Lock (Introduced in v4.0.0)
- `ALTER TABLE ... COMPACT` (Introduced in v6.1.0)
- Using `ALTER TABLE` to modify multiple columns or indexes (Introduced in v5.0.0)

2.4.5 Storage

- Disable Titan (Introduced in v4.0)
- Titan Level Merge (Introduced in v4.0)
- Divide Regions are divided into buckets. Buckets are used as the unit of concurrent query to improve the scan concurrency. (Introduced in v6.1.0)

- TiKV introduces API V2. (Introduced in v6.1.0)

2.4.6 Backup and restoration

- Back up and restore RawKV (Introduced in v3.1)

2.4.7 Data migration

- Use WebUI to manage migration tasks in DM. (Introduced in v6.0)

2.4.8 Garbage collection

- Green GC (Introduced in v5.0)

2.4.9 Diagnostics

- SQL diagnostics (Introduced in v4.0)
- Cluster diagnostics (Introduced in v4.0)
- TiKV-FastTune dashboard (Introduced in v4.0)

2.5 Benchmarks

2.5.1 TiDB Sysbench Performance Test Report – v6.1.0 vs. v6.0.0

2.5.1.1 Test overview

This test aims at comparing the Sysbench performance of TiDB v6.1.0 and TiDB v6.0.0 in the Online Transactional Processing (OLTP) scenario. The results show that performance of v6.1.0 is improved in the write workload. The performance of write-heavy workload is improved by 2.33% ~ 4.61%.

2.5.1.2 Test environment (AWS EC2)

2.5.1.2.1 Hardware configuration

Service type	EC2 type	Instance count
PD	m5.xlarge	3
TiKV	i3.4xlarge	3
TiDB	c5.4xlarge	3
Sysbench	c5.9xlarge	1

2.5.1.2.2 Software version

Service type	Software version
PD	v6.0.0 and v6.1.0
TiDB	v6.0.0 and v6.1.0
TiKV	v6.0.0 and v6.1.0
Sysbench	1.1.0-df89d34

2.5.1.2.3 Parameter configuration

TiDB v6.1.0 and TiDB v6.0.0 use the same configuration.

TiDB parameter configuration

```
log.level: "error"
prepared-plan-cache.enabled: true
tikv-client.max-batch-wait-time: 2000000
```

TiKV parameter configuration

```
storage.scheduler-worker-pool-size: 5
raftstore.store-pool-size: 3
raftstore.apply-pool-size: 3
rocksdb.max-background-jobs: 8
server.grpc-concurrency: 6
readpool.storage.normal-concurrency: 10
```

TiDB global variable configuration

```
set global tidb_hashagg_final_concurrency=1;
set global tidb_hashagg_partial_concurrency=1;
set global tidb_enable_async_commit = 1;
set global tidb_enable_1pc = 1;
set global tidb_guarantee_linearizability = 0;
set global tidb_enable_clustered_index = 1;
set global tidb_prepared_plan_cache_size=1000;
```

HAProxy configuration - haproxy.cfg

For more details about how to use HAProxy on TiDB, see Best Practices for Using HAProxy in TiDB.

```
global                                  # Global configuration.
  pidfile   /var/run/haproxy.pid        # Writes the PIDs of HAProxy processes
     ↪  into this file.
  maxconn   4000                        # The maximum number of concurrent
     ↪ connections for a single HAProxy process.
  user      haproxy                     # The same with the UID parameter.
  group     haproxy                     # The same with the GID parameter. A
     ↪ dedicated user group is recommended.
```

```
  nbproc    64                          # The number of processes created when
     ↪  going daemon. When starting multiple processes to forward requests
     ↪ , ensure that the value is large enough so that HAProxy does not
     ↪ block processes.
  daemon                                # Makes the process fork into
     ↪ background. It is equivalent to the command line "-D" argument. It
     ↪ can be disabled by the command line "-db" argument.

defaults                                # Default configuration.
  log global                            # Inherits the settings of the global
     ↪ configuration.
  retries 2                             # The maximum number of retries to
     ↪ connect to an upstream server. If the number of connection attempts
     ↪  exceeds the value, the backend server is considered unavailable.
  timeout connect 2s                    # The maximum time to wait for a
     ↪ connection attempt to a backend server to succeed. It should be set
     ↪  to a shorter time if the server is located on the same LAN as
     ↪ HAProxy.
  timeout client 30000s                 # The maximum inactivity time on the
     ↪ client side.
  timeout server 30000s                 # The maximum inactivity time on the
     ↪ server side.

listen tidb-cluster                     # Database load balancing.
  bind 0.0.0.0:3390                     # The Floating IP address and
     ↪ listening port.
  mode tcp                              # HAProxy uses layer 4, the transport
     ↪ layer.
  balance leastconn                    # The server with the fewest
     ↪ connections receives the connection. "leastconn" is recommended
     ↪ where long sessions are expected, such as LDAP, SQL and TSE, rather
     ↪  than protocols using short sessions, such as HTTP. The algorithm
     ↪ is dynamic, which means that server weights might be adjusted on
     ↪ the fly for slow starts for instance.
  server tidb-1 10.9.18.229:4000 check inter 2000 rise 2 fall 3 # Detects
     ↪ port 4000 at a frequency of once every 2000 milliseconds. If it is
     ↪ detected as successful twice, the server is considered available;
     ↪ if it is detected as failed three times, the server is considered
     ↪ unavailable.
  server tidb-2 10.9.39.208:4000 check inter 2000 rise 2 fall 3
  server tidb-3 10.9.64.166:4000 check inter 2000 rise 2 fall 3
```

2.5.1.3 Test plan

1. Deploy TiDB v6.1.0 and v6.0.0 using TiUP.
2. Use Sysbench to import 16 tables, each table with 10 million rows of data.
3. Execute the `analyze table` statement on each table.
4. Back up the data used for restore before different concurrency tests, which ensures data consistency for each test.
5. Start the Sysbench client to perform the `point_select`, `read_write`, `update_index`, and `update_non_index` tests. Perform stress tests on TiDB via HAProxy. For each concurrency under each workload, the test takes 20 minutes.
6. After each type of test is completed, stop the cluster, overwrite the cluster with the backup data in step 4, and restart the cluster.

2.5.1.3.1 Prepare test data

Run the following command to prepare the test data:

```
sysbench oltp_common \
    --threads=16 \
    --rand-type=uniform \
    --db-driver=mysql \
    --mysql-db=sbtest \
    --mysql-host=$aws_nlb_host \
    --mysql-port=$aws_nlb_port \
    --mysql-user=root \
    --mysql-password=password \
    prepare --tables=16 --table-size=10000000
```

2.5.1.3.2 Perform the test

Run the following command to perform the test:

```
sysbench $testname \
    --threads=$threads \
    --time=1200 \
    --report-interval=1 \
    --rand-type=uniform \
    --db-driver=mysql \
    --mysql-db=sbtest \
    --mysql-host=$aws_nlb_host \
    --mysql-port=$aws_nlb_port \
    run --tables=16 --table-size=10000000
```

2.5.1.4 Test results

2.5.1.4.1 Point Select performance

Threads	v6.0.0 TPS	v6.1.0 TPS	v6.0.0 95% latency (ms)	v6.1.0 95% latency (ms)	TPS improvement (%)
300	268934.84	265353.15	1.89	1.96	-1.33
600	365217.96	358976.94	2.57	2.66	-1.71
900	420799.64	407625.11	3.68	3.82	-3.13

Compared with v6.0.0, the Point Select performance of v6.1.0 slightly drops by 2.1%.

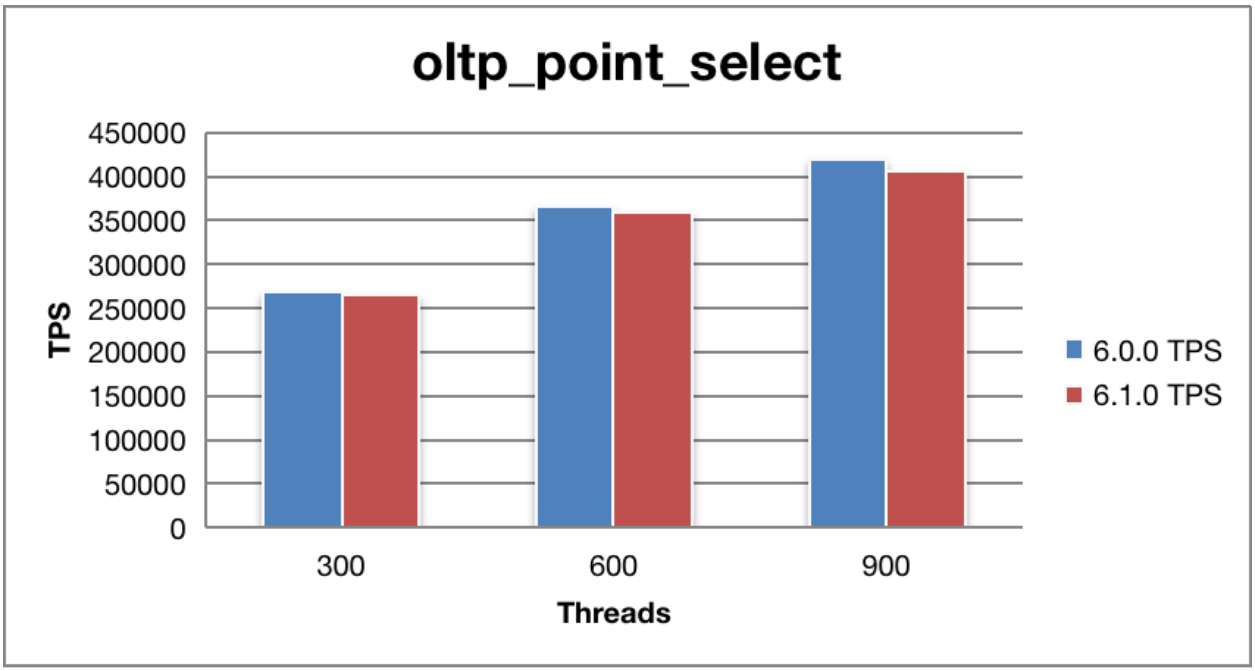


Figure 1: Point Select

2.5.1.4.2 Update Non-index performance

Threads	v6.0.0 TPS	v6.1.0 TPS	v6.0.0 95% latency (ms)	v6.1.0 95% latency (ms)	TPS improvement (%)
300	41778.95	42991.9	11.24	11.45	2.90
600	52045.39	54099.58	20.74	20.37	3.95
900	59243.35	62084.65	27.66	26.68	4.80

Compared with v6.0.0, the Update Non-index performance of v6.1.0 is improved by 3.88%.

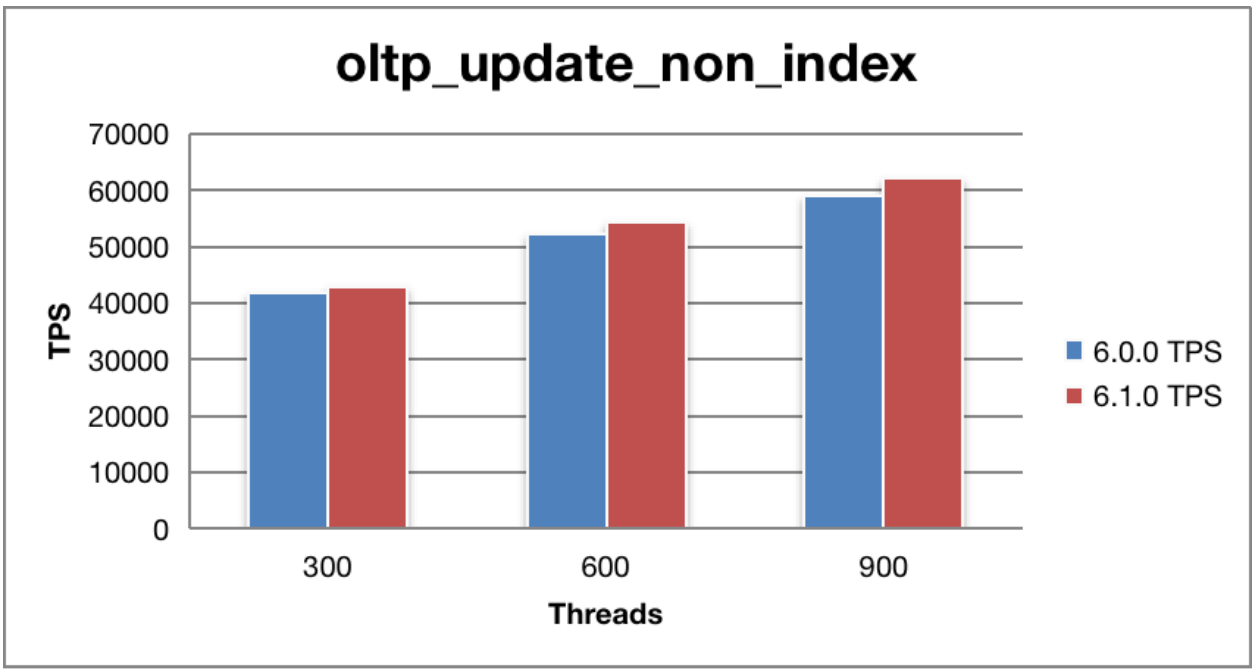


Figure 2: Update Non-index

2.5.1.4.3 Update Index performance

Threads	v6.0.0 TPS	v6.1.0 TPS	v6.0.0 95% latency (ms)	v6.1.0 95% latency (ms)	TPS improvement (%)
300	18085.79	19198.89	25.28	23.95	6.15
600	22210.8	22877.58	42.61	41.85	3.00
900	25249.81	26431.12	55.82	53.85	4.68

Compared with v6.0.0, the Update Index performance of v6.1.0 is improved by 4.61%.

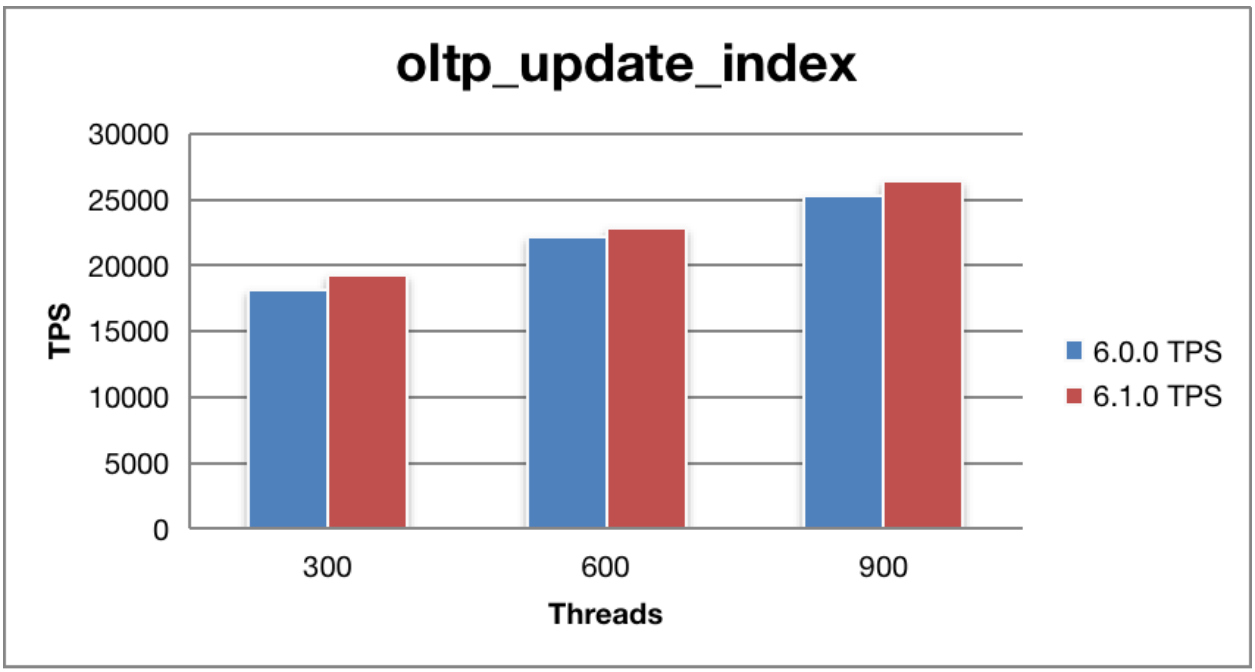


Figure 3: Update Index

2.5.1.4.4 Read Write performance

Threads	v6.0.0 TPS	v6.1.0 TPS	v6.0.0 95% latency (ms)	v6.1.0 95% latency (ms)	TPS improvement (%)
300	4856.23	4914.11	84.47	82.96	1.19
600	5676.46	5848.09	161.51	150.29	3.02
900	6072.97	6223.95	240.02	223.34	2.49

Compared with v6.0.0, the Read Write performance of v6.1.0 is improved by 2.23%.

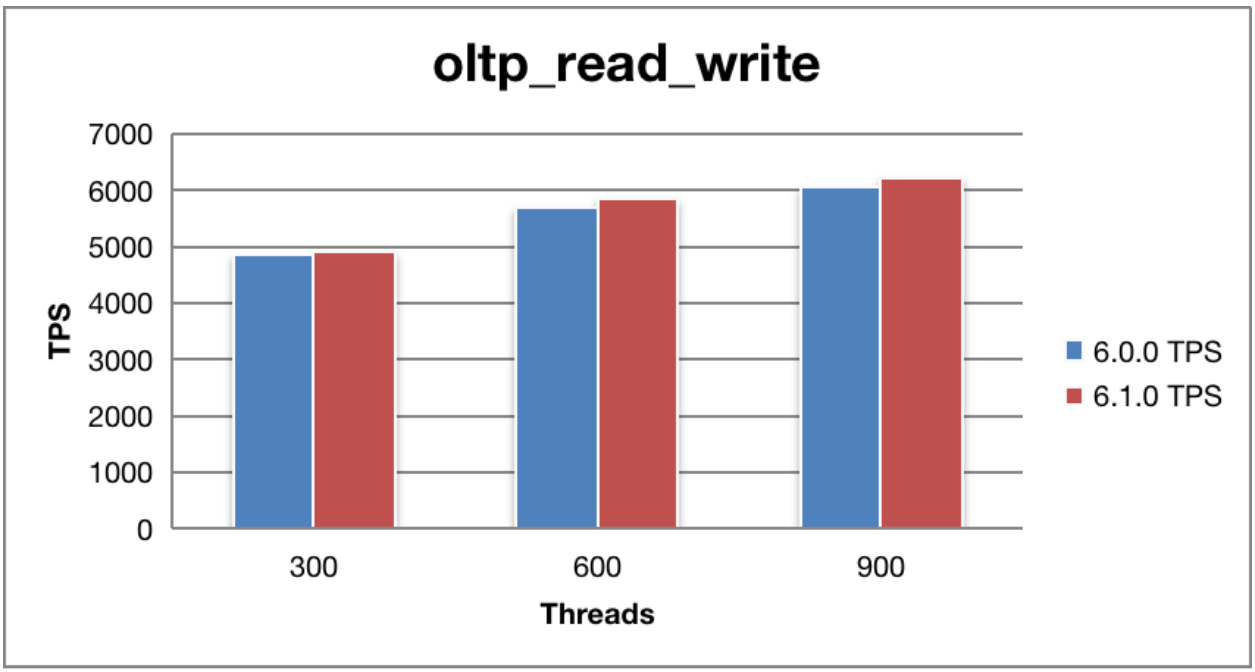


Figure 4: Read Write

2.5.2 TiDB TPC-C Performance Test Report – v6.1.0 vs. v6.0.0

2.5.2.1 Test overview

This test aims at comparing the TPC-C performance of TiDB v6.1.0 and v6.0.0 in the Online Transactional Processing (OLTP) scenario. The results show that compared with v6.0.0, the TPC-C performance of v6.1.0 is improved by 2.85%.

2.5.2.2 Test environment (AWS EC2)

2.5.2.2.1 Hardware configuration

Service type	EC2 type	Instance count
PD	m5.xlarge	3
TiKV	i3.4xlarge	3
TiDB	c5.4xlarge	3
TPC-C	c5.9xlarge	1

2.5.2.2.2 Software version

Service type	Software version
PD	v6.0.0 and v6.1.0

Service type	Software version
TiDB	v6.0.0 and v6.1.0
TiKV	v6.0.0 and v6.1.0
TiUP	1.9.3
HAProxy	2.5.0

2.5.2.2.3 Parameter configuration

TiDB v6.1.0 and TiDB v6.0.0 use the same configuration.

TiDB parameter configuration

```
log.level: "error"
prepared-plan-cache.enabled: true
tikv-client.max-batch-wait-time: 2000000
```

TiKV parameter configuration

```
raftstore.apply-max-batch-size: 2048
raftstore.apply-pool-size: 3
raftstore.store-max-batch-size: 2048
raftstore.store-pool-size: 2
readpool.storage.normal-concurrency: 10
server.grpc-concurrency: 6
```

TiDB global variable configuration

```
set global tidb_hashagg_final_concurrency=1;
set global tidb_hashagg_partial_concurrency=1;
set global tidb_enable_async_commit = 1;
set global tidb_enable_1pc = 1;
set global tidb_guarantee_linearizability = 0;
set global tidb_enable_clustered_index = 1;
set global tidb_prepared_plan_cache_size=1000;
```

HAProxy configuration - haproxy.cfg

For more details about how to use HAProxy on TiDB, see Best Practices for Using HAProxy in TiDB.

```
global                                  # Global configuration.
  pidfile   /var/run/haproxy.pid        # Writes the PIDs of HAProxy processes
     ↪  into this file.
  maxconn   4000                        # The maximum number of concurrent
     ↪ connections for a single HAProxy process.
  user      haproxy                     # The same with the UID parameter.
  group     haproxy                     # The same with the GID parameter. A
     ↪ dedicated user group is recommended.
```

```
  nbproc    64                        # The number of processes created when
     ↪  going daemon. When starting multiple processes to forward requests
     ↪ , ensure that the value is large enough so that HAProxy does not
     ↪ block processes.
  daemon                              # Makes the process fork into
     ↪ background. It is equivalent to the command line "-D" argument. It
     ↪ can be disabled by the command line "-db" argument.

defaults                              # Default configuration.
  log global                          # Inherits the settings of the global
     ↪ configuration.
  retries 2                           # The maximum number of retries to
     ↪ connect to an upstream server. If the number of connection attempts
     ↪  exceeds the value, the backend server is considered unavailable.
  timeout connect 2s                  # The maximum time to wait for a
     ↪ connection attempt to a backend server to succeed. It should be set
     ↪  to a shorter time if the server is located on the same LAN as
     ↪ HAProxy.
  timeout client 30000s               # The maximum inactivity time on the
     ↪ client side.
  timeout server 30000s               # The maximum inactivity time on the
     ↪ server side.

listen tidb-cluster                   # Database load balancing.
  bind 0.0.0.0:3390                   # The Floating IP address and
     ↪ listening port.
  mode tcp                            # HAProxy uses layer 4, the transport
     ↪ layer.
  balance leastconn                  # The server with the fewest
     ↪ connections receives the connection. "leastconn" is recommended
     ↪ where long sessions are expected, such as LDAP, SQL and TSE, rather
     ↪  than protocols using short sessions, such as HTTP. The algorithm
     ↪ is dynamic, which means that server weights might be adjusted on
     ↪ the fly for slow starts for instance.
  server tidb-1 10.9.18.229:4000 check inter 2000 rise 2 fall 3 # Detects
     ↪ port 4000 at a frequency of once every 2000 milliseconds. If it is
     ↪ detected as successful twice, the server is considered available;
     ↪ if it is detected as failed three times, the server is considered
     ↪ unavailable.
  server tidb-2 10.9.39.208:4000 check inter 2000 rise 2 fall 3
  server tidb-3 10.9.64.166:4000 check inter 2000 rise 2 fall 3
```

2.5.2.2.4 Prepare test data

1. Deploy TiDB v6.1.0 and v6.0.0 using TiUP.
2. Create a database named `tpcc`: `create database tpcc;`.
3. Use BenchmarkSQL to import the TPC-C 5000 Warehouse data: `tiup bench tpcc prepare --warehouse 5000 --db tpcc -H 127.0.0.1 -p 4000`.
4. Run the `tiup bench tpcc run -U root --db tpcc --host 127.0.0.1 --port 4000 --time 1800s --warehouses 5000 --threads {{thread}}` command to perform stress tests on TiDB via HAProxy. For each concurrency, the test takes 30 minutes.
5. Extract the tpmC data of New Order from the result.

2.5.2.3 Test result

Compared with v6.0.0, the TPC-C performance of v6.1.0 is **improved by 2.85%**.

Threads	v6.0.0 tpmC	v6.1.0 tpmC	tpmC improvement (%)
50	59059.2	60424.4	2.31
100	69357.6	71235.5	2.71
200	71364.8	74117.8	3.86
400	72694.3	74525.3	2.52

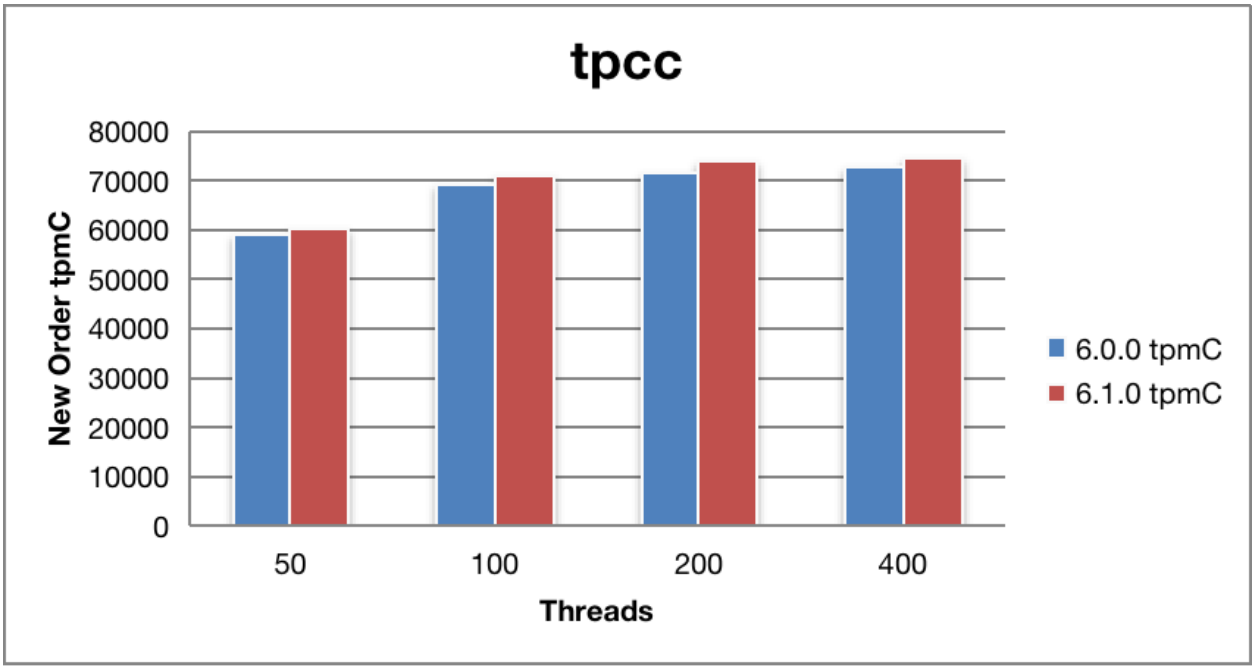


Figure 5: TPC-C

2.5.3 Performance Comparison between TiFlash and Greenplum/Spark

Refer to TiDB v5.4 TPC-H performance benchmarking report.

2.6 MySQL Compatibility

TiDB is highly compatible with the MySQL 5.7 protocol and the common features and syntax of MySQL 5.7. The ecosystem tools for MySQL 5.7 (PHPMyAdmin, Navicat, MySQL Workbench, mysqldump, and Mydumper/myloader) and the MySQL client can be used for TiDB.

However, some features of MySQL are not supported. This could be because there is now a better way to solve the problem (such as XML functions superseded by JSON), or a lack of current demand versus effort required (such as stored procedures and functions). Some features might also be difficult to implement as a distributed system.

In addition, TiDB does not support the MySQL replication protocol, but provides specific tools to replicate data with MySQL:

- Replicate data from MySQL: TiDB Data Migration (DM) is a tool that supports the full data migration and the incremental data replication from MySQL/MariaDB into TiDB.
- Replicate data to MySQL: TiCDC is a tool for replicating the incremental data of TiDB by pulling TiKV change logs. TiCDC uses the MySQL sink to replicate the incremental data of TiDB to MySQL.

> **Note:**
>
> This page describes general differences between MySQL and TiDB. See the dedicated pages for Security and Pessimistic Transaction Mode compatibility.

2.6.1 Unsupported features

- Stored procedures and functions
- Triggers
- Events
- User-defined functions
- `FOREIGN KEY` constraints #18209
- `FULLTEXT` syntax and indexes #1793
- `SPATIAL` (also known as `GIS/GEOMETRY`) functions, data types and indexes #6347
- Character sets other than `ascii`, `latin1`, `binary`, `utf8`, `utf8mb4`, and `gbk`.
- SYS schema
- Optimizer trace
- XML Functions
- X-Protocol #1109
- Savepoints #6840
- Column-level privileges #9766

- `XA` syntax (TiDB uses a two-phase commit internally, but this is not exposed via an SQL interface)
- `CREATE TABLE tblName AS SELECT stmt` syntax #4754
- `CHECK TABLE` syntax #4673
- `CHECKSUM TABLE` syntax #1895
- `REPAIR TABLE` syntax
- `OPTIMIZE TABLE` syntax
- `HANDLER` statement
- `CREATE TABLESPACE` statement
- "Session Tracker: Add GTIDs context to the OK packet"

2.6.2 Features that are different from MySQL

2.6.2.1 Auto-increment ID

- In TiDB, auto-incremental columns are globally unique. They are incremental on a single TiDB server, but *not* necessarily incremental among multiple TiDB servers or allocated sequentially. It is recommended that you do not mix default values and custom values. Otherwise, you might encounter the `Duplicated Error` error message.
- You can use the `tidb_allow_remove_auto_inc` system variable to allow or forbid removing the `AUTO_INCREMENT` column attribute. The syntax of removing the column attribute is `ALTER TABLE MODIFY` or `ALTER TABLE CHANGE`.
- TiDB does not support adding the `AUTO_INCREMENT` column attribute, and this attribute cannot be recovered once it is removed.
- See `AUTO_INCREMENT` for more details.

Note:

- If you have not specified the primary key when creating a table, TiDB uses `_tidb_rowid` to identify the row. The allocation of this value shares an allocator with the auto-increment column (if such a column exists). If you specify an auto-increment column as the primary key, TiDB uses this column to identify the row. In this situation, the following situation might happen:

```
mysql> CREATE TABLE t(id INT UNIQUE KEY AUTO_INCREMENT);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(),(),();
```

```
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT _tidb_rowid, id FROM t;
+-------------+------+
| _tidb_rowid | id |
+-------------+------+
|           4 |   1 |
|           5 |   2 |
|           6 |   3 |
+-------------+------+
3 rows in set (0.01 sec)
```

> **Note:**
>
> The `AUTO_INCREMENT` attribute might cause hotspot in production environments. See Troubleshoot HotSpot Issues for details. It is recommended to use `AUTO_RANDOM` instead.

2.6.2.2 Performance schema

TiDB uses a combination of Prometheus and Grafana to store and query the performance monitoring metrics. Performance schema tables return empty results in TiDB.

2.6.2.3 Query Execution Plan

The output format, output content, and the privilege setting of Query Execution Plan (`EXPLAIN`/`EXPLAIN FOR`) in TiDB is greatly different from those in MySQL.

The MySQL system variable `optimizer_switch` is read-only in TiDB and has no effect on query plans. You can also use optimizer hints in similar syntax to MySQL, but the available hints and implementation might differ.

See Understand the Query Execution Plan for more details.

2.6.2.4 Built-in functions

TiDB supports most of the MySQL built-in functions, but not all. The statement `SHOW BUILTINS` provides a list of functions that are available.

See also: TiDB SQL Grammar.

2.6.2.5 DDL

In TiDB, all supported DDL changes are performed online. Compared with DDL operations in MySQL, the DDL operations in TiDB have the following major restrictions:

- Multiple operations cannot be completed in a single `ALTER TABLE` statement. For example, it is not possible to add multiple columns or indexes in a single statement. Otherwise, the `Unsupported multi schema change` error might be output.
- `ALTER TABLE` in TiDB does not support the changes of some data types. For example, TiDB does not support the change from the `DECIMAL` type to the `DATE` type. If a data type change is unsupported, TiDB reports the `Unsupported modify column: type %d not match origin %d` error. Refer to ALTER TABLE for more details.
- The `ALGORITHM={INSTANT,INPLACE,COPY}` syntax functions only as an assertion in TiDB, and does not modify the `ALTER` algorithm. See ALTER TABLE for further details.
- Adding/Dropping the primary key of the `CLUSTERED` type is unsupported. For more details about the primary key of the `CLUSTERED` type, refer to clustered index.
- Different types of indexes (`HASH|BTREE|RTREE|FULLTEXT`) are not supported, and will be parsed and ignored when specified.
- TiDB supports `HASH`, `RANGE`, and `LIST` partitioning types. For an unsupported partition type, TiDB returns `Warning: Unsupported partition type, treat as normal table`, where `%s` is a specific partition type.
- Table partitioning supports `ADD`, `DROP`, and `TRUNCATE` operations. Other partition operations are ignored. The following table partition syntaxes are not supported:
  - `PARTITION BY KEY`
  - `SUBPARTITION`
  - `{CHECK|OPTIMIZE|REPAIR|IMPORT|DISCARD|REBUILD|REORGANIZE|COALESCE} PARTITION`

  For more details, see Partitioning.

2.6.2.6 Analyze table

Statistics Collection works differently in TiDB than in MySQL, in that it is a relatively lightweight and short-lived operation in MySQL/InnoDB, while in TiDB it completely rebuilds the statistics for a table and can take much longer to complete.

These differences are documented further in ANALYZE TABLE.

2.6.2.7 Limitations of `SELECT` syntax

- The syntax `SELECT ... INTO @variable` is not supported.
- The syntax `SELECT ... GROUP BY ... WITH ROLLUP` is not supported.
- The syntax `SELECT .. GROUP BY expr` does not imply `GROUP BY expr ORDER BY expr` as it does in MySQL 5.7.

For details, see the SELECT statement reference.

2.6.2.8 `UPDATE` statement

See the `UPDATE` statement reference.

2.6.2.9 Views

Views in TiDB are not updatable. They do not support write operations such as `UPDATE`, `INSERT`, and `DELETE`.

2.6.2.10 Temporary tables

For details, see Compatibility between TiDB local temporary tables and MySQL temporary tables.

2.6.2.11 Character sets and collations

- To learn the details of the character sets and collations supported by TiDB, see Character Set and Collation Overview.
- To learn the MySQL compatibility of the GBK character set, see GBK compatibility .
- TiDB inherits the character set used in the table as the national character set.

2.6.2.12 Storage engines

For compatibility reasons, TiDB supports the syntax to create tables with alternative storage engines. In implementation, TiDB describes the metadata as the InnoDB storage engine.

TiDB supports storage engine abstraction similar to MySQL, but you need to specify the storage engine using the `--store` option when you start the TiDB server.

2.6.2.13 SQL modes

TiDB supports most SQL modes:

- The compatibility modes, such as `Oracle` and `PostgreSQL` are parsed but ignored. Compatibility modes are deprecated in MySQL 5.7 and removed in MySQL 8.0.
- The `ONLY_FULL_GROUP_BY` mode has minor semantic differences from MySQL 5.7.
- The `NO_DIR_IN_CREATE` and `NO_ENGINE_SUBSTITUTION` SQL modes in MySQL are accepted for compatibility, but are not applicable to TiDB.

2.6.2.14 Default differences

- Default character set:
  - The default value in TiDB is `utf8mb4`.
  - The default value in MySQL 5.7 is `latin1`.
  - The default value in MySQL 8.0 is `utf8mb4`.
- Default collation:
  - The default collation of `utf8mb4` in TiDB is `utf8mb4_bin`.
  - The default collation of `utf8mb4` in MySQL 5.7 is `utf8mb4_general_ci`.
  - The default collation of `utf8mb4` in MySQL 8.0 is `utf8mb4_0900_ai_ci`.
- Default value of `foreign_key_checks`:
  - The default value in TiDB is `OFF` and currently TiDB only supports `OFF`.
  - The default value in MySQL 5.7 is `ON`.
- Default SQL mode:
  - The default SQL mode in TiDB includes these modes: `ONLY_FULL_GROUP_BY,`
    ↪ `STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO`
    ↪ `,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION`.
  - The default SQL mode in MySQL:
    - The default SQL mode in MySQL 5.7 is the same as TiDB.
    - The default SQL mode in MySQL 8.0 includes these modes: `ONLY_FULL_GROUP_BY`
      ↪ `,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_`
      ↪ `,NO_ENGINE_SUBSTITUTION`.
- Default value of `lower_case_table_names`:
  - The default value in TiDB is `2` and currently TiDB only supports `2`.
  - The default value in MySQL:
    - On Linux: `0`
    - On Windows: `1`
    - On macOS: `2`
- Default value of `explicit_defaults_for_timestamp`:
  - The default value in TiDB is `ON` and currently TiDB only supports `ON`.
  - The default value in MySQL:
    - For MySQL 5.7: `OFF`.
    - For MySQL 8.0: `ON`.

2.6.2.15 Date and Time

2.6.2.15.1 Named timezone

- TiDB uses all time zone rules currently installed in the system for calculation (usually the `tzdata` package). You can use all time zone names without importing the time zone table data. You cannot modify the calculation rules by importing the time zone table data.
- MySQL uses the local time zone by default and relies on the current time zone rules built into the system (such as when to start daylight saving time) for calculation; and the time zone cannot be specified by the time zone name without importing the time zone table data.

2.6.2.16 Type system differences

The following column types are supported by MySQL, but **NOT** by TiDB:

- FLOAT4/FLOAT8
- `SQL_TSI_*` (including SQL_TSI_MONTH, SQL_TSI_WEEK, SQL_TSI_DAY, SQL_TSI_HOUR, SQL_TSI_MINUTE and SQL_TSI_SECOND, excluding SQL_TSI_YEAR)

2.6.2.17 Incompatibility caused by deprecated features

TiDB does not implement certain features that have been marked as deprecated in MySQL, including:

- Specifying precision for floating point types. MySQL 8.0 deprecates this feature, and it is recommended to use the `DECIMAL` type instead.
- The `ZEROFILL` attribute. MySQL 8.0 deprecates this feature, and it is recommended to instead pad numeric values in your application.

2.7 TiDB Limitations

This document describes the common usage limitations of TiDB, including the maximum identifier length and the maximum number of supported databases, tables, indexes, partitioned tables, and sequences.

2.7.1 Limitations on identifier length

Identifier type	Maximum length (number of characters allowed)
Database	64
Table	64
Column	64
Index	64
View	64
Sequence	64

2.7.2 Limitations on the total number of databases, tables, views, and connections

Identifier type	Maximum number
Databases	unlimited
Tables	unlimited
Views	unlimited
Connections	unlimited

2.7.3 Limitations on a single database

Type	Upper limit
Tables	unlimited

2.7.4 Limitations on a single table

Type	Upper limit (default value)
Columns	Defaults to 1017 and can be adjusted up to 4096
Indexes	Defaults to 64 and can be adjusted up to 512
Rows	unlimited
Size	unlimited
Partitions	8192

- The upper limit of `Columns` can be modified via `table-column-count-limit`.
- The upper limit of `Indexes` can be modified via `index-limit`.

2.7.5 Limitation on a single row

Type	Upper limit (default value)
Size	Defaults to 6 MiB and can be adjusted to 120 MiB

You can adjust the size limit via the `txn-entry-size-limit` configuration item.

2.7.6 Limitations on data types

Type	Upper limit
CHAR	255 characters

Type	Upper limit
BINARY	255 characters
VARBINARY	65535 characters
VARCHAR	16383 characters
TEXT	Defaults to 6 MiB and can be adjusted to 120 MiB
BLOB	Defaults to 6 MiB and can be adjusted to 120 MiB

2.7.7 Limitations on SQL statements

Type	Upper limit
The maximum number of SQL statements in a single transaction	When the optimistic transaction is used and the transaction retry is enabled, the upper limit is 5000.

You can modify the limit via the `stmt-count-limit` configuration item.

2.8 Credits

Each contributor has played an important role in promoting the robust development of TiDB. We sincerely appreciate all contributors who have submitted code, written and translated documents for TiDB.

2.8.1 TiDB developers

TiDB developers contribute to new feature development, performance improvement, stability guarantee, and bug fixes. The following is the list of contributors in TiDB related repos:

- pingcap/tidb
- tikv/tikv
- pingcap/parser
- tikv/pd
- pingcap/tiflash
- pingcap/tidb-operator

- pingcap/tiup
- pingcap/br
- pingcap/dm
- pingcap/tidb-binlog
- pingcap/tidb-dashboard
- pingcap/tiflow
- pingcap/tidb-tools
- pingcap/tidb-lightning
- pingcap/tispark
- pingcap/dumpling
- tikv/client-java
- tidb-incubator/TiBigData
- ti-community-infra

2.8.2 Writers and translators for TiDB documentation

Writers and translators write and translate documents for TiDB and the related projects. The following is the list of contributors in TiDB documentation related repos:

- pingcap/docs-cn
- pingcap/docs
- pingcap/docs-tidb-operator
- tikv/website

3 Quick Start

3.1 Quick Start Guide for the TiDB Database Platform

This guide walks you through the quickest way to get started with TiDB. For non-production environments, you can deploy your TiDB database by either of the following methods:

- Deploy a local test cluster (for macOS and Linux)
- Simulate production deployment on a single machine (for Linux only)

> **Note:**
>
> The deployment method provided in this guide is **ONLY FOR** quick start, **NOT FOR** production.
>
> - To deploy a self-hosted production cluster, see production installation guide.

- To deploy TiDB on Kubernetes, see Get Started with TiDB on Kubernetes.
- To manage TiDB in the cloud, see TiDB Cloud Quick Start.

3.1.1 Deploy a local test cluster

- Scenario: Quickly deploy a local TiDB cluster for testing using a single macOS or Linux server. By deploying such a cluster, you can learn the basic architecture of TiDB and the operation of its components, such as TiDB, TiKV, PD, and the monitoring components.

As a distributed system, a basic TiDB test cluster usually consists of 2 TiDB instances, 3 TiKV instances, 3 PD instances, and optional TiFlash instances. With TiUP Playground, you can quickly build the test cluster by taking the following steps:

1. Download and install TiUP:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

If the following message is displayed, you have installed TiUP successfully:

```
Successfully set mirror to https://tiup-mirrors.pingcap.com
Detected shell: zsh
Shell profile: /Users/user/.zshrc
/Users/user/.zshrc has been modified to add tiup to PATH
open a new terminal or source /Users/user/.zshrc to use it
Installed path: /Users/user/.tiup/bin/tiup
===============================================
Have a try:   tiup playground
===============================================
```

Note the Shell profile path in the output above. You need to use the path in the next step.

2. Declare the global environment variable:

Note:

After the installation, TiUP displays the absolute path of the corresponding Shell profile file. You need to modify `${your_shell_profile}` in the following `source` command according to the path. In this case, `$ ↪ {your_shell_profile}` is `/Users/user/.zshrc` from the output of Step 1.

```
source ${your_shell_profile}
```

3. Start the cluster in the current session:

    - If you want to start a TiDB cluster of the latest version with 1 TiDB instance, 1 TiKV instance, 1 PD instance, and 1 TiFlash instance, run the following command:

    ```
    tiup playground
    ```

    - If you want to specify the TiDB version and the number of the instances of each component, run a command like this:

    ```
    tiup playground v6.1.7 --db 2 --pd 3 --kv 3
    ```

    The command downloads a version cluster to the local machine and starts it, such as v6.1.7. To view the latest version, run `tiup list tidb`.

    This command returns the access methods of the cluster:

    ```
    CLUSTER START SUCCESSFULLY, Enjoy it ^-^
    To connect TiDB: mysql --comments --host 127.0.0.1 --port 4001 -u
       ↪ root -p (no password)
    To connect TiDB: mysql --comments --host 127.0.0.1 --port 4000 -u
       ↪ root -p (no password)
    To view the dashboard: http://127.0.0.1:2379/dashboard
    PD client endpoints: [127.0.0.1:2379 127.0.0.1:2382 127.0.0.1:2384]
    To view Prometheus: http://127.0.0.1:9090
    To view Grafana: http://127.0.0.1:3000
    ```

Note:

- Since v5.2.0, TiDB supports running `tiup playground` on the machine that uses the Apple M1 chip.
- For the playground operated in this way, after the test deployment is finished, TiUP will clean up the original cluster data. You will get a new cluster after re-running the command.
- If you want the data to be persisted on storage, run `tiup --tag ↪ <your-tag> playground ...`. For details, refer to TiUP Reference Guide.

4. Start a new session to access TiDB:

    - Use the TiUP client to connect to TiDB.

    ```
    tiup client
    ```

- You can also use the MySQL client to connect to TiDB.

```
mysql --host 127.0.0.1 --port 4000 -u root
```

5. Access the Prometheus dashboard of TiDB at http://127.0.0.1:9090.
6. Access the TiDB Dashboard at http://127.0.0.1:2379/dashboard. The default username is `root`, with an empty password.
7. Access the Grafana dashboard of TiDB through http://127.0.0.1:3000. Both the default username and password are `admin`.
8. (Optional) Load data to TiFlash for analysis.
9. Clean up the cluster after the test deployment:
    1. Stop the above TiDB service by pressing Control+C.
    2. Run the following command after the service is stopped:

```
tiup clean --all
```

Note:

TiUP Playground listens on `127.0.0.1` by default, and the service is only locally accessible. If you want the service to be externally accessible, specify the listening address using the `--host` parameter to bind the network interface card (NIC) to an externally accessible IP address.

As a distributed system, a basic TiDB test cluster usually consists of 2 TiDB instances, 3 TiKV instances, 3 PD instances, and optional TiFlash instances. With TiUP Playground, you can quickly build the test cluster by taking the following steps:

1. Download and install TiUP:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

If the following message is displayed, you have installed TiUP successfully:

```
Successfully set mirror to https://tiup-mirrors.pingcap.com
Detected shell: zsh
Shell profile: /Users/user/.zshrc
/Users/user/.zshrc has been modified to add tiup to PATH
open a new terminal or source /Users/user/.zshrc to use it
```

```
Installed path: /Users/user/.tiup/bin/tiup
==============================================
Have a try:   tiup playground
==============================================
```

Note the Shell profile path in the output above. You need to use the path in the next step.

2. Declare the global environment variable:

> **Note:**
> After the installation, TiUP displays the absolute path of the corresponding Shell profile file. You need to modify `${your_shell_profile}` in the following `source` command according to the path.

```
source ${your_shell_profile}
```

3. Start the cluster in the current session:

   - If you want to start a TiDB cluster of the latest version with 1 TiDB instance, 1 TiKV instance, 1 PD instance, and 1 TiFlash instance, run the following command:

     ```
     tiup playground
     ```

   - If you want to specify the TiDB version and the number of the instances of each component, run a command like this:

     ```
     tiup playground v6.1.7 --db 2 --pd 3 --kv 3
     ```

     The command downloads a version cluster to the local machine and starts it, such as v6.1.7. To view the latest version, run `tiup list tidb`.

     This command returns the access methods of the cluster:

     ```
     CLUSTER START SUCCESSFULLY, Enjoy it ^-^
     To connect TiDB: mysql --host 127.0.0.1 --port 4000 -u root -p (no
        ↪ password) --comments
     To view the dashboard: http://127.0.0.1:2379/dashboard
     PD client endpoints: [127.0.0.1:2379]
     To view the Prometheus: http://127.0.0.1:9090
     To view the Grafana: http://127.0.0.1:3000
     ```

Note:
For the playground operated in this way, after the test deployment is finished, TiUP will clean up the original cluster data. You will get a new cluster after re-running the command. If you want the data to be persisted on storage, run `tiup --tag <your-tag> playground ...`. For details, refer to TiUP Reference Guide.

4. Start a new session to access TiDB:

    - Use the TiUP client to connect to TiDB.

    ```
    tiup client
    ```

    - You can also use the MySQL client to connect to TiDB.

    ```
    mysql --host 127.0.0.1 --port 4000 -u root
    ```

5. Access the Prometheus dashboard of TiDB at http://127.0.0.1:9090.

6. Access the TiDB Dashboard at http://127.0.0.1:2379/dashboard. The default username is `root`, with an empty password.

7. Access the Grafana dashboard of TiDB through http://127.0.0.1:3000. Both the default username and password are `admin`.

8. (Optional) Load data to TiFlash for analysis.

9. Clean up the cluster after the test deployment:

    1. Stop the process by pressing Control+C.
    2. Run the following command after the service is stopped:

    ```
    tiup clean --all
    ```

Note:

TiUP Playground listens on `127.0.0.1` by default, and the service is only locally accessible. If you want the service to be externally accessible, specify the listening address using the `--host` parameter to bind the network interface card (NIC) to an externally accessible IP address.

3.1.2 Simulate production deployment on a single machine

- Scenario: Experience the smallest TiDB cluster with the complete topology and simulate the production deployment steps on a single Linux server.

This section describes how to deploy a TiDB cluster using a YAML file of the smallest topology in TiUP.

3.1.2.1 Prepare

Prepare a target machine that meets the following requirements:

- CentOS 7.3 or a later version is installed
- The Linux OS has access to the Internet, which is required to download TiDB and related software installation packages

The smallest TiDB cluster topology is as follows:

> **Note:**
>
> The IP address of the following instances only serves as an example IP. In your actual deployment, you need to replace the IP with your actual IP.

Instance	Count	IP	Configuration
TiKV	3	10.0.1.1 10.0.1.2 10.0.1.3	Avoid conflict between the port and the directory

Instance	Count	IP	Configuration
TiDB	1	10.0.1.1	The default port Global directory configuration
PD	1	10.0.1.1	The default port Global directory configuration
TiFlash	1	10.0.1.1	The default port Global directory configuration

Instance	Count	IP	Configuration
Monitor	1	10.0.1.1	The default port Global directory configuration

Other requirements for the target machine:

- The `root` user and its password is required
- Stop the firewall service of the target machine, or open the port needed by the TiDB cluster nodes
- Currently, the TiUP cluster supports deploying TiDB on the x86_64 (AMD64) and ARM architectures:
    - It is recommended to use CentOS 7.3 or later versions on AMD64
    - It is recommended to use CentOS 7.6 1810 on ARM

3.1.2.2 Deploy

> **Note:**
>
> You can log in to the target machine as a regular user or the `root` user. The following steps use the `root` user as an example.

1. Download and install TiUP:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

2. Declare the global environment variable.

> **Note:**
> After the installation, TiUP displays the absolute path of the corresponding Shell profile file. You need to modify `${your_shell_profile}` in the following `source` command according to the path.

```
source ${your_shell_profile}
```

3. Install the cluster component of TiUP:

```
tiup cluster
```

4. If the TiUP cluster is already installed on the machine, update the software version:

```
tiup update --self && tiup update cluster
```

5. Use the root user privilege to increase the connection limit of the `sshd` service. This is because TiUP needs to simulate deployment on multiple machines.

    1. Modify `/etc/ssh/sshd_config`, and set `MaxSessions` to 20.
    2. Restart the `sshd` service:

    ```
    service sshd restart
    ```

6. Create and start the cluster:

    Edit the configuration file according to the following template, and name it as `topo.yaml`:

```
# # Global variables are applied to all deployments and used as the default value of
# # the deployments if a specific deployment value is missing.
global:
 user: "tidb"
 ssh_port: 22
 deploy_dir: "/tidb-deploy"
 data_dir: "/tidb-data"

# # Monitored variables are applied to all the machines.
monitored:
 node_exporter_port: 9100
 blackbox_exporter_port: 9115
```

```
server_configs:
 tidb:
   instance.tidb_slow_log_threshold: 300
 tikv:
   readpool.storage.use-unified-pool: false
   readpool.coprocessor.use-unified-pool: true
 pd:
   replication.enable-placement-rules: true
   replication.location-labels: ["host"]
 tiflash:
   logger.level: "info"

pd_servers:
 - host: 10.0.1.1

tidb_servers:
 - host: 10.0.1.1

tikv_servers:
 - host: 10.0.1.1
   port: 20160
   status_port: 20180
   config:
     server.labels: { host: "logic-host-1" }

 - host: 10.0.1.1
   port: 20161
   status_port: 20181
   config:
     server.labels: { host: "logic-host-2" }

 - host: 10.0.1.1
   port: 20162
   status_port: 20182
   config:
     server.labels: { host: "logic-host-3" }

tiflash_servers:
 - host: 10.0.1.1

monitoring_servers:
 - host: 10.0.1.1

grafana_servers:
 - host: 10.0.1.1
```

- `user: "tidb"`: Use the `tidb` system user (automatically created during deployment) to perform the internal management of the cluster. By default, use port 22 to log in to the target machine via SSH.
- `replication.enable-placement-rules`: This PD parameter is set to ensure that TiFlash runs normally.
- `host`: The IP of the target machine.

7. Execute the cluster deployment command:

```
tiup cluster deploy <cluster-name> <tidb-version> ./topo.yaml --user
    ↪ root -p
```

- `<cluster-name>`: Set the cluster name
- `<tidb-version>`: Set the TiDB cluster version. You can see all the supported TiDB versions by running the `tiup list tidb` command
- `-p`: Specify the password used to connect to the target machine.

> **Note:**
> If you use secret keys, you can specify the path of the keys through `-i`. Do not use `-i` and `-p` at the same time.

Enter "y" and the `root` user's password to complete the deployment:

```
Do you want to continue? [y/N]: y
Input SSH password:
```

8. Start the cluster:

```
tiup cluster start <cluster-name>
```

9. Access the cluster:

- Install the MySQL client. If it is already installed, skip this step.

```
yum -y install mysql
```

- Access TiDB. The password is empty:

```
mysql -h 10.0.1.1 -P 4000 -u root
```

- Access the Grafana monitoring dashboard at http://%7Bgrafana-ip%7D:3000. The default username and password are both `admin`.

- Access the TiDB Dashboard at http://%7Bpd-ip%7D:2379/dashboard. The default username is `root`, and the password is empty.
- To view the currently deployed cluster list:

```
tiup cluster list
```

- To view the cluster topology and status:

```
tiup cluster display <cluster-name>
```

3.1.3 What's next

- If you have just deployed a TiDB cluster for the local test environment:
  - Learn Basic SQL operations in TiDB
  - Migrate data to TiDB
- If you are ready to deploy a TiDB cluster for the production environment:
  - Deploy TiDB using TiUP
  - Deploy TiDB on Cloud using TiDB Operator
- If you're looking for analytics solution with TiFlash:
  - Use TiFlash
  - TiFlash Overview

3.2 Quick Start Guide for TiDB HTAP

This guide walks you through the quickest way to get started with TiDB's one-stop solution of Hybrid Transactional and Analytical Processing (HTAP).

> **Note:**
>
> The steps provided in this guide is ONLY for quick start in the test environment. For production environments, explore HTAP is recommended.

3.2.1 Basic concepts

Before using TiDB HTAP, you need to have some basic knowledge about TiKV, a row-based storage engine for TiDB Online Transactional Processing (OLTP), and TiFlash, a columnar storage engine for TiDB Online Analytical Processing (OLAP).

- Storage engines of HTAP: The row-based storage engine and the columnar storage engine co-exist for HTAP. Both storage engines can replicate data automatically and keep strong consistency. The row-based storage engine optimizes OLTP performance, and the columnar storage engine optimizes OLAP performance.
- Data consistency of HTAP: As a distributed and transactional key-value database, TiKV provides transactional interfaces with ACID compliance, and guarantees data consistency between multiple replicas and high availability with the implementation of the Raft consensus algorithm. As a columnar storage extension of TiKV, TiFlash replicates data from TiKV in real time according to the Raft Learner consensus algorithm, which ensures that data is strongly consistent between TiKV and TiFlash.
- Data isolation of HTAP: TiKV and TiFlash can be deployed on different machines as needed to solve the problem of HTAP resource isolation.
- MPP computing engine: MPP is a distributed computing framework provided by the TiFlash engine since TiDB 5.0, which allows data exchange between nodes and provides high-performance, high-throughput SQL algorithms. In the MPP mode, the run time of the analytic queries can be significantly reduced.

3.2.2 Steps

In this document, you can experience the convenience and high performance of TiDB HTAP by querying an example table in a TPC-H dataset. TPC-H is a popular decision support benchmark that consists of a suite of business oriented ad-hoc queries with large volumes of data and a high degree of complexity. To experience 22 complete SQL queries using TPC-H, visit tidb-bench repo or TPC-H for instructions on how to generate query statements and data.

3.2.2.1 Step 1. Deploy a local test environment

Before using TiDB HTAP, follow the steps in the Quick Start Guide for the TiDB Database Platform to prepare a local test environment, and run the following command to deploy a TiDB cluster:

```
tiup playground
```

> **Note:**
>
> `tiup playground` command is ONLY for quick start, NOT for production.

3.2.2.2 Step 2. Prepare test data

In the following steps, you can create a TPC-H dataset as the test data to use TiDB HTAP. If you are interested in TPC-H, see General Implementation Guidelines.

> **Note:**
>
> If you want to use your existing data for analytic queries, you can migrate your data to TiDB. If you want to design and create your own test data, you can create it by executing SQL statements or using related tools.

1. Install the test data generation tool by running the following command:

```
tiup install bench
```

2. Generate the test data by running the following command:

```
tiup bench tpch --sf=1 prepare
```

   If the output of this command shows `Finished`, it indicates that the data is created.

3. Execute the following SQL statement to view the generated data:

```
SELECT
  CONCAT(table_schema,'.',table_name) AS 'Table Name',
  table_rows AS 'Number of Rows',
  FORMAT_BYTES(data_length) AS 'Data Size',
  FORMAT_BYTES(index_length) AS 'Index Size',
  FORMAT_BYTES(data_length+index_length) AS'Total'
FROM
  information_schema.TABLES
WHERE
  table_schema='test';
```

   As you can see from the output, eight tables are created in total, and the largest table has 6.5 million rows (the number of rows created by the tool depends on the actual SQL query result because the data is randomly generated).

```
sql +---------------+---------------+-----------+------------+-----------+
↪  | Table Name | Number of Rows | Data Size | Index Size | Total |
↪ +---------------+---------------+-----------+------------+-----------+
↪  | test.nation | 25 | 2.44 KiB | 0 bytes | 2.44 KiB | | test.region
↪  | 5 | 416 bytes | 0 bytes | 416 bytes | | test.part |        200000
↪  | 25.07 MiB | 0 bytes | 25.07 MiB | | test.supplier |    10000 |
↪  1.45 MiB | 0 bytes | 1.45 MiB | | test.partsupp |     800000 |
```

```
↪ 120.17 MiB| 12.21 MiB | 132.38 MiB| | test.customer | 150000 | 24.77
↪  MiB | 0 bytes | 24.77 MiB | | test.orders |    1527648 | 174.40
↪ MiB| 0 bytes | 174.40 MiB| | test.lineitem |  6491711 | 849.07 MiB|
↪ 99.06 MiB | 948.13 MiB| +---------------+---------------+----------+----------
↪  8 rows in set (0.06 sec)
```

This is a database of a commercial ordering system. In which, the `test.nation` table indicates the information about countries, the `test.region` table indicates the information about regions, the `test.part` table indicates the information about parts, the `test.supplier` table indicates the information about suppliers, the `test.partsupp` ↪ table indicates the information about parts of suppliers, the `test.customer` table indicates the information about customers, the `test.customer` table indicates the information about orders, and the `test.lineitem` table indicates the information about online items.

3.2.2.3 Step 3. Query data with the row-based storage engine

To know the performance of TiDB with only the row-based storage engine, execute the following SQL statements:

```
SELECT
    l_orderkey,
    SUM(
        l_extendedprice * (1 - l_discount)
    ) AS revenue,
    o_orderdate,
    o_shippriority
FROM
    customer,
    orders,
    lineitem
WHERE
    c_mktsegment = 'BUILDING'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < DATE '1996-01-01'
AND l_shipdate > DATE '1996-02-01'
GROUP BY
    l_orderkey,
    o_orderdate,
    o_shippriority
ORDER BY
    revenue DESC,
    o_orderdate
limit 10;
```

This is a shipping priority query, which provides the priority and potential revenue of the highest-revenue order that has not been shipped before a specified date. The potential revenue is defined as the sum of `l_extendedprice * (1-l_discount)`. The orders are listed in the descending order of revenue. In this example, this query lists the unshipped orders with potential query revenue in the top 10.

3.2.2.4 Step 4. Replicate the test data to the columnar storage engine

After TiFlash is deployed, TiKV does not replicate data to TiFlash immediately. You need to execute the following DDL statements in a MySQL client of TiDB to specify which tables need to be replicated. After that, TiDB will create the specified replicas in TiFlash accordingly.

```
ALTER TABLE test.customer SET TIFLASH REPLICA 1;
ALTER TABLE test.orders SET TIFLASH REPLICA 1;
ALTER TABLE test.lineitem SET TIFLASH REPLICA 1;
```

To check the replication status of the specific tables, execute the following statements:

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = 'test
    ↪ ' and TABLE_NAME = 'customer';
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = 'test
    ↪ ' and TABLE_NAME = 'orders';
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = 'test
    ↪ ' and TABLE_NAME = 'lineitem';
```

In the result of the above statements:

- `AVAILABLE` indicates whether the TiFlash replica of a specific table is available or not. `1` means available and `0` means unavailable. Once a replica becomes available, this status does not change any more. If you use DDL statements to modify the number of replicas, the replication status will be recalculated.
- `PROGRESS` means the progress of the replication. The value is between 0.0 and 1.0. 1 means at least one replica is replicated.

3.2.2.5 Step 5. Analyze data faster using HTAP

Execute the SQL statements in Step 3 again, and you can see the performance of TiDB HTAP.

For tables with TiFlash replicas, the TiDB optimizer automatically determines whether to use TiFlash replicas based on the cost estimation. To check whether or not a TiFlash replica is selected, you can use the `desc` or `explain analyze` statement. For example:

```
explain analyze SELECT
    l_orderkey,
    SUM(
```

```
        l_extendedprice * (1 - l_discount)
    ) AS revenue,
    o_orderdate,
    o_shippriority
FROM
    customer,
    orders,
    lineitem
WHERE
    c_mktsegment = 'BUILDING'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < DATE '1996-01-01'
AND l_shipdate > DATE '1996-02-01'
GROUP BY
    l_orderkey,
    o_orderdate,
    o_shippriority
ORDER BY
    revenue DESC,
    o_orderdate
limit 10;
```

If the result of the `EXPLAIN` statement shows `ExchangeSender` and `ExchangeReceiver` operators, it indicates that the MPP mode has taken effect.

In addition, you can specify that each part of the entire query is computed using only the TiFlash engine. For detailed information, see Use TiDB to read TiFlash replicas.

You can compare query results and query performance of these two methods.

3.2.3 What's next

- Architecture of TiDB HTAP
- Explore HTAP
- Use TiFlash

3.3 Explore SQL with TiDB

TiDB is compatible with MySQL, you can use MySQL statements directly in most of the cases. For unsupported features, see Compatibility with MySQL.

To experiment with SQL and test out TiDB compatibility with MySQL queries, you can run TiDB directly in your web browser without installing it. You can also first deploy a TiDB cluster and then run SQL statements in it.

This page walks you through the basic TiDB SQL statements such as DDL, DML and CRUD operations. For a complete list of TiDB statements, see TiDB SQL Syntax Diagram.

3.3.1 Category

SQL is divided into the following 4 types according to their functions:

- DDL (Data Definition Language): It is used to define database objects, including databases, tables, views, and indexes.
- DML (Data Manipulation Language): It is used to manipulate application related records.
- DQL (Data Query Language): It is used to query the records after conditional filtering.
- DCL (Data Control Language): It is used to define access privileges and security levels.

Common DDL features are creating, modifying, and deleting objects (such as tables and indexes). The corresponding commands are `CREATE`, `ALTER`, and `DROP`.

3.3.2 Show, create and drop a database

A database in TiDB can be considered as a collection of objects such as tables and indexes.

To show the list of databases, use the `SHOW DATABASES` statement:

```
SHOW DATABASES;
```

To use the database named `mysql`, use the following statement:

```
USE mysql;
```

To show all the tables in a database, use the `SHOW TABLES` statement:

```
SHOW TABLES FROM mysql;
```

To create a database, use the `CREATE DATABASE` statement:

```
CREATE DATABASE db_name [options];
```

To create a database named `samp_db`, use the following statement:

```
CREATE DATABASE IF NOT EXISTS samp_db;
```

Add `IF NOT EXISTS` to prevent an error if the database exists.

To delete a database, use the `DROP DATABASE` statement:

```
DROP DATABASE samp_db;
```

3.3.3 Create, show, and drop a table

To create a table, use the `CREATE TABLE` statement:

```
CREATE TABLE table_name column_name data_type constraint;
```

For example, to create a table named `person` which includes fields such as number, name, and birthday, use the following statement:

```
CREATE TABLE person (
    id INT(11),
    name VARCHAR(255),
    birthday DATE
    );
```

To view the statement that creates the table (DDL), use the `SHOW CREATE` statement:

```
SHOW CREATE table person;
```

To delete a table, use the `DROP TABLE` statement:

```
DROP TABLE person;
```

3.3.4 Create, show, and drop an index

Indexes are used to speed up queries on indexed columns. To create an index for the column whose value is not unique, use the `CREATE INDEX` statement:

```
CREATE INDEX person_id ON person (id);
```

Or use the `ALTER TABLE` statement:

```
ALTER TABLE person ADD INDEX person_id (id);
```

To create a unique index for the column whose value is unique, use the `CREATE UNIQUE INDEX` statement:

```
CREATE UNIQUE INDEX person_unique_id ON person (id);
```

Or use the `ALTER TABLE` statement:

```
ALTER TABLE person ADD UNIQUE person_unique_id (id);
```

To show all the indexes in a table, use the `SHOW INDEX` statement:

```
SHOW INDEX FROM person;
```

To delete an index, use the `DROP INDEX` or `ALTER TABLE` statement. `DROP INDEX` can be nested in `ALTER TABLE`:

```
DROP INDEX person_id ON person;
```

```
ALTER TABLE person DROP INDEX person_unique_id;
```

> **Note:**
>
> DDL operations are not transactions. You don't need to run a `COMMIT` statement when executing DDL operations.

3.3.5 Insert, update, and delete data

Common DML features are adding, modifying, and deleting table records. The corresponding commands are `INSERT`, `UPDATE`, and `DELETE`.

To insert data into a table, use the `INSERT` statement:

```
INSERT INTO person VALUES(1,'tom','20170912');
```

To insert a record containing data of some fields into a table, use the `INSERT` statement:

```
INSERT INTO person(id,name) VALUES('2','bob');
```

To update some fields of a record in a table, use the `UPDATE` statement:

```
UPDATE person SET birthday='20180808' WHERE id=2;
```

To delete the data in a table, use the `DELETE` statement:

```
DELETE FROM person WHERE id=2;
```

> **Note:**
>
> The `UPDATE` and `DELETE` statements without the `WHERE` clause as a filter operate on the entire table.

3.3.6 Query data

DQL is used to retrieve the desired data rows from a table or multiple tables.

To view the data in a table, use the `SELECT` statement:

```
SELECT * FROM person;
```

To query a specific column, add the column name after the `SELECT` keyword:

```
SELECT name FROM person;
```

```
+------+
| name |
+------+
| tom |
+------+
1 rows in set (0.00 sec)
```

Use the `WHERE` clause to filter all records that match the conditions and then return the result:

```
SELECT * FROM person where id<5;
```

3.3.7 Create, authorize, and delete a user

DCL are usually used to create or delete users, and manage user privileges.

To create a user, use the `CREATE USER` statement. The following example creates a user named `tiuser` with the password `123456`:

```
CREATE USER 'tiuser'@'localhost' IDENTIFIED BY '123456';
```

To grant `tiuser` the privilege to retrieve the tables in the `samp_db` database:

```
GRANT SELECT ON samp_db.* TO 'tiuser'@'localhost';
```

To check the privileges of `tiuser`:

```
SHOW GRANTS for tiuser@localhost;
```

To delete `tiuser`:

```
DROP USER 'tiuser'@'localhost';
```

3.4 Explore HTAP

This guide describes how to explore and use the features of TiDB Hybrid Transactional and Analytical Processing (HTAP).

Note:

If you are new to TiDB HTAP and want to start using it quickly, see Quick start with HTAP.

3.4.1 Use cases

TiDB HTAP can handle the massive data that increases rapidly, reduce the cost of DevOps, and be deployed in either self-hosted or cloud environments easily, which brings the value of data assets in real time.

The following are the typical use cases of HTAP:

- Hybrid workload

  When using TiDB for real-time Online Analytical Processing (OLAP) in hybrid load scenarios, you only need to provide an entry point of TiDB to your data. TiDB automatically selects different processing engines based on the specific business.

- Real-time stream processing

  When using TiDB in real-time stream processing scenarios, TiDB ensures that all the data flowed in constantly can be queried in real time. At the same time, TiDB also can handle highly concurrent data workloads and Business Intelligence (BI) queries.

- Data hub

  When using TiDB as a data hub, TiDB can meet specific business needs by seamlessly connecting the data for the application and the data warehouse.

For more information about use cases of TiDB HTAP, see blogs about HTAP on the PingCAP website.

3.4.2 Architecture

In TiDB, a row-based storage engine TiKV for Online Transactional Processing (OLTP) and a columnar storage engine TiFlash for Online Analytical Processing (OLAP) co-exist, replicate data automatically, and keep strong consistency.

For more information about the architecture, see architecture of TiDB HTAP.

3.4.3 Environment preparation

Before exploring the features of TiDB HTAP, you need to deploy TiDB and the corresponding storage engines according to the data volume. If the data volume is large (for example, 100 T), it is recommended to use TiFlash Massively Parallel Processing (MPP) as the primary solution and TiSpark as the supplementary solution.

- TiFlash
    - If you have deployed a TiDB cluster with no TiFlash node, add the TiFlash nodes in the current TiDB cluster. For detailed information, see Scale out a TiFlash cluster.
    - If you have not deployed a TiDB cluster, see Deploy a TiDB Cluster using TiUP. Based on the minimal TiDB topology, you also need to deploy the topology of TiFlash.
    - When deciding how to choose the number of TiFlash nodes, consider the following scenarios:
        - If your use case requires OLTP with small-scale analytical processing and Ad-Hoc queries, deploy one or several TiFlash nodes. They can dramatically increase the speed of analytic queries.
        - If the OLTP throughput does not cause significant pressure to I/O usage rate of the TiFlash nodes, each TiFlash node uses more resources for computation, and thus the TiFlash cluster can have near-linear scalability. The number of TiFlash nodes should be tuned based on expected performance and response time.
        - If the OLTP throughput is relatively high (for example, the write or update throughput is higher than 10 million lines/hours), due to the limited write capacity of network and physical disks, the I/O between TiKV and TiFlash becomes a bottleneck and is also prone to read and write hotspots. In this case, the number of TiFlash nodes has a complex non-linear relationship with the computation volume of analytical processing, so you need to tune the number of TiFlash nodes based on the actual status of the system.
- TiSpark
    - If your data needs to be analyzed with Spark, deploy TiSpark. For specific process, see TiSpark User Guide.

3.4.4 Data preparation

After TiFlash is deployed, TiKV does not replicate data to TiFlash automatically. You need to manually specify which tables need to be replicated to TiFlash. After that, TiDB creates the corresponding TiFlash replicas.

- If there is no data in the TiDB Cluster, migrate the data to TiDB first. For detailed information, see data migration.
- If the TiDB cluster already has the replicated data from upstream, after TiFlash is deployed, data replication does not automatically begin. You need to manually specify the tables to be replicated to TiFlash. For detailed information, see Use TiFlash.

3.4.5 Data processing

With TiDB, you can simply enter SQL statements for query or write requests. For the tables with TiFlash replicas, TiDB uses the front-end optimizer to automatically choose the optimal execution plan.

> **Note:**
>
> The MPP mode of TiFlash is enabled by default. When an SQL statement is executed, TiDB automatically determines whether to run in the MPP mode through the optimizer.
>
> - To disable the MPP mode of TiFlash, set the value of the tidb_allow_mpp system variable to `OFF`.
> - To forcibly enable MPP mode of TiFlash for query execution, set the values of tidb_allow_mpp and tidb_enforce_mpp to `ON`.
> - To check whether TiDB chooses the MPP mode to execute a specific query, see Explain Statements in the MPP Mode. If the output of `EXPLAIN` statement includes the `ExchangeSender` and `ExchangeReceiver` operators, the MPP mode is in use.

3.4.6 Performance monitoring

When using TiDB, you can monitor the TiDB cluster status and performance metrics in either of the following ways:

- TiDB Dashboard: you can see the overall running status of the TiDB cluster, analyse distribution and trends of read and write traffic, and learn the detailed execution information of slow queries.
- Monitoring system (Prometheus & Grafana): you can see the monitoring parameters of TiDB cluster-related componants including PD, TiDB, TiKV, TiFlash,TiCDC, and Node_exporter.

To see the alert rules of TiDB cluster and TiFlash cluster, see TiDB cluster alert rules and TiFlash alert rules.

3.4.7 Troubleshooting

If any issue occurs during using TiDB, refer to the following documents:

- Analyze slow queries
- Identify expensive queries
- Troubleshoot hotspot issues
- TiDB cluster troubleshooting guide
- Troubleshoot a TiFlash Cluster

You are also welcome to create Github Issues or submit your questions on AskTUG.

3.4.8 What's next

- To check the TiFlash version, critical logs, system tables, see Maintain a TiFlash cluster.
- To remove a specific TiFlash node, see Scale out a TiFlash cluster.

3.5 Import Example Database

Examples used in the TiDB manual use System Data from Capital Bikeshare, released under the Capital Bikeshare Data License Agreement.

3.5.1 Download all data files

The system data is available for download in .zip files organized per year. Downloading and extracting all files requires approximately 3GB of disk space. To download all files for years 2010-2017 using a bash script:

```
mkdir -p bikeshare-data && cd bikeshare-data

curl -L --remote-name-all https://s3.amazonaws.com/capitalbikeshare-data
    ↪ /{2010..2017}-capitalbikeshare-tripdata.zip
unzip \*-tripdata.zip
```

3.5.2 Load data into TiDB

The system data can be imported into TiDB using the following schema:

```
CREATE DATABASE bikeshare;
USE bikeshare;

CREATE TABLE trips (
 trip_id bigint NOT NULL PRIMARY KEY AUTO_INCREMENT,
```

```
 duration integer not null,
 start_date datetime,
 end_date datetime,
 start_station_number integer,
 start_station varchar(255),
 end_station_number integer,
 end_station varchar(255),
 bike_number varchar(255),
 member_type varchar(255)
);
```

You can import files individually using the example `LOAD DATA` command here, or import all files using the bash loop below:

```
SET tidb_dml_batch_size = 20000;
LOAD DATA LOCAL INFILE '2017Q1-capitalbikeshare-tripdata.csv' INTO TABLE
    ↪ trips
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES
(duration, start_date, end_date, start_station_number, start_station,
end_station_number, end_station, bike_number, member_type);
```

3.5.2.1 Import all files

> **Note:**
>
> When you start the MySQL client, use the `--local-infile=1` option.

To import all `*.csv` files into TiDB in a bash loop:

```
for FILE in *.csv; do
 echo "== $FILE =="
 mysql bikeshare --local-infile=1 -e "SET tidb_dml_batch_size = 20000; LOAD
    ↪ DATA LOCAL INFILE '${FILE}' INTO TABLE trips FIELDS TERMINATED BY
    ↪ ',' ENCLOSED BY '\"' LINES TERMINATED BY '\r\n' IGNORE 1 LINES (
    ↪ duration, start_date, end_date, start_station_number, start_station,
    ↪  end_station_number, end_station, bike_number, member_type);"
done;
```

4 Develop

4.1 Developer Guide Overview

This guide is written for application developers, but if you are interested in the inner workings of TiDB or want to get involved in TiDB development, read the TiDB Kernel Development Guide for more information about TiDB.

This tutorial shows how to quickly build an application using TiDB, the possible use cases of TiDB and how to handle common problems.

Before reading this page, it is recommended that you read the Quick Start Guide for the TiDB Database Platform.

4.1.1 TiDB basics

Before you start working with TiDB, you need to understand some important mechanisms of how TiDB works:

- Read the TiDB Transaction Overview to understand how transactions work in TiDB, or check out the Transaction Notes for Application Developers to learn about transaction knowledge required for application development.
- Understand the way applications interact with TiDB.
- To learn core components and concepts of building up the distributed database TiDB and TiDB Cloud, refer to the free online course Introduction to TiDB.

4.1.2 TiDB transaction mechanisms

TiDB supports distributed transactions and offers both optimistic transaction and pessimistic transaction modes. The current version of TiDB uses the **pessimistic transaction** mode by default, which allows you to transact with TiDB as you would with a traditional monolithic database (for example, MySQL).

You can start a transaction using `BEGIN`, explicitly specify a **pessimistic transaction** using `BEGIN PESSIMISTIC`, or explicitly specify an **optimistic transaction** using `BEGIN OPTIMISTIC`. After that, you can either commit (`COMMIT`) or roll back (`ROLLBACK`) the transaction.

TiDB guarantees atomicity for all statements between the start of `BEGIN` and the end of `COMMIT` or `ROLLBACK`, that is, all statements that are executed during this period either succeed or fail as a whole. This is used to ensure data consistency you need for application development.

If you are not sure what an **optimistic transaction** is, do ***NOT*** use it yet. Because **optimistic transactions** require that the application can correctly handle all errors returned by the `COMMIT` statement. If you are not sure how your application handles them, use a **pessimistic transaction** instead.

4.1.3 The way applications interact with TiDB

TiDB is highly compatible with the MySQL protocol and supports most MySQL syntax and features, so most MySQL connection libraries are compatible with TiDB. If your application framework or language does not have an official adaptation from PingCAP, it is recommended that you use MySQL's client libraries. More and more third-party libraries are actively supporting TiDB's different features.

Since TiDB is compatible with the MySQL protocol and MySQL syntax, most of the ORMs that support MySQL are also compatible with TiDB.

4.1.4 Read More

- Quick Start
- Choose Driver or ORM

- Connect to TiDB

- Database Schema Design
- Write Data
- Read Data
- Transaction
- Optimize
- Example Applications

4.2 Quick Start

4.2.1 Build a TiDB Serverless Cluster

This document walks you through the quickest way to get started with TiDB. You will use TiDB Cloud to create a TiDB Serverless cluster, connect to it, and run a sample application on it.

If you need to run TiDB on your local machine, see Starting TiDB Locally.

4.2.1.1 Step 1. Create a TiDB Serverless cluster

1. If you do not have a TiDB Cloud account, click here to sign up for an account.
2. Log in to your TiDB Cloud account.

   The **Clusters** list page is displayed by default.
3. For new sign-up users, TiDB Cloud creates a default TiDB Serverless cluster `Cluster0` for you automatically. You can either use this default cluster for the subsequent steps or create a new TiDB Serverless cluster on your own.

   To create a new TiDB Serverless cluster on your own, take the following operations:

1. Click **Create Cluster**.
2. On the **Create Cluster** page, **Serverless** is selected by default. Update the default cluster name if necessary, select a target region of your cluster, and then click **Create**. Your TiDB Serverless cluster will be created in approximately 30 seconds.

4. Click the target cluster name to go to its overview page, and then click **Connect** in the upper-right corner. A connection dialog box is displayed.
5. In the dialog, select your preferred connection method and operating system to get the corresponding connection string. This document uses MySQL client as an example.
6. Click **Create password** to generate a random password. The generated password will not show again, so save your password in a secure location. If you do not set a root password, you cannot connect to the cluster.

> **Note:**
>
> For TiDB Serverless clusters, when you connect to your cluster, you must include the prefix for your cluster in the user name and wrap the name with quotation marks. For more information, see User name prefix.

4.2.1.2 Step 2. Connect to a cluster

1. If the MySQL client is not installed, select your operating system and follow the steps below to install it.

For macOS, install Homebrew if you do not have it, and then run the following command to install the MySQL client:

```
brew install mysql-client
```

The output is as follows:

```
mysql-client is keg-only, which means it was not symlinked into /opt/
    ↪ homebrew,
because it conflicts with mysql (which contains client libraries).

If you need to have mysql-client first in your PATH, run:
  echo 'export PATH="/opt/homebrew/opt/mysql-client/bin:$PATH"' >> ~/.zshrc

For compilers to find mysql-client you may need to set:
  export LDFLAGS="-L/opt/homebrew/opt/mysql-client/lib"
  export CPPFLAGS="-I/opt/homebrew/opt/mysql-client/include"
```

To add the MySQL client to your PATH, locate the following command in the above output (if your output is inconsistent with the above output in the document, use the corresponding command in your output instead) and run it:

```
echo 'export PATH="/opt/homebrew/opt/mysql-client/bin:$PATH"' >> ~/.zshrc
```

Then, declare the global environment variable by the `source` command and verify that the MySQL client is installed successfully:

```
source ~/.zshrc
mysql --version
```

An example of the expected output:

```
mysql Ver 8.0.28 for macos12.0 on arm64 (Homebrew)
```

For Linux, the following takes CentOS 7 as an example:

```
yum install mysql
```

Then, verify that the MySQL client is installed successfully:

```
mysql --version
```

An example of the expected output:

```
mysql Ver 15.1 Distrib 5.5.68-MariaDB, for Linux (x86_64) using readline
    ↪ 5.1
```

2. Run the connection string obtained in Step 1.

```
mysql --connect-timeout 15 -u '<prefix>.root' -h <host> -P 4000 -D test
    ↪  --ssl-mode=VERIFY_IDENTITY --ssl-ca=/etc/ssl/cert.pem -p
```

Note:

- When you connect to a TiDB Serverless cluster, you must use the TLS connection.
- If you encounter problems when connecting to a TiDB Serverless cluster, you can read Secure Connections to TiDB Serverless Clusters for more information.

3. Fill in the password to sign in.

4.2.1.3 Step 3. Execute a SQL statement

Let's try to execute your first SQL statement on TiDB Cloud.

```
SELECT 'Hello TiDB Cloud!';
```

Expected output:

```
+--------------------+
| Hello TiDB Cloud! |
+--------------------+
| Hello TiDB Cloud! |
+--------------------+
```

If your actual output is similar to the expected output, congratulations, you have successfully execute a SQL statement on TiDB Cloud.

4.2.2 CRUD SQL in TiDB

This document briefly introduces how to use TiDB's CURD SQL.

4.2.2.1 Before you start

Please make sure you are connected to a TiDB cluster. If not, refer to Build a TiDB Serverless Cluster to create a TiDB Serverless cluster.

4.2.2.2 Explore SQL with TiDB

> **Note:**
>
> This document references and simplifies Explore SQL with TiDB. For more details, see Explore SQL with TiDB.

TiDB is compatible with MySQL, you can use MySQL statements directly in most cases. For unsupported features, see Compatibility with MySQL.

To experiment with SQL and test out TiDB compatibility with MySQL queries, you can try Chat2Query (beta) in your TiDB Cloud console. To access Chat2Query, go to the **Clusters** page of your project, click your cluster name, and then click **Chat2Query** in the left navigation pane. In Chat2Query, you can either let AI generate SQL queries automatically or write SQL queries manually, and run SQL queries against databases without a terminal.

Note:

Chat2Query currently has limited support on SQL statements. DDLs such as `CREATE TABLE` or `DROP TABLE` are not supported yet.

If you want to explore all SQL statements with TiDB, you can deploy a TiDB cluster and then run SQL statements in it.

This page walks you through the basic TiDB SQL statements such as DDL, DML, and CRUD operations. For a complete list of TiDB statements, see TiDB SQL Syntax Diagram.

4.2.2.3 Category

SQL is divided into the following 4 types according to their functions:

- **DDL (Data Definition Language)**: It is used to define database objects, including databases, tables, views, and indexes.
- **DML (Data Manipulation Language)**: It is used to manipulate application related records.
- **DQL (Data Query Language)**: It is used to query the records after conditional filtering.
- **DCL (Data Control Language)**: It is used to define access privileges and security levels.

The following mainly introduces DML and DQL. For more information about DDL and DCL, see Explore SQL with TiDB or TiDB SQL syntax detailed explanation.

4.2.2.4 Data Manipulation Language

Common DML features are adding, modifying, and deleting table records. The corresponding commands are `INSERT`, `UPDATE`, and `DELETE`.

To insert data into a table, use the `INSERT` statement:

```
INSERT INTO person VALUES(1,'tom','20170912');
```

To insert a record containing data of some fields into a table, use the `INSERT` statement:

```
INSERT INTO person(id,name) VALUES('2','bob');
```

To update some fields of a record in a table, use the `UPDATE` statement:

```
UPDATE person SET birthday='20180808' WHERE id=2;
```

To delete the data in a table, use the `DELETE` statement:

```
DELETE FROM person WHERE id=2;
```

> **Note:**
>
> The `UPDATE` and `DELETE` statements without the `WHERE` clause as a filter operate on the entire table.

4.2.2.5 Data Query Language

DQL is used to retrieve the desired data rows from a table or multiple tables.

To view the data in a table, use the `SELECT` statement:

```
SELECT * FROM person;
```

To query a specific column, add the column name after the `SELECT` keyword:

```
SELECT name FROM person;
```

The result is as follows:

```
+------+
| name |
+------+
| tom |
+------+
1 rows in set (0.00 sec)
```

Use the `WHERE` clause to filter all records that match the conditions and then return the result:

```
SELECT * FROM person WHERE id < 5;
```

4.3 Example Applications

4.3.1 Build a Simple CRUD App with TiDB and Golang

This document describes how to use TiDB and Golang to build a simple CRUD application.

> **Note:**
>
> It is recommended to use Golang 1.20 or a later version.

4.3.1.1 Step 1. Launch your TiDB cluster

The following introduces how to start a TiDB cluster.

Use a TiDB Serverless cluster

For detailed steps, see Create a TiDB Serverless cluster.

Use a local cluster

For detailed steps, see Deploy a local test cluster or Deploy a TiDB Cluster Using TiUP.

4.3.1.2 Step 2. Get the code

```
git clone https://github.com/pingcap-inc/tidb-example-golang.git
```

Compared with GORM, the go-sql-driver/mysql implementation might be not a best practice, because you need to write error handling logic, close `*sql.Rows` manually and cannot reuse code easily, which makes your code slightly redundant.

GORM is a popular open-source ORM library for Golang. The following instructions take `v1.23.5` as an example.

To adapt TiDB transactions, write a toolkit util according to the following code:

```
package util

import (
    "context"
    "database/sql"
)

type TiDBSqlTx struct {
    *sql.Tx
    conn       *sql.Conn
    pessimistic bool
}

func TiDBSqlBegin(db *sql.DB, pessimistic bool) (*TiDBSqlTx, error) {
    ctx := context.Background()
    conn, err := db.Conn(ctx)
    if err != nil {
```

```
        return nil, err
    }
    if pessimistic {
        _, err = conn.ExecContext(ctx, "set @@tidb_txn_mode=?", "pessimistic
            ↪ ")
    } else {
        _, err = conn.ExecContext(ctx, "set @@tidb_txn_mode=?", "optimistic")
    }
    if err != nil {
        return nil, err
    }
    tx, err := conn.BeginTx(ctx, nil)
    if err != nil {
        return nil, err
    }
    return &TiDBSqlTx{
        conn:       conn,
        Tx:         tx,
        pessimistic: pessimistic,
    }, nil
}

func (tx *TiDBSqlTx) Commit() error {
    defer tx.conn.Close()
    return tx.Tx.Commit()
}

func (tx *TiDBSqlTx) Rollback() error {
    defer tx.conn.Close()
    return tx.Tx.Rollback()
}
```

Change to the `gorm` directory:

```
cd gorm
```

The structure of this directory is as follows:

```
.
├── Makefile
├── go.mod
├── go.sum
└── gorm.go
```

`gorm.go` is the main body of the `gorm`. Compared with go-sql-driver/mysql, GORM avoids differences in database creation between different databases. It also implements a lot

of operations, such as AutoMigrate and CRUD of objects, which greatly simplifies the code.

`Player` is a data entity struct that is a mapping for tables. Each property of a `Player` corresponds to a field in the `player` table. Compared with go-sql-driver/mysql, `Player` in GORM adds struct tags to indicate mapping relationships for more information, such as `gorm:"primaryKey;type:VARCHAR(36);column:id"`.

```
package main

import (
    "fmt"
    "math/rand"

    "github.com/google/uuid"
    "github.com/pingcap-inc/tidb-example-golang/util"

    "gorm.io/driver/mysql"
    "gorm.io/gorm"
    "gorm.io/gorm/clause"
    "gorm.io/gorm/logger"
)

type Player struct {
    ID    string `gorm:"primaryKey;type:VARCHAR(36);column:id"`
    Coins int   `gorm:"column:coins"`
    Goods int   `gorm:"column:goods"`
}

func (*Player) TableName() string {
    return "player"
}

func main() {
    // 1. Configure the example database connection.
    db := createDB()

    // AutoMigrate for player table
    db.AutoMigrate(&Player{})

    // 2. Run some simple examples.
    simpleExample(db)

    // 3. Explore more.
    tradeExample(db)
}
```

```
func tradeExample(db *gorm.DB) {
    // Player 1: id is "1", has only 100 coins.
    // Player 2: id is "2", has 114514 coins, and 20 goods.
    player1 := &Player{ID: "1", Coins: 100}
    player2 := &Player{ID: "2", Coins: 114514, Goods: 20}

    // Create two players "by hand", using the INSERT statement on the
    ↪ backend.
    db.Clauses(clause.OnConflict{UpdateAll: true}).Create(player1)
    db.Clauses(clause.OnConflict{UpdateAll: true}).Create(player2)

    // Player 1 wants to buy 10 goods from player 2.
    // It will cost 500 coins, but player 1 cannot afford it.
    fmt.Println("\nbuyGoods:\n => this trade will fail")
    if err := buyGoods(db, player2.ID, player1.ID, 10, 500); err == nil {
        panic("there shouldn't be success")
    }

    // So player 1 has to reduce the incoming quantity to two.
    fmt.Println("\nbuyGoods:\n => this trade will success")
    if err := buyGoods(db, player2.ID, player1.ID, 2, 100); err != nil {
        panic(err)
    }
}

func simpleExample(db *gorm.DB) {
    // Create a player, who has a coin and a goods.
    if err := db.Clauses(clause.OnConflict{UpdateAll: true}).
        Create(&Player{ID: "test", Coins: 1, Goods: 1}).Error; err != nil {
        panic(err)
    }

    // Get a player.
    var testPlayer Player
    db.Find(&testPlayer, "id = ?", "test")
    fmt.Printf("getPlayer: %+v\n", testPlayer)

    // Create players with bulk inserts. Insert 1919 players totally, with
    ↪ 114 players per batch.
    bulkInsertPlayers := make([]Player, 1919, 1919)
    total, batch := 1919, 114
    for i := 0; i < total; i++ {
        bulkInsertPlayers[i] = Player{
            ID:    uuid.New().String(),
```

```
            Coins: rand.Intn(10000),
            Goods: rand.Intn(10000),
        }
    }

    if err := db.Session(&gorm.Session{Logger: db.Logger.LogMode(logger.
        ↪ Error)}).
        CreateInBatches(bulkInsertPlayers, batch).Error; err != nil {
        panic(err)
    }

    // Count players amount.
    playersCount := int64(0)
    db.Model(&Player{}).Count(&playersCount)
    fmt.Printf("countPlayers: %d\n", playersCount)

    // Print 3 players.
    threePlayers := make([]Player, 3, 3)
    db.Limit(3).Find(&threePlayers)
    for index, player := range threePlayers {
        fmt.Printf("print %d player: %+v\n", index+1, player)
    }
}

func createDB() *gorm.DB {
    dsn := "root:@tcp(127.0.0.1:4000)/test?charset=utf8mb4"
    db, err := gorm.Open(mysql.Open(dsn), &gorm.Config{
        Logger: logger.Default.LogMode(logger.Info),
    })
    if err != nil {
        panic(err)
    }

    return db
}

func buyGoods(db *gorm.DB, sellID, buyID string, amount, price int) error {
    return util.TiDBGormBegin(db, true, func(tx *gorm.DB) error {
        var sellPlayer, buyPlayer Player
        if err := tx.Clauses(clause.Locking{Strength: "UPDATE"}).
            Find(&sellPlayer, "id = ?", sellID).Error; err != nil {
            return err
        }

        if sellPlayer.ID != sellID || sellPlayer.Goods < amount {
```

```
            return fmt.Errorf("sell player %s goods not enough", sellID)
        }

        if err := tx.Clauses(clause.Locking{Strength: "UPDATE"}).
            Find(&buyPlayer, "id = ?", buyID).Error; err != nil {
            return err
        }

        if buyPlayer.ID != buyID || buyPlayer.Coins < price {
            return fmt.Errorf("buy player %s coins not enough", buyID)
        }

        updateSQL := "UPDATE player set goods = goods + ?, coins = coins + ?
            ↪ WHERE id = ?"
        if err := tx.Exec(updateSQL, -amount, price, sellID).Error; err !=
            ↪ nil {
            return err
        }

        if err := tx.Exec(updateSQL, amount, -price, buyID).Error; err != nil
            ↪  {
            return err
        }

        fmt.Println("\n[buyGoods]:\n 'trade success'")
        return nil
    })
}
```

Change to the `sqldriver` directory:

```
cd sqldriver
```

The structure of this directory is as follows:

```
.
├── Makefile
├── dao.go
├── go.mod
├── go.sum
├── sql
│   └── dbinit.sql
├── sql.go
└── sqldriver.go
```

You can find initialization statements for the table creation in `dbinit.sql`:

```
USE test;
DROP TABLE IF EXISTS player;

CREATE TABLE player (
    `id` VARCHAR(36),
    `coins` INTEGER,
    `goods` INTEGER,
   PRIMARY KEY (`id`)
);
```

`sqldriver.go` is the main body of the `sqldriver`. TiDB is highly compatible with the MySQL protocol, so you need to initialize a MySQL source instance `db, err := sql.Open ↪ ("mysql", dsn)` to connect to TiDB. Then, you can use `dao.go` to read, edit, add, and delete data.

```
package main

import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)

func main() {
    // 1. Configure the example database connection.
    dsn := "root:@tcp(127.0.0.1:4000)/test?charset=utf8mb4"
    openDB("mysql", dsn, func(db *sql.DB) {
        // 2. Run some simple examples.
        simpleExample(db)

        // 3. Explore more.
        tradeExample(db)
    })
}

func simpleExample(db *sql.DB) {
    // Create a player, who has a coin and a goods.
    err := createPlayer(db, Player{ID: "test", Coins: 1, Goods: 1})
    if err != nil {
        panic(err)
    }

    // Get a player.
    testPlayer, err := getPlayer(db, "test")
```

```
    if err != nil {
        panic(err)
    }
    fmt.Printf("getPlayer: %+v\n", testPlayer)

    // Create players with bulk inserts. Insert 1919 players totally, with
        ↪ 114 players per batch.

    err = bulkInsertPlayers(db, randomPlayers(1919), 114)
    if err != nil {
        panic(err)
    }

    // Count players amount.
    playersCount, err := getCount(db)
    if err != nil {
        panic(err)
    }
    fmt.Printf("countPlayers: %d\n", playersCount)

    // Print 3 players.
    threePlayers, err := getPlayerByLimit(db, 3)
    if err != nil {
        panic(err)
    }
    for index, player := range threePlayers {
        fmt.Printf("print %d player: %+v\n", index+1, player)
    }
}

func tradeExample(db *sql.DB) {
    // Player 1: id is "1", has only 100 coins.
    // Player 2: id is "2", has 114514 coins, and 20 goods.
    player1 := Player{ID: "1", Coins: 100}
    player2 := Player{ID: "2", Coins: 114514, Goods: 20}

    // Create two players "by hand", using the INSERT statement on the
        ↪ backend.
    if err := createPlayer(db, player1); err != nil {
        panic(err)
    }
    if err := createPlayer(db, player2); err != nil {
        panic(err)
    }
```

```
    // Player 1 wants to buy 10 goods from player 2.
    // It will cost 500 coins, but player 1 cannot afford it.
    fmt.Println("\nbuyGoods:\n => this trade will fail")
    if err := buyGoods(db, player2.ID, player1.ID, 10, 500); err == nil {
        panic("there shouldn't be success")
    }

    // So player 1 has to reduce the incoming quantity to two.
    fmt.Println("\nbuyGoods:\n => this trade will success")
    if err := buyGoods(db, player2.ID, player1.ID, 2, 100); err != nil {
        panic(err)
    }
}

func openDB(driverName, dataSourceName string, runnable func(db *sql.DB)) {
    db, err := sql.Open(driverName, dataSourceName)
    if err != nil {
        panic(err)
    }
    defer db.Close()

    runnable(db)
}
```

To adapt TiDB transactions, write a toolkit util according to the following code:

```
package util

import (
    "context"
    "database/sql"
)

type TiDBSqlTx struct {
    *sql.Tx
    conn       *sql.Conn
    pessimistic bool
}

func TiDBSqlBegin(db *sql.DB, pessimistic bool) (*TiDBSqlTx, error) {
    ctx := context.Background()
    conn, err := db.Conn(ctx)
    if err != nil {
        return nil, err
    }
```

```
    if pessimistic {
        _, err = conn.ExecContext(ctx, "set @@tidb_txn_mode=?", "pessimistic
            ↪ ")
    } else {
        _, err = conn.ExecContext(ctx, "set @@tidb_txn_mode=?", "optimistic")
    }
    if err != nil {
        return nil, err
    }
    tx, err := conn.BeginTx(ctx, nil)
    if err != nil {
        return nil, err
    }
    return &TiDBSqlTx{
        conn:       conn,
        Tx:         tx,
        pessimistic: pessimistic,
    }, nil
}

func (tx *TiDBSqlTx) Commit() error {
    defer tx.conn.Close()
    return tx.Tx.Commit()
}

func (tx *TiDBSqlTx) Rollback() error {
    defer tx.conn.Close()
    return tx.Tx.Rollback()
}
```

dao.go defines a set of data manipulation methods to provide the ability to write data. This is also the core part of this example.

```
package main

import (
    "database/sql"
    "fmt"
    "math/rand"
    "strings"

    "github.com/google/uuid"
    "github.com/pingcap-inc/tidb-example-golang/util"
)
```

```
type Player struct {
    ID    string
    Coins int
    Goods int
}

// createPlayer create a player
func createPlayer(db *sql.DB, player Player) error {
    _, err := db.Exec(CreatePlayerSQL, player.ID, player.Coins, player.Goods
        ↪ )
    return err
}

// getPlayer get a player
func getPlayer(db *sql.DB, id string) (Player, error) {
    var player Player

    rows, err := db.Query(GetPlayerSQL, id)
    if err != nil {
        return player, err
    }
    defer rows.Close()

    if rows.Next() {
        err = rows.Scan(&player.ID, &player.Coins, &player.Goods)
        if err == nil {
            return player, nil
        } else {
            return player, err
        }
    }

    return player, fmt.Errorf("can not found player")
}

// getPlayerByLimit get players by limit
func getPlayerByLimit(db *sql.DB, limit int) ([]Player, error) {
    var players []Player

    rows, err := db.Query(GetPlayerByLimitSQL, limit)
    if err != nil {
        return players, err
    }
    defer rows.Close()
```

```
    for rows.Next() {
        player := Player{}
        err = rows.Scan(&player.ID, &player.Coins, &player.Goods)
        if err == nil {
            players = append(players, player)
        } else {
            return players, err
        }
    }

    return players, nil
}

// bulk-insert players
func bulkInsertPlayers(db *sql.DB, players []Player, batchSize int) error {
    tx, err := util.TiDBSqlBegin(db, true)
    if err != nil {
        return err
    }

    stmt, err := tx.Prepare(buildBulkInsertSQL(batchSize))
    if err != nil {
        return err
    }

    defer stmt.Close()

    for len(players) > batchSize {
        if _, err := stmt.Exec(playerToArgs(players[:batchSize])...); err !=
            ↪ nil {
            tx.Rollback()
            return err
        }

        players = players[batchSize:]
    }

    if len(players) != 0 {
        if _, err := tx.Exec(buildBulkInsertSQL(len(players)), playerToArgs(
            ↪ players)...); err != nil {
            tx.Rollback()
            return err
        }
    }
```

```
    if err := tx.Commit(); err != nil {
        tx.Rollback()
        return err
    }

    return nil
}

func getCount(db *sql.DB) (int, error) {
    count := 0

    rows, err := db.Query(GetCountSQL)
    if err != nil {
        return count, err
    }

    defer rows.Close()

    if rows.Next() {
        if err := rows.Scan(&count); err != nil {
            return count, err
        }
    }

    return count, nil
}

func buyGoods(db *sql.DB, sellID, buyID string, amount, price int) error {
    var sellPlayer, buyPlayer Player

    tx, err := util.TiDBSqlBegin(db, true)
    if err != nil {
        return err
    }

    buyExec := func() error {
        stmt, err := tx.Prepare(GetPlayerWithLockSQL)
        if err != nil {
            return err
        }
        defer stmt.Close()

        sellRows, err := stmt.Query(sellID)
        if err != nil {
            return err
```

```
    }
    defer sellRows.Close()

    if sellRows.Next() {
        if err := sellRows.Scan(&sellPlayer.ID, &sellPlayer.Coins, &
            ↪ sellPlayer.Goods); err != nil {
            return err
        }
    }
    sellRows.Close()

    if sellPlayer.ID != sellID || sellPlayer.Goods < amount {
        return fmt.Errorf("sell player %s goods not enough", sellID)
    }

    buyRows, err := stmt.Query(buyID)
    if err != nil {
        return err
    }
    defer buyRows.Close()

    if buyRows.Next() {
        if err := buyRows.Scan(&buyPlayer.ID, &buyPlayer.Coins, &
            ↪ buyPlayer.Goods); err != nil {
            return err
        }
    }
    buyRows.Close()

    if buyPlayer.ID != buyID || buyPlayer.Coins < price {
        return fmt.Errorf("buy player %s coins not enough", buyID)
    }

    updateStmt, err := tx.Prepare(UpdatePlayerSQL)
    if err != nil {
        return err
    }
    defer updateStmt.Close()

    if _, err := updateStmt.Exec(-amount, price, sellID); err != nil {
        return err
    }

    if _, err := updateStmt.Exec(amount, -price, buyID); err != nil {
        return err
```

```
        }

        return nil
    }

    err = buyExec()
    if err == nil {
        fmt.Println("\n[buyGoods]:\n 'trade success'")
        tx.Commit()
    } else {
        tx.Rollback()
    }

    return err
}

func playerToArgs(players []Player) []interface{} {
    var args []interface{}
    for _, player := range players {
        args = append(args, player.ID, player.Coins, player.Goods)
    }
    return args
}

func buildBulkInsertSQL(amount int) string {
    return CreatePlayerSQL + strings.Repeat(",(?,?,?)", amount-1)
}

func randomPlayers(amount int) []Player {
    players := make([]Player, amount, amount)
    for i := 0; i < amount; i++ {
        players[i] = Player{
            ID:    uuid.New().String(),
            Coins: rand.Intn(10000),
            Goods: rand.Intn(10000),
        }
    }

    return players
}
```

sql.go defines SQL statements as constants:

```
package main
```

```
const (
    CreatePlayerSQL    = "INSERT INTO player (id, coins, goods) VALUES (?, ?,
        ↪  ?)"
    GetPlayerSQL       = "SELECT id, coins, goods FROM player WHERE id = ?"
    GetCountSQL        = "SELECT count(*) FROM player"
    GetPlayerWithLockSQL = GetPlayerSQL + " FOR UPDATE"
    UpdatePlayerSQL    = "UPDATE player set goods = goods + ?, coins = coins
        ↪ + ? WHERE id = ?"
    GetPlayerByLimitSQL = "SELECT id, coins, goods FROM player LIMIT ?"
)
```

4.3.1.3 Step 3. Run the code

The following content introduces how to run the code step by step.

4.3.1.3.1 Step 3.1 Table initialization

No need to initialize tables manually.

When using go-sql-driver/mysql, you need to initialize the database tables manually. If you are using a local cluster, and MySQL client has been installed locally, you can run it directly in the `sqldriver` directory:

```
make mysql
```

Or you can execute the following command:

```
mysql --host 127.0.0.1 --port 4000 -u root<sql/dbinit.sql
```

If you are using a non-local cluster or MySQL client has not been installed, connect to your cluster and run the statement in the `sql/dbinit.sql` file.

4.3.1.3.2 Step 3.2 Modify parameters for TiDB Cloud

If you are using a TiDB Serverless cluster, modify the value of the `dsn` in `gorm.go`:

```
dsn := "root:@tcp(127.0.0.1:4000)/test?charset=utf8mb4"
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the `mysql.RegisterTLSConfig` and `dsn` as follows:

```
mysql.RegisterTLSConfig("register-tidb-tls", &tls.Config {
    MinVersion: tls.VersionTLS12,
    ServerName: "xxx.tidbcloud.com",
})

dsn := "2aEp24QWEDLqRFs.root:123456@tcp(xxx.tidbcloud.com:4000)/test?charset
    ↪ =utf8mb4&tls=register-tidb-tls"
```

If you are using a TiDB Serverless cluster, modify the value of the `dsn` in `sqldriver.go`:

```
dsn := "root:@tcp(127.0.0.1:4000)/test?charset=utf8mb4"
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the `mysql.RegisterTLSConfig` and `dsn` as follows:

```
mysql.RegisterTLSConfig("register-tidb-tls", &tls.Config {
    MinVersion: tls.VersionTLS12,
    ServerName: "xxx.tidbcloud.com",
})

dsn := "2aEp24QWEDLqRFs.root:123456@tcp(xxx.tidbcloud.com:4000)/test?charset
    ↪ =utf8mb4&tls=register-tidb-tls"
```

4.3.1.1.3.3 Step 3.3 Run

To run the code, you can run `make build` and `make run` respectively:

```
make build # this command executes `go build -o bin/gorm-example`
make run # this command executes `./bin/gorm-example`
```

Or you can use the native commands:

```
go build -o bin/gorm-example
./bin/gorm-example
```

Or run the `make` command directly, which is a combination of `make build` and `make run`.

To run the code, you can run `make mysql`, `make build` and `make run` respectively:

```
make mysql # this command executes `mysql --host 127.0.0.1 --port 4000 -u
    ↪ root<sql/dbinit.sql`
make build # this command executes `go build -o bin/sql-driver-example`
make run # this command executes `./bin/sql-driver-example`
```

Or you can use the native commands:

```
mysql --host 127.0.0.1 --port 4000 -u root<sql/dbinit.sql
go build -o bin/sql-driver-example
./bin/sql-driver-example
```

Or run the `make all` command directly, which is a combination of `make mysql`, `make build` and `make run`.

4.3.1.4 Step 4. Expected output

GORM Expected Output

go-sql-driver/mysql Expected Output

4.3.2 Build a TiDB Application Using Spring Boot

This tutorial shows you how to build a Spring Boot web application using TiDB. The Spring Data JPA module is used as the framework for data access capabilities. You can download the code for this sample application from GitHub.

This is a sample application for building a RESTful API, which shows a generic **Spring Boot** backend service using **TiDB** as the database. The following process was designed to recreate a real-world scenario:

This is an example of a game where each player has two attributes: `coins` and `goods` . Each player is uniquely identified by an `id` field. Players can trade freely if they have sufficient coins and goods.

You can build your own application based on this example.

4.3.2.1 Step 1: Launch your TiDB cluster

The following introduces how to start a TiDB cluster.

Use a TiDB Serverless cluster

For detailed steps, see Create a TiDB Serverless cluster.

Use a local cluster

For detailed steps, see Deploy a local test cluster or Deploy a TiDB Cluster Using TiUP.

4.3.2.2 Step 2: Install JDK

Download and install the **Java Development Kit** (JDK) on your computer. It is a necessary tool for Java development. **Spring Boot** supports JDK for Java 8 and later versions. However, due to the **Hibernate** version, it is recommended that you use JDK for Java 11 and later versions.

Both **Oracle JDK** and **OpenJDK** are supported. You can choose at your own discretion. This tutorial uses JDK 17 from **OpenJDK**.

4.3.2.3 Step 3: Install Maven

This sample application uses **Apache Maven** to manage the application's dependencies. Spring supports Maven 3.3 and later versions. As dependency management software, the latest stable version of **Maven** is recommended.

To install **Maven** from the command line.

- macOS:

```
brew install maven
```

- Debian-based Linux distributions (such as Ubuntu):

```
apt-get install maven
```

- Red Hat-based Linux distributions (such as Fedora, CentOS):
  - dnf:

```
dnf install maven
```

  - yum:

```
yum install maven
```

For other installation methods, refer to the Maven official documentation.

4.3.2.4 Step 4: Get the application code

Download or clone the sample code repository and navigate to the `spring-jpa-hibernate` directory.

4.3.2.5 Step 5: Run the application

In this step, the application code is compiled and run, which produces a web application. Hibernate creates a `player_jpa` table within the `test` database. If you make requests using the application's RESTful API, these requests run database transactions on the TiDB cluster.

If you want to learn more about the code of this application, refer to Implementation details.

4.3.2.5.1 Step 5.1 Change parameters

If you are using a TiDB Serverless cluster, change the `spring.datasource.url`, `spring.datasource.username`, `spring.datasource.password` parameters in the `application.yml` (located in `src/main/resources`).

```
spring:
  datasource:
    url: jdbc:mysql://localhost:4000/test
    username: root
    #    password: xxx
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    show-sql: true
    database-platform: org.hibernate.dialect.TiDBDialect
    hibernate:
      ddl-auto: create-drop
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

Accordingly, the parameters must be set as folows:

```
spring:
  datasource:
    url: jdbc:mysql://xxx.tidbcloud.com:4000/test?sslMode=VERIFY_IDENTITY&
       ↪ enabledTLSProtocols=TLSv1.2,TLSv1.3
    username: 2aEp24QWEDLqRFs.root
    password: 123456
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    show-sql: true
    database-platform: org.hibernate.dialect.TiDBDialect
    hibernate:
      ddl-auto: create-drop
```

4.3.2.5.2 Step 5.2 Run

Open a terminal session and make sure you are in the `spring-jpa-hibernate` directory. If you are not already in this directory, navigate to the directory with the following command:

```
cd <path>/tidb-example-java/spring-jpa-hibernate
```

Build and run with Make (recommended)

```
make
```

Build and run manually

If you prefer to build manually, follow these steps:

1. Clear cache and package:

```
mvn clean package
```

2. Run applications with JAR files:

```
java -jar target/spring-jpa-hibernate-0.0.1.jar
```

4.3.2.5.3 Step 5.3 Output

The final part of the output should look like the following:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.0.1)

2023-01-05T14:06:54.427+08:00 INFO 22005 --- [    main] com.pingcap.App
    ↪                    : Starting App using Java 17.0.2 with PID 22005 (/
    ↪ Users/cheese/IdeaProjects/tidb-example-java/spring-jpa-hibernate/
    ↪ target/classes started by cheese in /Users/cheese/IdeaProjects/tidb-
    ↪ example-java)
2023-01-05T14:06:54.428+08:00 INFO 22005 --- [    main] com.pingcap.App
    ↪                    : No active profile set, falling back to 1 default
    ↪  profile: "default"
2023-01-05T14:06:54.642+08:00 INFO 22005 --- [    main] .s.d.r.c.
    ↪ RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA
    ↪ repositories in DEFAULT mode.
2023-01-05T14:06:54.662+08:00 INFO 22005 --- [    main] .s.d.r.c.
    ↪ RepositoryConfigurationDelegate : Finished Spring Data repository
    ↪ scanning in 17 ms. Found 1 JPA repository interfaces.
2023-01-05T14:06:54.830+08:00 INFO 22005 --- [    main] o.s.b.w.embedded.
    ↪ tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-01-05T14:06:54.833+08:00 INFO 22005 --- [    main] o.apache.catalina.
    ↪ core.StandardService : Starting service [Tomcat]
```

```
2023-01-05T14:06:54.833+08:00 INFO 22005 --- [    main] o.apache.catalina.
    ↪ core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.4]
2023-01-05T14:06:54.865+08:00 INFO 22005 --- [    main] o.a.c.c.C.[Tomcat
    ↪ ].[localhost].[/] : Initializing Spring embedded
    ↪ WebApplicationContext
2023-01-05T14:06:54.865+08:00 INFO 22005 --- [    main] w.s.c.
    ↪ ServletWebServerApplicationContext : Root WebApplicationContext:
    ↪ initialization completed in 421 ms
2023-01-05T14:06:54.916+08:00 INFO 22005 --- [    main] o.hibernate.jpa.
    ↪ internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [
    ↪ name: default]
2023-01-05T14:06:54.929+08:00 INFO 22005 --- [    main] org.hibernate.
    ↪ Version              : HHH000412: Hibernate ORM core version 6.1.6.
    ↪ Final
2023-01-05T14:06:54.969+08:00 WARN 22005 --- [    main] org.hibernate.orm.
    ↪ deprecation      : HHH90000021: Encountered deprecated setting [javax
    ↪ .persistence.sharedCache.mode], use [jakarta.persistence.sharedCache.
    ↪ mode] instead
2023-01-05T14:06:55.005+08:00 INFO 22005 --- [    main] com.zaxxer.hikari.
    ↪ HikariDataSource : HikariPool-1 - Starting...
2023-01-05T14:06:55.074+08:00 INFO 22005 --- [    main] com.zaxxer.hikari.
    ↪ pool.HikariPool  : HikariPool-1 - Added connection com.mysql.cj.jdbc.
    ↪ ConnectionImpl@5e905f2c
2023-01-05T14:06:55.075+08:00 INFO 22005 --- [    main] com.zaxxer.hikari.
    ↪ HikariDataSource : HikariPool-1 - Start completed.
2023-01-05T14:06:55.089+08:00 INFO 22005 --- [    main] SQL dialect
    ↪                          : HHH000400: Using dialect: org.hibernate.
    ↪ dialect.TiDBDialect
Hibernate: drop table if exists player_jpa
Hibernate: drop sequence player_jpa_id_seq
Hibernate: create sequence player_jpa_id_seq start with 1 increment by 1
Hibernate: create table player_jpa (id bigint not null, coins integer, goods
    ↪  integer, primary key (id)) engine=InnoDB
2023-01-05T14:06:55.332+08:00 INFO 22005 --- [    main] o.h.e.t.j.p.i.
    ↪ JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [
    ↪ org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-01-05T14:06:55.335+08:00 INFO 22005 --- [    main] j.
    ↪ LocalContainerEntityManagerFactoryBean : Initialized JPA
    ↪ EntityManagerFactory for persistence unit 'default'
2023-01-05T14:06:55.579+08:00 WARN 22005 --- [    main]
    ↪ JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is
    ↪  enabled by default. Therefore, database queries may be performed
    ↪ during view rendering. Explicitly configure spring.jpa.open-in-view
    ↪ to disable this warning
2023-01-05T14:06:55.710+08:00 INFO 22005 --- [    main] o.s.b.w.embedded.
```

```
    ↪ tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with
    ↪ context path ''
2023-01-05T14:06:55.714+08:00 INFO 22005 --- [    main] com.pingcap.App
    ↪                     : Started App in 1.432 seconds (process running
    ↪ for 1.654)
```

The output log indicates the application behavior during startup. In this example, the application starts a **Servlet** using Tomcat, uses Hibernate as the ORM, uses HikariCP as the database connection pool implementation, and uses `org.hibernate.dialect.TiDBDialect` as the database dialect. After startup, Hibernate deletes and re-creates the `player_jpa` table and the `player_jpa_id_seq` sequence. At the end of startup, the application listens on port `8080` to provide HTTP services to the outside.

If you want to learn more about the code of this application, refer to implementation details.

4.3.2.6 Step 6: HTTP requests

After the service is up and running, you can send the HTTP requests to the backend application. http://localhost:8080 is the base URL that provides services. This tutorial uses a series of HTTP requests to show how to use the service.

4.3.2.6.1 Step 6.1 Use Postman requests (recommended)

You can download this configuration file locally and import it into Postman as shown here:

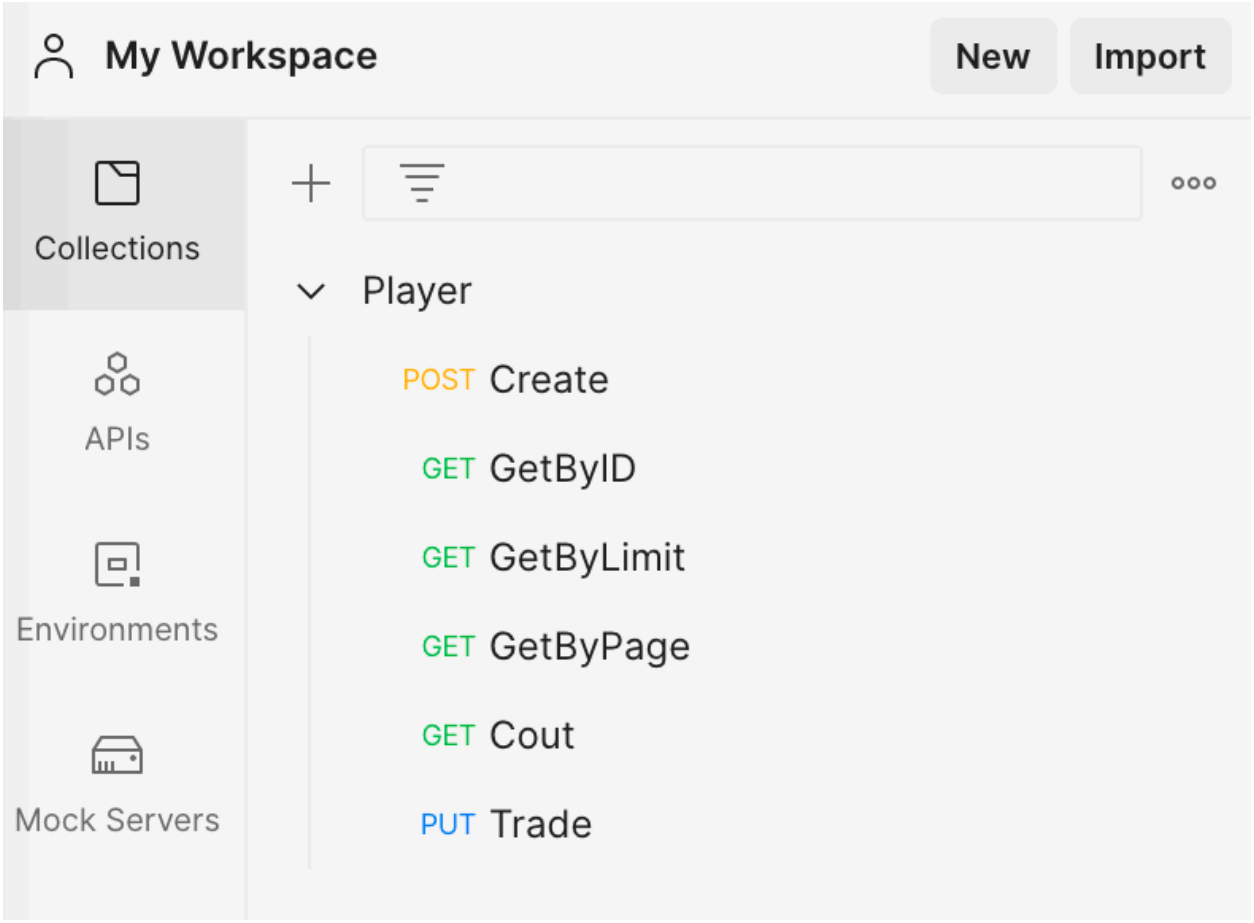


Figure 6: import the collection into Postman

Create players

Click on the **Create** tab and the **Send** button to send a POST request to `http://localhost:8080/player/`. The return value is the number of players added, which is expected to be 1.

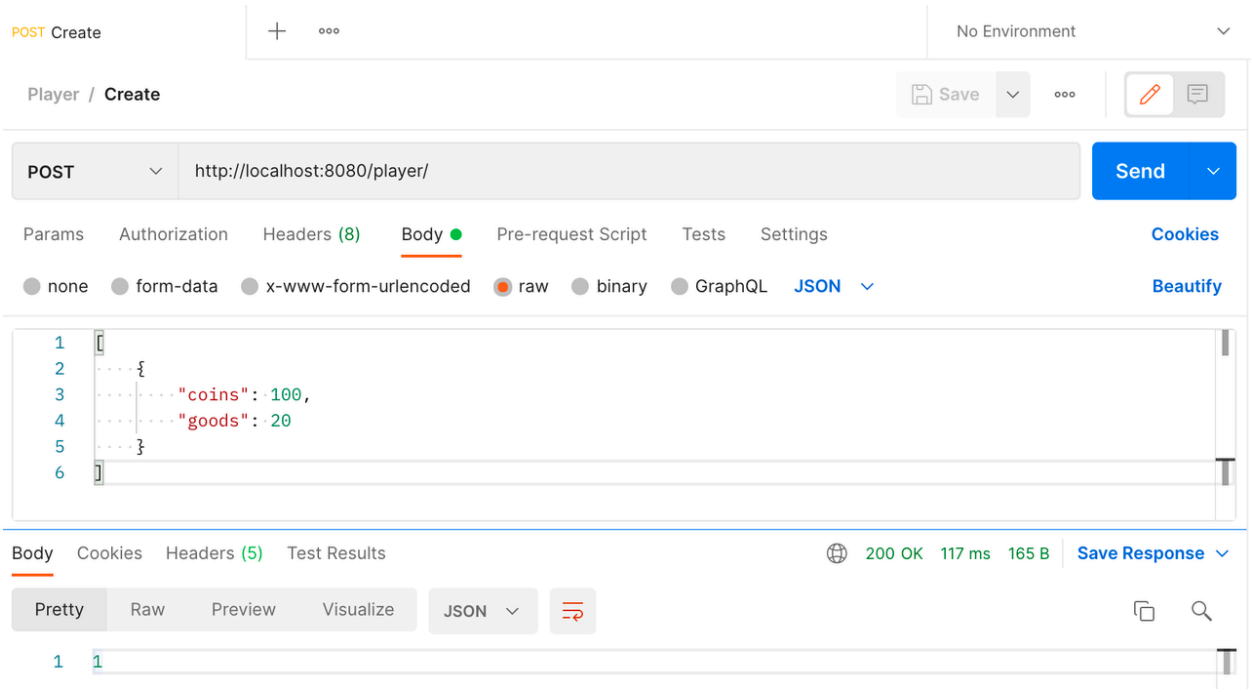


Figure 7: Postman-Create a player

Get player information by ID

Click on the **GetByID** tab and the **Send** button to send a GET request to `http://localhost:8080/player/1`. The return value is the information of the player with ID 1.

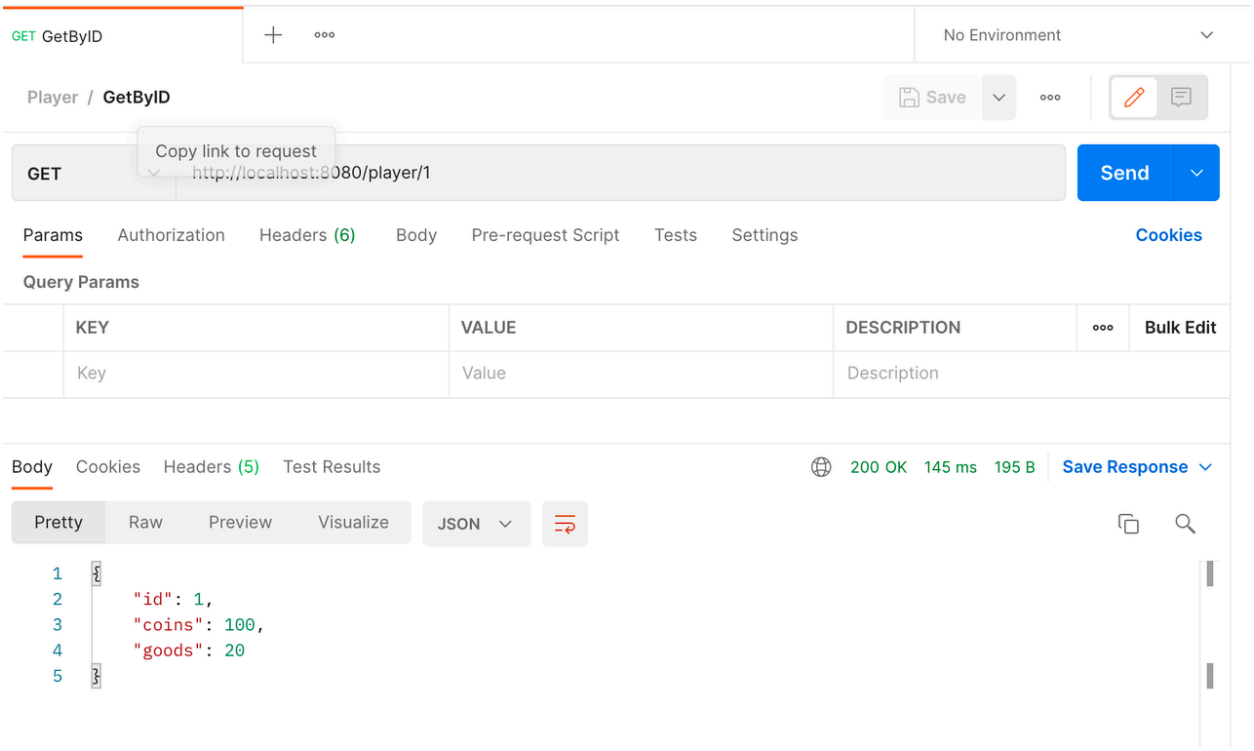


Figure 8: Postman-GetByID

Get player information in bulk by limit

Click on the **GetByLimit** tab and the **Send** button to send a GET request to `http://localhost:8080/player/limit/3`. The return value is a list of information for up to 3 players.

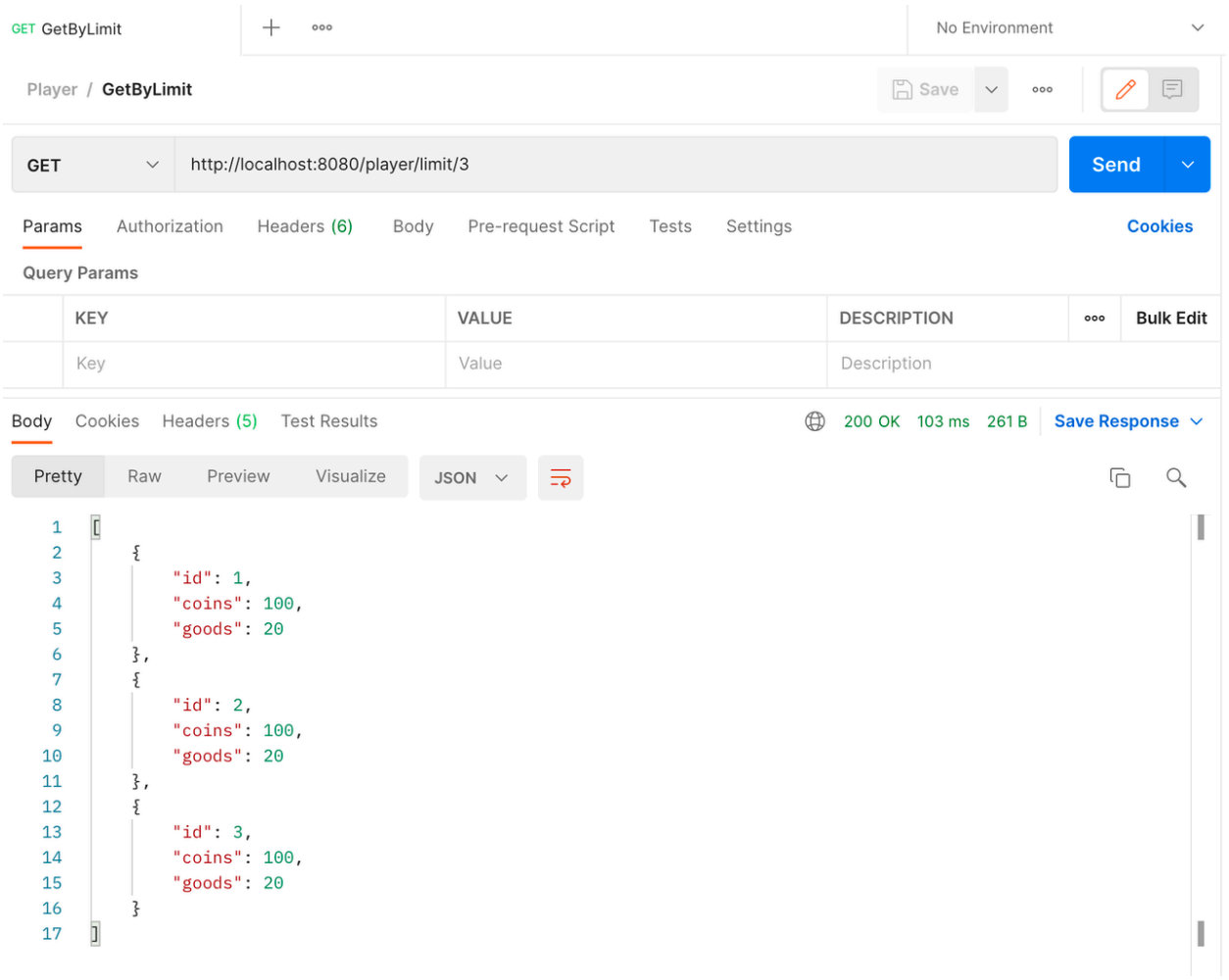


Figure 9: Postman-GetByLimit

Get player information by page

Click on the **GetByPage** tab and the **Send** button to send a GET request to `http ↪ ://localhost:8080/player/page?index=0&size=2`. The return value is the page with index `0`, with `2` players per page. The return value also contains the paging information such as offset, totalPages, and sort.

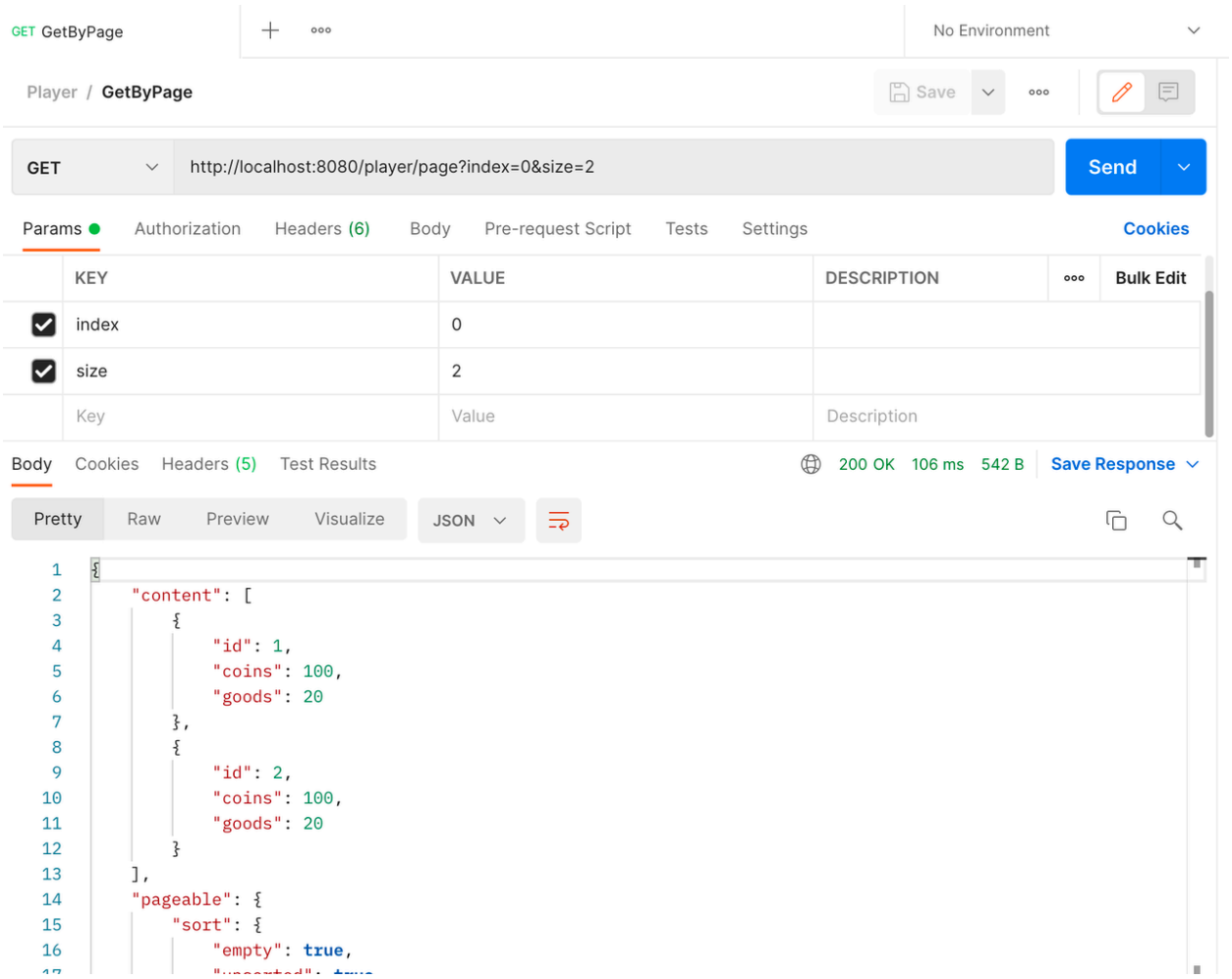


Figure 10: Postman-GetByPage

Count players

Click the **Count** tab and the **Send** button to send a GET request to `http://localhost:8080/player/count`. The return value is the number of players.

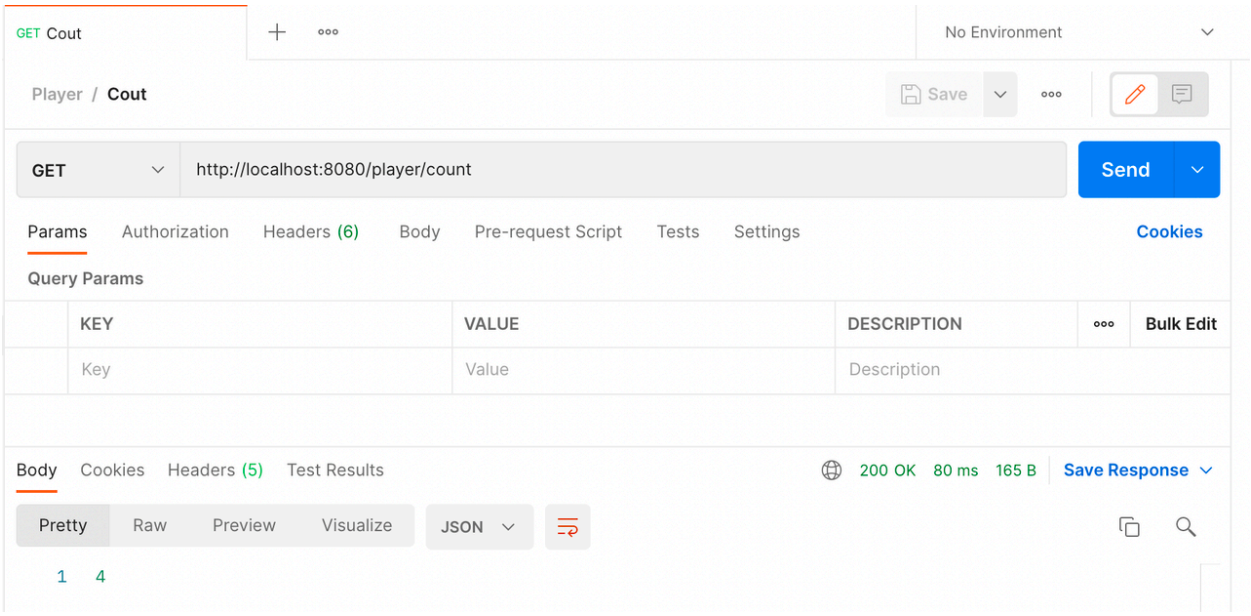


Figure 11: Postman-Count

Player trading

Click on the **Trade** tab and the **Send** button to send a PUT request to `http://localhost:8080/player/trade`. The request parameters are the seller's ID `sellID`, the buyer's ID `buyID`, the number of goods purchased `amount`, the number of coins consumed for the purchase `price`.

The return value is whether the transaction is successful or not. When there are insufficient goods for the seller, insufficient coins for the buyer, or a database error, the database transaction guarantees that the trade is not successful and no player's coins or goods are lost.

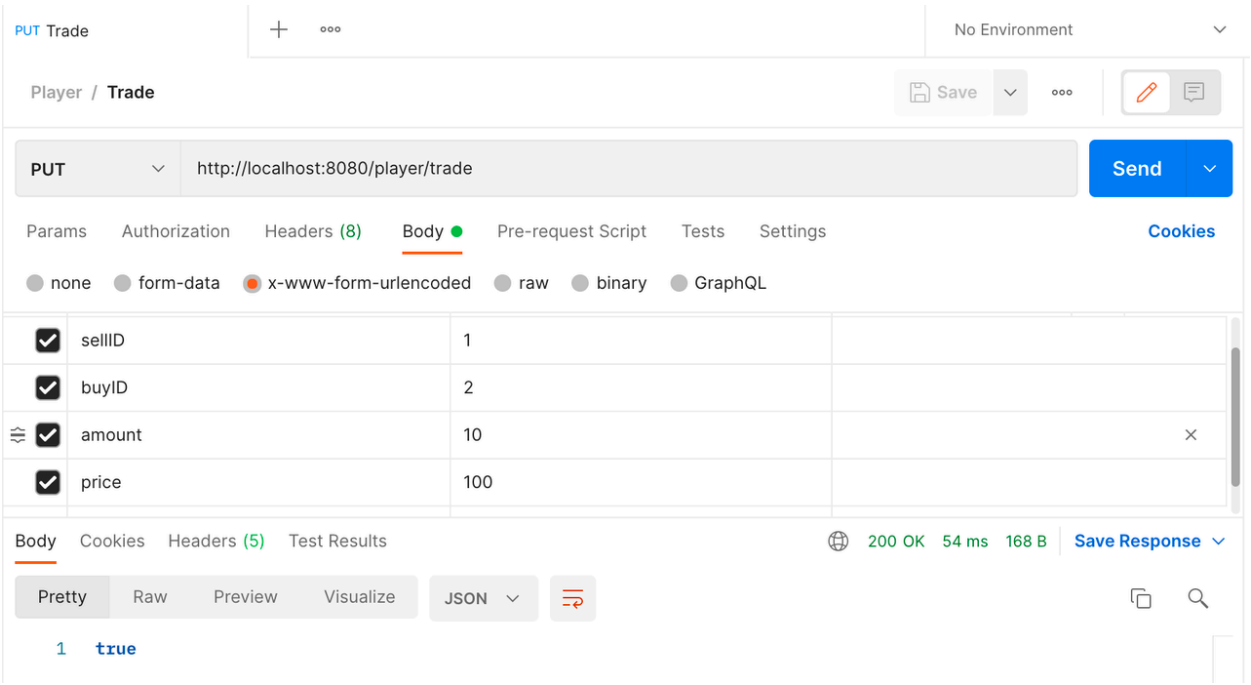


Figure 12: Postman-Trade

4.3.2.6.2 Step 6.2 Using curl requests

You can also use curl to make requests directly.

Create players

To create players, you can send a **POST** request to the `/player` endpoint. For example:

```
curl --location --request POST 'http://localhost:8080/player/' --header '
    ↪ Content-Type: application/json' --data-raw '[{"coins":100,"goods
    ↪ ":20}]'
```

The request uses JSON as the payload. The example above indicates creating a player with 100 `coins` and 20 `goods`. The return value is the number of players created.

```
1
```

Get player information by ID

To get the player information, you can send a **GET** request to the `/player` endpoint. You need to specify the `id` of the player in the path parameter as follows: `/player/{id}`. The following example shows how to get the information of a player with `id` 1:

```
curl --location --request GET 'http://localhost:8080/player/1'
```

The return value is the player's information:

```
{
  "coins": 200,
  "goods": 10,
  "id": 1
}
```

Get player information in bulk by limit

To get the player information in bulk, you can send a **GET** request to the `/player/limit` endpoint. You need to specify the total number of players in the path parameter as follows: `/player/limit/{limit}`. The following example shows how to get the information of up to 3 players:

```
curl --location --request GET 'http://localhost:8080/player/limit/3'
```

The return value is a list of player information:

```
[
  {
    "coins": 200,
    "goods": 10,
    "id": 1
  },
  {
    "coins": 0,
    "goods": 30,
    "id": 2
  },
  {
    "coins": 100,
    "goods": 20,
    "id": 3
  }
]
```

Get player information by page

To get paginated player information, you can send a **GET** request to the `/player/page` endpoint. To specify additional parameters, you need to use the URL parameter. The following example shows how to get the information from a page whose `index` is 0, where each page has a maximum `size` of 2 players.

```
curl --location --request GET 'http://localhost:8080/player/page?index=0&
    ↪ size=2'
```

The return value is the page with `index` 0, where 2 players are listed per page. In addition, the return value contains pagination information such as offset, total pages, and whether the results are sorted.

```
{
  "content": [
    {
      "coins": 200,
      "goods": 10,
      "id": 1
    },
    {
      "coins": 0,
      "goods": 30,
      "id": 2
    }
  ],
  "empty": false,
  "first": true,
  "last": false,
  "number": 0,
  "numberOfElements": 2,
  "pageable": {
    "offset": 0,
    "pageNumber": 0,
    "pageSize": 2,
    "paged": true,
    "sort": {
      "empty": true,
      "sorted": false,
      "unsorted": true
    },
    "unpaged": false
  },
  "size": 2,
  "sort": {
    "empty": true,
    "sorted": false,
    "unsorted": true
  },
  "totalElements": 4,
  "totalPages": 2
}
```


Count players

To get the number of players, you can send a **GET** request to the `/player/count` endpoint:

```
curl --location --request GET 'http://localhost:8080/player/count'
```

The return value is the number of players:

```
4
```

Player trading

To initiate a transaction between players, you can send a **PUT** request to the `/player/trade` endpoint. For example:

```
curl --location --request PUT 'http://localhost:8080/player/trade' \
  --header 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode 'sellID=1' \
  --data-urlencode 'buyID=2' \
  --data-urlencode 'amount=10' \
  --data-urlencode 'price=100'
```

The request uses **Form Data** as the payload. The example request indicates that the seller's ID (`sellID`) is 1, the buyer's ID (`buyID`) is 2, the number of goods purchased (`amount`) is 10, and the number of coins consumed for purchase (`price`) is 100.

The return value is whether the transaction is successful or not. When there are insufficient goods for the seller, insufficient coins for the buyer, or a database error, the database transaction guarantees that the trade is not successful and no player's coins or goods are lost.

```
true
```

4.3.2.6.3 Step 6.3 Requests with Shell script

You can download this shell script for testing purposes. The script performs the following operations:

1. Create 10 players in a loop.
2. Get the information of players with the `id` of 1.
3. Get a list of up to 3 players.
4. Get a page of players with the `index` of 0 and the `size` of 2.
5. Get the total number of players.
6. Perform a transaction, where the player with the `id` of 1 is the seller and the player with the `id` of 2 is the buyer, and 10 `goods` are purchased at the cost of 100 `coins`.

You can run this script with `make request` or `./request.sh`. The result should look like this:

```
cheese@CheesedeMacBook-Pro spring-jpa-hibernate % make request
./request.sh
loop to create 10 players:
1111111111

get player 1:
{"id":1,"coins":200,"goods":10}

get players by limit 3:
[{"id":1,"coins":200,"goods":10},{"id":2,"coins":0,"goods":30},{"id":3,"
    ↪ coins":100,"goods":20}]

get first players:
{"content":[{"id":1,"coins":200,"goods":10},{"id":2,"coins":0,"goods":30}],"
    ↪ pageable":{"sort":{"empty":true,"unsorted":true,"sorted":false},"
    ↪ offset":0,"pageNumber":0,"pageSize":2,"paged":true,"unpaged":false},"
    ↪ last":false,"totalPages":7,"totalElements":14,"first":true,"size":2,"
    ↪ number":0,"sort":{"empty":true,"unsorted":true,"sorted":false},"
    ↪ numberOfElements":2,"empty":false}

get players count:
14

trade by two players:
false
```

4.3.2.7 Implementation details

This subsection describes the components in the sample application project.

4.3.2.7.1 Overview

The catalog tree for this example project is shown below (some incomprehensible parts are removed):

```
.
├── pom.xml
├── src
    ├── main
        ├── java
            ├── com
                ├── pingcap
                    ├── App.java
                    ├── controller
```

```
                    ├── PlayerController.java
                ├── dao
                    ├── PlayerBean.java
                    ├── PlayerRepository.java
                ├── service
                    ├── PlayerService.java
                    ├── impl
                        ├── PlayerServiceImpl.java
        ├── resources
            ├── application.yml
```

- `pom.xml` declares the project's Maven configuration, such as dependencies and packaging.
- `application.yml` declares the project's user configuration, such as database address, password, and database dialect used.
- `App.java` is the entry point of the project.
- `controller` is the package that exposes the HTTP interface to the outside.
- `service` is the package that implements the interface and logic of the project.
- `dao` is the package that implements the connection to the database and the persistence of the data.

4.3.2.7.2 Configuration

This part briefly describes the Maven configuration in the `pom.xml` file and the user configuration in the `application.yml` file.

Maven configuration

The `pom.xml` file is a Maven configuration file that declares the project's Maven dependencies, packaging methods, and packaging information. You can replicate the process of generating this configuration file by creating a blank application with the same dependency, or copying it directly to your project.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
    ↪ org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.
        ↪ apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.1</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
```

```
<groupId>com.pingcap</groupId>
<artifactId>spring-jpa-hibernate</artifactId>
<version>0.0.1</version>
<name>spring-jpa-hibernate</name>
<description>an example for spring boot, jpa, hibernate and TiDB</
    ↪ description>

<properties>
    <java.version>17</java.version>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```
</project>
```

User configuration

The `application.yml` configuration file declares the user configuration, such as database address, password, and the database dialect used.

```
spring:
  datasource:
    url: jdbc:mysql://localhost:4000/test
    username: root
    #    password: xxx
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    show-sql: true
    database-platform: org.hibernate.dialect.TiDBDialect
    hibernate:
      ddl-auto: create-drop
```

The configuration is written in YAML. The fields are described as follows:

- `spring.datasource.url` : URL of the database connection.
- `spring.datasource.username` : the database username.
- `spring.datasource.password` : the database password. Empty. You need to comment out or delete this field.
- `spring.datasource.driver-class-name` : the database driver. Because TiDB is compatible with MySQL, use a mysql-connector-java driver class `com.mysql.cj.jdbc`.
- `jpa.show-sql` : when this field is set to `true`, the SQL statements run by JPA are printed.
- `jpa.database-platform` : the selected database dialect. Because the application connects to TiDB, choose **TiDB dialect**. Note that this dialect is only available in Hibernate `6.0.0.Beta2` and later versions, so choose the applicable dependency version.
- `jpa.hibernate.ddl-auto` : `create-drop` creates the table at the beginning of the program and deletes the table on exit. Do not set this option in a production environment. Because this is a sample application, this option is set to minimize the impact on the database data.

4.3.2.7.3 Entry point

The `App.java` file is the entry point:

```
package com.pingcap;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.boot.context.ApplicationPidFileWriter;

@SpringBootApplication
public class App {
   public static void main(String[] args) {
      SpringApplication springApplication = new SpringApplication(App.class)
         ↪ ;
      springApplication.addListeners(new ApplicationPidFileWriter("spring-
         ↪ jpa-hibernate.pid"));
      springApplication.run(args);
   }
}
```

The entry class starts with the standard configuration annotation `@SpringBootApplication` ↪ for Spring Boot applications. For more information, see Using the `@SpringBootApplication` ↪ Annotation in the Spring Boot official documentation. Then, the program uses the `ApplicationPidFileWriter` to write a PID (process identification number) file called `spring-jpa-hibernate.pid` during application startup. The PID file can be used to close this application from an external source.

4.3.2.7.4 Data access object

The `dao` (Data Access Object) package implements the persistence of data objects.

Entity objects

The `PlayerBean.java` file is an entity object, which corresponds to a table in the database:

```
package com.pingcap.dao;

import jakarta.persistence.*;

/**
 * it's core entity in hibernate
 * @Table appoint to table name
 */
@Entity
@Table(name = "player_jpa")
public class PlayerBean {
    /**
     * @ID primary key
     * @GeneratedValue generated way. this field will use generator named "
        ↪ player_id"
     * @SequenceGenerator using `sequence` feature to create a generator,
     *   and it named "player_jpa_id_seq" in database, initial form 1 (by
        ↪ `initialValue`
```

```
     *   parameter default), and every operator will increase 1 (by `
        ↪ allocationSize`)
     */
    @Id
    @GeneratedValue(generator="player_id")
    @SequenceGenerator(name="player_id", sequenceName="player_jpa_id_seq",
        ↪ allocationSize=1)
    private Long id;

    /**
     * @Column field
     */
    @Column(name = "coins")
    private Integer coins;
    @Column(name = "goods")
    private Integer goods;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Integer getCoins() {
        return coins;
    }

    public void setCoins(Integer coins) {
        this.coins = coins;
    }

    public Integer getGoods() {
        return goods;
    }

    public void setGoods(Integer goods) {
        this.goods = goods;
    }
}
```

The entity class has several annotations that give Hibernate additional information to bind the entity class to the table.

- `@Entity` declares that `PlayerBean` is an entity class.
- `@Table` relates this entity class to the `player_jpa` table using the annotation attribute `name`.
- `@Id` declares that this property is related to the primary key column of the table.
- `@GeneratedValue` indicates that the value of this column is generated automatically and should not be set manually. The attribute `generator` is used to specify the name of the generator as `player_id`.
- `@SequenceGenerator` declares a generator that uses sequence, and uses the annotation attribute `name` to declare the name of the generator as `player_id` (consistent with the name specified in `@GeneratedValue`). The annotation attribute `sequenceName` is used to specify the name of the sequence in the database. Finally, the annotation attribute `allocationSize` is used to declare the sequence's step size to be 1.
- `@Column` declares each private attribute as a column of the `player_jpa` table, and uses the annotation attribute `name` to determine the name of the column corresponding to the attribute.

Repository

To abstract the database layer, Spring applications use the `Repository` interface, or a sub-interface of the `Repository`. This interface maps to a database object, such as a table. JPA implements some pre-built methods, such as `INSERT`, or `SELECT` using the primay key.

```
package com.pingcap.dao;

import jakarta.persistence.LockModeType;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Lock;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface PlayerRepository extends JpaRepository<PlayerBean, Long> {
    /**
     * use HQL to query by page
     * @param pageable a pageable parameter required by hibernate
     * @return player list package by page message
     */
    @Query(value = "SELECT player_jpa FROM PlayerBean player_jpa")
    Page<PlayerBean> getPlayersByPage(Pageable pageable);

    /**
```

```
     * use SQL to query by limit, using named parameter
     * @param limit sql parameter
     * @return player list (max size by limit)
     */
    @Query(value = "SELECT * FROM player_jpa LIMIT :limit", nativeQuery =
        ↪ true)
    List<PlayerBean> getPlayersByLimit(@Param("limit") Integer limit);

    /**
     * query player and add a lock for update
     * @param id player id
     * @return player
     */
    @Lock(value = LockModeType.PESSIMISTIC_WRITE)
    @Query(value = "SELECT player FROM PlayerBean player WHERE player.id = :
        ↪ id")
    // @Query(value = "SELECT * FROM player_jpa WHERE id = :id FOR UPDATE",
        ↪  nativeQuery = true)
    PlayerBean getPlayerAndLock(@Param("id") Long id);
}
```

The `PlayerRepository` interface extends the `JpaRepository` interface used by Spring for JPA data access. The `@Query` annotation is used to tell Hibernate how to implement queries in this interface. Two query syntaxes are used:

- In the `getPlayersByPage` interface, Hibernate Query Language (HQL) is used.
- In the `getPlayersByLimit` interface, native SQL is used. When the interface uses the native SQL syntax, the `@Query` annotation parameter `nativeQuery` must be set to `true`.

In the SQL for the `getPlayersByLimit` annotation, `:limit` is called a named parameter in Hibernate. Hibernate automatically finds and splices the parameter by name within the interface where the annotation resides. You can also use `@Param` to specify a name different from the parameter for injection.

In `getPlayerAndLock`, an annotation `@Lock` is used to declare that pessimistic locking is applied. For details on other locking methods, see Entity Locking. The `@Lock` annotation must be used with `HQL`; otherwise, an error occurs. If you want to use SQL directly for locking, you can use the annotation from the comment:

```
@Query(value = "SELECT * FROM player_jpa WHERE id = :id FOR UPDATE",
    ↪ nativeQuery = true)
```

The SQL statement above uses `FOR UPDATE` to add locks directly. You can also dive deeper into the principles with the TiDB `SELECT` statement.

4.3.2.7.5 Logic implementation

The logic implementation layer is the `service` package, which contains the interfaces and logic implemented by the project.

Interface

The `PlayerService.java` file defines the logical interface and implements the interface rather than writing a class directly. This is to keep the example as close to actual use as possible and to reflect the open-closed principle of the design. You may omit this interface and inject the implementation class directly in the dependency class, but this approach is not recommended.

```
package com.pingcap.service;

import com.pingcap.dao.PlayerBean;
import org.springframework.data.domain.Page;

import java.util.List;

public interface PlayerService {
    /**
     * create players by passing in a List of PlayerBean
     *
     * @param players will create players list
     * @return The number of create accounts
     */
    Integer createPlayers(List<PlayerBean> players);

    /**
     * buy goods and transfer funds between one player and another in one
        ↪ transaction
     * @param sellId sell player id
     * @param buyId buy player id
     * @param amount goods amount, if sell player has not enough goods, the
        ↪  trade will break
     * @param price price should pay, if buy player has not enough coins,
        ↪ the trade will break
     */
    void buyGoods(Long sellId, Long buyId, Integer amount, Integer price)
       ↪ throws RuntimeException;

    /**
     * get the player info by id.
     *
     * @param id player id
     * @return the player of this id
```

```
     */
    PlayerBean getPlayerByID(Long id);

    /**
     * get a subset of players from the data store by limit.
     *
     * @param limit return max size
     * @return player list
     */
    List<PlayerBean> getPlayers(Integer limit);

    /**
     * get a page of players from the data store.
     *
     * @param index page index
     * @param size page size
     * @return player list
     */
    Page<PlayerBean> getPlayersByPage(Integer index, Integer size);

    /**
     * count players from the data store.
     *
     * @return all players count
     */
    Long countPlayers();
}
```

Implementation (Important)

The `PlayerService.java` file implements the `PlayerService` interface, which contains all the data processing logic.

```
package com.pingcap.service.impl;

import com.pingcap.dao.PlayerBean;
import com.pingcap.dao.PlayerRepository;
import com.pingcap.service.PlayerService;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;

import java.util.List;
```

```
/**
 * PlayerServiceImpl implements PlayerService interface
 * @Transactional it means every method in this class, will package by a
    ↪ pair of
 *    transaction.begin() and transaction.commit(). and it will be call
 *    transaction.rollback() when method throw an exception
 */
@Service
@Transactional
public class PlayerServiceImpl implements PlayerService {
    @Autowired
    private PlayerRepository playerRepository;

    @Override
    public Integer createPlayers(List<PlayerBean> players) {
        return playerRepository.saveAll(players).size();
    }

    @Override
    public void buyGoods(Long sellId, Long buyId, Integer amount, Integer
        ↪ price) throws RuntimeException {
        PlayerBean buyPlayer = playerRepository.getPlayerAndLock(buyId);
        PlayerBean sellPlayer = playerRepository.getPlayerAndLock(sellId);
        if (buyPlayer == null || sellPlayer == null) {
            throw new RuntimeException("sell or buy player not exist");
        }

        if (buyPlayer.getCoins() < price || sellPlayer.getGoods() < amount)
            ↪ {
            throw new RuntimeException("coins or goods not enough, rollback")
                ↪ ;
        }

        buyPlayer.setGoods(buyPlayer.getGoods() + amount);
        buyPlayer.setCoins(buyPlayer.getCoins() - price);
        playerRepository.save(buyPlayer);

        sellPlayer.setGoods(sellPlayer.getGoods() - amount);
        sellPlayer.setCoins(sellPlayer.getCoins() + price);
        playerRepository.save(sellPlayer);
    }

    @Override
    public PlayerBean getPlayerByID(Long id) {
        return playerRepository.findById(id).orElse(null);
```

```
    }

    @Override
    public List<PlayerBean> getPlayers(Integer limit) {
        return playerRepository.getPlayersByLimit(limit);
    }

    @Override
    public Page<PlayerBean> getPlayersByPage(Integer index, Integer size) {
        return playerRepository.getPlayersByPage(PageRequest.of(index, size)
            ↪ );
    }

    @Override
    public Long countPlayers() {
        return playerRepository.count();
    }
}
```

The `@Service` annotation is used to declare that the lifecycle of this object is managed by `Spring`.

The `PlayerServiceImpl` implementation class also has a `@Transactional` annotation in addition to the `@Service` annotation. When transaction management is enabled in the application (which can be turned on using `@EnableTransactionManagement`, but is turned on by default by `Spring Boot`. You don not need to manually configure it.), `Spring` automatically wraps all objects with the `@Transactional` annotation in a proxy and uses this proxy for object invocation processing.

You can simply assume that when the agent calls a function inside an object with the `@Transactional` annotation:

- At the top of the function, it starts the transaction with `transaction.begin()`.
- When the function returns, it calls `transaction.commit()` to commit the transaction.
- When any runtime error occurs, the agent calls `transaction.rollback()` to roll back.

You can refer to Database Transactions for more information on transactions, or read Understanding the Spring Framework's Declarative Transaction Implementation on the `Spring` website.

In all implementation classes, the `buyGoods` function is requires attention. When the function encounters an illogical operation, it throws an exception and directs Hibernate to perform a transaction rollback to prevent incorrect data.

4.3.2.7.6 External HTTP Interface

The `controller` package exposes the HTTP interface to the outside world and allows access to the service via the REST API.

```
package com.pingcap.controller;

import com.pingcap.dao.PlayerBean;
import com.pingcap.service.PlayerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.lang.NonNull;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/player")
public class PlayerController {
    @Autowired
    private PlayerService playerService;

    @PostMapping
    public Integer createPlayer(@RequestBody @NonNull List<PlayerBean>
        ↪ playerList) {
        return playerService.createPlayers(playerList);
    }

    @GetMapping("/{id}")
    public PlayerBean getPlayerByID(@PathVariable Long id) {
        return playerService.getPlayerByID(id);
    }

    @GetMapping("/limit/{limit_size}")
    public List<PlayerBean> getPlayerByLimit(@PathVariable("limit_size")
        ↪ Integer limit) {
        return playerService.getPlayers(limit);
    }

    @GetMapping("/page")
    public Page<PlayerBean> getPlayerByPage(@RequestParam Integer index,
        ↪ @RequestParam("size") Integer size) {
        return playerService.getPlayersByPage(index, size);
    }

    @GetMapping("/count")
    public Long getPlayersCount() {
```

```
        return playerService.countPlayers();
    }

    @PutMapping("/trade")
    public Boolean trade(@RequestParam Long sellID, @RequestParam Long buyID
        ↪ , @RequestParam Integer amount, @RequestParam Integer price) {
        try {
            playerService.buyGoods(sellID, buyID, amount, price);
        } catch (RuntimeException e) {
            return false;
        }

        return true;
    }
}
```

`PlayerController` uses annotations as many as possible to demonstrate features. In real projects, keep the style consistent while following the rules of your company or team. The annotations in `PlayerController` are explained as follows:

- `@RestController` declares `PlayerController` as a Web Controller and serializes the return value as `JSON` output.
- `@RequestMapping` maps the URL endpoint to `/player`, that is, this `Web Controller` only listens for requests sent to the `/player` URL.
- `@Autowired` means `Spring` container can autowire relationships between collaborating beans. The declaration requires a `PlayerService` object, which is an interface and does not specify which implementation class to use. This is assembled by Spring. For the rules of this assembly, see The IoC container on Spring's official website.
- `@PostMapping` declares that this function responds to a POST request in HTTP.
    - `@RequestBody` declares that the entire HTTP payload is parsed into the `playerList` parameter.
    - `@NonNull` declares that the parameter must not be null; otherwise, it returns an error.
- `@GetMapping` declares that this function responds to a GET request in HTTP.
    - `@PathVariable` shows that the annotation has placeholders like `{id}` and `{ ↪ limit_size}`, which are bound to the variable annotated by `@PathVariable`. Such binding is based on the annotation attribute `name`. If the annotation attribute `name` is not specified, it is the same as the variable name. The variable name can be omitted, that is, `@PathVariable(name="limit_size")` can be written as `@PathVariable("limit_size")`.
- `@PutMapping` declares that this function responds to a PUT request in HTTP.
- `@RequestParam` declares that this function parses URL parameters, form parameters, and other parameters in the request and binds them to the annotated variables.

4.3.2.8 Create a blank application with the same dependency (optional)

This application is built using Spring Initializr. You can quickly get a blank application with the same dependencies as this sample application by clicking on the following options and changing a few configuration items:

Project

- Maven Project

Language

- Java

Spring Boot

- Latest stable version

Project Metadata

- Group: com.pingcap
- Artifact: spring-jpa-hibernate
- Name: spring-jpa-hibernate
- Package name: com.pingcap
- Packaging: Jar
- Java: 17

Dependencies

- Spring Web
- Spring Data JPA
- MySQL Driver

The complete configuration is as follows:

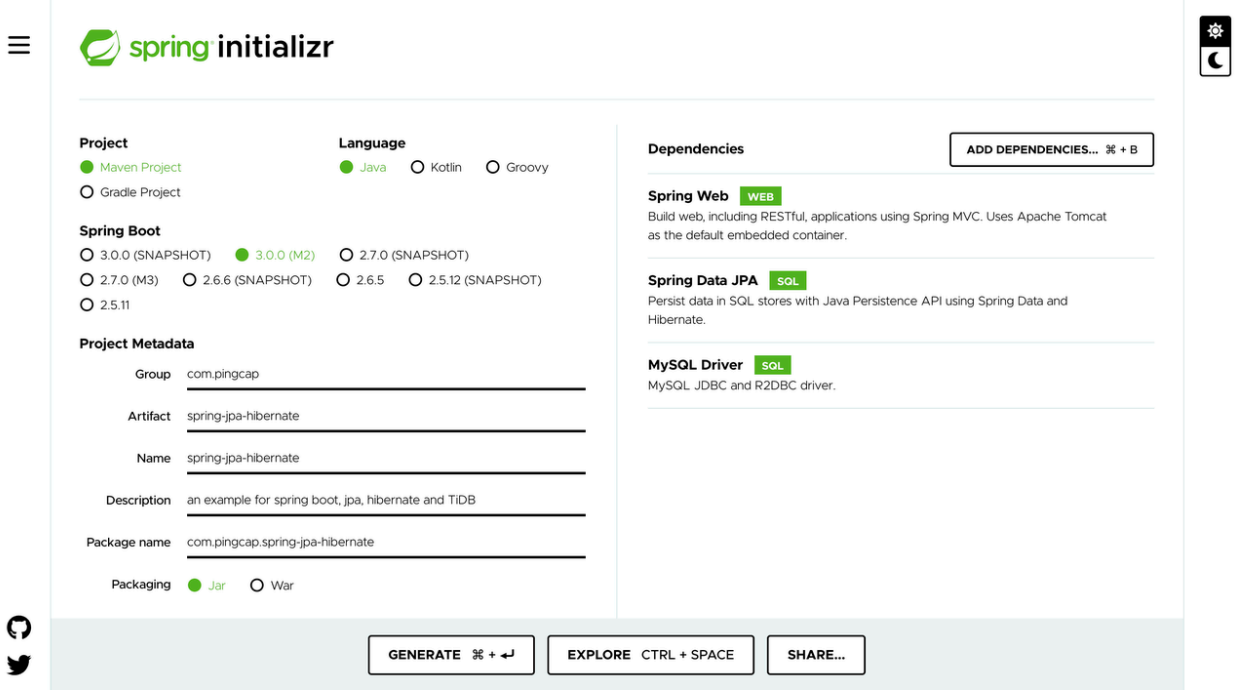


Figure 13: Spring Initializr Configuration

> **Note:**
>
> Although SQL is relatively standardized, each database vendor uses a subset and superset of ANSI SQL defined syntax. This is referred to as the database's dialect. Hibernate handles variations across these dialects through its `org.hibernate.dialect.Dialect` class and the various subclasses for each database vendor.
>
> In most cases, Hibernate will be able to determine the proper Dialect to use by asking some questions of the JDBC Connection during bootstrap. For information on Hibernate's ability to determine the proper Dialect to use (and your ability to influence that resolution), see Dialect resolution.
>
> If for some reason it is not able to determine the proper one or you want to use a custom Dialect, you will need to set the `hibernate.dialect` setting.
>
> —— *Excerpt from the Hibernate official documentation: Database Dialect*

After the configuration, you can get a blank **Spring Boot** application with the same dependencies as the sample application.

4.3.3 Build a Simple CRUD App with TiDB and Java

This document describes how to use TiDB and Java to build a simple CRUD application.

> **Note:**
>
> It is recommended to use Java 8 or a later Java version.
>
> If you want to use Spring Boot for application development, refer to Build the TiDB Application using Spring Boot

4.3.3.1 Step 1. Launch your TiDB cluster

The following introduces how to start a TiDB cluster.

Use a TiDB Serverless cluster

For detailed steps, see Create a TiDB Serverless cluster.

Use a local cluster

For detailed steps, see Deploy a local test cluster or Deploy a TiDB Cluster Using TiUP.

4.3.3.2 Step 2. Get the code

```
git clone https://github.com/pingcap-inc/tidb-example-java.git
```

Compared with Mybatis, the JDBC implementation might be not a best practice, because you need to write error handling logic manually and cannot reuse code easily, which makes your code slightly redundant.

Mybatis is a popular open-source Java class persistence framework. The following uses MyBatis Generator as a Maven plugin to generate the persistence layer code.

Change to the `plain-java-mybatis` directory:

```
cd plain-java-mybatis
```

The structure of this directory is as follows:

```
.
├── Makefile
├── pom.xml
└── src
    └── main
        └── java
            └── com
                └── pingcap
```

```
                ├── MybatisExample.java
                ├── dao
                │   ├── PlayerDAO.java
                ├── model
                │   ├── Player.java
                │   ├── PlayerMapper.java
                │   ├── PlayerMapperEx.java
        ├── resources
            ├── dbinit.sql
            ├── log4j.properties
            ├── mapper
            │   ├── PlayerMapper.xml
            │   ├── PlayerMapperEx.xml
            ├── mybatis-config.xml
            ├── mybatis-generator.xml
```

The automatically generated files are:

- `src/main/java/com/pingcap/model/Player.java`: The `Player` entity class.
- `src/main/java/com/pingcap/model/PlayerMapper.java`: The interface of `PlayerMapper`.
- `src/main/resources/mapper/PlayerMapper.xml`: The XML mapping of `Player`. Mybatis uses this configuration to automatically generate the implementation class of the `PlayerMapper` interface.

The strategy for generating these files is written in `mybatis-generator.xml`, which is the configuration file for Mybatis Generator. There are comments in the following configuration file to describe how to use it.

```
<!DOCTYPE generatorConfiguration PUBLIC
 "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
 "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">

<generatorConfiguration>
    <!--
        <context/> entire document: https://mybatis.org/generator/
            ↪ configreference/context.html

        context.id: A unique identifier you like
        context.targetRuntime: Used to specify the runtime target for
            ↪ generated code.
            It has MyBatis3DynamicSql / MyBatis3Kotlin / MyBatis3 /
                ↪ MyBatis3Simple 4 selection to choice.
    -->
    <context id="simple" targetRuntime="MyBatis3">
```

```
<!--
    <commentGenerator/> entire document: https://mybatis.org/
        ↪ generator/configreference/commentGenerator.html

    commentGenerator:
        - property(suppressDate): remove timestamp in comments
        - property(suppressAllComments): remove all comments
-->
<commentGenerator>
    <property name="suppressDate" value="true"/>
    <property name="suppressAllComments" value="true" />
</commentGenerator>

<!--
    <jdbcConnection/> entire document: https://mybatis.org/generator
        ↪ /configreference/jdbcConnection.html

    jdbcConnection.driverClass: The fully qualified class name for
        ↪ the JDBC driver used to access the database.
        Used mysql-connector-java:5.1.49, should specify JDBC is com
            ↪ .mysql.jdbc.Driver
    jdbcConnection.connectionURL: The JDBC connection URL used to
        ↪ access the database.
-->
<jdbcConnection driverClass="com.mysql.jdbc.Driver"
    connectionURL="jdbc:mysql://localhost:4000/test?user=root" />

<!--
    <javaModelGenerator/> entire document: https://mybatis.org/
        ↪ generator/configreference/javaModelGenerator.html
    Model code file will be generated at ${targetProject}/${
        ↪ targetPackage}

    javaModelGenerator:
        - property(constructorBased): If it's true, generator will
            ↪ create constructor function in model
-->
<javaModelGenerator targetPackage="com.pingcap.model" targetProject="
    ↪ src/main/java">
    <property name="constructorBased" value="true"/>
</javaModelGenerator>

<!--
    <sqlMapGenerator/> entire document: https://mybatis.org/generator
        ↪ /configreference/sqlMapGenerator.html
```

```
        XML SQL mapper file will be generated at ${targetProject}/${
            ↪ targetPackage}
    -->
    <sqlMapGenerator targetPackage="." targetProject="src/main/resources/
        ↪ mapper"/>

    <!--
        <javaClientGenerator/> entire document: https://mybatis.org/
            ↪ generator/configreference/javaClientGenerator.html
        Java code mapper interface file will be generated at ${
            ↪ targetProject}/${targetPackage}

        javaClientGenerator.type (context.targetRuntime is MyBatis3):
            This attribute indicated Mybatis how to implement interface.
            It has ANNOTATEDMAPPER / MIXEDMAPPER / XMLMAPPER 3 selection
                ↪ to choice.
    -->
    <javaClientGenerator type="XMLMAPPER" targetPackage="com.pingcap.
        ↪ model" targetProject="src/main/java"/>

    <!--
        <table/> entire document: https://mybatis.org/generator/
            ↪ configreference/table.html

        table.tableName: The name of the database table.
        table.domainObjectName: The base name from which generated object
            ↪  names will be generated. If not specified, MBG will
            ↪ generate a name automatically based on the tableName.
        table.enableCountByExample: Signifies whether a count by example
            ↪ statement should be generated.
        table.enableUpdateByExample: Signifies whether an update by
            ↪ example statement should be generated.
        table.enableDeleteByExample: Signifies whether a delete by
            ↪ example statement should be generated.
        table.enableSelectByExample: Signifies whether a select by
            ↪ example statement should be generated.
        table.selectByExampleQueryId: This value will be added to the
            ↪ select list of the select by example statement in this
            ↪ form: "'<value>' as QUERYID".
    -->
    <table tableName="player" domainObjectName="Player"
           enableCountByExample="false" enableUpdateByExample="false"
           enableDeleteByExample="false" enableSelectByExample="false"
           selectByExampleQueryId="false"/>
</context>
```

```
</generatorConfiguration>
```

`mybatis-generator.xml` is included in `pom.xml` as the configuration of `mybatis-generator-maven-plugin`.

```
<plugin>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-maven-plugin</artifactId>
    <version>1.4.1</version>
    <configuration>
        <configurationFile>src/main/resources/mybatis-generator.xml</
            ↪ configurationFile>
        <verbose>true</verbose>
        <overwrite>true</overwrite>
    </configuration>

    <dependencies>
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java
            ↪ -->
        <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.49</version>
        </dependency>
    </dependencies>
</plugin>
```

Once included in the Maven plugin, you can delete the old generated files and make new ones using `mvn mybatis-generate`. Or you can use `make gen` to delete the old file and generate a new one at the same time.

> **Note:**
>
> The property `configuration.overwrite` in `mybatis-generator.xml` only ensures that the generated Java code files are overwritten. But the XML mapping files are still written as appended. Therefore, it is recommended to delete the old file before Mybaits Generator generating a new one.

`Player.java` is a data entity class file generated using Mybatis Generator, which is a mapping of database tables in the application. Each property of the `Player` class corresponds to a field in the `player` table.

```
package com.pingcap.model;
```

```
public class Player {
    private String id;

    private Integer coins;

    private Integer goods;

    public Player(String id, Integer coins, Integer goods) {
        this.id = id;
        this.coins = coins;
        this.goods = goods;
    }

    public Player() {
        super();
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public Integer getCoins() {
        return coins;
    }

    public void setCoins(Integer coins) {
        this.coins = coins;
    }

    public Integer getGoods() {
        return goods;
    }

    public void setGoods(Integer goods) {
        this.goods = goods;
    }
}
```

`PlayerMapper.java` is a mapping interface file generated using Mybatis Generator. This file only defines the interface, and the implementation classes of interface are automatically

generated using XML or annotations.

```
package com.pingcap.model;

import com.pingcap.model.Player;

public interface PlayerMapper {
    int deleteByPrimaryKey(String id);

    int insert(Player row);

    int insertSelective(Player row);

    Player selectByPrimaryKey(String id);

    int updateByPrimaryKeySelective(Player row);

    int updateByPrimaryKey(Player row);
}
```

`PlayerMapper.xml` is a mapping XML file generated using Mybatis Generator. Mybatis uses this to automatically generate the implementation class of the `PlayerMapper` interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://
    ↪ mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.pingcap.model.PlayerMapper">
  <resultMap id="BaseResultMap" type="com.pingcap.model.Player">
    <constructor>
      <idArg column="id" javaType="java.lang.String" jdbcType="VARCHAR" />
      <arg column="coins" javaType="java.lang.Integer" jdbcType="INTEGER" />
      <arg column="goods" javaType="java.lang.Integer" jdbcType="INTEGER" />
    </constructor>
  </resultMap>
  <sql id="Base_Column_List">
    id, coins, goods
  </sql>
  <select id="selectByPrimaryKey" parameterType="java.lang.String" resultMap
      ↪ ="BaseResultMap">
    select
    <include refid="Base_Column_List" />
    from player
    where id = #{id,jdbcType=VARCHAR}
  </select>
  <delete id="deleteByPrimaryKey" parameterType="java.lang.String">
    delete from player
```

```
  where id = #{id,jdbcType=VARCHAR}
</delete>
<insert id="insert" parameterType="com.pingcap.model.Player">
  insert into player (id, coins, goods
    )
  values (#{id,jdbcType=VARCHAR}, #{coins,jdbcType=INTEGER}, #{goods,
    ↪ jdbcType=INTEGER}
    )
</insert>
<insert id="insertSelective" parameterType="com.pingcap.model.Player">
  insert into player
  <trim prefix="(" suffix=")" suffixOverrides=",">
    <if test="id != null">
      id,
    </if>
    <if test="coins != null">
      coins,
    </if>
    <if test="goods != null">
      goods,
    </if>
  </trim>
  <trim prefix="values (" suffix=")" suffixOverrides=",">
    <if test="id != null">
      #{id,jdbcType=VARCHAR},
    </if>
    <if test="coins != null">
      #{coins,jdbcType=INTEGER},
    </if>
    <if test="goods != null">
      #{goods,jdbcType=INTEGER},
    </if>
  </trim>
</insert>
<update id="updateByPrimaryKeySelective" parameterType="com.pingcap.model.
  ↪ Player">
  update player
  <set>
    <if test="coins != null">
      coins = #{coins,jdbcType=INTEGER},
    </if>
    <if test="goods != null">
      goods = #{goods,jdbcType=INTEGER},
    </if>
  </set>
```

```
    where id = #{id,jdbcType=VARCHAR}
  </update>
  <update id="updateByPrimaryKey" parameterType="com.pingcap.model.Player">
    update player
    set coins = #{coins,jdbcType=INTEGER},
      goods = #{goods,jdbcType=INTEGER}
    where id = #{id,jdbcType=VARCHAR}
  </update>
</mapper>
```

Since Mybatis Generator needs to generate the source code from the table definition, the table needs to be created first. To create the table, you can use `dbinit.sql`.

```
USE test;
DROP TABLE IF EXISTS player;

CREATE TABLE player (
    `id` VARCHAR(36),
    `coins` INTEGER,
    `goods` INTEGER,
    PRIMARY KEY (`id`)
);
```

Split the interface `PlayerMapperEx` additionally to extend from `PlayerMapper` and write a matching `PlayerMapperEx.xml` file. Avoid changing `PlayerMapper.java` and `PlayerMapper.xml` directly. This is to avoid overwrite by Mybatis Generator.

Define the added interface in `PlayerMapperEx.java`:

```
package com.pingcap.model;

import java.util.List;

public interface PlayerMapperEx extends PlayerMapper {
    Player selectByPrimaryKeyWithLock(String id);

    List<Player> selectByLimit(Integer limit);

    Integer count();
}
```

Define the mapping rules in `PlayerMapperEx.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://
    ↪ mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.pingcap.model.PlayerMapperEx">
```

```
  <resultMap id="BaseResultMap" type="com.pingcap.model.Player">
    <constructor>
      <idArg column="id" javaType="java.lang.String" jdbcType="VARCHAR" />
      <arg column="coins" javaType="java.lang.Integer" jdbcType="INTEGER" />
      <arg column="goods" javaType="java.lang.Integer" jdbcType="INTEGER" />
    </constructor>
  </resultMap>
  <sql id="Base_Column_List">
    id, coins, goods
  </sql>

  <select id="selectByPrimaryKeyWithLock" parameterType="java.lang.String"
     ↪ resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List" />
    from player
    where `id` = #{id,jdbcType=VARCHAR}
    for update
  </select>

  <select id="selectByLimit" parameterType="java.lang.Integer" resultMap="
     ↪ BaseResultMap">
    select
    <include refid="Base_Column_List" />
    from player
    limit #{id,jdbcType=INTEGER}
  </select>

  <select id="count" resultType="java.lang.Integer">
    select count(*) from player
  </select>

</mapper>
```

`PlayerDAO.java` is a class used to manage data, in which `DAO` means Data Access Object. The class defines a set of data manipulation methods for writing data. In it, Mybatis encapsulates a large number of operations such as object mapping and CRUD of basic objects, which greatly simplifies the code.

```
package com.pingcap.dao;

import com.pingcap.model.Player;
import com.pingcap.model.PlayerMapperEx;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
```

```
import java.util.List;
import java.util.function.Function;

public class PlayerDAO {
    public static class NotEnoughException extends RuntimeException {
        public NotEnoughException(String message) {
            super(message);
        }
    }

    // Run SQL code in a way that automatically handles the
    // transaction retry logic, so we don't have to duplicate it in
    // various places.
    public Object runTransaction(SqlSessionFactory sessionFactory, Function<
        ↪ PlayerMapperEx, Object> fn) {
        Object resultObject = null;
        SqlSession session = null;

        try {
            // open a session with autoCommit is false
            session = sessionFactory.openSession(false);

            // get player mapper
            PlayerMapperEx playerMapperEx = session.getMapper(PlayerMapperEx.
                ↪ class);

            resultObject = fn.apply(playerMapperEx);
            session.commit();
            System.out.println("APP: COMMIT;");
        } catch (Exception e) {
            if (e instanceof NotEnoughException) {
                System.out.printf("APP: ROLLBACK BY LOGIC; \n%s\n", e.
                    ↪ getMessage());
            } else {
                System.out.printf("APP: ROLLBACK BY ERROR; \n%s\n", e.
                    ↪ getMessage());
            }

            if (session != null) {
                session.rollback();
            }
        } finally {
            if (session != null) {
                session.close();
```

```
        }
    }

    return resultObject;
}

public Function<PlayerMapperEx, Object> createPlayers(List<Player>
    ↪ players) {
    return playerMapperEx -> {
        Integer addedPlayerAmount = 0;
        for (Player player: players) {
            playerMapperEx.insert(player);
            addedPlayerAmount ++;
        }
        System.out.printf("APP: createPlayers() --> %d\n",
            ↪ addedPlayerAmount);
        return addedPlayerAmount;
    };
}

public Function<PlayerMapperEx, Object> buyGoods(String sellId, String
    ↪ buyId, Integer amount, Integer price) {
    return playerMapperEx -> {
        Player sellPlayer = playerMapperEx.selectByPrimaryKeyWithLock(
            ↪ sellId);
        Player buyPlayer = playerMapperEx.selectByPrimaryKeyWithLock(
            ↪ buyId);

        if (buyPlayer == null || sellPlayer == null) {
            throw new NotEnoughException("sell or buy player not exist");
        }

        if (buyPlayer.getCoins() < price || sellPlayer.getGoods() <
            ↪ amount) {
            throw new NotEnoughException("coins or goods not enough,
                ↪ rollback");
        }

        int affectRows = 0;
        buyPlayer.setGoods(buyPlayer.getGoods() + amount);
        buyPlayer.setCoins(buyPlayer.getCoins() - price);
        affectRows += playerMapperEx.updateByPrimaryKey(buyPlayer);

        sellPlayer.setGoods(sellPlayer.getGoods() - amount);
        sellPlayer.setCoins(sellPlayer.getCoins() + price);
```

```
            affectRows += playerMapperEx.updateByPrimaryKey(sellPlayer);

            System.out.printf("APP: buyGoods --> sell: %s, buy: %s, amount: %
                ↪ d, price: %d\n", sellId, buyId, amount, price);
            return affectRows;
        };
    }

    public Function<PlayerMapperEx, Object> getPlayerByID(String id) {
        return playerMapperEx -> playerMapperEx.selectByPrimaryKey(id);
    }

    public Function<PlayerMapperEx, Object> printPlayers(Integer limit) {
        return playerMapperEx -> {
            List<Player> players = playerMapperEx.selectByLimit(limit);

            for (Player player: players) {
                System.out.println("\n[printPlayers]:\n" + player);
            }
            return 0;
        };
    }

    public Function<PlayerMapperEx, Object> countPlayers() {
        return PlayerMapperEx::count;
    }
}
```

`MybatisExample` is the main class of the `plain-java-mybatis` sample application. It defines the entry functions:

```
package com.pingcap;

import com.pingcap.dao.PlayerDAO;
import com.pingcap.model.Player;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.Collections;

public class MybatisExample {
```

```
public static void main( String[] args ) throws IOException {
    // 1. Create a SqlSessionFactory based on our mybatis-config.xml
       ↪ configuration
    // file, which defines how to connect to the database.
    InputStream inputStream = Resources.getResourceAsStream("mybatis-
       ↪ config.xml");
    SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().
       ↪ build(inputStream);

    // 2. And then, create DAO to manager your data
    PlayerDAO playerDAO = new PlayerDAO();

    // 3. Run some simple examples.

    // Create a player who has 1 coin and 1 goods.
    playerDAO.runTransaction(sessionFactory, playerDAO.createPlayers(
           Collections.singletonList(new Player("test", 1, 1))));

    // Get a player.
    Player testPlayer = (Player)playerDAO.runTransaction(sessionFactory,
       ↪ playerDAO.getPlayerByID("test"));
    System.out.printf("PlayerDAO.getPlayer:\n => id: %s\n => coins: %s\n
       ↪ => goods: %s\n",
           testPlayer.getId(), testPlayer.getCoins(), testPlayer.getGoods
              ↪ ());

    // Count players amount.
    Integer count = (Integer)playerDAO.runTransaction(sessionFactory,
       ↪ playerDAO.countPlayers());
    System.out.printf("PlayerDAO.countPlayers:\n => %d total players\n",
       ↪ count);

    // Print 3 players.
    playerDAO.runTransaction(sessionFactory, playerDAO.printPlayers(3));

    // 4. Getting further.

    // Player 1: id is "1", has only 100 coins.
    // Player 2: id is "2", has 114514 coins, and 20 goods.
    Player player1 = new Player("1", 100, 0);
    Player player2 = new Player("2", 114514, 20);

    // Create two players "by hand", using the INSERT statement on the
       ↪ backend.
    int addedCount = (Integer)playerDAO.runTransaction(sessionFactory,
```

```
                playerDAO.createPlayers(Arrays.asList(player1, player2)));
        System.out.printf("PlayerDAO.createPlayers:\n => %d total inserted
            ↪ players\n", addedCount);

        // Player 1 wants to buy 10 goods from player 2.
        // It will cost 500 coins, but player 1 cannot afford it.
        System.out.println("\nPlayerDAO.buyGoods:\n => this trade will fail")
            ↪ ;
        Integer updatedCount = (Integer)playerDAO.runTransaction(
            ↪ sessionFactory,
                playerDAO.buyGoods(player2.getId(), player1.getId(), 10, 500))
                    ↪ ;
        System.out.printf("PlayerDAO.buyGoods:\n => %d total update players\n
            ↪ ", updatedCount);

        // So player 1 has to reduce the incoming quantity to two.
        System.out.println("\nPlayerDAO.buyGoods:\n => this trade will
            ↪ success");
        updatedCount = (Integer)playerDAO.runTransaction(sessionFactory,
                playerDAO.buyGoods(player2.getId(), player1.getId(), 2, 100));
        System.out.printf("PlayerDAO.buyGoods:\n => %d total update players\n
            ↪ ", updatedCount);
    }
}
```

Compared with Hibernate, the JDBC implementation might be not a best practice, because you need to write error handling logic manually and cannot reuse code easily, which makes your code slightly redundant.

Hibernate is a popular open-source Java ORM, and it supports TiDB dialect starting from `v6.0.0.Beta2`, which fits TiDB features well. The following instructions take `v6.0.0.Beta2` as an example.

Change to the `plain-java-hibernate` directory:

```
cd plain-java-hibernate
```

The structure of this directory is as follows:

```
.
├── Makefile
├── plain-java-hibernate.iml
├── pom.xml
└── src
    └── main
        └── java
            └── com
```

```
            ── pingcap
                ── HibernateExample.java
        ── resources
            ── hibernate.cfg.xml
```

`hibernate.cfg.xml` is the Hibernate configuration file:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>

        <!-- Database connection settings -->
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.
            ↪ Driver</property>
        <property name="hibernate.dialect">org.hibernate.dialect.TiDBDialect<
            ↪ /property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:4000
            ↪ /test</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property name="hibernate.connection.autocommit">false</property>

        <!-- Required so a table can be created from the 'PlayerDAO' class
            ↪ -->
        <property name="hibernate.hbm2ddl.auto">create-drop</property>

        <!-- Optional: Show SQL output for debugging -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
    </session-factory>
</hibernate-configuration>
```

`HibernateExample.java` is the main body of the `plain-java-hibernate`. Compared with JDBC, when using Hibernate, you only need to write the path of the configuration file, because Hibernate avoids differences in database creation between different databases.

`PlayerDAO` is a class used to manage data, in which `DAO` means Data Access Object. The class defines a set of data manipulation methods for writing data. Compared with JDBC, Hibernate encapsulates a large number of operations such as object mapping and CRUD of basic objects, which greatly simplifies the code.

`PlayerBean` is a data entity class that is a mapping for tables. Each property of a `PlayerBean` corresponds to a field in the `player` table. Compared with JDBC, `PlayerBean` in Hibernate adds annotations to indicate mapping relationships for more information.

```
package com.pingcap;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import org.hibernate.JDBCException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.NativeQuery;
import org.hibernate.query.Query;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.function.Function;

@Entity
@Table(name = "player_hibernate")
class PlayerBean {
    @Id
    private String id;
    @Column(name = "coins")
    private Integer coins;
    @Column(name = "goods")
    private Integer goods;

    public PlayerBean() {
    }

    public PlayerBean(String id, Integer coins, Integer goods) {
        this.id = id;
        this.coins = coins;
        this.goods = goods;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
```

```
    }

    public Integer getCoins() {
        return coins;
    }

    public void setCoins(Integer coins) {
        this.coins = coins;
    }

    public Integer getGoods() {
        return goods;
    }

    public void setGoods(Integer goods) {
        this.goods = goods;
    }

    @Override
    public String toString() {
        return String.format(" %-8s => %10s\n %-8s => %10s\n  %-8s => %10s\n
            ↪ ",
                "id", this.id, "coins", this.coins, "goods", this.goods);
    }
}

/**
 * Main class for the basic Hibernate example.
 **/
public class HibernateExample
{
    public static class PlayerDAO {
        public static class NotEnoughException extends RuntimeException {
            public NotEnoughException(String message) {
                super(message);
            }
        }

        // Run SQL code in a way that automatically handles the
        // transaction retry logic so we don't have to duplicate it in
        // various places.
        public Object runTransaction(Session session, Function<Session,
            ↪ Object> fn) {
            Object resultObject = null;
```

```
        Transaction txn = session.beginTransaction();
        try {
            resultObject = fn.apply(session);
            txn.commit();
            System.out.println("APP: COMMIT;");
        } catch (JDBCException e) {
            System.out.println("APP: ROLLBACK BY JDBC ERROR;");
            txn.rollback();
        } catch (NotEnoughException e) {
            System.out.printf("APP: ROLLBACK BY LOGIC; %s", e.getMessage()
                ↪ );
            txn.rollback();
        }
        return resultObject;
    }

    public Function<Session, Object> createPlayers(List<PlayerBean>
        ↪ players) throws JDBCException {
        return session -> {
            Integer addedPlayerAmount = 0;
            for (PlayerBean player: players) {
                session.persist(player);
                addedPlayerAmount ++;
            }
            System.out.printf("APP: createPlayers() --> %d\n",
                ↪ addedPlayerAmount);
            return addedPlayerAmount;
        };
    }

    public Function<Session, Object> buyGoods(String sellId, String
        ↪ buyId, Integer amount, Integer price) throws JDBCException {
        return session -> {
            PlayerBean sellPlayer = session.get(PlayerBean.class, sellId)
                ↪ ;
            PlayerBean buyPlayer = session.get(PlayerBean.class, buyId);

            if (buyPlayer == null || sellPlayer == null) {
                throw new NotEnoughException("sell or buy player not exist
                    ↪ ");
            }

            if (buyPlayer.getCoins() < price || sellPlayer.getGoods() <
                ↪ amount) {
```

```
            throw new NotEnoughException("coins or goods not enough,
                ↪ rollback");
        }

        buyPlayer.setGoods(buyPlayer.getGoods() + amount);
        buyPlayer.setCoins(buyPlayer.getCoins() - price);
        session.persist(buyPlayer);

        sellPlayer.setGoods(sellPlayer.getGoods() - amount);
        sellPlayer.setCoins(sellPlayer.getCoins() + price);
        session.persist(sellPlayer);

        System.out.printf("APP: buyGoods --> sell: %s, buy: %s, amount
            ↪ : %d, price: %d\n", sellId, buyId, amount, price);
        return 0;
    };
}

public Function<Session, Object> getPlayerByID(String id) throws
    ↪ JDBCException {
    return session -> session.get(PlayerBean.class, id);
}

public Function<Session, Object> printPlayers(Integer limit) throws
    ↪ JDBCException {
    return session -> {
        NativeQuery<PlayerBean> limitQuery = session.createNativeQuery
            ↪ ("SELECT * FROM player_hibernate LIMIT :limit",
            ↪ PlayerBean.class);
        limitQuery.setParameter("limit", limit);
        List<PlayerBean> players = limitQuery.getResultList();

        for (PlayerBean player: players) {
            System.out.println("\n[printPlayers]:\n" + player);
        }
        return 0;
    };
}

public Function<Session, Object> countPlayers() throws JDBCException
    ↪  {
    return session -> {
        Query<Long> countQuery = session.createQuery("SELECT count(
            ↪ player_hibernate) FROM PlayerBean player_hibernate",
            ↪ Long.class);
```

```
            return countQuery.getSingleResult();
        };
    }
}

public static void main(String[] args) {
    // 1. Create a SessionFactory based on our hibernate.cfg.xml
       ↪ configuration
    // file, which defines how to connect to the database.
    SessionFactory sessionFactory
            = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(PlayerBean.class)
            .buildSessionFactory();

    try (Session session = sessionFactory.openSession()) {
        // 2. And then, create DAO to manager your data.
        PlayerDAO playerDAO = new PlayerDAO();

        // 3. Run some simple example.

        // Create a player who has 1 coin and 1 goods.
        playerDAO.runTransaction(session, playerDAO.createPlayers(
           ↪ Collections.singletonList(
               new PlayerBean("test", 1, 1))));

        // Get a player.
        PlayerBean testPlayer = (PlayerBean)playerDAO.runTransaction(
           ↪ session, playerDAO.getPlayerByID("test"));
        System.out.printf("PlayerDAO.getPlayer:\n => id: %s\n => coins: %
           ↪ s\n => goods: %s\n",
               testPlayer.getId(), testPlayer.getCoins(), testPlayer.
                  ↪ getGoods());

        // Count players amount.
        Long count = (Long)playerDAO.runTransaction(session, playerDAO.
           ↪ countPlayers());
        System.out.printf("PlayerDAO.countPlayers:\n => %d total players\
           ↪ n", count);

        // Print 3 players.
        playerDAO.runTransaction(session, playerDAO.printPlayers(3));

        // 4. Getting further.
```

```
            // Player 1: id is "1", has only 100 coins.
            // Player 2: id is "2", has 114514 coins, and 20 goods.
            PlayerBean player1 = new PlayerBean("1", 100, 0);
            PlayerBean player2 = new PlayerBean("2", 114514, 20);

            // Create two players "by hand", using the INSERT statement on
               ↪ the backend.
            int addedCount = (Integer)playerDAO.runTransaction(session,
                    playerDAO.createPlayers(Arrays.asList(player1, player2)));
            System.out.printf("PlayerDAO.createPlayers:\n => %d total
               ↪ inserted players\n", addedCount);

            // Player 1 wants to buy 10 goods from player 2.
            // It will cost 500 coins, but player 1 can't afford it.
            System.out.println("\nPlayerDAO.buyGoods:\n => this trade will
               ↪ fail");
            Integer updatedCount = (Integer)playerDAO.runTransaction(session,
                    playerDAO.buyGoods(player2.getId(), player1.getId(), 10,
                       ↪ 500));
            System.out.printf("PlayerDAO.buyGoods:\n => %d total update
               ↪ players\n", updatedCount);

            // So player 1 have to reduce his incoming quantity to two.
            System.out.println("\nPlayerDAO.buyGoods:\n => this trade will
               ↪ success");
            updatedCount = (Integer)playerDAO.runTransaction(session,
                    playerDAO.buyGoods(player2.getId(), player1.getId(), 2,
                       ↪ 100));
            System.out.printf("PlayerDAO.buyGoods:\n => %d total update
               ↪ players\n", updatedCount);
        } finally {
            sessionFactory.close();
        }
    }
}
```

Change to the `plain-java-jdbc` directory:

```
cd plain-java-jdbc
```

The structure of this directory is as follows:

```
.
├── Makefile
├── plain-java-jdbc.iml
└── pom.xml
```

```
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── com
│   │   │   │   ├── pingcap
│   │   │   │   │   ├── JDBCExample.java
│   │   ├── resources
│   │   │   ├── dbinit.sql
```

You can find initialization statements for the table creation in `dbinit.sql`:

```
USE test;
DROP TABLE IF EXISTS player;

CREATE TABLE player (
    `id` VARCHAR(36),
    `coins` INTEGER,
    `goods` INTEGER,
   PRIMARY KEY (`id`)
);
```

`JDBCExample.java` is the main body of the `plain-java-jdbc`. TiDB is highly compatible with the MySQL protocol, so you need to initialize a MySQL source instance `MysqlDataSource` to connect to TiDB. Then, you can initialize `PlayerDAO` for object management and use it to read, edit, add, and delete data.

`PlayerDAO` is a class used to manage data, in which `DAO` means Data Access Object. The class defines a set of data manipulation methods to provide the ability to write data.

`PlayerBean` is a data entity class that is a mapping for tables. Each property of a `PlayerBean` corresponds to a field in the `player` table.

```
package com.pingcap;

import com.mysql.cj.jdbc.MysqlDataSource;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.*;

/**
 * Main class for the basic JDBC example.
 **/
public class JDBCExample
{
```

```
public static class PlayerBean {
    private String id;
    private Integer coins;
    private Integer goods;

    public PlayerBean() {
    }

    public PlayerBean(String id, Integer coins, Integer goods) {
        this.id = id;
        this.coins = coins;
        this.goods = goods;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public Integer getCoins() {
        return coins;
    }

    public void setCoins(Integer coins) {
        this.coins = coins;
    }

    public Integer getGoods() {
        return goods;
    }

    public void setGoods(Integer goods) {
        this.goods = goods;
    }

    @Override
    public String toString() {
        return String.format(" %-8s => %10s\n %-8s => %10s\n  %-8s => %10
            ↪ s\n",
                "id", this.id, "coins", this.coins, "goods", this.goods);
    }
}
```

```
/**
 * Data access object used by 'ExampleDataSource'.
 * Example for CURD and bulk insert.
 */
public static class PlayerDAO {
    private final MysqlDataSource ds;
    private final Random rand = new Random();

    PlayerDAO(MysqlDataSource ds) {
        this.ds = ds;
    }

    /**
     * Create players by passing in a List of PlayerBean.
     *
     * @param players Will create players list
     * @return The number of create accounts
     */
    public int createPlayers(List<PlayerBean> players){
        int rows = 0;

        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            connection = ds.getConnection();
            preparedStatement = connection.prepareStatement("INSERT INTO
                ↪ player (id, coins, goods) VALUES (?, ?, ?)");
        } catch (SQLException e) {
            System.out.printf("[createPlayers] ERROR: { state => %s, cause
                ↪  => %s, message => %s }\n",
                    e.getSQLState(), e.getCause(), e.getMessage());
            e.printStackTrace();

            return -1;
        }

        try {
            for (PlayerBean player : players) {
                preparedStatement.setString(1, player.getId());
                preparedStatement.setInt(2, player.getCoins());
                preparedStatement.setInt(3, player.getGoods());

                preparedStatement.execute();
                rows += preparedStatement.getUpdateCount();
```

```
            }
        } catch (SQLException e) {
            System.out.printf("[createPlayers] ERROR: { state => %s, cause
                ↪  => %s, message => %s }\n",
                    e.getSQLState(), e.getCause(), e.getMessage());
            e.printStackTrace();
        } finally {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        System.out.printf("\n[createPlayers]:\n '%s'\n",
            ↪ preparedStatement);
        return rows;
    }

    /**
     * Buy goods and transfer funds between one player and another in
        ↪ one transaction.
     * @param sellId Sell player id.
     * @param buyId Buy player id.
     * @param amount Goods amount, if sell player has not enough goods,
        ↪  the trade will break.
     * @param price Price should pay, if buy player has not enough
        ↪ coins, the trade will break.
     *
     * @return The number of effected players.
     */
    public int buyGoods(String sellId, String buyId, Integer amount,
        ↪ Integer price) {
        int effectPlayers = 0;

        Connection connection = null;
        try {
            connection = ds.getConnection();
        } catch (SQLException e) {
            System.out.printf("[buyGoods] ERROR: { state => %s, cause => %
                ↪ s, message => %s }\n",
                    e.getSQLState(), e.getCause(), e.getMessage());
            e.printStackTrace();
            return effectPlayers;
        }
```

```
try {
    connection.setAutoCommit(false);

    PreparedStatement playerQuery = connection.prepareStatement("
        ↪ SELECT * FROM player WHERE id=? OR id=? FOR UPDATE");
    playerQuery.setString(1, sellId);
    playerQuery.setString(2, buyId);
    playerQuery.execute();

    PlayerBean sellPlayer = null;
    PlayerBean buyPlayer = null;

    ResultSet playerQueryResultSet = playerQuery.getResultSet();
    while (playerQueryResultSet.next()) {
        PlayerBean player = new PlayerBean(
                playerQueryResultSet.getString("id"),
                playerQueryResultSet.getInt("coins"),
                playerQueryResultSet.getInt("goods")
        );

        System.out.println("\n[buyGoods]:\n 'check goods and coins
            ↪  enough'");
        System.out.println(player);

        if (sellId.equals(player.getId())) {
            sellPlayer = player;
        } else {
            buyPlayer = player;
        }
    }

    if (sellPlayer == null || buyPlayer == null) {
        throw new SQLException("player not exist.");
    }

    if (sellPlayer.getGoods().compareTo(amount) < 0) {
        throw new SQLException(String.format("sell player %s goods
            ↪  not enough.", sellId));
    }

    if (buyPlayer.getCoins().compareTo(price) < 0) {
        throw new SQLException(String.format("buy player %s coins
            ↪ not enough.", buyId));
    }
```

```
            PreparedStatement transfer = connection.prepareStatement("
                ↪ UPDATE player set goods = goods + ?, coins = coins + ?
                ↪ WHERE id=?");
            transfer.setInt(1, -amount);
            transfer.setInt(2, price);
            transfer.setString(3, sellId);
            transfer.execute();
            effectPlayers += transfer.getUpdateCount();

            transfer.setInt(1, amount);
            transfer.setInt(2, -price);
            transfer.setString(3, buyId);
            transfer.execute();
            effectPlayers += transfer.getUpdateCount();

            connection.commit();

            System.out.println("\n[buyGoods]:\n 'trade success'");
        } catch (SQLException e) {
            System.out.printf("[buyGoods] ERROR: { state => %s, cause => %
                ↪ s, message => %s }\n",
                    e.getSQLState(), e.getCause(), e.getMessage());

            try {
                System.out.println("[buyGoods] Rollback");

                connection.rollback();
            } catch (SQLException ex) {
                // do nothing
            }
        } finally {
            try {
                connection.close();
            } catch (SQLException e) {
                // do nothing
            }
        }

        return effectPlayers;
    }

    /**
     * Get the player info by id.
     *
```

```
     * @param id Player id.
     * @return The player of this id.
     */
    public PlayerBean getPlayer(String id) {
        PlayerBean player = null;

        try (Connection connection = ds.getConnection()) {
            PreparedStatement preparedStatement = connection.
                ↪ prepareStatement("SELECT * FROM player WHERE id = ?");
            preparedStatement.setString(1, id);
            preparedStatement.execute();

            ResultSet res = preparedStatement.executeQuery();
            if(!res.next()) {
                System.out.printf("No players in the table with id %s", id
                    ↪ );
            } else {
                player = new PlayerBean(res.getString("id"), res.getInt("
                    ↪ coins"), res.getInt("goods"));
            }
        } catch (SQLException e) {
            System.out.printf("PlayerDAO.getPlayer ERROR: { state => %s,
                ↪ cause => %s, message => %s }\n",
                    e.getSQLState(), e.getCause(), e.getMessage());
        }

        return player;
    }

    /**
     * Insert randomized account data (id, coins, goods) using the JDBC
        ↪  fast path for
     * bulk inserts. The fastest way to get data into TiDB is using the
     * TiDB Lightning(https://docs.pingcap.com/tidb/stable/tidb-
        ↪ lightning-overview).
     * However, if you must bulk insert from the application using
        ↪ INSERT SQL, the best
     * option is the method shown here. It will require the following:
     *
     *   Add `rewriteBatchedStatements=true` to your JDBC connection
        ↪ settings.
     *   Setting rewriteBatchedStatements to true now causes
        ↪ CallableStatements
     *   with batched arguments to be re-written in the form "CALL
        ↪ (...); CALL (...); ..."
```

```
     *   to send the batch in as few client/server round trips as
         ↪ possible.
     *   https://dev.mysql.com/doc/relnotes/connector-j/5.1/en/news
         ↪ -5-1-3.html
     *
     *   You can see the `rewriteBatchedStatements` param effect logic
         ↪ at
     *   implement function: `com.mysql.cj.jdbc.StatementImpl.
         ↪ executeBatchUsingMultiQueries`
     *
     * @param total Add players amount.
     * @param batchSize Bulk insert size for per batch.
     *
     * @return The number of new accounts inserted.
     */
    public int bulkInsertRandomPlayers(Integer total, Integer batchSize)
        ↪ {
        int totalNewPlayers = 0;

        try (Connection connection = ds.getConnection()) {
            // We're managing the commit lifecycle ourselves, so we can
            // control the size of our batch inserts.
            connection.setAutoCommit(false);

            // In this example we are adding 500 rows to the database,
            // but it could be any number. What's important is that
            // the batch size is 128.
            try (PreparedStatement pstmt = connection.prepareStatement("
                ↪ INSERT INTO player (id, coins, goods) VALUES (?, ?, ?)"
                ↪ )) {
                for (int i=0; i<=(total/batchSize);i++) {
                    for (int j=0; j<batchSize; j++) {
                        String id = UUID.randomUUID().toString();
                        pstmt.setString(1, id);
                        pstmt.setInt(2, rand.nextInt(10000));
                        pstmt.setInt(3, rand.nextInt(10000));
                        pstmt.addBatch();
                    }

                    int[] count = pstmt.executeBatch();
                    totalNewPlayers += count.length;
                    System.out.printf("\nPlayerDAO.bulkInsertRandomPlayers
                        ↪ :\n '%s'\n", pstmt);
                    System.out.printf(" => %s row(s) updated in this batch
                        ↪ \n", count.length);
```

```
            }
            connection.commit();
        } catch (SQLException e) {
            System.out.printf("PlayerDAO.bulkInsertRandomPlayers ERROR
                ↪ : { state => %s, cause => %s, message => %s }\n",
                    e.getSQLState(), e.getCause(), e.getMessage());
        }
    } catch (SQLException e) {
        System.out.printf("PlayerDAO.bulkInsertRandomPlayers ERROR: {
            ↪ state => %s, cause => %s, message => %s }\n",
                e.getSQLState(), e.getCause(), e.getMessage());
    }
    return totalNewPlayers;
}


/**
 * Print a subset of players from the data store by limit.
 *
 * @param limit Print max size.
 */
public void printPlayers(Integer limit) {
    try (Connection connection = ds.getConnection()) {
        PreparedStatement preparedStatement = connection.
            ↪ prepareStatement("SELECT * FROM player LIMIT ?");
        preparedStatement.setInt(1, limit);
        preparedStatement.execute();

        ResultSet res = preparedStatement.executeQuery();
        while (!res.next()) {
            PlayerBean player = new PlayerBean(res.getString("id"),
                    res.getInt("coins"), res.getInt("goods"));
            System.out.println("\n[printPlayers]:\n" + player);
        }
    } catch (SQLException e) {
        System.out.printf("PlayerDAO.printPlayers ERROR: { state => %s
            ↪ , cause => %s, message => %s }\n",
                e.getSQLState(), e.getCause(), e.getMessage());
    }
}


/**
 * Count players from the data store.
 *
```

```
     * @return All players count
     */
    public int countPlayers() {
        int count = 0;

        try (Connection connection = ds.getConnection()) {
            PreparedStatement preparedStatement = connection.
                ↪ prepareStatement("SELECT count(*) FROM player");
            preparedStatement.execute();

            ResultSet res = preparedStatement.executeQuery();
            if(res.next()) {
                count = res.getInt(1);
            }
        } catch (SQLException e) {
            System.out.printf("PlayerDAO.countPlayers ERROR: { state => %s
                ↪ , cause => %s, message => %s }\n",
                    e.getSQLState(), e.getCause(), e.getMessage());
        }

        return count;
    }
}

public static void main(String[] args) {
    // 1. Configure the example database connection.

    // 1.1 Create a mysql data source instance.
    MysqlDataSource mysqlDataSource = new MysqlDataSource();

    // 1.2 Set server name, port, database name, username and password.
    mysqlDataSource.setServerName("localhost");
    mysqlDataSource.setPortNumber(4000);
    mysqlDataSource.setDatabaseName("test");
    mysqlDataSource.setUser("root");
    mysqlDataSource.setPassword("");

    // Or you can use jdbc string instead.
    // mysqlDataSource.setURL("jdbc:mysql://{host}:{port}/test?user={
        ↪ user}&password={password}");

    // 2. And then, create DAO to manager your data.
    PlayerDAO dao = new PlayerDAO(mysqlDataSource);

    // 3. Run some simple example.
```

```
// Create a player, has a coin and a goods.
dao.createPlayers(Collections.singletonList(new PlayerBean("test",
    ↪ 1, 1)));

// Get a player.
PlayerBean testPlayer = dao.getPlayer("test");
System.out.printf("PlayerDAO.getPlayer:\n => id: %s\n => coins: %s\n
    ↪ => goods: %s\n",
        testPlayer.getId(), testPlayer.getCoins(), testPlayer.getGoods
            ↪ ());

// Create players with bulk inserts, insert 1919 players totally,
    ↪ and per batch for 114 players.
int addedCount = dao.bulkInsertRandomPlayers(1919, 114);
System.out.printf("PlayerDAO.bulkInsertRandomPlayers:\n => %d total
    ↪ inserted players\n", addedCount);

// Count players amount.
int count = dao.countPlayers();
System.out.printf("PlayerDAO.countPlayers:\n => %d total players\n",
    ↪ count);

// Print 3 players.
dao.printPlayers(3);

// 4. Getting further.

// Player 1: id is "1", has only 100 coins.
// Player 2: id is "2", has 114514 coins, and 20 goods.
PlayerBean player1 = new PlayerBean("1", 100, 0);
PlayerBean player2 = new PlayerBean("2", 114514, 20);

// Create two players "by hand", using the INSERT statement on the
    ↪ backend.
addedCount = dao.createPlayers(Arrays.asList(player1, player2));
System.out.printf("PlayerDAO.createPlayers:\n => %d total inserted
    ↪ players\n", addedCount);

// Player 1 wants to buy 10 goods from player 2.
// It will cost 500 coins, but player 1 can't afford it.
System.out.println("\nPlayerDAO.buyGoods:\n => this trade will fail")
    ↪ ;
int updatedCount = dao.buyGoods(player2.getId(), player1.getId(),
    ↪ 10, 500);
```

```
        System.out.printf("PlayerDAO.buyGoods:\n => %d total update players\n
            ↪ ", updatedCount);

        // So player 1 have to reduce his incoming quantity to two.
        System.out.println("\nPlayerDAO.buyGoods:\n => this trade will
            ↪ success");
        updatedCount = dao.buyGoods(player2.getId(), player1.getId(), 2, 100)
            ↪ ;
        System.out.printf("PlayerDAO.buyGoods:\n => %d total update players\n
            ↪ ", updatedCount);
    }
}
```

4.3.3.3 Step 3. Run the code

The following content introduces how to run the code step by step.

4.3.3.3.1 Step 3.1 Table initialization

When using Mybatis, you need to initialize the database tables manually. If you are using a local cluster, and MySQL client has been installed locally, you can run it directly in the `plain-java-mybatis` directory:

```
make prepare
```

Or you can execute the following command:

```
mysql --host 127.0.0.1 --port 4000 -u root < src/main/resources/dbinit.sql
```

If you are using a non-local cluster or MySQL client has not been installed, connect to your cluster and run the statement in the `src/main/resources/dbinit.sql` file.

No need to initialize tables manually.

When using JDBC, you need to initialize the database tables manually. If you are using a local cluster, and MySQL client has been installed locally, you can run it directly in the `plain-java-jdbc` directory:

```
make mysql
```

Or you can execute the following command:

```
mysql --host 127.0.0.1 --port 4000 -u root<src/main/resources/dbinit.sql
```

If you are using a non-local cluster or MySQL client has not been installed, connect to your cluster and run the statement in the `src/main/resources/dbinit.sql` file.

4.3.3.3.2 Step 3.2 Modify parameters for TiDB Cloud

If you are using a TiDB Serverless cluster, modify the `dataSource.url`, `dataSource.username`, `dataSource.password` in `mybatis-config.xml`.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE configuration
       PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
       "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
    <settings>
        <setting name="cacheEnabled" value="true"/>
        <setting name="lazyLoadingEnabled" value="false"/>
        <setting name="aggressiveLazyLoading" value="true"/>
        <setting name="logImpl" value="LOG4J"/>
    </settings>

    <typeAliases>
        <package name="com.pingcap.dao"/>
    </typeAliases>

    <environments default="development">
        <environment id="development">
            <!-- JDBC transaction manager -->
            <transactionManager type="JDBC"/>
            <!-- Database pool -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://127.0.0.1:4000/test"/
                    ↪ >
                <property name="username" value="root"/>
                <property name="password" value=""/>
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <mapper resource="mapper/PlayerMapper.xml"/>
        <mapper resource="mapper/PlayerMapperEx.xml"/>
    </mappers>

</configuration>
```

Suppose that the password you set is `123456`, and the connection parameters you get

from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the parameters in `dataSource` node as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE configuration
       PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
       "http://mybatis.org/dtd/mybatis-3-config.dtd">

       ...
           <!-- Database pool -->
           <dataSource type="POOLED">
               <property name="driver" value="com.mysql.jdbc.Driver"/>
               <property name="url" value="jdbc:mysql://xxx.tidbcloud.
                   ↪ com:4000/test?sslMode=VERIFY_IDENTITY&amp;
                   ↪ enabledTLSProtocols=TLSv1.2,TLSv1.3"/>
               <property name="username" value="2aEp24QWEDLqRFs.root"/>
               <property name="password" value="123456"/>
           </dataSource>
       ...

</configuration>
```

If you are using a TiDB Serverless cluster, modify the `hibernate.connection.url`, `hibernate.connection.username`, `hibernate.connection.password` in `hibernate.cfg.xml`.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
       "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
       "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>

       <!-- Database connection settings -->
       <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.
           ↪ Driver</property>
       <property name="hibernate.dialect">org.hibernate.dialect.TiDBDialect<
           ↪ /property>
       <property name="hibernate.connection.url">jdbc:mysql://localhost:4000
           ↪ /test</property>
```

```
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property name="hibernate.connection.autocommit">false</property>

        <!-- Required so a table can be created from the 'PlayerDAO' class
            ↪ -->
        <property name="hibernate.hbm2ddl.auto">create-drop</property>

        <!-- Optional: Show SQL output for debugging -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
    </session-factory>
</hibernate-configuration>
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the parameters as follows:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>

        <!-- Database connection settings -->
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.
            ↪ Driver</property>
        <property name="hibernate.dialect">org.hibernate.dialect.TiDBDialect<
            ↪ /property>
        <property name="hibernate.connection.url">jdbc:mysql://xxx.tidbcloud.
            ↪ com:4000/test?sslMode=VERIFY_IDENTITY&amp;enabledTLSProtocols=
            ↪ TLSv1.2,TLSv1.3</property>
        <property name="hibernate.connection.username">2aEp24QWEDLqRFs.root</
            ↪ property>
        <property name="hibernate.connection.password">123456</property>
        <property name="hibernate.connection.autocommit">false</property>

        <!-- Required so a table can be created from the 'PlayerDAO' class
            ↪ -->
```

```
        <property name="hibernate.hbm2ddl.auto">create-drop</property>

        <!-- Optional: Show SQL output for debugging -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
    </session-factory>
</hibernate-configuration>
```

If you are using a TiDB Serverless cluster, modify the parameters of the host, port, user, and password in `JDBCExample.java`:

```
mysqlDataSource.setServerName("localhost");
mysqlDataSource.setPortNumber(4000);
mysqlDataSource.setDatabaseName("test");
mysqlDataSource.setUser("root");
mysqlDataSource.setPassword("");
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the parameters as follows:

```
mysqlDataSource.setServerName("xxx.tidbcloud.com");
mysqlDataSource.setPortNumber(4000);
mysqlDataSource.setDatabaseName("test");
mysqlDataSource.setUser("2aEp24QWEDLqRFs.root");
mysqlDataSource.setPassword("123456");
mysqlDataSource.setSslMode(PropertyDefinitions.SslMode.VERIFY_IDENTITY.name
    ↪ ());
mysqlDataSource.setEnabledTLSProtocols("TLSv1.2,TLSv1.3");
```

4.3.3.3.3 Step 3.3 Run

To run the code, you can run `make prepare`, `make gen`, `make build` and `make run` respectively:

```
make prepare
### this command executes :
### - `mysql --host 127.0.0.1 --port 4000 -u root < src/main/resources/
    ↪ dbinit.sql`
### - `mysql --host 127.0.0.1 --port 4000 -u root -e "TRUNCATE test.player"`
```

```
make gen
### this command executes :
### - `rm -f src/main/java/com/pingcap/model/Player.java`
### - `rm -f src/main/java/com/pingcap/model/PlayerMapper.java`
### - `rm -f src/main/resources/mapper/PlayerMapper.xml`
### - `mvn mybatis-generator:generate`

make build # this command executes `mvn clean package`
make run # this command executes `java -jar target/plain-java-mybatis-0.0.1-
    ↪ jar-with-dependencies.jar`
```

Or you can use the native commands:

```
mysql --host 127.0.0.1 --port 4000 -u root < src/main/resources/dbinit.sql
mysql --host 127.0.0.1 --port 4000 -u root -e "TRUNCATE test.player"
rm -f src/main/java/com/pingcap/model/Player.java
rm -f src/main/java/com/pingcap/model/PlayerMapper.java
rm -f src/main/resources/mapper/PlayerMapper.xml
mvn mybatis-generator:generate
mvn clean package
java -jar target/plain-java-mybatis-0.0.1-jar-with-dependencies.jar
```

Or run the `make` command directly, which is a combination of `make prepare`, `make gen`, `make build` and `make run`.

To run the code, you can run `make build` and `make run` respectively:

```
make build # this command executes `mvn clean package`
make run # this command executes `java -jar target/plain-java-jdbc-0.0.1-jar
    ↪ -with-dependencies.jar`
```

Or you can use the native commands:

```
mvn clean package
java -jar target/plain-java-jdbc-0.0.1-jar-with-dependencies.jar
```

Or run the `make` command directly, which is a combination of `make build` and `make run`.

To run the code, you can run `make build` and `make run` respectively:

```
make build # this command executes `mvn clean package`
make run # this command executes `java -jar target/plain-java-jdbc-0.0.1-jar
    ↪ -with-dependencies.jar`
```

Or you can use the native commands:

```
mvn clean package
java -jar target/plain-java-jdbc-0.0.1-jar-with-dependencies.jar
```

Or run the `make` command directly, which is a combination of `make build` and `make run`.

4.3.3.4 Step 4. Expected output

Mybatis Expected Output

Hibernate Expected Output

JDBC Expected Output

4.3.4 Build a Simple CRUD App with TiDB and Python

This document describes how to use TiDB and Python to build a simple CRUD application.

> **Note:**
>
> It is recommended to use Python 3.10 or a later Python version.

4.3.4.1 Step 1. Launch your TiDB cluster

The following introduces how to start a TiDB cluster.

Use a TiDB Serverless cluster

For detailed steps, see Create a TiDB Serverless cluster.

Use a local cluster

For detailed steps, see Deploy a local test cluster or Deploy a TiDB cluster using TiUP.

4.3.4.2 Step 2. Get the code

```
git clone https://github.com/pingcap-inc/tidb-example-python.git
```

SQLAlchemy is a popular open-source ORM library for Python. The following uses SQLAlchemy 1.44 as an example.

```
import uuid
from typing import List

from sqlalchemy import create_engine, String, Column, Integer, select, func
from sqlalchemy.orm import declarative_base, sessionmaker

engine = create_engine('mysql://root:@127.0.0.1:4000/test')
Base = declarative_base()
```

```
Base.metadata.create_all(engine)
Session = sessionmaker(bind=engine)


class Player(Base):
    __tablename__ = "player"

    id = Column(String(36), primary_key=True)
    coins = Column(Integer)
    goods = Column(Integer)

    def __repr__(self):
        return f'Player(id={self.id!r}, coins={self.coins!r}, goods={self.
            ↪ goods!r})'


def random_player(amount: int) -> List[Player]:
    players = []
    for _ in range(amount):
        players.append(Player(id=uuid.uuid4(), coins=10000, goods=10000))

    return players


def simple_example() -> None:
    with Session() as session:
        # create a player, who has a coin and a goods.
        session.add(Player(id="test", coins=1, goods=1))

        # get this player, and print it.
        get_test_stmt = select(Player).where(Player.id == "test")
        for player in session.scalars(get_test_stmt):
            print(player)

        # create players with bulk inserts.
        # insert 1919 players totally, with 114 players per batch.
        # each player has a random UUID
        player_list = random_player(1919)
        for idx in range(0, len(player_list), 114):
            session.bulk_save_objects(player_list[idx:idx + 114])

        # print the number of players
        count = session.query(func.count(Player.id)).scalar()
        print(f'number of players: {count}')
```

```
        # print 3 players.
        three_players = session.query(Player).limit(3).all()
        for player in three_players:
            print(player)

        session.commit()


def trade_check(session: Session, sell_id: str, buy_id: str, amount: int,
    ↪ price: int) -> bool:
    # sell player goods check
    sell_player = session.query(Player.goods).filter(Player.id == sell_id).
        ↪ with_for_update().one()
    if sell_player.goods < amount:
        print(f'sell player {sell_id} goods not enough')
        return False

    # buy player coins check
    buy_player = session.query(Player.coins).filter(Player.id == buy_id).
        ↪ with_for_update().one()
    if buy_player.coins < price:
        print(f'buy player {buy_id} coins not enough')
        return False


def trade(sell_id: str, buy_id: str, amount: int, price: int) -> None:
    with Session() as session:
        if trade_check(session, sell_id, buy_id, amount, price) is False:
            return

        # deduct the goods of seller, and raise his/her the coins
        session.query(Player).filter(Player.id == sell_id). \
            update({'goods': Player.goods - amount, 'coins': Player.coins +
                ↪ price})
        # deduct the coins of buyer, and raise his/her the goods
        session.query(Player).filter(Player.id == buy_id). \
            update({'goods': Player.goods + amount, 'coins': Player.coins -
                ↪ price})

        session.commit()
        print("trade success")


def trade_example() -> None:
    with Session() as session:
```

```
        # create two players
        # player 1: id is "1", has only 100 coins.
        # player 2: id is "2", has 114514 coins, and 20 goods.
        session.add(Player(id="1", coins=100, goods=0))
        session.add(Player(id="2", coins=114514, goods=20))
        session.commit()

    # player 1 wants to buy 10 goods from player 2.
    # it will cost 500 coins, but player 1 cannot afford it.
    # so this trade will fail, and nobody will lose their coins or goods
    trade(sell_id="2", buy_id="1", amount=10, price=500)

    # then player 1 has to reduce the incoming quantity to 2.
    # this trade will be successful
    trade(sell_id="2", buy_id="1", amount=2, price=100)

    with Session() as session:
        traders = session.query(Player).filter(Player.id.in_(("1", "2"))).
            ↪ all()
        for player in traders:
            print(player)
        session.commit()


simple_example()
trade_example()
```

Compared with using drivers directly, SQLAlchemy provides an abstraction for the specific details of different databases when you create a database connection. In addition, SQLAlchemy encapsulates some operations such as session management and CRUD of basic objects, which greatly simplifies the code.

The `Player` class is a mapping of a table to attributes in the application. Each attribute of `Player` corresponds to a field in the `player` table. To provide SQLAlchemy with more information, the attribute is defined as `id = Column(String(36), primary_key=True)` to indicate the field type and its additional attributes. For example, `id = Column(String(36) ↪ , primary_key=True)` indicates that the `id` attribute is `String` type, the corresponding field in database is `VARCHAR` type, the length is `36`, and it is a primary key.

For more information about how to use SQLAlchemy, refer to SQLAlchemy documentation.

peewee is a popular open-source ORM library for Python. The following uses peewee 3.15.4 as an example.

```
import os
import uuid
```

```python
from typing import List

from peewee import *

from playhouse.db_url import connect

db = connect('mysql://root:@127.0.0.1:4000/test')


class Player(Model):
    id = CharField(max_length=36, primary_key=True)
    coins = IntegerField()
    goods = IntegerField()

    class Meta:
        database = db
        table_name = "player"


def random_player(amount: int) -> List[Player]:
    players = []
    for _ in range(amount):
        players.append(Player(id=uuid.uuid4(), coins=10000, goods=10000))

    return players


def simple_example() -> None:
    # create a player, who has a coin and a goods.
    Player.create(id="test", coins=1, goods=1)

    # get this player, and print it.
    test_player = Player.select().where(Player.id == "test").get()
    print(f'id:{test_player.id}, coins:{test_player.coins}, goods:{
        ↪ test_player.goods}')

    # create players with bulk inserts.
    # insert 1919 players totally, with 114 players per batch.
    # each player has a random UUID
    player_list = random_player(1919)
    Player.bulk_create(player_list, 114)

    # print the number of players
    count = Player.select().count()
    print(f'number of players: {count}')
```

```
    # print 3 players.
    three_players = Player.select().limit(3)
    for player in three_players:
        print(f'id:{player.id}, coins:{player.coins}, goods:{player.goods}')


def trade_check(sell_id: str, buy_id: str, amount: int, price: int) -> bool
    ↪ :
    sell_goods = Player.select(Player.goods).where(Player.id == sell_id).get
        ↪ ().goods
    if sell_goods < amount:
        print(f'sell player {sell_id} goods not enough')
        return False

    buy_coins = Player.select(Player.coins).where(Player.id == buy_id).get()
        ↪ .coins
    if buy_coins < price:
        print(f'buy player {buy_id} coins not enough')
        return False

    return True


def trade(sell_id: str, buy_id: str, amount: int, price: int) -> None:
    with db.atomic() as txn:
        try:
            if trade_check(sell_id, buy_id, amount, price) is False:
                txn.rollback()
                return

            # deduct the goods of seller, and raise his/her the coins
            Player.update(goods=Player.goods - amount, coins=Player.coins +
                ↪ price).where(Player.id == sell_id).execute()
            # deduct the coins of buyer, and raise his/her the goods
            Player.update(goods=Player.goods + amount, coins=Player.coins -
                ↪ price).where(Player.id == buy_id).execute()

        except Exception as err:
            txn.rollback()
            print(f'something went wrong: {err}')
        else:
            txn.commit()
            print("trade success")
```

```
def trade_example() -> None:
    # create two players
    # player 1: id is "1", has only 100 coins.
    # player 2: id is "2", has 114514 coins, and 20 goods.
    Player.create(id="1", coins=100, goods=0)
    Player.create(id="2", coins=114514, goods=20)

    # player 1 wants to buy 10 goods from player 2.
    # it will cost 500 coins, but player 1 cannot afford it.
    # so this trade will fail, and nobody will lose their coins or goods
    trade(sell_id="2", buy_id="1", amount=10, price=500)

    # then player 1 has to reduce the incoming quantity to 2.
    # this trade will be successful
    trade(sell_id="2", buy_id="1", amount=2, price=100)

    # let's take a look for player 1 and player 2 currently
    after_trade_players = Player.select().where(Player.id.in_(["1", "2"]))
    for player in after_trade_players:
        print(f'id:{player.id}, coins:{player.coins}, goods:{player.goods}')


db.connect()

### recreate the player table
db.drop_tables([Player])
db.create_tables([Player])

simple_example()
trade_example()
```

Compared with using drivers directly, peewee provides an abstraction for the specific details of different databases when you create a database connection. In addition, peewee encapsulates some operations such as session management and CRUD of basic objects, which greatly simplifies the code.

The `Player` class is a mapping of a table to attributes in the application. Each attribute of `Player` corresponds to a field in the `player` table. To provide SQLAlchemy with more information, the attribute is defined as `id = Column(String(36), primary_key=True)` to indicate the field type and its additional attributes. For example, `id = Column(String(36)` ↪ `, primary_key=True)` indicates that the `id` attribute is `String` type, the corresponding field in database is `VARCHAR` type, the length is `36`, and it is a primary key.

For more information about how to use peewee, refer to peewee documentation.

mysqlclient is a popular open-source driver for Python. The following uses mysqlclient 2.1.1 as an example. Drivers for Python are more convenient to use than other languages, but they do not shield the underlying implementation and require manual management of transactions. If there are not a lot of scenarios where SQL is required, it is recommended to use ORM, which can help reduce the coupling of your program.

```
import uuid
from typing import List

import MySQLdb
from MySQLdb import Connection
from MySQLdb.cursors import Cursor

def get_connection(autocommit: bool = True) -> MySQLdb.Connection:
    return MySQLdb.connect(
        host="127.0.0.1",
        port=4000,
        user="root",
        password="",
        database="test",
        autocommit=autocommit
    )


def create_player(cursor: Cursor, player: tuple) -> None:
    cursor.execute("INSERT INTO player (id, coins, goods) VALUES (%s, %s, %s
        ↪ )", player)


def get_player(cursor: Cursor, player_id: str) -> tuple:
    cursor.execute("SELECT id, coins, goods FROM player WHERE id = %s", (
        ↪ player_id,))
    return cursor.fetchone()


def get_players_with_limit(cursor: Cursor, limit: int) -> List[tuple]:
    cursor.execute("SELECT id, coins, goods FROM player LIMIT %s", (limit,))
    return cursor.fetchall()


def random_player(amount: int) -> List[tuple]:
    players = []
    for _ in range(amount):
        players.append((uuid.uuid4(), 10000, 10000))
```

```
    return players


def bulk_create_player(cursor: Cursor, players: List[tuple]) -> None:
    cursor.executemany("INSERT INTO player (id, coins, goods) VALUES (%s, %s
        ↪ , %s)", players)


def get_count(cursor: Cursor) -> None:
    cursor.execute("SELECT count(*) FROM player")
    return cursor.fetchone()[0]


def trade_check(cursor: Cursor, sell_id: str, buy_id: str, amount: int,
    ↪ price: int) -> bool:
    get_player_with_lock_sql = "SELECT coins, goods FROM player WHERE id = %
        ↪ s FOR UPDATE"

    # sell player goods check
    cursor.execute(get_player_with_lock_sql, (sell_id,))
    _, sell_goods = cursor.fetchone()
    if sell_goods < amount:
        print(f'sell player {sell_id} goods not enough')
        return False

    # buy player coins check
    cursor.execute(get_player_with_lock_sql, (buy_id,))
    buy_coins, _ = cursor.fetchone()
    if buy_coins < price:
        print(f'buy player {buy_id} coins not enough')
        return False


def trade_update(cursor: Cursor, sell_id: str, buy_id: str, amount: int,
    ↪ price: int) -> None:
    update_player_sql = "UPDATE player set goods = goods + %s, coins = coins
        ↪  + %s WHERE id = %s"

    # deduct the goods of seller, and raise his/her the coins
    cursor.execute(update_player_sql, (-amount, price, sell_id))
    # deduct the coins of buyer, and raise his/her the goods
    cursor.execute(update_player_sql, (amount, -price, buy_id))
```

```
def trade(connection: Connection, sell_id: str, buy_id: str, amount: int,
    ↪ price: int) -> None:
    with connection.cursor() as cursor:
        if trade_check(cursor, sell_id, buy_id, amount, price) is False:
            connection.rollback()
            return

        try:
            trade_update(cursor, sell_id, buy_id, amount, price)
        except Exception as err:
            connection.rollback()
            print(f'something went wrong: {err}')
        else:
            connection.commit()
            print("trade success")


def simple_example() -> None:
    with get_connection(autocommit=True) as conn:
        with conn.cursor() as cur:
            # create a player, who has a coin and a goods.
            create_player(cur, ("test", 1, 1))

            # get this player, and print it.
            test_player = get_player(cur, "test")
            print(f'id:{test_player[0]}, coins:{test_player[1]}, goods:{
                ↪ test_player[2]}')

            # create players with bulk inserts.
            # insert 1919 players totally, with 114 players per batch.
            # each player has a random UUID
            player_list = random_player(1919)
            for idx in range(0, len(player_list), 114):
                bulk_create_player(cur, player_list[idx:idx + 114])

            # print the number of players
            count = get_count(cur)
            print(f'number of players: {count}')

            # print 3 players.
            three_players = get_players_with_limit(cur, 3)
            for player in three_players:
                print(f'id:{player[0]}, coins:{player[1]}, goods:{player[2]}'
                    ↪ )
```

```
def trade_example() -> None:
    with get_connection(autocommit=False) as conn:
        with conn.cursor() as cur:
            # create two players
            # player 1: id is "1", has only 100 coins.
            # player 2: id is "2", has 114514 coins, and 20 goods.
            create_player(cur, ("1", 100, 0))
            create_player(cur, ("2", 114514, 20))
            conn.commit()

        # player 1 wants to buy 10 goods from player 2.
        # it will cost 500 coins, but player 1 cannot afford it.
        # so this trade will fail, and nobody will lose their coins or
            ↪ goods
        trade(conn, sell_id="2", buy_id="1", amount=10, price=500)

        # then player 1 has to reduce the incoming quantity to 2.
        # this trade will be successful
        trade(conn, sell_id="2", buy_id="1", amount=2, price=100)

        # let's take a look for player 1 and player 2 currently
        with conn.cursor() as cur:
            _, player1_coin, player1_goods = get_player(cur, "1")
            print(f'id:1, coins:{player1_coin}, goods:{player1_goods}')
            _, player2_coin, player2_goods = get_player(cur, "2")
            print(f'id:2, coins:{player2_coin}, goods:{player2_goods}')


simple_example()
trade_example()
```

The driver has a lower level of encapsulation than ORM, so there are a lot of SQL statements in the program. Unlike ORM, there is no data object in drivers, so the `Player` queried by the driver is represented as a tuple.

For more information about how to use mysqlclient, refer to mysqlclient documentation.

PyMySQL is a popular open-source driver for Python. The following uses PyMySQL 1.0.2 as an example. Drivers for Python are more convenient to use than other languages, but they do not shield the underlying implementation and require manual management of transactions. If there are not a lot of scenarios where SQL is required, it is recommended to use ORM, which can help reduce the coupling of your program.

```
import uuid
from typing import List
```

```
import pymysql.cursors
from pymysql import Connection
from pymysql.cursors import DictCursor


def get_connection(autocommit: bool = False) -> Connection:
    return pymysql.connect(host='127.0.0.1',
                           port=4000,
                           user='root',
                           password='',
                           database='test',
                           cursorclass=DictCursor,
                           autocommit=autocommit)


def create_player(cursor: DictCursor, player: tuple) -> None:
    cursor.execute("INSERT INTO player (id, coins, goods) VALUES (%s, %s, %s
        ↪ )", player)


def get_player(cursor: DictCursor, player_id: str) -> dict:
    cursor.execute("SELECT id, coins, goods FROM player WHERE id = %s", (
        ↪ player_id,))
    return cursor.fetchone()


def get_players_with_limit(cursor: DictCursor, limit: int) -> tuple:
    cursor.execute("SELECT id, coins, goods FROM player LIMIT %s", (limit,))
    return cursor.fetchall()


def random_player(amount: int) -> List[tuple]:
    players = []
    for _ in range(amount):
        players.append((uuid.uuid4(), 10000, 10000))

    return players


def bulk_create_player(cursor: DictCursor, players: List[tuple]) -> None:
    cursor.executemany("INSERT INTO player (id, coins, goods) VALUES (%s, %s
        ↪ , %s)", players)


def get_count(cursor: DictCursor) -> int:
```

```
    cursor.execute("SELECT count(*) as count FROM player")
    return cursor.fetchone()['count']


def trade_check(cursor: DictCursor, sell_id: str, buy_id: str, amount: int,
    ↪  price: int) -> bool:
    get_player_with_lock_sql = "SELECT coins, goods FROM player WHERE id = %
        ↪ s FOR UPDATE"

    # sell player goods check
    cursor.execute(get_player_with_lock_sql, (sell_id,))
    seller = cursor.fetchone()
    if seller['goods'] < amount:
        print(f'sell player {sell_id} goods not enough')
        return False

    # buy player coins check
    cursor.execute(get_player_with_lock_sql, (buy_id,))
    buyer = cursor.fetchone()
    if buyer['coins'] < price:
        print(f'buy player {buy_id} coins not enough')
        return False


def trade_update(cursor: DictCursor, sell_id: str, buy_id: str, amount: int
    ↪ , price: int) -> None:
    update_player_sql = "UPDATE player set goods = goods + %s, coins = coins
        ↪  + %s WHERE id = %s"

    # deduct the goods of seller, and raise his/her the coins
    cursor.execute(update_player_sql, (-amount, price, sell_id))
    # deduct the coins of buyer, and raise his/her the goods
    cursor.execute(update_player_sql, (amount, -price, buy_id))


def trade(connection: Connection, sell_id: str, buy_id: str, amount: int,
    ↪ price: int) -> None:
    with connection.cursor() as cursor:
        if trade_check(cursor, sell_id, buy_id, amount, price) is False:
            connection.rollback()
            return

        try:
            trade_update(cursor, sell_id, buy_id, amount, price)
        except Exception as err:
```

```
            connection.rollback()
            print(f'something went wrong: {err}')
        else:
            connection.commit()
            print("trade success")


def simple_example() -> None:
    with get_connection(autocommit=True) as connection:
        with connection.cursor() as cur:
            # create a player, who has a coin and a goods.
            create_player(cur, ("test", 1, 1))

            # get this player, and print it.
            test_player = get_player(cur, "test")
            print(test_player)

            # create players with bulk inserts.
            # insert 1919 players totally, with 114 players per batch.
            # each player has a random UUID
            player_list = random_player(1919)
            for idx in range(0, len(player_list), 114):
                bulk_create_player(cur, player_list[idx:idx + 114])

            # print the number of players
            count = get_count(cur)
            print(f'number of players: {count}')

            # print 3 players.
            three_players = get_players_with_limit(cur, 3)
            for player in three_players:
                print(player)


def trade_example() -> None:
    with get_connection(autocommit=False) as connection:
        with connection.cursor() as cur:
            # create two players
            # player 1: id is "1", has only 100 coins.
            # player 2: id is "2", has 114514 coins, and 20 goods.
            create_player(cur, ("1", 100, 0))
            create_player(cur, ("2", 114514, 20))
            connection.commit()

        # player 1 wants to buy 10 goods from player 2.
```

```
        # it will cost 500 coins, but player 1 cannot afford it.
        # so this trade will fail, and nobody will lose their coins or
            ↪ goods
        trade(connection, sell_id="2", buy_id="1", amount=10, price=500)

        # then player 1 has to reduce the incoming quantity to 2.
        # this trade will be successful
        trade(connection, sell_id="2", buy_id="1", amount=2, price=100)

        # let's take a look for player 1 and player 2 currently
        with connection.cursor() as cur:
            print(get_player(cur, "1"))
            print(get_player(cur, "2"))


simple_example()
trade_example()
```

The driver has a lower level of encapsulation than ORM, so there are a lot of SQL statements in the program. Unlike ORM, there is no data object in drivers, so the `Player` queried by the driver is represented as a dictionary.

For more information about how to use PyMySQL, refer to PyMySQL documentation.

mysql-connector-python is a popular open-source driver for Python. The following uses mysql-connector-python 8.0.31 as an example. Drivers for Python are more convenient to use than other languages, but they do not shield the underlying implementation and require manual management of transactions. If there are not a lot of scenarios where SQL is required, it is recommended to use ORM, which can help reduce the coupling of your program.

```
import uuid
from typing import List

from mysql.connector import connect, MySQLConnection
from mysql.connector.cursor import MySQLCursor


def get_connection(autocommit: bool = True) -> MySQLConnection:
    connection = connect(host='127.0.0.1',
                         port=4000,
                         user='root',
                         password='',
                         database='test')
    connection.autocommit = autocommit
    return connection
```

```
def create_player(cursor: MySQLCursor, player: tuple) -> None:
    cursor.execute("INSERT INTO player (id, coins, goods) VALUES (%s, %s, %s
        ↪ )", player)


def get_player(cursor: MySQLCursor, player_id: str) -> tuple:
    cursor.execute("SELECT id, coins, goods FROM player WHERE id = %s", (
        ↪ player_id,))
    return cursor.fetchone()


def get_players_with_limit(cursor: MySQLCursor, limit: int) -> List[tuple]:
    cursor.execute("SELECT id, coins, goods FROM player LIMIT %s", (limit,))
    return cursor.fetchall()


def random_player(amount: int) -> List[tuple]:
    players = []
    for _ in range(amount):
        players.append((str(uuid.uuid4()), 10000, 10000))

    return players


def bulk_create_player(cursor: MySQLCursor, players: List[tuple]) -> None:
    cursor.executemany("INSERT INTO player (id, coins, goods) VALUES (%s, %s
        ↪ , %s)", players)


def get_count(cursor: MySQLCursor) -> int:
    cursor.execute("SELECT count(*) FROM player")
    return cursor.fetchone()[0]


def trade_check(cursor: MySQLCursor, sell_id: str, buy_id: str, amount: int
    ↪ , price: int) -> bool:
    get_player_with_lock_sql = "SELECT coins, goods FROM player WHERE id = %
        ↪ s FOR UPDATE"

    # sell player goods check
    cursor.execute(get_player_with_lock_sql, (sell_id,))
    _, sell_goods = cursor.fetchone()
    if sell_goods < amount:
        print(f'sell player {sell_id} goods not enough')
        return False
```

```
    # buy player coins check
    cursor.execute(get_player_with_lock_sql, (buy_id,))
    buy_coins, _ = cursor.fetchone()
    if buy_coins < price:
        print(f'buy player {buy_id} coins not enough')
        return False


def trade_update(cursor: MySQLCursor, sell_id: str, buy_id: str, amount:
    ↪ int, price: int) -> None:
    update_player_sql = "UPDATE player set goods = goods + %s, coins = coins
        ↪  + %s WHERE id = %s"

    # deduct the goods of seller, and raise his/her the coins
    cursor.execute(update_player_sql, (-amount, price, sell_id))
    # deduct the coins of buyer, and raise his/her the goods
    cursor.execute(update_player_sql, (amount, -price, buy_id))


def trade(connection: MySQLConnection, sell_id: str, buy_id: str, amount:
    ↪ int, price: int) -> None:
    with connection.cursor() as cursor:
        if trade_check(cursor, sell_id, buy_id, amount, price) is False:
            connection.rollback()
            return

        try:
            trade_update(cursor, sell_id, buy_id, amount, price)
        except Exception as err:
            connection.rollback()
            print(f'something went wrong: {err}')
        else:
            connection.commit()
            print("trade success")


def simple_example() -> None:
    with get_connection(autocommit=True) as connection:
        with connection.cursor() as cur:
            # create a player, who has a coin and a goods.
            create_player(cur, ("test", 1, 1))

            # get this player, and print it.
            test_player = get_player(cur, "test")
```

```
            print(f'id:{test_player[0]}, coins:{test_player[1]}, goods:{
                ↪ test_player[2]}')

            # create players with bulk inserts.
            # insert 1919 players totally, with 114 players per batch.
            # each player has a random UUID
            player_list = random_player(1919)
            for idx in range(0, len(player_list), 114):
                bulk_create_player(cur, player_list[idx:idx + 114])

            # print the number of players
            count = get_count(cur)
            print(f'number of players: {count}')

            # print 3 players.
            three_players = get_players_with_limit(cur, 3)
            for player in three_players:
                print(f'id:{player[0]}, coins:{player[1]}, goods:{player[2]}'
                    ↪ )


def trade_example() -> None:
    with get_connection(autocommit=False) as conn:
        with conn.cursor() as cur:
            # create two players
            # player 1: id is "1", has only 100 coins.
            # player 2: id is "2", has 114514 coins, and 20 goods.
            create_player(cur, ("1", 100, 0))
            create_player(cur, ("2", 114514, 20))
            conn.commit()

        # player 1 wants to buy 10 goods from player 2.
        # it will cost 500 coins, but player 1 cannot afford it.
        # so this trade will fail, and nobody will lose their coins or
            ↪ goods
        trade(conn, sell_id="2", buy_id="1", amount=10, price=500)

        # then player 1 has to reduce the incoming quantity to 2.
        # this trade will be successful
        trade(conn, sell_id="2", buy_id="1", amount=2, price=100)

        # let's take a look for player 1 and player 2 currently
        with conn.cursor() as cur:
            _, player1_coin, player1_goods = get_player(cur, "1")
            print(f'id:1, coins:{player1_coin}, goods:{player1_goods}')
```

```
            _, player2_coin, player2_goods = get_player(cur, "2")
            print(f'id:2, coins:{player2_coin}, goods:{player2_goods}')


simple_example()
trade_example()
```

The driver has a lower level of encapsulation than ORM, so there are a lot of SQL statements in the program. Unlike ORM, there is no data object in drivers, so the `Player` queried by the driver is represented as a tuple.

For more information about how to use mysql-connector-python, refer to mysql-connector-python documentation.

4.3.4.3 Step 3. Run the code

The following content introduces how to run the code step by step.

4.3.4.3.1 Step 3.1 Initialize table

Before running the code, you need to initialize the table manually. If you are using a local TiDB cluster, you can run the following command:

```
mysql --host 127.0.0.1 --port 4000 -u root < player_init.sql
```

```
mycli --host 127.0.0.1 --port 4000 -u root --no-warn < player_init.sql
```

If you are not using a local cluster, or have not installed a MySQL client, connect to your cluster using your preferred method (such as Navicat, DBeaver, or other GUI tools) and run the SQL statements in the `player_init.sql` file.

4.3.4.3.2 Step 3.2 Modify parameters for TiDB Cloud

If you are using a TiDB Serverless cluster, you need to provide your CA root path and replace `<ca_path>` in the following examples with your CA path. To get the CA root path on your system, refer to Where is the CA root path on my system?.

If you are using a TiDB Serverless cluster, modify the parameters of the `create_engine` function in `sqlalchemy_example.py`:

```
engine = create_engine('mysql://root:@127.0.0.1:4000/test')
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`

- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the `create_engine` as follows:

```
engine = create_engine('mysql://2aEp24QWEDLqRFs.root:123456@xxx.tidbcloud.
    ↪ com:4000/test', connect_args={
    "ssl_mode": "VERIFY_IDENTITY",
    "ssl": {
        "ca": "<ca_path>"
    }
})
```

If you are using a TiDB Serverless cluster, modify the parameters of the `create_engine` function in `sqlalchemy_example.py`:

```
db = connect('mysql://root:@127.0.0.1:4000/test')
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the `connect` as follows:

- When peewee uses PyMySQL as the driver:

```
db = connect('mysql://2aEp24QWEDLqRFs.root:123456@xxx.tidbcloud.com
    ↪ :4000/test',
    ssl_verify_cert=True, ssl_ca="<ca_path>")
```

- When peewee uses mysqlclient as the driver:

```
db = connect('mysql://2aEp24QWEDLqRFs.root:123456@xxx.tidbcloud.com
    ↪ :4000/test',
    ssl_mode="VERIFY_IDENTITY", ssl={"ca": "<ca_path>"})
```

Because peewee will pass parameters to the driver, you need to pay attention to the usage type of the driver when using peewee.

If you are using a TiDB Serverless cluster, change the `get_connection` function in `mysqlclient_example.py`:

```
def get_connection(autocommit: bool = True) -> MySQLdb.Connection:
    return MySQLdb.connect(
        host="127.0.0.1",
        port=4000,
        user="root",
        password="",
        database="test",
        autocommit=autocommit
    )
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the `get_connection` as follows:

```
def get_connection(autocommit: bool = True) -> MySQLdb.Connection:
    return MySQLdb.connect(
        host="xxx.tidbcloud.com",
        port=4000,
        user="2aEp24QWEDLqRFs.root",
        password="123456",
        database="test",
        autocommit=autocommit,
        ssl_mode="VERIFY_IDENTITY",
        ssl={
            "ca": "<ca_path>"
        }
    )
```

If you are using a TiDB Serverless cluster, change the `get_connection` function in `pymysql_example.py`:

```
def get_connection(autocommit: bool = False) -> Connection:
    return pymysql.connect(host='127.0.0.1',
                           port=4000,
                           user='root',
                           password='',
                           database='test',
                           cursorclass=DictCursor,
                           autocommit=autocommit)
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the `get_connection` as follows:

```
def get_connection(autocommit: bool = False) -> Connection:
    return pymysql.connect(host='xxx.tidbcloud.com',
                           port=4000,
                           user='2aEp24QWEDLqRFs.root',
                           password='123546',
                           database='test',
                           cursorclass=DictCursor,
                           autocommit=autocommit,
                           ssl_ca='<ca_path>',
                           ssl_verify_cert=True,
                           ssl_verify_identity=True)
```

If you are using a TiDB Serverless cluster, change the `get_connection` function in `mysql_connector_python_example.py`:

```
def get_connection(autocommit: bool = True) -> MySQLConnection:
    connection = connect(host='127.0.0.1',
                         port=4000,
                         user='root',
                         password='',
                         database='test')
    connection.autocommit = autocommit
    return connection
```

Suppose that the password you set is `123456`, and the connection parameters you get from the cluster details page are the following:

- Endpoint: `xxx.tidbcloud.com`
- Port: `4000`
- User: `2aEp24QWEDLqRFs.root`

In this case, you can modify the `get_connection` as follows:

```
def get_connection(autocommit: bool = True) -> MySQLConnection:
    connection = connect(
        host="xxx.tidbcloud.com",
```

```
        port=4000,
        user="2aEp24QWEDLqRFs.root",
        password="123456",
        database="test",
        autocommit=autocommit,
        ssl_ca='<ca_path>',
        ssl_verify_identity=True
    )
    connection.autocommit = autocommit
    return connection
```

4.3.4.4.3.3 Step 3.3 Run the code

Before running the code, use the following command to install dependencies:

```
pip3 install -r requirement.txt
```

If you need to run the script multiple times, follow the Table initialization section to initialize the table again before each run.

```
python3 sqlalchemy_example.py
```

```
python3 peewee_example.py
```

```
python3 mysqlclient_example.py
```

```
python3 pymysql_example.py
```

```
python3 mysql_connector_python_example.py
```

4.3.4.4 Step 4. Expected output

SQLAlchemy Expected Output

peewee Expected Output

mysqlclient Expected Output

PyMySQL Expected Output

mysql-connector-python Expected Output

4.4 Connect to TiDB

4.4.1 Choose Driver or ORM

Note:

TiDB provides the following two support levels for drivers and ORMs:

- **Full**: indicates that TiDB is compatible with most features of the tool and maintains compatibility with its newer versions. PingCAP will periodically conduct compatibility tests with the latest version of Third-party tools supported by TiDB.
- **Compatible**: indicates that because the corresponding third-party tool is adapted to MySQL and TiDB is highly compatible with the MySQL protocol, so TiDB can use most features of the tool. However, PingCAP has not completed a full test on all features of the tool, which might lead to some unexpected behaviors.

For more information, refer to Third-Party Tools Supported by TiDB.

TiDB is highly compatible with the MySQL protocol but some features are incompatible with MySQL.

For example:

- Features that are not supported by TiDB:
  - Stored procedures and functions
  - Triggers
  - `FOREIGN KEY` constraints
- Features that are different from MySQL:
  - Auto-increment ID: auto-incremental columns are globally unique in TiDB. They are incremental on a single TiDB server, but ***not*** necessarily incremental among multiple TiDB servers or allocated sequentially.

For a full list of compatibility differences, see MySQL Compatibility.

4.4.1.1 Java

This section describes how to use drivers and ORM frameworks in Java.

4.4.1.1.1 Java drivers

Support level: **Full**

You can follow the MySQL documentation to download and configure a Java JDBC driver. It is recommended to use the latest GA version of MySQL Connector/J with TiDB v6.3.0 or later.

> **Warning:**
>
> There is a bug in the MySQL Connector/J 8.0 versions before 8.0.31 (see MySQL JDBC bugs for details), which might cause threads to hang when using TiDB versions earlier than v6.3.0. To avoid this issue, do **NOT** use MySQL Connector/J 8.0.31 or an earlier version.

For an example of how to build a complete application, see Build a Simple CRUD App with TiDB and JDBC.

Support level: **Full**

TiDB-JDBC is a customized Java driver based on MySQL 8.0.29. Compiled based on MySQL official version 8.0.29, TiDB-JDBC fixes the bug of multi-parameter and multi-field EOF in the prepare mode in the original JDBC, and adds features such as automatic TiCDC snapshot maintenance and the SM3 authentication plugin.

The authentication based on SM3 is only supported in TiDB's TiDB-JDBC.

If you use Maven, add the following content to the `<dependencies></dependencies>` section in the `pom.xml` file:

```
<dependency>
  <groupId>io.github.lastincisor</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.29-tidb-1.0.0</version>
</dependency>
```

If you need to enable SM3 authentication, add the following content to the `<dependencies></dependencies>` section in the `pom.xml` file:

```
<dependency>
  <groupId>io.github.lastincisor</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.29-tidb-1.0.0</version>
</dependency>
<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcprov-jdk15on</artifactId>
```

```
    <version>1.67</version>
</dependency>
<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcpkix-jdk15on</artifactId>
    <version>1.67</version>
</dependency>
```

If you use Gradle, add the following content to `dependencies`:

```
implementation group: 'io.github.lastincisor', name: 'mysql-connector-java',
    ↪  version: '8.0.29-tidb-1.0.0'
implementation group: 'org.bouncycastle', name: 'bcprov-jdk15on', version:
    ↪ '1.67'
implementation group: 'org.bouncycastle', name: 'bcpkix-jdk15on', version:
    ↪ '1.67'
```

4.4.1.1.2 Java ORM frameworks

Note:

- Currently, Hibernate does not support nested transactions.
- Since v6.2.0, TiDB supports savepoint. To use the `Propagation.NESTED` transaction propagation option in `@Transactional`, that is, to set `@Transactional(propagation = Propagation.NESTED)`, make sure that your TiDB is v6.2.0 or later.

Support level: **Full**

To avoid manually managing complex relationships between different dependencies of an application, you can use Gradle or Maven to get all dependencies of your application, including those indirect ones. Note that only Hibernate `6.0.0.Beta2` or above supports the TiDB dialect.

If you are using **Maven**, add the following to your `<dependencies></dependencies>`:

```
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.0.0.CR2</version>
</dependency>
```

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.49</version>
</dependency>
```

If you are using **Gradle**, add the following to your `dependencies`:

```
implementation 'org.hibernate:hibernate-core:6.0.0.CR2'
implementation 'mysql:mysql-connector-java:5.1.49'
```

- For an example of using Hibernate to build a TiDB application by native Java, see Build a Simple CRUD App with TiDB and Java.
- For an example of using Spring Data JPA or Hibernate to build a TiDB application by Spring, see Build a TiDB Application using Spring Boot.

In addition, you need to specify the TiDB dialect in your Hibernate configuration file: `org.hibernate.dialect.TiDBDialect`, which is only supported by Hibernate `6.0.0.Beta2` or above. If your `Hibernate` version is earlier than `6.0.0.Beta2`, upgrade it first.

> **Note:**
>
> If you are unable to upgrade your `Hibernate` version, use the MySQL 5.7 dialect `org.hibernate.dialect.MySQL57Dialect` instead. However, this setting might cause unpredictable results and the absence of some TiDB-specific features, such as sequences.

Support level: **Full**

To avoid manually managing complex relationships between different dependencies of an application, you can use Gradle or Maven to get all dependencies of your application, including those indirect dependencies.

If you are using Maven, add the following to your `<dependencies></dependencies>`:

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.9</version>
</dependency>

<dependency>
    <groupId>mysql</groupId>
```

```
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.49</version>
</dependency>
```

If you are using Gradle, add the following to your `dependencies`:

```
implementation 'org.mybatis:mybatis:3.5.9'
implementation 'mysql:mysql-connector-java:5.1.49'
```

For an example of using MyBatis to build a TiDB application, see Build a Simple CRUD App with TiDB and Java.

4.4.1.1.3 Java client load balancing

tidb-loadbalance

Support level: **Full**

tidb-loadbalance is a load balancing component on the application side. With tidb-loadbalance, you can automatically maintain the node information of TiDB server and distribute JDBC connections on the client using the tidb-loadbalance policies. Using a direct JDBC connection between the client application and TiDB server has higher performance than using the load balancing component.

Currently, tidb-loadbalance supports the following policies: roundrobin, random, and weight.

> **Note:**
>
> tidb-loadbalance must be used with mysql-connector-j.

If you use Maven, add the following content to the element body of `<dependencies></dependencies>` in the `pom.xml` file:

```
<dependency>
  <groupId>io.github.lastincisor</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.29-tidb-1.0.0</version>
</dependency>
<dependency>
  <groupId>io.github.lastincisor</groupId>
  <artifactId>tidb-loadbalance</artifactId>
  <version>0.0.5</version>
</dependency>
```

If you use Gradle, add the following content to `dependencies`:

```
implementation group: 'io.github.lastincisor', name: 'mysql-connector-java',
    ↪ version: '8.0.29-tidb-1.0.0'
implementation group: 'io.github.lastincisor', name: 'tidb-loadbalance',
    ↪ version: '0.0.5'
```

4.4.1.2 Golang

This section describes how to use drivers and ORM frameworks in Golang.

4.4.1.2.1 Golang drivers

go-sql-driver/mysql

Support level: **Full**

To download and configure a Golang driver, refer to the go-sql-driver/mysql documentation.

For an example of how to build a complete application, see Build a Simple CRUD App with TiDB and Golang.

4.4.1.2.2 Golang ORM frameworks

GORM

Support level: **Full**

GORM is a popular ORM framework for Golang. To get all dependencies in your application, you can use the `go get` command.

```
go get -u gorm.io/gorm
go get -u gorm.io/driver/mysql
```

For an example of using GORM to build a TiDB application, see Build a Simple CRUD App with TiDB and Golang.

4.4.1.3 Python

This section describes how to use drivers and ORM frameworks in Python.

4.4.1.3.1 Python drivers

Support level: **Compatible**

You can follow the PyMySQL documentation to download and configure the driver. It is recommended to use PyMySQL 1.0.2 or later versions.

For an example of using PyMySQL to build a TiDB application, see Build a Simple CRUD App with TiDB and Python.

Support level: **Compatible**

You can follow the mysqlclient documentation to download and configure the driver. It is recommended to use mysqlclient 2.1.1 or later versions.

For an example of using mysqlclient to build a TiDB application, see Build a Simple CRUD App with TiDB and Python.

Support level: **Compatible**

You can follow the mysql-connector-python documentation to download and configure the driver. It is recommended to use Connector/Python 8.0.31 or later versions.

For an example of using mysql-connector-python to build a TiDB application, see Build a Simple CRUD App with TiDB and Python.

4.4.1.3.2 Python ORM frameworks

Support level: **Full**

SQLAlchemy is a popular ORM framework for Python. To get all dependencies in your application, you can use the `pip install SQLAlchemy==1.4.44` command. It is recommended to use SQLAlchemy 1.4.44 or later versions.

For an example of using SQLAlchemy to build a TiDB application, see Build a Simple CRUD App with TiDB and Python.

Support level: **Compatible**

peewee is a popular ORM framework for Python. To get all dependencies in your application, you can use the `pip install peewee==3.15.4` command. It is recommended to use peewee 3.15.4 or later versions.

For an example of using peewee to build a TiDB application, see Build a Simple CRUD App with TiDB and Python.

4.4.2 Connect to TiDB

TiDB is highly compatible with the MySQL protocol. For a full list of client link parameters, see MySQL Client Options.

TiDB supports the MySQL Client/Server Protocol, which allows most client drivers and ORM frameworks to connect to TiDB just as they connect to MySQL.

4.4.2.1 MySQL

You can choose to use MySQL Client or MySQL Shell based on your personal preferences.

You can connect to TiDB using MySQL Client, which can be used as a command-line tool for TiDB. To install MySQL Client, follow the instructions below for YUM based Linux distributions.

```
sudo yum install mysql
```

After the installation, you can connect to TiDB using the following command:

```
mysql --host <tidb_server_host> --port 4000 -u root -p --comments
```

You can connect to TiDB using MySQL Shell, which can be used as a command-line tool for TiDB. To install MySQL Shell, follow the instructions in the MySQL Shell documentation. After the installation, you can connect to TiDB using the following command:

```
mysqlsh --sql mysql://root@<tidb_server_host>:4000
```

4.4.2.2 JDBC

You can connect to TiDB using the JDBC driver. To do that, you need to create a `MysqlDataSource` or `MysqlConnectionPoolDataSource` object (both objects support the `DataSource` interface), and then set the connection string using the `setURL` function.

For example:

```
MysqlDataSource mysqlDataSource = new MysqlDataSource();
mysqlDataSource.setURL("jdbc:mysql://{host}:{port}/{database}?user={username
    ↪ }&password={password}");
```

For more information on JDBC connections, see the JDBC documentation

4.4.2.2.1 Connection parameters

Parameter name	Description
{username}	A SQL user to connect to the TiDB cluster
{password}	The password of the SQL user
{host}	Host of a TiDB node
{port}	Port that the TiDB node is listening on
{database}	Name of an existing database

For more information about TiDB SQL users, see TiDB User Account Management.

4.4.2.3 Hibernate

You can connect to TiDB using the Hibernate ORM. To do that, you need to set `hibernate.connection.url` in the Hibernate configuration file to a legal TiDB connection string.

For example, if you use a `hibernate.cfg.xml` configuration file, set `hibernate.connection.url` as follows:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
```

```
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.
            ↪ Driver</property>
        <property name="hibernate.dialect">org.hibernate.dialect.TiDBDialect<
            ↪ /property>
        <property name="hibernate.connection.url">jdbc:mysql://{host}:{port
            ↪ }/{database}?user={user}&amp;password={password}</property>
    </session-factory>
</hibernate-configuration>
```

After the configuration is done, you can use the following command to read the configuration file and get the `SessionFactory` object:

```
SessionFactory sessionFactory = new Configuration().configure("hibernate.
    ↪ cfg.xml").buildSessionFactory();
```

Note the following:

- Because the `hibernate.cfg.xml` configuration file is in the XML format and `&` is a special character in XML, you need to change `&` to `&` when configuring the file. For example, you need to change the connection string `hibernate.connection.url` from `jdbc:mysql://{host}:{port}/{database}?user={user}&password={password}` to `jdbc:mysql://{host}:{ port}/{database}?user={user}&password={password}`.
- It is recommended that you use the `TiDB` dialect by setting `hibernate.dialect` to `org.hibernate.dialect.TiDBDialect`.
- Hibernate supports TiDB dialects starting from `6.0.0.Beta2`, so it is recommended that you use Hibernate `6.0.0.Beta2` or a later version to connect to TiDB.

For more information about Hibernate connection parameters, see Hibernate documentation.

4.4.2.3.1 Connection parameters

Parameter name	Description
`{username}`	A SQL user to connect to the TiDB cluster
`{password}`	The password of the SQL user
`{host}`	Host of a TiDB node
`{port}`	Port that the TiDB node is listening on
`{database}`	Name of an existing database

For more information about TiDB SQL users, see TiDB User Account Management.

For more information about TiDB SQL users, see TiDB User Account Management.

4.4.3 Connection Pools and Connection Parameters

This document describes how to configure connection pools and connection parameters when you use a driver or ORM framework to connect to TiDB.

If you are interested in more tips about Java application development, see Best Practices for Developing Java Applications with TiDB

4.4.3.1 Connection pool

Building TiDB (MySQL) connections is relatively expensive (for OLTP scenarios at least). Because in addition to building a TCP connection, connection authentication is also required. Therefore, the client usually saves the TiDB (MySQL) connections to the connection pool for reuse.

Java has many connection pool implementations such as HikariCP, tomcat-jdbc, druid, c3p0, and dbcp. TiDB does not limit which connection pool you use, so you can choose whichever you like for your application.

4.4.3.1.1 Configure the number of connections

It is a common practice that the connection pool size is well adjusted according to the application's own needs. Take HikariCP as an example:

- **maximumPoolSize**: The maximum number of connections in the connection pool. If this value is too large, TiDB consumes resources to maintain useless connections. If this value is too small, the application gets slow connections. Therefore, you need to configure this value according to your application characteristics. For details, see About Pool Sizing.
- **minimumIdle**: The minimum number of idle connections in the connection pool. It is mainly used to reserve some connections to respond to sudden requests when the application is idle. You also need to configure it according to your application characteristics.

The application needs to return the connection after finishing using it. It is recommended that the application uses the corresponding connection pool monitoring (such as **metricRegistry**) to locate connection pool issues in time.

4.4.3.1.2 Probe configuration

The connection pool maintains persistent connections from clients to TiDB as follows:

- Before v5.4, TiDB does not proactively close client connections by default (unless an error is reported).
- Starting from v5.4, TiDB automatically closes client connections after `28800` seconds (this is, `8` hours) of inactivity by default. You can control this timeout setting using the TiDB and MySQL compatible `wait_timeout` variable. For more information, see JDBC Query Timeout.

Moreover, there might be network proxies such as LVS or HAProxy between clients and TiDB. These proxies typically proactively clean up connections after a specific idle period (determined by the proxy's idle configuration). In addition to monitoring the proxy's idle configuration, connection pools also need to maintain or probe connections for keep-alive.

If you often see the following error in your Java application:

```
The last packet sent successfully to the server was 3600000 milliseconds ago
    ↪ . The driver has not received any packets from the server. com.mysql.
    ↪ jdbc.exceptions.jdbc4.CommunicationsException: Communications link
    ↪ failure
```

If `n` in `n milliseconds ago` is `0` or a very small value, it is usually because the executed SQL operation causes TiDB to exit abnormally. To find the cause, it is recommended to check the TiDB stderr log.

If `n` is a very large value (such as `3600000` in the above example), it is likely that this connection was idle for a long time and then closed by the proxy. The usual solution is to increase the value of the proxy's idle configuration and allow the connection pool to:

- Check whether the connection is available before using the connection every time.
- Regularly check whether the connection is available using a separate thread.
- Send a test query regularly to keep alive connections.

Different connection pool implementations might support one or more of the above methods. You can check your connection pool documentation to find the corresponding configuration.

4.4.3.1.3 Formulas based on experience

According to the About Pool Sizing article of HikariCP, if you have no idea about how to set a proper size for the database connection pool, you can get started with a formula based on experience. Then, based on the performance result of the pool size calculated from the formula, you can further adjust the size to achieve the best performance.

The formula based on experience is as follows:

```
connections = ((core_count * 2) + effective_spindle_count)
```

The description of each parameter in the formula is as follows:

- **connections**: the size of connections obtained.
- **core_count**: the number of CPU cores.
- **effective_spindle_count**: the number of hard drives (not SSD). Because each spinning hard disk can be called a spindle. For example, if you are using a server with a RAID of 16 disks, the **effective_spindle_count** should be 16. Because **HDD** usually can handle only one request at a time, the formula here is actually measuring how many concurrent I/O requests your server can manage.

In particular, pay attention to the following note below the formula.

```
A formula which has held up pretty well across a lot of
    ↪ benchmarks for years is
that for optimal throughput the number of active connections
    ↪ should be somewhere
near ((core_count * 2) + effective_spindle_count). Core count
    ↪ should not include
HT threads, even if hyperthreading is enabled. Effective
    ↪ spindle count is zero if
the active data set is fully cached, and approaches the actual
    ↪ number of spindles
as the cache hit rate falls. ... There hasn't been any analysis
    ↪  so far regarding
how well the formula works with SSDs.
```

This note indicates that:

- **core_count** is the number of physical cores, regardless of whether you enable Hyper-Threading or not.
- When data is fully cached, you need to set **effective_spindle_count** to `0`. As the hit rate of cache decreases, the count is closer to the actual number of `HDD`.
- **Whether the formula works for *SSD* is not tested and unknown.**

When using SSDs, it is recommended that you use the following formula based on experience instead:

```
connections = (number of cores * 4)
```

Therefore, you can set the maximum connection size of the initial connection pool to `cores * 4` in the case of SSDs and further adjust the size to tune the performance.

4.4.3.1.4 Tuning direction

As you can see, a size calculated from the formulas based on experience is just a recommended base value. To get the optimal size on a specific machine, you need to try other values around the base value and test the performance.

Here are some basic rules to help you get the optimal size:

- If your network or storage latency is high, increase your maximum number of connections to reduce the latency waiting time. Once a thread is blocked by latency, other threads can take over and continue processing.
- If you have multiple services deployed on your server and each service has a separate connection pool, consider the sum of the maximum number of connections to all connection pools.

4.4.3.2 Connection parameters

Java applications can be encapsulated with various frameworks. In most of the frameworks, JDBC API is called on the bottommost level to interact with the database server. For JDBC, it is recommended that you focus on the following things:

- JDBC API usage choice
- API Implementer's parameter configuration

4.4.3.2.1 JDBC API

For JDBC API usage, see JDBC official tutorial. This section covers the usage of several important APIs.

Use Prepare API

For OLTP (Online Transactional Processing) scenarios, the SQL statements sent by the program to the database are several types that can be exhausted after removing parameter changes. Therefore, it is recommended to use Prepared Statements instead of regular execution from a text file and reuse Prepared Statements to execute directly. This avoids the overhead of repeatedly parsing and generating SQL execution plans in TiDB.

At present, most upper-level frameworks call the Prepare API for SQL execution. If you use the JDBC API directly for development, pay attention to choosing the Prepare API.

In addition, with the default implementation of MySQL Connector/J, only client-side statements are preprocessed, and the statements are sent to the server in a text file after `?` ↪ is replaced on the client. Therefore, in addition to using the Prepare API, you also need to configure `useServerPrepStmts = true` in JDBC connection parameters before you perform statement preprocessing on the TiDB server. For detailed parameter configuration, see MySQL JDBC parameters.

Use Batch API

For batch inserts, you can use the addBatch/executeBatch API. The `addBatch()` ↪ method is used to cache multiple SQL statements first on the client, and then send them to the database server together when calling the `executeBatch` method.

> **Note:**
>
> In the default MySQL Connector/J implementation, the sending time of the SQL statements that are added to batch with `addBatch()` is delayed to the time when `executeBatch()` is called, but the statements will still be sent one by one during the actual network transfer. Therefore, this method usually does not reduce the amount of communication overhead.
>
> If you want to batch network transfer, you need to configure `rewriteBatchedStatements = true` in the JDBC connection parameters. For the detailed parameter configuration, see Batch-related parameters.

Use `StreamingResult` to get the execution result

In most scenarios, to improve execution efficiency, JDBC obtains query results in advance and saves them in client memory by default. But when the query returns a super large result set, the client often wants the database server to reduce the number of records returned at a time and waits until the client's memory is ready and it requests for the next batch.

Usually, there are two kinds of processing methods in JDBC:

- Set **FetchSize** to `Integer.MIN_VALUE` to ensure that the client does not cache. The client will read the execution result from the network connection through `StreamingResult`.

  When the client uses the streaming read method, it needs to finish reading or close `resultset` before continuing to use the statement to make a query. Otherwise, the error `No statements may be issued when any streaming result sets are open and in use on a given connection. Ensure that you have called .close()on any active streaming result sets before attempting more queries.` is returned.

  To avoid such an error in queries before the client finishes reading or closes `resultset` , you can add the `clobberStreamingResults=true` parameter in the URL. Then, `resultset` is automatically closed but the result set to be read in the previous streaming query is lost.

- To use Cursor Fetch, first set `FetchSize` as a positive integer and configure `useCursorFetch=true` in the JDBC URL.

TiDB supports both methods, but it is preferred that you use the first method, because it is a simpler implementation and has a better execution efficiency.

4.4.3.2.2 MySQL JDBC parameters

JDBC usually provides implementation-related configurations in the form of JDBC URL parameters. This section introduces MySQL Connector/J's parameter configurations (If you use MariaDB, see MariaDB's parameter configurations). Because this document cannot cover all configuration items, it mainly focuses on several parameters that might affect performance.

Prepare-related parameters

This section introduces parameters related to `Prepare`.

- **useServerPrepStmts**

  useServerPrepStmts is set to `false` by default, that is, even if you use the Prepare API, the "prepare" operation will be done only on the client. To avoid the parsing overhead of the server, if the same SQL statement uses the Prepare API multiple times, it is recommended to set this configuration to `true`.

  To verify that this setting already takes effect, you can do:

  - Go to TiDB monitoring dashboard and view the request command type through **Query Summary** > **CPS By Instance**.
  - If `COM_QUERY` is replaced by `COM_STMT_EXECUTE` or `COM_STMT_PREPARE` in the request, it means this setting already takes effect.

- **cachePrepStmts**

  Although `useServerPrepStmts=true` allows the server to execute Prepared Statements, by default, the client closes the Prepared Statements after each execution and does not reuse them. This means that the "prepare" operation is not even as efficient as text file execution. To solve this, it is recommended that after setting `useServerPrepStmts=true`, you should also configure `cachePrepStmts=true`. This allows the client to cache Prepared Statements.

  To verify that this setting already takes effect, you can do:

  - Go to TiDB monitoring dashboard and view the request command type through **Query Summary** > **CPS By Instance**.
  - If the number of `COM_STMT_EXECUTE` in the request is far more than the number of `COM_STMT_PREPARE`, it means this setting already takes effect.

  In addition, configuring `useConfigs=maxPerformance` will configure multiple parameters at the same time, including `cachePrepStmts=true`.

- **prepStmtCacheSqlLimit**

  After configuring `cachePrepStmts`, also pay attention to the `prepStmtCacheSqlLimit` ↪ configuration (the default value is `256`). This configuration controls the maximum length of the Prepared Statements cached on the client.

The Prepared Statements that exceed this maximum length will not be cached, so they cannot be reused. In this case, you may consider increasing the value of this configuration depending on the actual SQL length of the application.

You need to check whether this setting is too small if you:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > CPS By Instance**.
- And find that `cachePrepStmts=true` has been configured, but `COM_STMT_PREPARE` is still mostly equal to `COM_STMT_EXECUTE` and `COM_STMT_CLOSE` exists.

- **prepStmtCacheSize**

  prepStmtCacheSize controls the number of cached Prepared Statements (the default value is `25`). If your application requires "preparing" many types of SQL statements and wants to reuse Prepared Statements, you can increase this value.

  To verify that this setting already takes effect, you can do:

  - Go to TiDB monitoring dashboard and view the request command type through **Query Summary > CPS By Instance**.
  - If the number of `COM_STMT_EXECUTE` in the request is far more than the number of `COM_STMT_PREPARE`, it means this setting already takes effect.

Batch-related parameters

While processing batch writes, it is recommended to configure `rewriteBatchedStatements=true`. After using `addBatch()` or `executeBatch()`, JDBC still sends SQL one by one by default, for example:

```
pstmt = prepare("INSERT INTO `t` (a) values(?)");
pstmt.setInt(1, 10);
pstmt.addBatch();
pstmt.setInt(1, 11);
pstmt.addBatch();
pstmt.setInt(1, 12);
pstmt.executeBatch();
```

Although `Batch` methods are used, the SQL statements sent to TiDB are still individual `INSERT` statements:

```
INSERT INTO `t` (`a`) VALUES(10);
INSERT INTO `t` (`a`) VALUES(11);
INSERT INTO `t` (`a`) VALUES(12);
```

But if you set `rewriteBatchedStatements=true`, the SQL statements sent to TiDB will be a single `INSERT` statement:

```
INSERT INTO `t` (`a`) values(10),(11),(12);
```

Note that the rewrite of the `INSERT` statements is to concatenate the values after multiple "values" keywords into a whole SQL statement. If the `INSERT` statements have other differences, they cannot be rewritten, for example:

```
INSERT INTO `t` (`a`) VALUES (10) ON DUPLICATE KEY UPDATE `a` = 10;
INSERT INTO `t` (`a`) VALUES (11) ON DUPLICATE KEY UPDATE `a` = 11;
INSERT INTO `t` (`a`) VALUES (12) ON DUPLICATE KEY UPDATE `a` = 12;
```

The above `INSERT` statements cannot be rewritten into one statement. But if you change the three statements into the following ones:

```
INSERT INTO `t` (`a`) VALUES (10) ON DUPLICATE KEY UPDATE `a` = VALUES(`a`)
    ↪ ;
INSERT INTO `t` (`a`) VALUES (11) ON DUPLICATE KEY UPDATE `a` = VALUES(`a`)
    ↪ ;
INSERT INTO `t` (`a`) VALUES (12) ON DUPLICATE KEY UPDATE `a` = VALUES(`a`)
    ↪ ;
```

Then they meet the rewrite requirement. The above `INSERT` statements will be rewritten into the following one statement:

```
INSERT INTO `t` (`a`) VALUES (10), (11), (12) ON DUPLICATE KEY UPDATE a =
    ↪ VALUES(`a`);
```

If there are three or more updates during the batch update, the SQL statements will be rewritten and sent as multiple queries. This effectively reduces the client-to-server request overhead, but the side effect is that a larger SQL statement is generated. For example:

```
UPDATE `t` SET `a` = 10 WHERE `id` = 1; UPDATE `t` SET `a` = 11 WHERE `id`
    ↪ = 2; UPDATE `t` SET `a` = 12 WHERE `id` = 3;
```

In addition, because of a client bug, if you want to configure `rewriteBatchedStatements ↪ =true` and `useServerPrepStmts=true` during batch update, it is recommended that you also configure the `allowMultiQueries=true` parameter to avoid this bug.

Integrate parameters

Through monitoring, you might notice that although the application only performs `INSERT` operations to the TiDB cluster, there are a lot of redundant `SELECT` statements. Usually this happens because JDBC sends some SQL statements to query the settings, for example, `select @@session.transaction_read_only`. These SQL statements are useless for TiDB, so it is recommended that you configure `useConfigs=maxPerformance` to avoid extra overhead.

`useConfigs=maxPerformance` includes a group of configurations. To get the detailed configurations in MySQL Connector/J 8.0 and those in MySQL Connector/J 5.1, see mysql-connector-j 8.0 and mysql-connector-j 5.1 respectively.

After it is configured, you can check the monitoring to see a decreased number of `SELECT` statements.

Timeout-related parameters

TiDB provides two MySQL-compatible parameters to control the timeout: `wait_timeout` and `max_execution_time`. These two parameters respectively control the connection idle timeout with the Java application and the timeout of the SQL execution in the connection; that is to say, these parameters control the longest idle time and the longest busy time for the connection between TiDB and the Java application. Since TiDB v5.4, the default value of `wait_timeout` is `28800` seconds, which is 8 hours. For TiDB versions earlier than v5.4, the default value is `0`, which means the timeout is unlimited. The default value of `max_execution_time` is `0`, which means the maximum execution time of a SQL statement is unlimited.

However, in an actual production environment, idle connections and SQL statements with excessively long execution time negatively affect databases and applications. To avoid idle connections and SQL statements that are executed for too long, you can configure these two parameters in your application's connection string. For example, set `sessionVariables=wait_timeout=3600` (1 hour) and `sessionVariables=max_execution_time=300000` (5 minutes).

4.5 Design Database Schema

4.5.1 TiDB Database Schema Design Overview

This document provides the basics of TiDB database schema design, including the objects in TiDB, access control, database schema changes, and object limitations.

In the subsequent documents, Bookshop will be taken as an example to show you how to design a database and perform data read and write operations in a database.

4.5.1.1 Objects in TiDB

To distinguish some general terms, here is a brief agreement on the terms used in TiDB:

- To avoid confusion with the generic term database, **database** in this document refers to a logical object, **TiDB** refers to TiDB itself, and **cluster** refers to a deployed instance of TiDB.
- TiDB uses MySQL-compatible syntax, in which **schema** means the generic term schema instead of a logical object in a database. For more information, see MySQL documentation. Make sure that you note this difference if you are migrating from databases that have schemas as logical objects (for example, PostgreSQL, Oracle, and Microsoft SQL Server).

4.5.1.1.1 Database

A database in TiDB is a collection of objects such as tables and indexes.

TiDB comes with a default database named `test`. However, it is recommended that you create your own database instead of using the `test` database.

4.5.1.1.2 Table

A table is a collection of related data in a database.

Each table consists of **rows** and **columns**. Each value in a row belongs to a specific **column**. Each column allows only a single data type. To further qualify columns, you can add some constraints. To accelerate calculations, you can add generated columns (experimental feature).

4.5.1.1.3 Index

An index is a copy of selected columns in a table. You can create an index using one or more columns of a table. With indexes, TiDB can quickly locate data without having to search every row in a table every time, which greatly improves your query performance.

There are two common types of indexes:

- **Primary Key**: indexes on the primary key column.
- **Secondary Index**: indexes on non-primary key columns.

> **Note:**
>
> In TiDB, the default definition of **Primary Key** is different from that in InnoDB (a common storage engine of MySQL).
>
> - In InnoDB, the definition of **Primary Key** is unique, not null, and a **clustered index**.
> - In TiDB, the definition of **Primary Key** is unique and not null. But the primary key is not guaranteed to be a **clustered index**. To specify whether the primary key is a clustered index, you can add non-reserved keywords `CLUSTERED` or `NONCLUSTERED` after `PRIMARY KEY` in a `CREATE TABLE` statement. If a statement does not explicitly specify these keywords, the default behavior is controlled by the system variable `@@global.tidb_enable_clustered_index`. For more information, see Clustered Indexes.

Specialized indexes

To improve query performance of various user scenarios, TiDB provides you with some specialized types of indexes. For details of each type, see the following links:

- Expression indexes (Experimental)
- Columnar storage (TiFlash)
- RocksDB engine

- Titan plugin

- Invisible indexes
- Composite `PRIMARY KEY`
- Unique indexes
- Clustered indexes on integer `PRIMARY KEY`
- Clustered indexes on composite or non-integer key

4.5.1.1.4 Other supported logical objects

TiDB supports the following logical objects at the same level as **table**:

- View: a view acts as a virtual table, whose schema is defined by the `SELECT` statement that creates the view.
- Sequence: a sequence generates and stores sequential data.
- Temporary table: a table whose data is not persistent.

4.5.1.2 Access Control

TiDB supports both user-based and role-based access control. To allow users to view, modify, or delete data objects and data schemas, you can either grant privileges to users directly or grant privileges to users through roles.

4.5.1.3 Database schema changes

As a best practice, it is recommended that you use a MySQL client or a GUI client instead of a driver or ORM to execute database schema changes.

4.5.1.4 Object limitations

This section lists the object limitations on identifier length, a single table, and string types. For more information, see TiDB Limitations.

4.5.1.4.1 Limitations on identifier length

Identifier type	Maximum length (number of characters allowed)
Database	64
Table	64
Column	64
Index	64
View	64
Sequence	64

4.5.1.4.2 Limitations on a single table

Type	Upper limit (default value)
Columns	Defaults to 1017 and can be adjusted up to 4096
Indexes	Defaults to 64 and can be adjusted up to 512
Partitions	8192
Size of a single line	6 MB by default.
Size of a single column in a line	6 MB

You can adjust the size limit of a single line via the **txn-entry-size-limit** configuration item.

4.5.1.4.3 Limitations on string types

Type	Upper limit
CHAR	256 characters
BINARY	256 characters
VARBINARY	65535 characters
VARCHAR	16383 characters
TEXT	6 MB
BLOB	6 MB

4.5.1.4.4 Number of rows

TiDB supports an **unlimited** number of rows by adding nodes to the cluster. For the relevant principles, see TiDB Best Practices.

4.5.2 Create a Database

This document describes how to create a database using SQL and various programming languages and lists the rules of database creation. In this document, the Bookshop application is taken as an example to walk you through the steps of database creation.

4.5.2.1 Before you start

Before creating a database, do the following:

- Build a TiDB Serverless Cluster.
- Read Schema Design Overview.

4.5.2.2 What is database

Database objects in TiDB contain **tables**, **views**, **sequences**, and other objects.

4.5.2.3 Create databases

To create a database, you can use the `CREATE DATABASE` statement.

For example, to create a database named `bookshop` if it does not exist, use the following statement:

```
CREATE DATABASE IF NOT EXISTS `bookshop`;
```

For more information and examples of the `CREATE DATABASE` statement, see the CREATE DATABASE document.

To execute the library build statement as the `root` user, run the following command:

```
mysql
    -u root \
    -h {host} \
    -P {port} \
    -p {password} \
    -e "CREATE DATABASE IF NOT EXISTS bookshop;"
```

4.5.2.4 View databases

To view the databases in a cluster, use the `SHOW DATABASES` statement.

For example:

```
mysql
    -u root \
    -h {host} \
    -P {port} \
    -p {password} \
    -e "SHOW DATABASES;"
```

The following is an example output:

```
+--------------------+
| Database           |
+--------------------+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| bookshop           |
| mysql              |
| test               |
+--------------------+
```

4.5.2.5 Rules in database creation

- Follow the Database Naming Conventions and name your database meaningfully.
- TiDB comes with a default database named `test`. However, it is not recommended that you use it in a production environment if you do not have to. You can create your own database using the `CREATE DATABASE` statement and change the current database using the `USE {databasename};` statement in a SQL session.
- Use the `root` user to create objects such as database, roles, and users. Grant only the necessary privileges to roles and users.
- As a best practice, it is recommended that you use a **MySQL command-line client** or a **MySQL GUI client** instead of a driver or ORM to execute database schema changes.

4.5.2.6 Next step

After creating a database, you can add **tables** to it. For more information, see Create a Table.

4.5.3 Create a Table

This document introduces how to create tables using the SQL statement and the related best practices. An example of the TiDB-based Bookshop application) is provided to illustrate the best practices.

4.5.3.1 Before you start

Before reading this document, make sure that the following tasks are completed:

- Build a TiDB Serverless Cluster.
- Read Schema Design Overview.
- Create a Database.

4.5.3.2 What is a table

A table is a logical object in TiDB cluster that is subordinate to the database. It is used to store the data sent from SQL statements. Tables save data records in the form of rows and columns. A table has at least one column. If you have defined `n` columns, each row of data has exactly the same fields as the `n` columns.

4.5.3.3 Name a table

The first step for creating a table is to give your table a name. Do not use meaningless names that will cause great distress to yourself or your colleagues in the future. It is recommended that you follow your company or organization's table naming convention.

The `CREATE TABLE` statement usually takes the following form:

```
CREATE TABLE {table_name} ( {elements} );
```

Parameter description

- `{table_name}`: The name of the table to be created.
- `{elements}`: A comma-separated list of table elements, such as column definitions and primary key definitions.

Suppose you need to create a table to store the user information in the `bookshop` database.

Note that you cannot execute the following SQL statement yet because not a single column has been added.

```
CREATE TABLE `bookshop`.`users` (
);
```

4.5.3.4 Define columns

A **column** is subordinate to a table. Each table has at least one column. Columns provide a structure to a table by dividing the values in each row into small cells of a single data type.

Column definitions typically take the following form.

```
{column_name} {data_type} {column_qualification}
```

Parameter description

- `{column_name}`: The column name.
- `{data_type}`: The column data type.
- `{column_qualification}`: Column qualifications, such as **column-level constraints** or generated column (experimental feature) clauses.

You can add some columns to the `users` table, such as the unique identifier `id`, `balance` and `nickname`.

```
CREATE TABLE `bookshop`.`users` (
  `id` bigint,
  `nickname` varchar(100),
  `balance` decimal(15,2)
);
```

In the above statement, a field is defined with the name `id` and the type bigint. This is used to represent a unique user identifier. This means that all user identifiers should be of the `bigint` type.

Then, a field named `nickname` is defined, which is the varchar type, with a length limit of 100 characters. This means that the `nicknames` of the users use the `varchar` type and are not longer than 100 characters.

Finally, a field named `balance` is added, which is the decimal type, with a **precision** of `15` and a **scale** of `2`. **Precision** represents the total number of digits in the field, and **scale** represents the number of decimal places. For example, `decimal(5,2)` means a precision of `5` and a scale of `2`, with the range from `-999.99` to `999.99`. `decimal(6,1)` means a precision of `6` and a scale of `1`, with the range from `-99999.9` to `99999.9`. **decimal** is a fixed-point types, which can be used to store numbers accurately. In scenarios where accurate numbers are needed (for example, user property-related), make sure that you use the **decimal** type.

TiDB supports many other column data types, including the integer types, floating-point types, fixed-point types, date and time types, and the enum type. You can refer to the supported column data types and use the **data types** that match the data you want to save in the database.

To make it a bit more complex, you can define a `books` table which will be the core of the `bookshop` data. The `books` table contains fields for the book's ids, titles, types (for example, magazine, novel, life, arts), stock, prices, and publication dates.

```
CREATE TABLE `bookshop`.`books` (
  `id` bigint NOT NULL,
  `title` varchar(100),
  `type` enum('Magazine', 'Novel', 'Life', 'Arts', 'Comics', 'Education &
    ↪ Reference', 'Humanities & Social Sciences', 'Science & Technology',
    ↪ 'Kids', 'Sports'),
  `published_at` datetime,
  `stock` int,
  `price` decimal(15,2)
);
```

This table contains more data types than the `users` table.

- int: It is recommended to use the type of right size to avoid using too much disk or even affecting performance (too large a type range) or data overflow (too small a data type range).
- datetime: The **datetime** type can be used to store time values.
- enum: The enum type can be used to store a limited selection of values.

4.5.3.5 Select primary key

A primary key is a column or a set of columns in a table whose values uniquely identify a row in the table.

Note:

The default definition of **primary key** in TiDB is different from that in InnoDB(the common storage engine of MySQL).

- In **InnoDB**: A **primary key** is unique, not null, and **index clustered**.
- In TiDB: A **primary key** is unique and is not null. But the primary key is not guaranteed to be a **clustered index**. Instead, another set of keywords `CLUSTERED` / `NONCLUSTERED` additionally controls whether the **primary key** is a **clustered index**. If the keyword is not specified, it is controlled by the system variable `@@global.tidb_enable_clustered_index`, as described in clustered indexes.

The **primary key** is defined in the `CREATE TABLE` statement. The primary key constraint requires that all constrained columns contain only non-NULL values.

A table can be created without a **primary key** or with a non-integer **primary key**. In this case, TiDB creates a `_tidb_rowid` as an **implicit primary key**. The implicit primary key `_tidb_rowid`, because of its monotonically increasing nature, might cause write hotspots in write-intensive scenarios. Therefore, if your application is write-intensive, consider sharding data using the `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` parameters. However, this might lead to read amplification, so you need to make your own trade-off.

When the **primary key** of a table is an integer type and `AUTO_INCREMENT` is used, hotspots cannot be avoided by using `SHARD_ROW_ID_BITS`. If you need to avoid hotspots and do not need a continuous and incremental primary key, you can use `AUTO_RANDOM` instead of `AUTO_INCREMENT` to eliminate row ID continuity.

For more information on how to handle hotspot issues, refer to Troubleshoot Hotspot Issues.

Following the guidelines for selecting primary key, the following example shows how an `AUTO_RANDOM` primary key is defined in the `users` table.

```
CREATE TABLE `bookshop`.`users` (
  `id` bigint AUTO_RANDOM,
  `balance` decimal(15,2),
  `nickname` varchar(100),
  PRIMARY KEY (`id`)
);
```

4.5.3.6 Clustered or not

TiDB supports the clustered index feature since v5.0. This feature controls how data is stored in tables containing primary keys. It provides TiDB the ability to organize tables in a way that can improve the performance of certain queries.

The term clustered in this context refers to the organization of how data is stored and not a group of database servers working together. Some database management systems refer to clustered indexes as index-organized tables (IOT).

Currently, tables ***containing primary*** keys in TiDB are divided into the following two categories:

- `NONCLUSTERED`: The primary key of the table is non-clustered index. In tables with non-clustered indexes, the keys for row data consist of internal `_tidb_rowid` implicitly assigned by TiDB. Because primary keys are essentially unique indexes, tables with non-clustered indexes need at least two key-value pairs to store a row, which are:
    - `_tidb_rowid` (key) - row data (value)
    - Primary key data (key) - `_tidb_rowid` (value)
- `CLUSTERED`: The primary key of the table is clustered index. In tables with clustered indexes, the keys for row data consist of primary key data given by the user. Therefore, tables with clustered indexes need only one key-value pair to store a row, which is:
    - Primary key data (key) - row data (value)

As described in select primary key, **clustered indexes** are controlled in TiDB using the keywords `CLUSTERED` and `NONCLUSTERED`.

> **Note:**
>
> TiDB supports clustering only by a table's `PRIMARY KEY`. With clustered indexes enabled, the terms *the* `PRIMARY KEY` and *the clustered index* might be used interchangeably. `PRIMARY KEY` refers to the constraint (a logical property), and clustered index describes the physical implementation of how the data is stored.

Following the guidelines for selecting clustered index, the following example creates a table with an association between `books` and `users`, which represents the `ratings` of a `book` by `users`. The example creates the table and constructs a composite primary key using `book_id` and `user_id`, and creates a **clustered index** on that **primary key**.

```
CREATE TABLE `bookshop`.`ratings` (
  `book_id` bigint,
  `user_id` bigint,
  `score` tinyint,
```

```
  `rated_at` datetime,
  PRIMARY KEY (`book_id`,`user_id`) CLUSTERED
);
```

4.5.3.7 Add column constraints

In addition to primary key constraints, TiDB also supports other **column constraints** such as NOT NULL constraint, UNIQUE KEY constraint, and `DEFAULT`. For complete constraints, refer to the TiDB constraints document.

4.5.3.7.1 Set default value

To set a default value on a column, use the `DEFAULT` constraint. The default value allows you to insert data without specifying a value for each column.

You can use `DEFAULT` together with supported SQL functions to move the calculation of defaults out of the application layer, thus saving resources of the application layer. The resources consumed by the calculation do not disappear and are moved to the TiDB cluster. Commonly, you can insert data with the default time. The following exemplifies setting the default value in the `ratings` table:

```
CREATE TABLE `bookshop`.`ratings` (
  `book_id` bigint,
  `user_id` bigint,
  `score` tinyint,
  `rated_at` datetime DEFAULT NOW(),
  PRIMARY KEY (`book_id`,`user_id`) CLUSTERED
);
```

In addition, if the current time is also filled in by default when the data is being updated, the following statements can be used (but only the current time related statements can be filled in after `ON UPDATE`, and more options are supported after `DEFAULT`):

```
CREATE TABLE `bookshop`.`ratings` (
  `book_id` bigint,
  `user_id` bigint,
  `score` tinyint,
  `rated_at` datetime DEFAULT NOW() ON UPDATE NOW(),
  PRIMARY KEY (`book_id`,`user_id`) CLUSTERED
);
```

4.5.3.7.2 Prevent duplicate values

If you need to prevent duplicate values in a column, you can use the `UNIQUE` constraint.

For example, to make sure that users' nicknames are unique, you can rewrite the table creation SQL statement for the `users` table like this:

```
CREATE TABLE `bookshop`.`users` (
  `id` bigint AUTO_RANDOM,
  `balance` decimal(15,2),
  `nickname` varchar(100) UNIQUE,
  PRIMARY KEY (`id`)
);
```

If you try to insert the same `nickname` in the `users` table, an error is returned.

4.5.3.7.3 Prevent null values

If you need to prevent null values in a column, you can use the `NOT NULL` constraint.

Take user nicknames as an example. To ensure that a nickname is not only unique but is also not null, you can rewrite the SQL statement for creating the `users` table as follows:

```
CREATE TABLE `bookshop`.`users` (
  `id` bigint AUTO_RANDOM,
  `balance` decimal(15,2),
  `nickname` varchar(100) UNIQUE NOT NULL,
  PRIMARY KEY (`id`)
);
```

4.5.3.8 Use HTAP capabilities

> **Note:**
>
> The steps provided in this guide is ***ONLY*** for quick start in the test environment. For production environments, refer to explore HTAP.

Suppose that you want to perform OLAP analysis on the `ratings` table using the `bookshop` application, for example, to query **whether the rating of a book has a significant correlation with the time of the rating**, which is to analyze whether the user's rating of the book is objective or not. Then you need to query the `score` and `rated_at` fields of the entire `ratings` table. This operation is resource-intensive for an OLTP-only database. Or you can use some ETL or other data synchronization tools to export the data from the OLTP database to a dedicated OLAP database for analysis.

In this scenario, TiDB, an **HTAP (Hybrid Transactional and Analytical Processing)** database that supports both OLTP and OLAP scenarios, is an ideal one-stop database solution.

4.5.3.8.1 Replicate column-based data

Currently, TiDB supports two data analysis engines, **TiFlash** and **TiSpark**. For the large data scenarios (100 T), **TiFlash MPP** is recommended as the primary solution for HTAP, and **TiSpark** as a complementary solution.

To learn more about TiDB HTAP capabilities, refer to the following documents: Quick Start Guide for TiDB HTAP and Explore HTAP.

In this example, TiFlash has been chosen as the data analysis engine for the `bookshop` database.

TiFlash does not automatically replicate data after deployment. Therefore, you need to manually specify the tables to be replicated:

```
ALTER TABLE {table_name} SET TIFLASH REPLICA {count};
```

Parameter description

- `{table_name}`: The table name.
- `{count}`: The number of replicated replicas. If it is 0, replicated replicas are deleted.

TiFlash will then replicate the table. When a query is performed, TiDB automatically selects TiKV (row-based) or TiFlash (column-based) for the query based on cost optimization. Alternatively, you can manually specify whether the query uses a **TiFlash** replica. To learn how to specify it, refer to Use TiDB to read TiFlash replicas.

4.5.3.8.2 An example of using HTAP capabilities

The `ratings` table opens `1` replica of TiFlash:

```
ALTER TABLE `bookshop`.`ratings` SET TIFLASH REPLICA 1;
```

> **Note:**
>
> If your cluster does not contain **TiFlash** nodes, this SQL statement will report an error: `1105 - the tiflash replica count: 1 should be less than the total tiflash server count: 0`. You can use Build a TiDB Serverless Cluster to create a TiDB Serverless cluster that includes **TiFlash**.

Then you can go on to perform the following query:

```
SELECT HOUR(`rated_at`), AVG(`score`) FROM `bookshop`.`ratings` GROUP BY
    ↪ HOUR(`rated_at`);
```

You can also execute the `EXPLAIN ANALYZE` statement to see whether this statement is using the **TiFlash**:

```
EXPLAIN ANALYZE SELECT HOUR(`rated_at`), AVG(`score`) FROM `bookshop`.`
    ↪ ratings` GROUP BY HOUR(`rated_at`);
```

Running results:

```
+--
    ↪ -------------------------+----------+--------+-------------+--------------
    ↪
| id                        | estRows  | actRows | task      | access object |
    ↪  execution info
    ↪
    ↪ | operator info
    ↪
    ↪ | memory  | disk |
+--
    ↪ -------------------------+----------+--------+-------------+--------------
    ↪
| Projection_4              | 299821.99 | 24    | root       |              |
    ↪ time:60.8ms, loops:6, Concurrency:5
    ↪
    ↪ | hour(cast(bookshop.ratings.rated_at, time))->Column#6, Column#5
    ↪                                                                | 17.7 KB
    ↪  | N/A |
|  ─HashAgg_5               | 299821.99 | 24    | root       |              |
    ↪ time:60.7ms, loops:6, partial_worker:{wall_time:60.660079ms,
    ↪ concurrency:5, task_num:293, tot_wait:262.536669ms, tot_exec
    ↪ :40.171833ms, tot_time:302.827753ms, max:60.636886ms, p95:60.636886ms
    ↪ }, final_worker:{wall_time:60.701437ms, concurrency:5, task_num:25,
    ↪ tot_wait:303.114278ms, tot_exec:176.564µs, tot_time:303.297475ms, max
    ↪ :60.69326ms, p95:60.69326ms} | group by:Column#10, funcs:avg(Column
    ↪ #8)->Column#5, funcs:firstrow(Column#9)->bookshop.ratings.rated_at |
    ↪ 714.0 KB | N/A |
|    ─Projection_15         | 300000.00 | 300000 | root      |              |
    ↪ time:58.5ms, loops:294, Concurrency:5
    ↪
    ↪ | cast(bookshop.ratings.score, decimal(8,4) BINARY)->Column#8,
    ↪ bookshop.ratings.rated_at, hour(cast(bookshop.ratings.rated_at, time)
    ↪ )->Column#10 | 366.2 KB | N/A |
|      ─TableReader_10      | 300000.00 | 300000 | root      |              |
    ↪ time:43.5ms, loops:294, cop_task: {num: 1, max: 43.1ms, proc_keys: 0,
    ↪  rpc_num: 1, rpc_time: 43ms, copr_cache_hit_ratio: 0.00}
    ↪
    ↪ | data:TableFullScan_9
```

```
    ↪
    ↪ | 4.58 MB | N/A |
|       └─TableFullScan_9   | 300000.00 | 300000 | cop[tiflash] | table:
    ↪ ratings | tiflash_task:{time:5.98ms, loops:8, threads:1}
    ↪
    ↪ | keep order:false
    ↪
    ↪ | N/A     | N/A |
+--
    ↪ ------------------------+----------+---------+-------------+--------------
    ↪
```

When the field `cop[tiflash]` appears, it means that the task is sent to **TiFlash** for processing.

4.5.3.9 Execute the `CREATE TABLE` statement

After creating all the tables as above rules, our database initialization script should look like this. If you need to see the table information in detail, please refer to Description of the Tables.

To name the database initialization script `init.sql` and save it, you can execute the following statement to initialize the database.

```
mysql
    -u root \
    -h {host} \
    -P {port} \
    -p {password} \
    < init.sql
```

To view all tables under the `bookshop` database, use the `SHOW TABLES` statement.

```
SHOW TABLES IN `bookshop`;
```

Running results:

```
+--------------------+
| Tables_in_bookshop |
+--------------------+
| authors            |
| book_authors       |
| books              |
| orders             |
| ratings            |
| users              |
+--------------------+
```

4.5.3.10 Guidelines to follow when creating a table

This section provides guidelines you need to follow when creating a table.

4.5.3.10.1 Guidelines to follow when naming a table

- Use a **fully-qualified** table name (for example, `CREATE TABLE {database_name}.{table_name}`). If you do not specify the database name, TiDB uses the current database in your **SQL session**. If you do not use `USE {databasename};` to specify the database in your SQL session, TiDB returns an error.
- Use meaningful table names. For example, if you need to create a user table, you can use names: `user`, `t_user`,`users`, or follow your company or organization's naming convention. If your company or organization does not have a naming convention, you can refer to the table naming convention. Do not use such table names as: `t1`, `table1`.
- Multiple words are separated by an underscore, and it is recommended that the name is no more than 32 characters.
- Create a separate `DATABASE` for tables of different business modules and add comments accordingly.

4.5.3.10.2 Guidelines to follow when defining columns

- Check the data types supported by columns and organize your data according to the data type restrictions. Select the appropriate type for the data you plan to store in the column.
- Check the guidelines to follow for selecting primary keys and decide whether to use primary key columns.
- Check the guidelines to follow for selecting clustered indexes and decide whether to specify **clustered indexes**.
- Check adding column constraints and decide whether to add constraints to the columns.
- Use meaningful column names. It is recommended that you follow your company or organization's table naming convention. If your company or organization does not have a corresponding naming convention, refer to the column naming convention.

4.5.3.10.3 Guidelines to follow when selecting primary key

- Define a **primary key** or **unique index** within the table.
- Try to select meaningful **columns** as **primary keys**.
- For performance reasons, try to avoid storing extra-wide tables. It is not recommended that the number of table fields is over `60` and that the total data size of a single row is over `64K`. It is recommended to split fields with too much data length to another table.
- It is not recommended to use complex data types.
- For the fields to be joined, ensure that the data types are consistent and avoid implicit conversion.

- Avoid defining **primary keys** on a single monotonic data column. If you use a single monotonic data column (for example, a column with the `AUTO_INCREMENT` attribute) to define the **primary key**, it might impact the write performance. If possible, use `AUTO_RANDOM` instead of `AUTO_INCREMENT`, which discards the continuous and incremental attribute of the primary key.
- If you really need to create an index on a single monotonic data column at write-intensive scenarios, instead of defining this monotonic data column as the **primary key**, you can use `AUTO_RANDOM` to create the **primary key** for that table, or use `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` to shard `_tidb_rowid`.

4.5.3.10.4 Guidelines to follow when selecting clustered index

- Follow guidelines for selecting primary key to build **clustered indexes**.
- Compared to tables with non-clustered indexes, tables with clustered indexes offer greater performance and throughput advantages in the following scenarios:
    - When data is inserted, the clustered index reduces one write of the index data from the network.
    - When a query with an equivalent condition only involves the primary key, the clustered index reduces one read of index data from the network.
    - When a query with a range condition only involves the primary key, the clustered index reduces multiple reads of index data from the network.
    - When a query with an equivalent or range condition only involves the primary key prefix, the clustered index reduces multiple reads of index data from the network.
- On the other hand, tables with clustered indexes might have the following issues:
    - There might be write hotspot issues when you insert a large number of primary keys with close values. Follow the guidelines to follow when selecting primary key.
    - The table data takes up more storage space if the data type of the primary key is larger than 64 bits, especially when there are multiple secondary indexes.
- To control the default behavior of whether to use clustered indexes, you can explicitly specify whether to use clustered indexes instead of using the system variable `@@global ↪ .tidb_enable_clustered_index` and the configuration `alter-primary-key`.

4.5.3.10.5 Guidelines to follow when executing the `CREATE TABLE` statement

- It is not recommended to use a client-side Driver or ORM to perform database schema changes. It is recommended to use a MySQL client or use a GUI client to perform database schema changes. In this document, the **MySQL client** is used to pass in SQL files to perform database schema changes in most scenarios.
- Follow the SQL development specification for creating and deleting tables. It is recommended to wrap the build and delete statements inside the business application to add judgment logic.

4.5.3.11 One more step

Note that all the tables that have been created in this document do not contain secondary indexes. For a guide to add secondary indexes, refer to Creating Secondary Indexes.

4.5.4 Create a Secondary Index

This document describes how to create a secondary index using SQL and various programming languages and lists the rules of index creation. In this document, the Bookshop application is taken as an example to walk you through the steps of secondary index creation.

4.5.4.1 Before you start

Before creating a secondary index, do the following:

- Build a TiDB Serverless Cluster.
- Read Schema Design Overview.
- Create a Database.
- Create a Table.

4.5.4.2 What is secondary index

A secondary index is a logical object in a TiDB cluster. You can simply regard it as a sorting type of data that TiDB uses to improve the query performance. In TiDB, creating a secondary index is an online operation, which does not block any data read and write operations on a table. For each index, TiDB creates references for each row in a table and sorts the references by selected columns instead of by data directly.

For more information about secondary indexes, see Secondary Indexes.

In TiDB, you can either add a secondary index to an existing table or create a secondary index when creating a new table.

4.5.4.3 Add a secondary index to an existing table

To add a secondary index to an existing table, you can use the CREATE INDEX statement as follows:

```
CREATE INDEX {index_name} ON {table_name} ({column_names});
```

Parameter description:

- `{index_name}`: the name of a secondary index.
- `{table_name}`: the table name.
- `{column_names}`: the names of the columns to be indexed, separated by semi-colon commas.

4.5.4.4 Create a secondary index when creating a new table

To create a secondary index at the same time as table creation, you can add a clause containing the `KEY` keyword to the end of the CREATE TABLE statement:

```
KEY `{index_name}` (`{column_names}`)
```

Parameter description:

- `{index_name}`: the name of a secondary index.
- `{column_names}`: the names of the columns to be indexed, separated by semi-colon commas.

4.5.4.5 Rules in secondary index creation

See Best Practices for Indexing.

4.5.4.6 Example

Suppose you want the `bookshop` application to support **searching for all books published in a given year**.

The fields in the `books` table are as follows:

Field name	Type	Field description
id	bigint(20)	Unique ID of the book
title	varchar(100)	Book title
type	enum	Types of books (for example, magazines, animations, and teaching aids)
stock	bigint(20)	Stock
price	decimal(15,2)	Price
published_at	datetime	Date of publishing

The `books` table is created using the following SQL statement:

```
CREATE TABLE `bookshop`.`books` (
  `id` bigint(20) AUTO_RANDOM NOT NULL,
  `title` varchar(100) NOT NULL,
  `type` enum('Magazine', 'Novel', 'Life', 'Arts', 'Comics', 'Education &
    ↪ Reference', 'Humanities & Social Sciences', 'Science & Technology',
    ↪ 'Kids', 'Sports') NOT NULL,
  `published_at` datetime NOT NULL,
  `stock` int(11) DEFAULT '0',
  `price` decimal(15,2) DEFAULT '0.0',
  PRIMARY KEY (`id`) CLUSTERED
) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

To support the searching by year feature, you need to write a SQL statement to **search for all books published in a given year**. Taking 2022 as an example, write a SQL statement as follows:

```
SELECT * FROM `bookshop`.`books` WHERE `published_at` >= '2022-01-01
    ↪ 00:00:00' AND `published_at` < '2023-01-01 00:00:00';
```

To check the execution plan of the SQL statement, you can use the `EXPLAIN` statement.

```
EXPLAIN SELECT * FROM `bookshop`.`books` WHERE `published_at` >= '
    ↪ 2022-01-01 00:00:00' AND `published_at` < '2023-01-01 00:00:00';
```

The following is an example output of the execution plan:

```
+------------------------+----------+-----------+---------------+----------------------
    ↪
| id                     | estRows | task      | access object | operator info
    ↪
    ↪ |
+------------------------+----------+-----------+---------------+----------------------
    ↪
| TableReader_7          | 346.32  | root      |               | data:Selection_6
    ↪
    ↪ |
|  ─Selection_6          | 346.32  | cop[tikv] |               | ge(bookshop.
    ↪ books.published_at, 2022-01-01 00:00:00.000000), lt(bookshop.books.
    ↪ published_at, 2023-01-01 00:00:00.000000) |
|    ─TableFullScan_5    | 20000.00 | cop[tikv] | table:books | keep order:
    ↪ false
    ↪
    ↪ |
+------------------------+----------+-----------+---------------+----------------------
    ↪
3 rows in set (0.61 sec)
```

In the example output, **TableFullScan** is displayed in the `id` column, which means that TiDB is ready to do a full table scan on the `books` table in this query. In the case of a large amount of data, however, a full table scan might be quite slow and cause a fatal impact.

To avoid such impact, you can add an index for the `published_at` column to the `books` table as follows:

```
CREATE INDEX `idx_book_published_at` ON `bookshop`.`books` (`bookshop`.`
    ↪ books`.`published_at`);
```

After adding the index, execute the `EXPLAIN` statement again to check the execution plan.

The following is an example output.

```
+-------------------------------+---------+-----------+-------------------------------
   ↪
| id                          | estRows | task    | access object
   ↪                                     | operator info
   ↪                                               |
+-------------------------------+---------+-----------+-------------------------------
   ↪
| IndexLookUp_10              | 146.01 | root     |
   ↪                                                  |
   ↪                                                            |
|  ─IndexRangeScan_8(Build)  | 146.01 | cop[tikv] | table:books, index:
   ↪ idx_book_published_at(published_at) | range:[2022-01-01
   ↪ 00:00:00,2023-01-01 00:00:00), keep order:false |
|  ─TableRowIDScan_9(Probe)  | 146.01 | cop[tikv] | table:books
   ↪                                     | keep order:false
   ↪                                           |
+-------------------------------+---------+-----------+-------------------------------
   ↪
3 rows in set (0.18 sec)
```

In the output, **IndexRangeScan** is displayed instead of **TableFullScan**, which means that TiDB is ready to use indexes to do this query.

The words such as **TableFullScan** and **IndexRangeScan** in the execution plan are operators in TiDB. For more information about execution plans and operators, see TiDB Query Execution Plan Overview.

The execution plan does not return the same operator every time. This is because TiDB uses a **Cost-Based Optimization (CBO)** approach, in which an execution plan depends on both rules and data distribution. For more information about TiDB SQL performance, see SQL Tuning Overview.

> **Note:**
>
> TiDB also supports explicit use of indexes when querying, and you can use Optimizer Hints or SQL Plan Management (SPM) to artificially control the use of indexes. But if you do not know well about indexes, optimizer hints, or SPM, **DO NOT** use this feature to avoid any unexpected results.

To query the indexes on a table, you can use the SHOW INDEXES statement:

```
SHOW INDEXES FROM `bookshop`.`books`;
```

The following is an example output:

```
+-------+------------+----------------------+-------------+-------------+-----------+
   ↪
| Table | Non_unique | Key_name             | Seq_in_index | Column_name |
   ↪ Collation | Cardinality | Sub_part | Packed | Null | Index_type |
   ↪ Comment | Index_comment | Visible | Expression | Clustered |
+-------+------------+----------------------+-------------+-------------+-----------+
   ↪
| books |          0 | PRIMARY              |            1 | id          | A
   ↪         |           0 |     NULL | NULL |      | BTREE     |         |
   ↪               | YES     | NULL       | YES       |
| books |          1 | idx_book_published_at |            1 | published_at | A
   ↪      |           0 |     NULL | NULL |      | BTREE     |         |
   ↪               | YES     | NULL       | NO        |
+-------+------------+----------------------+-------------+-------------+-----------+
   ↪
2 rows in set (1.63 sec)
```

4.5.4.7 Next step

After creating a database and adding tables and secondary indexes to it, you can start adding the data write and read features to your application.

4.6 Write Data

4.6.1 Insert Data

This document describes how to insert data into TiDB by using the SQL language with different programming languages.

4.6.1.1 Before you start

Before reading this document, you need to prepare the following:

- Build a TiDB Serverless Cluster.
- Read Schema Design Overview, Create a Database, Create a Table, and Create Secondary Indexes

4.6.1.2 Insert rows

There are two ways to insert multiple rows of data. For example, if you need to insert **3** players' data.

- A **multi-line insertion statement**:

```
INSERT INTO `player` (`id`, `coins`, `goods`) VALUES (1, 1000, 1), (2,
    ↪ 230, 2), (3, 300, 5);
```

- Multiple **single-line insertion statements**:

```
INSERT INTO `player` (`id`, `coins`, `goods`) VALUES (1, 1000, 1);
INSERT INTO `player` (`id`, `coins`, `goods`) VALUES (2, 230, 2);
INSERT INTO `player` (`id`, `coins`, `goods`) VALUES (3, 300, 5);
```

Generally the `multi-line insertion statement` runs faster than the multiple `single-line insertion statements`.

```
CREATE TABLE `player` (`id` INT, `coins` INT, `goods` INT);
INSERT INTO `player` (`id`, `coins`, `goods`) VALUES (1, 1000, 1), (2, 230,
    ↪ 2);
```

For more information on how to use this SQL, see Connecting to a TiDB Cluster and follow the steps to enter the SQL statement after connecting to a TiDB cluster using a client.

```
// ds is an entity of com.mysql.cj.jdbc.MysqlDataSource
try (Connection connection = ds.getConnection()) {
    connection.setAutoCommit(false);

    PreparedStatement pstmt = connection.prepareStatement("INSERT INTO
        ↪ player (id, coins, goods) VALUES (?, ?, ?)"))

    // first player
    pstmt.setInt(1, 1);
    pstmt.setInt(2, 1000);
    pstmt.setInt(3, 1);
    pstmt.addBatch();

    // second player
    pstmt.setInt(1, 2);
    pstmt.setInt(2, 230);
    pstmt.setInt(3, 2);
    pstmt.addBatch();

    pstmt.executeBatch();
    connection.commit();
} catch (SQLException e) {
    e.printStackTrace();
}
```

Due to the default MySQL JDBC Driver settings, you need to change some parameters to get better bulk insert performance.

Parameter	Means	Recommended Scenario	Recommended Configuration
`useServerPrepStmts`	Whether to use the server side to enable prepared statements	When you need to use a prepared statement more than once	`true`
`cachePrepStmts`	Whether the client caches prepared statements	`useServerPrepStmts=true` 时	`true`
`prepStmtCacheSqlLimit`	Maximum size of a prepared statement (256 characters by default)	When the prepared statement is greater than 256 characters	Configured according to the actual size of the prepared statement
`prepStmtCacheSize`	Maximum number of prepared statement caches (25 by default)	When the number of prepared statements is greater than 25	Configured according to the actual number of prepared statements
`rewriteBatchedStatements`	Whether to rewrite **Batched** statements	When batch operations are required	`true`
`allowMultiQueries`	Support batch operations	Because a client bug requires this to be set when `rewriteBatchedStatements = true` and `useServerPrepStmts = true`	`true`

MySQL JDBC Driver also provides an integrated configuration: `useConfigs`. When it is configured with `maxPerformance`, it is equivalent to configuring a set of configurations. Taking `mysql:mysql-connector-java:8.0.28` as an example, `useConfigs=maxPerformance`

contains:

```
cachePrepStmts=true
cacheCallableStmts=true
cacheServerConfiguration=true
useLocalSessionState=true
elideSetAutoCommits=true
alwaysSendSetIsolation=false
enableQueryTimeouts=false
connectionAttributes=none
useInformationSchema=true
```

You can check `mysql-connector-java-{version}.jar!/com/mysql/cj/configurations/maxPerformance.properties` to get the configurations contained in `useConfigs=maxPerformance` for the corresponding version of MySQL JDBC Driver.

The following is a typical scenario of JDBC connection string configurations. In this example, Host: `127.0.0.1`, Port: `4000`, User name: `root`, Password: null, Default database: `test`:

```
jdbc:mysql://127.0.0.1:4000/test?user=root&useConfigs=maxPerformance&useServerPrepStmts=true&prepStmtCacheSqlLimit=2048&prepStmtCacheSize=256&rewriteBatchedStatements=true&allowMultiQueries=true
```

For complete examples in Java, see:

- Build a simple CRUD application with TiDB and Java - using JDBC
- Build a simple CRUD application with TiDB and Java - using Hibernate
- Build the TiDB application using Spring Boot

```
package main

import (
    "database/sql"
    "strings"

    _ "github.com/go-sql-driver/mysql"
)

type Player struct {
    ID    string
    Coins int
    Goods int
}

func bulkInsertPlayers(db *sql.DB, players []Player, batchSize int) error {
```

```
    tx, err := db.Begin()
    if err != nil {
        return err
    }

    stmt, err := tx.Prepare(buildBulkInsertSQL(batchSize))
    if err != nil {
        return err
    }

    defer stmt.Close()

    for len(players) > batchSize {
        if _, err := stmt.Exec(playerToArgs(players[:batchSize])...); err !=
            ↪ nil {
            tx.Rollback()
            return err
        }

        players = players[batchSize:]
    }

    if len(players) != 0 {
        if _, err := tx.Exec(buildBulkInsertSQL(len(players)), playerToArgs(
            ↪ players)...); err != nil {
            tx.Rollback()
            return err
        }
    }

    if err := tx.Commit(); err != nil {
        tx.Rollback()
        return err
    }

    return nil
}

func playerToArgs(players []Player) []interface{} {
    var args []interface{}
    for _, player := range players {
        args = append(args, player.ID, player.Coins, player.Goods)
    }
    return args
}
```

```
func buildBulkInsertSQL(amount int) string {
    return "INSERT INTO player (id, coins, goods) VALUES (?, ?, ?)" +
        ↪ strings.Repeat(",(?,?,?)", amount-1)
}
```

For complete examples in Golang, see:

- Use go-sql-driver/mysql to build a simple CRUD application with TiDB and Golang
- Use GORM to build a simple CRUD application with TiDB and Golang

```
import MySQLdb
connection = MySQLdb.connect(
    host="127.0.0.1",
    port=4000,
    user="root",
    password="",
    database="bookshop",
    autocommit=True
)

with get_connection(autocommit=True) as connection:
    with connection.cursor() as cur:
        player_list = random_player(1919)
        for idx in range(0, len(player_list), 114):
            cur.executemany("INSERT INTO player (id, coins, goods) VALUES (%s
                ↪ , %s, %s)", player_list[idx:idx + 114])
```

For complete examples in Python, see:

- Use PyMySQL to build a simple CRUD application with TiDB and Python
- Use mysqlclient to build a simple CRUD application with TiDB and Python
- Use mysql-connector-python to build a simple CRUD application with TiDB and Python
- Use SQLAlchemy to build a simple CRUD application with TiDB and Python
- Use peewee to build a simple CRUD application with TiDB and Python

4.6.1.3 Bulk-Insert

If you need to quickly import a large amount of data into a TiDB cluster, it is recommended that you use a range of tools provided by **PingCAP** for data migration. Using the `INSERT` statement is not the best way, because it is not efficient and requires to handle exceptions and other issues on your own.

The following are the recommended tools for bulk-insert:

- Data export: Dumpling. You can export MySQL or TiDB data to local or Amazon S3.

- Data import: TiDB Lightning. You can import **Dumpling** exported data, a **CSV** file, or Migrate Data from Amazon Aurora to TiDB. It also supports reading data from a local disk or Amazon S3 cloud disk.
- Data replication: TiDB Data Migration. You can replicate MySQL, MariaDB, and Amazon Aurora databases to TiDB. It also supports merging and migrating the sharded instances and tables from the source databases.
- Data backup and restore: Backup & Restore (BR). Compared to **Dumpling**, **BR** is more suitable for ***big data*** scenario.

4.6.1.4 Avoid hotspots

When designing a table, you need to consider if there is a large number of insert operations. If so, you need to avoid hotspots during table design. See the Select primary key section and follow the Rules when selecting primary key.

For more information on how to handle hotspot issues, see Troubleshoot Hotspot Issues.

4.6.1.5 Insert data to a table with the `AUTO_RANDOM` primary key

If the primary key of the table you insert has the `AUTO_RANDOM` attribute, then by default the primary key cannot be specified. For example, in the `bookshop` database, you can see that the `id` field of the `users` table contains the `AUTO_RANDOM` attribute.

In this case, you **cannot** use SQL like the following to insert:

```
INSERT INTO `bookshop`.`users` (`id`, `balance`, `nickname`) VALUES (1,
    ↪ 0.00, 'nicky');
```

An error will occur:

```
ERROR 8216 (HY000): Invalid auto random: Explicit insertion on auto_random
    ↪ column is disabled. Try to set @@allow_auto_random_explicit_insert =
    ↪ true.
```

It is not recommended to manually specify the `AUTO_RANDOM` column during insertion time.

There are two solutions to handle this error:

- (Recommended) Remove this column from the insert statement and use the `AUTO_RANDOM` value that TiDB initialized for you. This fits the semantics of `AUTO_RANDOM`.

  ```
  INSERT INTO `bookshop`.`users` (`balance`, `nickname`) VALUES (0.00, '
      ↪ nicky');
  ```

- If you are sure that you ***must*** specify this column, then you can use the SET statement to allow the column of AUTO_RANDOM to be specified during insertion time by changing the user variable.

```
SET @@allow_auto_random_explicit_insert = true;
INSERT INTO `bookshop`.`users` (`id`, `balance`, `nickname`) VALUES
    ↪ (1, 0.00, 'nicky');
```

4.6.1.6 Use HTAP

In TiDB, HTAP capabilities save you from performing additional operations when inserting data. There is no additional insertion logic. TiDB automatically guarantees data consistency. All you need to do is turn on column-oriented replica synchronization after creating the table, and use the column-oriented replica to speed up your queries directly.

4.6.2 Update Data

This document describes how to use the following SQL statements to update the data in TiDB with various programming languages:

- UPDATE: Used to modify the data in the specified table.
- INSERT ON DUPLICATE KEY UPDATE: Used to insert data and update this data if there is a primary key or unique key conflict. It is **not recommended** to use this statement if there are multiple unique keys (including primary keys). This is because this statement updates the data once it detects any unique key (including primary key) conflict. When there are more than one row conflicts, it updates only one row.

4.6.2.1 Before you start

Before reading this document, you need to prepare the following:

- Build a TiDB Serverless Cluster.
- Read Schema Design Overview, Create a Database, Create a Table, and Create Secondary Indexes.
- If you want to UPDATE data, you need to insert data first.

4.6.2.2 Use UPDATE

To update an existing row in a table, you need to use an UPDATE statement with a WHERE clause to filter the columns for updating.

Note:

If you need to update a large number of rows, for example, more than ten thousand, it is recommended that you do ***NOT*** doing a complete update at once, but rather updating a portion at a time iteratively until all rows are updated. You can write scripts or programs to loop this operation. See Bulk-update for details.

4.6.2.2.1 `UPDATE` SQL syntax

In SQL, the `UPDATE` statement is generally in the following form:

```
UPDATE {table} SET {update_column} = {update_value} WHERE {filter_column} =
    ↪ {filter_value}
```

Parameter Name	Description
`{table}`	Table Name
`{update_column}`	Column names to be updated
`{update_value}`	Column values to be updated
`{filter_column}`	Column names matching filters
`{filter_value}`	Column values matching filters

For detailed information, see UPDATE syntax.

4.6.2.2.2 `UPDATE` best practices

The following are some best practices for updating data:

- Always specify the `WHERE` clause in the `UPDATE` statement. If the `UPDATE` statement does not have a `WHERE` clause, TiDB will update ***ALL ROWS*** in the table.

- Use bulk-update when you need to update a large number of rows (for example, more than ten thousand). Because TiDB limits the size of a single transaction (txn-total-size-limit, 100 MB by default), too many data updates at once will result in holding locks for too long (pessimistic transactions) or cause conflicts (optimistic transactions).

4.6.2.2.3 `UPDATE` example

Suppose an author changes her name to **Helen Haruki**. You need to change the authors table. Assume that her unique `id` is **1**, and the filter should be: `id = 1`.

```
UPDATE `authors` SET `name` = "Helen Haruki" WHERE `id` = 1;
```

```java
// ds is an entity of com.mysql.cj.jdbc.MysqlDataSource
try (Connection connection = ds.getConnection()) {
    PreparedStatement pstmt = connection.prepareStatement("UPDATE `authors`
        ↪ SET `name` = ? WHERE `id` = ?");
    pstmt.setString(1, "Helen Haruki");
    pstmt.setInt(2, 1);
    pstmt.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
```

4.6.2.3 Use `INSERT ON DUPLICATE KEY UPDATE`

If you need to insert new data into a table, but if there are unique key (a primary key is also a unique key) conflicts, the first conflicted record will be updated. You can use `INSERT ... ON DUPLICATE KEY UPDATE ...` statement to insert or update.

4.6.2.3.1 `INSERT ON DUPLICATE KEY UPDATE` SQL Syntax

In SQL, the `INSERT ... ON DUPLICATE KEY UPDATE ...` statement is generally in the following form:

```
INSERT INTO {table} ({columns}) VALUES ({values})
    ON DUPLICATE KEY UPDATE {update_column} = {update_value};
```

Parameter Name	Description
`{table}`	Table name
`{columns}`	Column names to be inserted
`{values}`	Column values to be inserted
`{update_column}`	Column names to be updated
`{update_value}`	Column values to be updated

4.6.2.3.2 `INSERT ON DUPLICATE KEY UPDATE` best practices

- Use `INSERT ON DUPLICATE KEY UPDATE` only for a table with one unique key. This statement updates the data if any ***UNIQUE KEY*** (including the primary key) conflicts are detected. If there are more than one row of conflicts, only one row will be updated. Therefore, it is not recommended to use the `INSERT ON DUPLICATE KEY ↪ UPDATE` statement in tables with multiple unique keys unless you can guarantee that there is only one row of conflict.
- Use this statement when you create data or update data.

4.6.2.3.3 `INSERT ON DUPLICATE KEY UPDATE` example

For example, you need to update the ratings table to include the user's ratings for the book. If the user has not yet rated the book, a new rating will be created. If the user has already rated it, his previous rating will be updated.

In the following example, the primary key is the joint primary keys of `book_id` and `user_id`. A user `user_id = 1` gives a rating of `5` to a book `book_id = 1000`.

```
INSERT INTO `ratings`
    (`book_id`, `user_id`, `score`, `rated_at`)
VALUES
    (1000, 1, 5, NOW())
ON DUPLICATE KEY UPDATE `score` = 5, `rated_at` = NOW();
```

```
// ds is an entity of com.mysql.cj.jdbc.MysqlDataSource

try (Connection connection = ds.getConnection()) {
    PreparedStatement p = connection.prepareStatement("INSERT INTO `ratings`
        ↪ (`book_id`, `user_id`, `score`, `rated_at`)
VALUES (?, ?, ?, NOW()) ON DUPLICATE KEY UPDATE `score` = ?, `rated_at` =
    ↪ NOW()");
    p.setInt(1, 1000);
    p.setInt(2, 1);
    p.setInt(3, 5);
    p.setInt(4, 5);
    p.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
```

4.6.2.4 Bulk-update

When you need to update multiple rows of data in a table, you can use `INSERT ON DUPLICATE KEY UPDATE` with the `WHERE` clause to filter the data that needs to be updated.

However, if you need to update a large number of rows (for example, more than ten thousand), it is recommended that you update the data iteratively, that is, updating only a portion of the data at each iteration until the update is complete. This is because TiDB limits the size of a single transaction (txn-total-size-limit, 100 MB by default). Too many data updates at once will result in holding locks for too long (pessimistic transactions, or causing conflicts (optimistic transactions). You can use a loop in your program or script to complete the operation.

This section provides examples of writing scripts to handle iterative updates. This example shows how a combination of `SELECT` and `UPDATE` should be done to complete a bulk-update.

4.6.2.4.1 Write bulk-update loop

First, you should write a `SELECT` query in a loop of your application or script. The return value of this query can be used as the primary key for the rows that need to be updated. Note that when defining this `SELECT` query, you need to use the `WHERE` clause to filter the rows that need to be updated.

4.6.2.4.2 Example

Suppose that you have had a lot of book ratings from users on your `bookshop` website over the past year, but the original design of a 5-point scale has resulted in a lack of differentiation in book ratings. Most books are rated `3`. You decide to switch from a 5-point scale to a 10-point scale to differentiate ratings.

You need to multiply by `2` the data in the `ratings` table from the previous 5-point scale, and add a new column to the ratings table to indicate whether the rows have been updated. Using this column, you can filter out rows that have been updated in `SELECT`, which will prevent the script from crashing and updating the rows multiple times, resulting in unreasonable data.

For example, you create a column named `ten_point` with the data type BOOL as an identifier of whether it is a 10-point scale:

```
ALTER TABLE `bookshop`.`ratings` ADD COLUMN `ten_point` BOOL NOT NULL
    ↪ DEFAULT FALSE;
```

> **Note:**
>
> This bulk-update application uses the **DDL** statements to make schema changes to the data tables. All DDL change operations for TiDB are executed online. For more information, see ADD COLUMN.

In Golang, a bulk-update application is similar to the following:

```
package main

import (
    "database/sql"
    "fmt"
    _ "github.com/go-sql-driver/mysql"
    "strings"
    "time"
)

func main() {
```

```
    db, err := sql.Open("mysql", "root:@tcp(127.0.0.1:4000)/bookshop")
    if err != nil {
        panic(err)
    }
    defer db.Close()

    bookID, userID := updateBatch(db, true, 0, 0)
    fmt.Println("first time batch update success")
    for {
        time.Sleep(time.Second)
        bookID, userID = updateBatch(db, false, bookID, userID)
        fmt.Printf("batch update success, [bookID] %d, [userID] %d\n", bookID
            ↪ , userID)
    }
}

// updateBatch select at most 1000 lines data to update score
func updateBatch(db *sql.DB, firstTime bool, lastBookID, lastUserID int64) (
    ↪ bookID, userID int64) {
    // select at most 1000 primary keys in five-point scale data
    var err error
    var rows *sql.Rows

    if firstTime {
        rows, err = db.Query("SELECT `book_id`, `user_id` FROM `bookshop`.`
            ↪ ratings` " +
            "WHERE `ten_point` != true ORDER BY `book_id`, `user_id` LIMIT
                ↪ 1000")
    } else {
        rows, err = db.Query("SELECT `book_id`, `user_id` FROM `bookshop`.`
            ↪ ratings` "+
            "WHERE `ten_point` != true AND `book_id` > ? AND `user_id` > ? "+
            "ORDER BY `book_id`, `user_id` LIMIT 1000", lastBookID,
                ↪ lastUserID)
    }

    if err != nil || rows == nil {
        panic(fmt.Errorf("error occurred or rows nil: %+v", err))
    }

    // joint all id with a list
    var idList []interface{}
    for rows.Next() {
        var tempBookID, tempUserID int64
        if err := rows.Scan(&tempBookID, &tempUserID); err != nil {
```

```
            panic(err)
        }
        idList = append(idList, tempBookID, tempUserID)
        bookID, userID = tempBookID, tempUserID
    }

    bulkUpdateSql := fmt.Sprintf("UPDATE `bookshop`.`ratings` SET `ten_point
        ↪ ` = true, "+
        "`score` = `score` * 2 WHERE (`book_id`, `user_id`) IN (%s)",
            ↪ placeHolder(len(idList)))
    db.Exec(bulkUpdateSql, idList...)

    return bookID, userID
}

// placeHolder format SQL place holder
func placeHolder(n int) string {
    holderList := make([]string, n/2, n/2)
    for i := range holderList {
        holderList[i] = "(?,?)"
    }
    return strings.Join(holderList, ",")
}
```

In each iteration, `SELECT` queries in order of the primary key. It selects primary key values for up to 1000 rows that have not been updated to the 10-point scale (`ten_point` is `false`). Each `SELECT` statement selects primary keys larger than the largest of the previous `SELECT` results to prevent duplication. Then, it uses bulk-update, multiples its `score` column by 2, and sets `ten_point` to `true`. The purpose of updating `ten_point` is to prevent the update application from repeatedly updating the same row in case of restart after crashing, which can cause data corruption. `time.Sleep(time.Second)` in each loop makes the update application pause for 1 second to prevent the update application from consuming too many hardware resources.

In Java (JDBC), a bulk-update application might be similar to the following:

Code:

```
package com.pingcap.bulkUpdate;

import com.mysql.cj.jdbc.MysqlDataSource;

import java.sql.*;
import java.util.LinkedList;
import java.util.List;
import java.util.concurrent.TimeUnit;
```

```
public class BatchUpdateExample {
    static class UpdateID {
        private Long bookID;
        private Long userID;

        public UpdateID(Long bookID, Long userID) {
            this.bookID = bookID;
            this.userID = userID;
        }

        public Long getBookID() {
            return bookID;
        }

        public void setBookID(Long bookID) {
            this.bookID = bookID;
        }

        public Long getUserID() {
            return userID;
        }

        public void setUserID(Long userID) {
            this.userID = userID;
        }

        @Override
        public String toString() {
            return "[bookID] " + bookID + ", [userID] " + userID ;
        }
    }

    public static void main(String[] args) throws InterruptedException {
        // Configure the example database connection.

        // Create a mysql data source instance.
        MysqlDataSource mysqlDataSource = new MysqlDataSource();

        // Set server name, port, database name, username and password.
        mysqlDataSource.setServerName("localhost");
        mysqlDataSource.setPortNumber(4000);
        mysqlDataSource.setDatabaseName("bookshop");
        mysqlDataSource.setUser("root");
        mysqlDataSource.setPassword("");
```

```
        UpdateID lastID = batchUpdate(mysqlDataSource, null);

        System.out.println("first time batch update success");
        while (true) {
            TimeUnit.SECONDS.sleep(1);
            lastID = batchUpdate(mysqlDataSource, lastID);
            System.out.println("batch update success, [lastID] " + lastID);
        }
    }

    public static UpdateID batchUpdate (MysqlDataSource ds, UpdateID lastID)
        ↪ {
        try (Connection connection = ds.getConnection()) {
            UpdateID updateID = null;

            PreparedStatement selectPs;

            if (lastID == null) {
                selectPs = connection.prepareStatement(
                        "SELECT `book_id`, `user_id` FROM `bookshop`.`ratings`
                            ↪ " +
                        "WHERE `ten_point` != true ORDER BY `book_id`, `user_id
                            ↪ ` LIMIT 1000");
            } else {
                selectPs = connection.prepareStatement(
                        "SELECT `book_id`, `user_id` FROM `bookshop`.`ratings`
                            ↪ "+
                            "WHERE `ten_point` != true AND `book_id` > ? AND `
                                ↪ user_id` > ? "+
                            "ORDER BY `book_id`, `user_id` LIMIT 1000");

                selectPs.setLong(1, lastID.getBookID());
                selectPs.setLong(2, lastID.getUserID());
            }

            List<Long> idList = new LinkedList<>();
            ResultSet res = selectPs.executeQuery();
            while (res.next()) {
                updateID = new UpdateID(
                        res.getLong("book_id"),
                        res.getLong("user_id")
                );
                idList.add(updateID.getBookID());
                idList.add(updateID.getUserID());
            }
```

```
            if (idList.isEmpty()) {
                System.out.println("no data should update");
                return null;
            }

            String updateSQL = "UPDATE `bookshop`.`ratings` SET `ten_point` =
                ↪ true, "+
                    "`score` = `score` * 2 WHERE (`book_id`, `user_id`) IN ("
                        ↪ +
                    placeHolder(idList.size() / 2) + ")";
            PreparedStatement updatePs = connection.prepareStatement(
                ↪ updateSQL);
            for (int i = 0; i < idList.size(); i++) {
                updatePs.setLong(i + 1, idList.get(i));
            }
            int count = updatePs.executeUpdate();
            System.out.println("update " + count + " data");

            return updateID;
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return null;
    }

    public static String placeHolder(int n) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < n ; i++) {
            sb.append(i == 0 ? "(?,?)" : ",(?,?)");
        }

        return sb.toString();
    }
}
```

- `hibernate.cfg.xml` configuration:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
```

```
    <session-factory>

        <!-- Database connection settings -->
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.
            ↪ Driver</property>
        <property name="hibernate.dialect">org.hibernate.dialect.TiDBDialect<
            ↪ /property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:4000
            ↪ /movie</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property name="hibernate.connection.autocommit">false</property>
        <property name="hibernate.jdbc.batch_size">20</property>

        <!-- Optional: Show SQL output for debugging -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
    </session-factory>
</hibernate-configuration>
```

In each iteration, `SELECT` queries in order of the primary key. It selects primary key values for up to `1000` rows that have not been updated to the 10-point scale (`ten_point` is `false`). Each `SELECT` statement selects primary keys larger than the largest of the previous `SELECT` results to prevent duplication. Then, it uses bulk-update, multiples its `score` column by `2`, and sets `ten_point` to `true`. The purpose of updating `ten_point` is to prevent the update application from repeatedly updating the same row in case of restart after crashing, which can cause data corruption. `TimeUnit.SECONDS.sleep(1);` in each loop makes the update application pause for 1 second to prevent the update application from consuming too many hardware resources.

4.6.3 Delete Data

This document describes how to use the DELETE SQL statement to delete the data in TiDB.

4.6.3.1 Before you start

Before reading this document, you need to prepare the following:

- Build a TiDB Serverless Cluster
- Read Schema Design Overview, Create a Database, Create a Table, and Create Secondary Indexes
- Insert Data

4.6.3.2 SQL syntax

The `DELETE` statement is generally in the following form:

```
DELETE FROM {table} WHERE {filter}
```

Parameter Name	Description
`{table}`	Table name
`{filter}`	Matching conditions of the filter

This example only shows a simple use case of `DELETE`. For detailed information, see DELETE syntax.

4.6.3.3 Best practices

The following are some best practices to follow when you delete data:

- Always specify the `WHERE` clause in the `DELETE` statement. If the `WHERE` clause is not specified, TiDB will delete ***ALL ROWS*** in the table.
- Use bulk-delete when you delete a large number of rows (for example, more than ten thousand), because TiDB limits the size of a single transaction (txn-total-size-limit, 100 MB by default).
- If you delete all the data in a table, do not use the `DELETE` statement. Instead, use the `TRUNCATE` statement.
- For performance considerations, see Performance Considerations.
- In scenarios where large batches of data need to be deleted, Non-Transactional bulk-delete can significantly improve performance. However, this will lose the transactional of the deletion and therefore **CANNOT** be rolled back. Make sure that you select the correct operation.

4.6.3.4 Example

Suppose you find an application error within a specific time period and you need to delete all the data for the ratings within this period, for example, from `2022-04-15 00:00:00` to `2022-04-15 00:15:00`. In this case, you can use the `SELECT` statement to check the number of records to be deleted.

```
SELECT COUNT(*) FROM `ratings` WHERE `rated_at` >= "2022-04-15 00:00:00"
    ↪ AND `rated_at` <= "2022-04-15 00:15:00";
```

If more than 10,000 records are returned, use Bulk-Delete to delete them.

If fewer than 10,000 records are returned, use the following example to delete them.

In SQL, the example is as follows:

```
DELETE FROM `ratings` WHERE `rated_at` >= "2022-04-15 00:00:00" AND `
    ↪ rated_at` <= "2022-04-15 00:15:00";
```

In Java, the example is as follows:

```
// ds is an entity of com.mysql.cj.jdbc.MysqlDataSource

try (Connection connection = ds.getConnection()) {
    String sql = "DELETE FROM `bookshop`.`ratings` WHERE `rated_at` >= ? AND
        ↪  `rated_at` <= ?";
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    Calendar calendar = Calendar.getInstance();
    calendar.set(Calendar.MILLISECOND, 0);

    calendar.set(2022, Calendar.APRIL, 15, 0, 0, 0);
    preparedStatement.setTimestamp(1, new Timestamp(calendar.getTimeInMillis
        ↪ ()));

    calendar.set(2022, Calendar.APRIL, 15, 0, 15, 0);
    preparedStatement.setTimestamp(2, new Timestamp(calendar.getTimeInMillis
        ↪ ()));

    preparedStatement.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
```

In Golang, the example is as follows:

```
package main

import (
    "database/sql"
    "fmt"
    "time"

    _ "github.com/go-sql-driver/mysql"
)

func main() {
    db, err := sql.Open("mysql", "root:@tcp(127.0.0.1:4000)/bookshop")
    if err != nil {
        panic(err)
    }
    defer db.Close()
```

```
    startTime := time.Date(2022, 04, 15, 0, 0, 0, 0, time.UTC)
    endTime := time.Date(2022, 04, 15, 0, 15, 0, 0, time.UTC)

    bulkUpdateSql := fmt.Sprintf("DELETE FROM `bookshop`.`ratings` WHERE `
        ↪ rated_at` >= ? AND `rated_at` <= ?")
    result, err := db.Exec(bulkUpdateSql, startTime, endTime)
    if err != nil {
        panic(err)
    }
    _, err = result.RowsAffected()
    if err != nil {
        panic(err)
    }
}
```

In Python, the example is as follows:

```
import MySQLdb
import datetime
import time
connection = MySQLdb.connect(
    host="127.0.0.1",
    port=4000,
    user="root",
    password="",
    database="bookshop",
    autocommit=True
)
with connection:
    with connection.cursor() as cursor:
        start_time = datetime.datetime(2022, 4, 15)
        end_time = datetime.datetime(2022, 4, 15, 0, 15)
        delete_sql = "DELETE FROM `bookshop`.`ratings` WHERE `rated_at` >= %s
            ↪  AND `rated_at` <= %s"
        affect_rows = cursor.execute(delete_sql, (start_time, end_time))
        print(f'delete {affect_rows} data')
```

The `rated_at` field is of the `DATETIME` type in Date and Time Types. You can assume that it is stored as a literal quantity in TiDB, independent of the time zone. On the other hand, the `TIMESTAMP` type stores a timestamp and thus displays a different time string in a different time zone.

Note:

Like MySQL, the `TIMESTAMP` data type is affected by the year 2038 problem. It is recommended to use the `DATETIME` type if you store values larger than 2038.

4.6.3.5 Performance considerations

4.6.3.5.1 TiDB GC mechanism

TiDB does not delete the data immediately after you run the `DELETE` statement. Instead, it marks the data as ready for deletion. Then it waits for TiDB GC (Garbage Collection) to clean up the outdated data. Therefore, the `DELETE` statement ***DOES NOT*** immediately reduce disk usage.

GC is triggered once every 10 minutes by default. Each GC calculates a time point called **safe_point**. Any data earlier than this time point will not be used again, so TiDB can safely clean it up.

For more information, see GC mechanism.

4.6.3.5.2 Update statistical information

TiDB uses statistical information to determine index selection. There is a high risk that the index is not correctly selected after a large volume of data is deleted. You can use manual collection to update the statistics. It provides the TiDB optimizer with more accurate statistical information for SQL performance optimization.

4.6.3.6 Bulk-delete

When you need to delete multiple rows of data from a table, you can choose the `DELETE` example and use the `WHERE` clause to filter the data that needs to be deleted.

However, if you need to delete a large number of rows (more than ten thousand), it is recommended that you delete the data in an iterative way, that is, deleting a portion of the data at each iteration until the deletion is completed. This is because TiDB limits the size of a single transaction (`txn-total-size-limit`, 100 MB by default). You can use loops in your programs or scripts to perform such operations.

This section provides an example of writing a script to handle an iterative delete operation that demonstrates how you should do a combination of `SELECT` and `DELETE` to complete a bulk-delete.

4.6.3.6.1 Write a bulk-delete loop

You can write a `DELETE` statement in the loop of your application or script, use the `WHERE` clause to filter data, and use `LIMIT` to constrain the number of rows to be deleted in a single statement.

4.6.3.6.2 Bulk-delete example

Suppose you find an application error within a specific time period. You need to delete all the data for the rating within this period, for example, from `2022-04-15 00:00:00` to `2022-04-15 00:15:00`, and more than 10,000 records are written in 15 minutes. You can perform as follows.

In Java, the bulk-delete example is as follows:

```
package com.pingcap.bulkDelete;

import com.mysql.cj.jdbc.MysqlDataSource;

import java.sql.*;
import java.util.*;
import java.util.concurrent.TimeUnit;

public class BatchDeleteExample
{
    public static void main(String[] args) throws InterruptedException {
        // Configure the example database connection.

        // Create a mysql data source instance.
        MysqlDataSource mysqlDataSource = new MysqlDataSource();

        // Set server name, port, database name, username and password.
        mysqlDataSource.setServerName("localhost");
        mysqlDataSource.setPortNumber(4000);
        mysqlDataSource.setDatabaseName("bookshop");
        mysqlDataSource.setUser("root");
        mysqlDataSource.setPassword("");

        while (true) {
            batchDelete(mysqlDataSource);
            TimeUnit.SECONDS.sleep(1);
        }
    }

    public static void batchDelete (MysqlDataSource ds) {
        try (Connection connection = ds.getConnection()) {
```

```
            String sql = "DELETE FROM `bookshop`.`ratings` WHERE `rated_at`
                ↪ >= ? AND `rated_at` <= ? LIMIT 1000";
            PreparedStatement preparedStatement = connection.prepareStatement
                ↪ (sql);
            Calendar calendar = Calendar.getInstance();
            calendar.set(Calendar.MILLISECOND, 0);

            calendar.set(2022, Calendar.APRIL, 15, 0, 0, 0);
            preparedStatement.setTimestamp(1, new Timestamp(calendar.
                ↪ getTimeInMillis()));

            calendar.set(2022, Calendar.APRIL, 15, 0, 15, 0);
            preparedStatement.setTimestamp(2, new Timestamp(calendar.
                ↪ getTimeInMillis()));

            int count = preparedStatement.executeUpdate();
            System.out.println("delete " + count + " data");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

In each iteration, `DELETE` deletes up to 1000 rows from `2022-04-15 00:00:00` to `2022-04-15 00:15:00`.

In Golang, the bulk-delete example is as follows:

```
package main

import (
    "database/sql"
    "fmt"
    "time"

    _ "github.com/go-sql-driver/mysql"
)

func main() {
    db, err := sql.Open("mysql", "root:@tcp(127.0.0.1:4000)/bookshop")
    if err != nil {
        panic(err)
    }
    defer db.Close()

    affectedRows := int64(-1)
```

```
    startTime := time.Date(2022, 04, 15, 0, 0, 0, 0, time.UTC)
    endTime := time.Date(2022, 04, 15, 0, 15, 0, 0, time.UTC)

    for affectedRows != 0 {
        affectedRows, err = deleteBatch(db, startTime, endTime)
        if err != nil {
            panic(err)
        }
    }
}

// deleteBatch delete at most 1000 lines per batch
func deleteBatch(db *sql.DB, startTime, endTime time.Time) (int64, error) {
    bulkUpdateSql := fmt.Sprintf("DELETE FROM `bookshop`.`ratings` WHERE `
        ↪ rated_at` >= ? AND `rated_at` <= ? LIMIT 1000")
    result, err := db.Exec(bulkUpdateSql, startTime, endTime)
    if err != nil {
        return -1, err
    }
    affectedRows, err := result.RowsAffected()
    if err != nil {
        return -1, err
    }

    fmt.Printf("delete %d data\n", affectedRows)
    return affectedRows, nil
}
```

In each iteration, `DELETE` deletes up to 1000 rows from `2022-04-15 00:00:00` to `2022-04-15 00:15:00`.

In Python, the bulk-delete example is as follows:

```
import MySQLdb
import datetime
import time
connection = MySQLdb.connect(
    host="127.0.0.1",
    port=4000,
    user="root",
    password="",
    database="bookshop",
    autocommit=True
)
with connection:
    with connection.cursor() as cursor:
```

```
        start_time = datetime.datetime(2022, 4, 15)
        end_time = datetime.datetime(2022, 4, 15, 0, 15)
        affect_rows = -1
        while affect_rows != 0:
            delete_sql = "DELETE FROM `bookshop`.`ratings` WHERE `rated_at`
                ↪ >= %s AND `rated_at` <= %s LIMIT 1000"
            affect_rows = cursor.execute(delete_sql, (start_time, end_time))
            print(f'delete {affect_rows} data')
            time.sleep(1)
```

In each iteration, `DELETE` deletes up to 1000 rows from `2022-04-15 00:00:00` to `2022-04-15 00:15:00`.

4.6.3.7 Non-transactional bulk-delete

> **Note:**
>
> Since v6.1.0, TiDB supports the non-transactional DML statements. This feature is not available for versions earlier than TiDB v6.1.0.

4.6.3.7.1 Prerequisites of non-transactional bulk-delete

Before using the non-transactional bulk-delete, make sure you have read the Non-Transactional DML statements documentation first. The non-transactional bulk-delete improves the performance and ease of use in batch data processing scenarios but compromises transactional atomicity and isolation.

Therefore, you should use it carefully to avoid serious consequences (such as data loss) due to mishandling.

4.6.3.7.2 SQL syntax for non-transactional bulk-delete

The SQL syntax for non-transactional bulk-delete statement is as follows:

```
BATCH ON {shard_column} LIMIT {batch_size} {delete_statement};
```

Parameter Name	Description
`{shard_column}`	The column used to divide batches.
`{batch_size}`	Control the size of each batch.
`{delete_statement}`	The `DELETE` statement.

The preceding example only shows a simple use case of a non-transactional bulk-delete

statement. For detailed information, see Non-transactional DML Statements.

4.6.3.7.3 Example of non-transactional bulk-delete

In the same scenario as the Bulk-delete example, the following SQL statement shows how to perform a non-transactional bulk-delete:

```
BATCH ON `rated_at` LIMIT 1000 DELETE FROM `ratings` WHERE `rated_at` >= "
    ↪ 2022-04-15 00:00:00" AND `rated_at` <= "2022-04-15 00:15:00";
```

4.6.4 Prepared Statements

A prepared statement templatizes multiple SQL statements in which only parameters are different. It separates the SQL statements from the parameters. You can use it to improve the following aspects of SQL statements:

- **Security**: Because parameters and statements are separated, the risk of SQL injection attacks is avoided.
- **Performance**: Because the statement is parsed in advance on the TiDB server, only parameters are passed for subsequent executions, saving the cost of parsing the entire SQL statements, splicing SQL statement strings, and network transmission.

In most applications, SQL statements can be enumerated. You can use a limited number of SQL statements to complete data queries for the entire application. So using a prepared statement is a best practice.

4.6.4.1 SQL syntax

This section describes the SQL syntax for creating, running and deleting a prepared statement.

4.6.4.1.1 Create a prepared statement

```
PREPARE {prepared_statement_name} FROM '{prepared_statement_sql}';
```

Parameter Name	Description
{prepared_statement_name}	name of the prepared statement
{prepared_statement_sql}	the prepared statement SQL with a question mark as a placeholder

See PREPARE statement for more information.

4.6.4.1.2 Use the prepared statement

A prepared statement can only use **user variables** as parameters, so use the SET statement to set the variables before the EXECUTE statement can call the prepared statement.

```
SET @{parameter_name} = {parameter_value};
EXECUTE {prepared_statement_name} USING @{parameter_name};
```

Parameter Name	Description
{parameter_name}	user variable name
{parameter_value}	user variable value
{ prepared_statement_name }	The name of the preprocessing statement, which must be the same as the name defined in the Create a prepared statement

See the EXECUTE statement for more information.

4.6.4.1.3 Delete the prepared statement

```
DEALLOCATE PREPARE {prepared_statement_name};
```

Parameter Name	Description
{ prepared_statement_name }	The name of the preprocessing statement, which must be the same as the name defined in the Create a prepared statement

See the DEALLOCATE statement for more information.

4.6.4.2 Examples

This section describes two examples of prepared statements: SELECT data and INSERT data.

4.6.4.2.1 SELECT example

For example, you need to query a book with id = 1 in the bookshop application.

```
PREPARE `books_query` FROM 'SELECT * FROM `books` WHERE `id` = ?';
```

Running result:

```
Query OK, 0 rows affected (0.01 sec)
```

```
SET @id = 1;
```

Running result:

```
Query OK, 0 rows affected (0.04 sec)
```

```
EXECUTE `books_query` USING @id;
```

Running result:

```
+---------+----------------------------------+--------+---------------------+-------+----
    ↪
| id      | title                            | type   | published_at        | stock |
    ↪ price |
+---------+----------------------------------+--------+---------------------+-------+----
    ↪
| 1       | The Adventures of Pierce Wehner | Comics | 1904-06-06 20:46:25 |
    ↪ 586 | 411.66 |
+---------+----------------------------------+--------+---------------------+-------+----
    ↪
1 row in set (0.05 sec)
```

```
// ds is an entity of com.mysql.cj.jdbc.MysqlDataSource
try (Connection connection = ds.getConnection()) {
    PreparedStatement preparedStatement = connection.prepareStatement("
        ↪ SELECT * FROM `books` WHERE `id` = ?");
    preparedStatement.setLong(1, 1);

    ResultSet res = preparedStatement.executeQuery();
    if(!res.next()) {
        System.out.println("No books in the table with id 1");
    } else {
        // got book's info, which id is 1
        System.out.println(res.getLong("id"));
        System.out.println(res.getString("title"));
        System.out.println(res.getString("type"));
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

4.6.4.2.2 `INSERT` example

Using the books table as an example, you need to insert a book with `title = TiDB ↪ Developer Guide`, `type = Science & Technology`, `stock = 100`, `price = 0.0`, and `published_at = NOW()` (current time of insertion). Note that you don't need to specify the `AUTO_RANDOM` attribute in the **primary key** of the `books` table. For more information about inserting data, see Insert Data.

```
PREPARE `books_insert` FROM 'INSERT INTO `books` (`title`, `type`, `stock`,
    ↪  `price`, `published_at`) VALUES (?, ?, ?, ?, ?);';
```

Running result:

```
Query OK, 0 rows affected (0.03 sec)
```

```
SET @title = 'TiDB Developer Guide';
SET @type = 'Science & Technology';
SET @stock = 100;
SET @price = 0.0;
SET @published_at = NOW();
```

Running result:

```
Query OK, 0 rows affected (0.04 sec)
```

```
EXECUTE `books_insert` USING @title, @type, @stock, @price, @published_at;
```

Running result:

```
Query OK, 1 row affected (0.03 sec)
```

```
try (Connection connection = ds.getConnection()) {
    String sql = "INSERT INTO `books` (`title`, `type`, `stock`, `price`, `
        ↪ published_at`) VALUES (?, ?, ?, ?, ?);";
    PreparedStatement preparedStatement = connection.prepareStatement(sql);

    preparedStatement.setString(1, "TiDB Developer Guide");
    preparedStatement.setString(2, "Science & Technology");
    preparedStatement.setInt(3, 100);
    preparedStatement.setBigDecimal(4, new BigDecimal("0.0"));
    preparedStatement.setTimestamp(5, new Timestamp(Calendar.getInstance().
        ↪ getTimeInMillis()));

    preparedStatement.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
```

As you can see, JDBC helps you control the life cycle of prepared statements and you don't need to manually create, use, or delete prepared statements in your application. However, note that because TiDB is compatible with MySQL, the default configuration for using MySQL JDBC Driver on the client-side is not to enable the ***server-side*** prepared statement option, but to use the client-side prepared statement.

The following configurations help you use the TiDB server-side prepared statements under JDBC:

Parameter	Means	Recommended Scenario	Recommended Configuration
`useServerPrepStmts`	Whether to use the server side to enable prepared statements	When you need to use a prepared statement more than once	`true`
`cachePrepStmts`	Whether the client caches prepared statements	`useServerPrepStmts=true` 时	`true`
`prepStmtCacheSqlLimit`	The size of a prepared statement (256 characters by default)	When the prepared statement is greater than 256 characters	Configured according to the actual size of the prepared statement
`prepStmtCacheSize`	The maximum number of prepared statements (25 by default)	When the number of prepared statements is greater than 25	Configured according to the actual number of prepared statements

The following is a typical scenario of JDBC connection string configurations. Host: `127.0.0.1`, Port: `4000`, User name: `root`, Password: null, Default database: `test`:

```
jdbc:mysql://127.0.0.1:4000/test?user=root&useConfigs=maxPerformance&
    ↪ useServerPrepStmts=true&prepStmtCacheSqlLimit=2048&prepStmtCacheSize
    ↪ =256&rewriteBatchedStatements=true&allowMultiQueries=true
```

You can also see the insert rows chapter if you need to change other JDBC parameters when inserting data.

For a complete example in Java, see:

- Build a Simple CRUD App with TiDB and Java - Using JDBC
- Build a Simple CRUD App with TiDB and Java - Using Hibernate
- Build the TiDB Application using Spring Boot

4.7 Read Data

4.7.1 Query Data from a Single Table

This document describes how to use SQL and various programming languages to query data from a single table in a database.

4.7.1.1 Before you begin

The following content takes the Bookshop application as an example to show how to query data from a single table in TiDB.

Before querying data, make sure that you have completed the following steps:

1. Build a TiDB cluster (using TiDB Cloud or TiUP is recommended).
2. Import table schema and sample data of the Bookshop application.
3. Connect to TiDB.

4.7.1.2 Execute a simple query

In the database of the Bookshop application, the `authors` table stores the basic information of authors. You can use the `SELECT ... FROM ...` statement to query data from the database.

Execute the following SQL statement in a MySQL client:

```
SELECT id, name FROM authors;
```

The output is as follows:

```
+------------+--------------------------+
| id         | name                     |
+------------+--------------------------+
|       6357 | Adelle Bosco             |
|     345397 | Chanelle Koepp           |
|     807584 | Clementina Ryan          |
```

```
|     839921 | Gage Huel             |
|     850070 | Ray Armstrong         |
|     850362 | Ford Waelchi          |
|     881210 | Jayme Gutkowski       |
|    1165261 | Allison Kuvalis       |
|    1282036 | Adela Funk            |
...
| 4294957408 | Lyla Nitzsche         |
+------------+-----------------------+
20000 rows in set (0.05 sec)
```

In Java, to store the basic information of authors, you can declare a class `Author`. You should choose appropriate Java data types according to the Data types and Value range in the database. For example:

- Use a variable of type `Int` to store data of type `int`.
- Use a variable of type `Long` to store data of type `bigint`.
- Use a variable of type `Short` to store data of type `tinyint`.
- Use a variable of type `String` to store data of type `varchar`.

```java
public class Author {
    private Long id;
    private String name;
    private Short gender;
    private Short birthYear;
    private Short deathYear;

    public Author() {}

     // Skip the getters and setters.
}
```

```java
public class AuthorDAO {

    // Omit initialization of instance variables.

    public List<Author> getAuthors() throws SQLException {
        List<Author> authors = new ArrayList<>();

        try (Connection conn = ds.getConnection()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT id, name FROM authors");
            while (rs.next()) {
                Author author = new Author();
```

```
                author.setId(rs.getLong("id"));
                author.setName(rs.getString("name"));
                authors.add(author);
            }
        }
        return authors;
    }
}
```

- After connecting to TiDB using the JDBC driver, you can create a `Statement` object with `conn.createStatus()`.

- Then call `stmt.executeQuery("query_sql")` to initiate a database query request to TiDB.
- The query results are stored in a `ResultSet` object. By traversing `ResultSet`, the returned results can be mapped to the `Author` object.

4.7.1.3 Filter results

To filter query results, you can use the `WHERE` statement.

For example, the following command queries authors who were born in 1998 among all authors:

Add filter conditions in the `WHERE` statement:

```
SELECT * FROM authors WHERE birth_year = 1998;
```

In Java, you can use the same SQL to handle data query requests with dynamic parameters.

This can be done by concatenating parameters into a SQL statement. However, this method poses a potential SQL Injection risk to the security of the application.

To deal with such queries, use a Prepared statement instead of a normal statement.

```
public List<Author> getAuthorsByBirthYear(Short birthYear) throws
    ↪ SQLException {
    List<Author> authors = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        PreparedStatement stmt = conn.prepareStatement("""
        SELECT * FROM authors WHERE birth_year = ?;
        """);
        stmt.setShort(1, birthYear);
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            Author author = new Author();
```

```
            author.setId(rs.getLong("id"));
            author.setName(rs.getString("name"));
            authors.add(author);
        }
    }
    return authors;
}
```

4.7.1.4 Sort results

To sort query results, you can use the `ORDER BY` statement.

For example, the following SQL statement is to get a list of the youngest authors by sorting the `authors` table in descending order (`DESC`) according to the `birth_year` column.

```
SELECT id, name, birth_year
FROM authors
ORDER BY birth_year DESC;
```

```
public List<Author> getAuthorsSortByBirthYear() throws SQLException {
    List<Author> authors = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("""
            SELECT id, name, birth_year
            FROM authors
            ORDER BY birth_year DESC;
            """);

        while (rs.next()) {
            Author author = new Author();
            author.setId(rs.getLong("id"));
            author.setName(rs.getString("name"));
            author.setBirthYear(rs.getShort("birth_year"));
            authors.add(author);
        }
    }
    return authors;
}
```

The result is as follows:

```
+----------+----------------------+------------+
| id       | name                 | birth_year |
+----------+----------------------+------------+
| 83420726 | Terrance Dach        | 2000       |
```

```
| 57938667 | Margarita Christiansen | 2000  |
| 77441404 | Otto Dibbert           | 2000  |
| 61338414 | Danial Cormier         | 2000  |
| 49680887 | Alivia Lemke           | 2000  |
| 45460101 | Itzel Cummings         | 2000  |
| 38009380 | Percy Hodkiewicz       | 2000  |
| 12943560 | Hulda Hackett          | 2000  |
| 1294029  | Stanford Herman        | 2000  |
| 111453184 | Jeffrey Brekke        | 2000  |
...
300000 rows in set (0.23 sec)
```

4.7.1.5 Limit the number of query results

To limit the number of query results, you can use the `LIMIT` statement.

```
SELECT id, name, birth_year
FROM authors
ORDER BY birth_year DESC
LIMIT 10;
```

```
public List<Author> getAuthorsWithLimit(Integer limit) throws SQLException
    ↪ {
    List<Author> authors = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        PreparedStatement stmt = conn.prepareStatement("""
            SELECT id, name, birth_year
            FROM authors
            ORDER BY birth_year DESC
            LIMIT ?;
            """);
        stmt.setInt(1, limit);
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            Author author = new Author();
            author.setId(rs.getLong("id"));
            author.setName(rs.getString("name"));
            author.setBirthYear(rs.getShort("birth_year"));
            authors.add(author);
        }
    }
    return authors;
}
```

The result is as follows:

```
+----------+-----------------------+------------+
| id       | name                  | birth_year |
+----------+-----------------------+------------+
| 83420726 | Terrance Dach         | 2000       |
| 57938667 | Margarita Christiansen | 2000      |
| 77441404 | Otto Dibbert          | 2000       |
| 61338414 | Danial Cormier        | 2000       |
| 49680887 | Alivia Lemke          | 2000       |
| 45460101 | Itzel Cummings        | 2000       |
| 38009380 | Percy Hodkiewicz      | 2000       |
| 12943560 | Hulda Hackett         | 2000       |
| 1294029  | Stanford Herman       | 2000       |
| 111453184 | Jeffrey Brekke       | 2000       |
+----------+-----------------------+------------+
10 rows in set (0.11 sec)
```

With the `LIMIT` statement, the query time is significantly reduced from `0.23 sec` to `0.11 sec` in this example. For more information, see TopN and Limit.

4.7.1.6 Aggregate queries

To have a better understanding of the overall data situation, you can use the `GROUP BY` statement to aggregate query results.

For example, if you want to know which years there are more authors born, you can group the `authors` table by the `birth_year` column, and then count for each year:

```
SELECT birth_year, COUNT (DISTINCT id) AS author_count
FROM authors
GROUP BY birth_year
ORDER BY author_count DESC;
```

```
public class AuthorCount {
    private Short birthYear;
    private Integer authorCount;

    public AuthorCount() {}

     // Skip the getters and setters.
}

public List<AuthorCount> getAuthorCountsByBirthYear() throws SQLException {
    List<AuthorCount> authorCounts = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        Statement stmt = conn.createStatement();
```

```
        ResultSet rs = stmt.executeQuery("""
            SELECT birth_year, COUNT(DISTINCT id) AS author_count
            FROM authors
            GROUP BY birth_year
            ORDER BY author_count DESC;
            """);

        while (rs.next()) {
            AuthorCount authorCount = new AuthorCount();
            authorCount.setBirthYear(rs.getShort("birth_year"));
            authorCount.setAuthorCount(rs.getInt("author_count"));
            authorCounts.add(authorCount);
        }
    }
    return authorCount;
}
```

The result is as follows:

```
+------------+--------------+
| birth_year | author_count |
+------------+--------------+
|       1932 |          317 |
|       1947 |          290 |
|       1939 |          282 |
|       1935 |          289 |
|       1968 |          291 |
|       1962 |          261 |
|       1961 |          283 |
|       1986 |          289 |
|       1994 |          280 |
...
|       1972 |          306 |
+------------+--------------+
71 rows in set (0.00 sec)
```

In addition to the `COUNT` function, TiDB also supports other aggregate functions. For more information, see Aggregate (GROUP BY) Functions.

4.7.2 Multi-table Join Queries

In many scenarios, you need to use one query to get data from multiple tables. You can use the `JOIN` statement to combine the data from two or more tables.

4.7.2.1 Join types

This section describes the Join types in detail.

4.7.2.1.1 INNER JOIN

The join result of an inner join returns only rows that match the join condition.

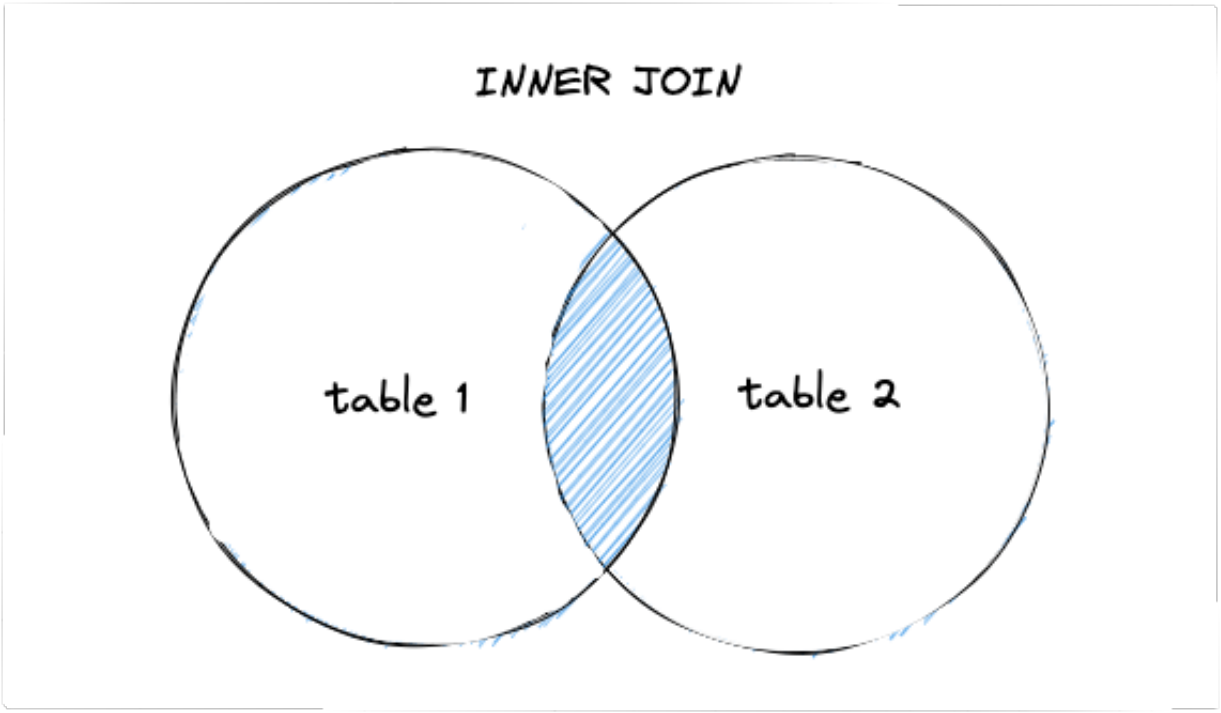


Figure 14: Inner Join

For example, if you want to know the most prolific author, you need to join the author table named `authors` with the book author table named `book_authors`.

In the following SQL statement, use the keyword `JOIN` to declare that you want to join the rows of the left table `authors` and the right table `book_authors` as an inner join with the join condition `a.id = ba.author_id`. The result set will only contain rows that satisfy the join condition. If an author has not written any books, then his record in `authors` table will not satisfy the join condition and will therefore not appear in the result set.

```
SELECT ANY_VALUE(a.id) AS author_id, ANY_VALUE(a.name) AS author_name,
    ↪ COUNT(ba.book_id) AS books
FROM authors a
JOIN book_authors ba ON a.id = ba.author_id
GROUP BY ba.author_id
ORDER BY books DESC
LIMIT 10;
```

The query results are as follows:

```
+------------+---------------+-------+
| author_id | author_name  | books |
+------------+---------------+-------+
|  431192671 | Emilie Cassin |  7 |
|  865305676 | Nola Howell |    7 |
|  572207928 | Lamar Koch  |    6 |
| 3894029860 | Elijah Howe |    6 |
| 1150614082 | Cristal Stehr |  6 |
| 4158341032 | Roslyn Rippin |  6 |
| 2430691560 | Francisca Hahn | 6 |
| 3346415350 | Leta Weimann |   6 |
| 1395124973 | Albin Cole |    6 |
| 2768150724 | Caleb Wyman |    6 |
+------------+---------------+-------+
10 rows in set (0.01 sec)
```

```
public List<Author> getTop10AuthorsOrderByBooks() throws SQLException {
    List<Author> authors = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("""
        SELECT ANY_VALUE(a.id) AS author_id, ANY_VALUE(a.name) AS author_name
            ↪ , COUNT(ba.book_id) AS books
        FROM authors a
        JOIN book_authors ba ON a.id = ba.author_id
        GROUP BY ba.author_id
        ORDER BY books DESC
        LIMIT 10;
        """);
        while (rs.next()) {
            Author author = new Author();
            author.setId(rs.getLong("author_id"));
            author.setName(rs.getString("author_name"));
            author.setBooks(rs.getInt("books"));
            authors.add(author);
        }
    }
    return authors;
}
```

4.7.2.1.2 LEFT OUTER JOIN

The left outer join returns all the rows in the left table and the values in the right table

that match the join condition. If no rows are matched in the right table, it will be filled with `NULL`.

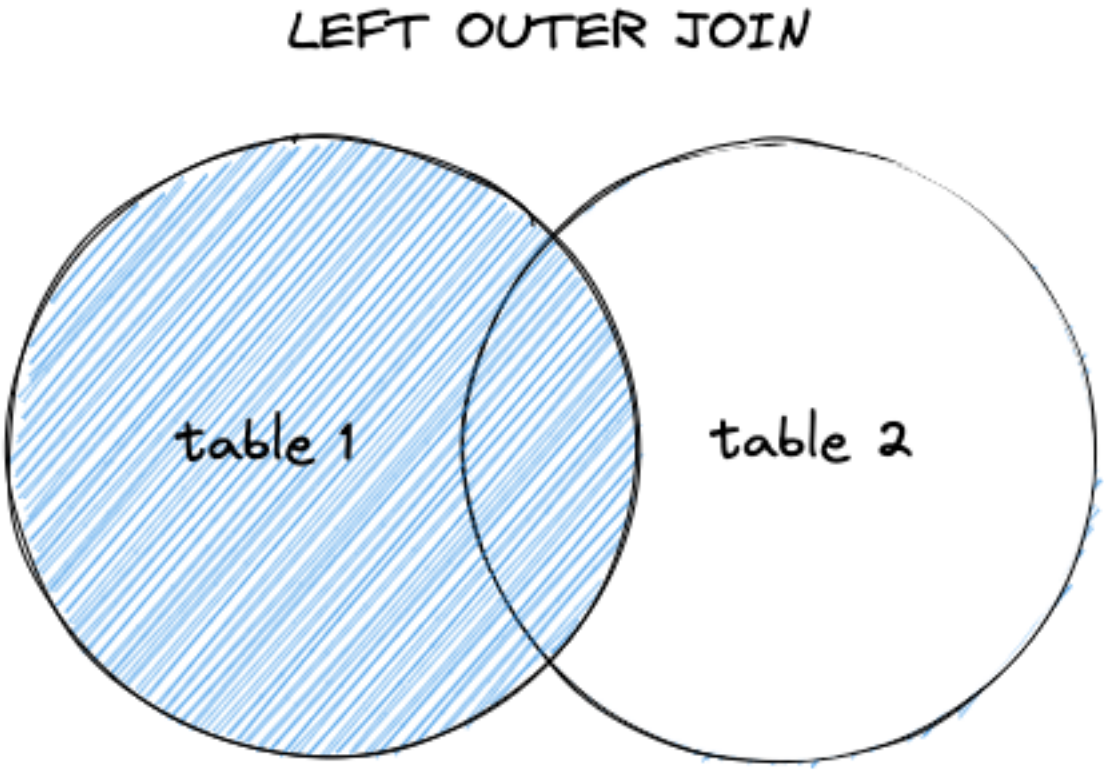


Figure 15: Left Outer Join

In some cases, you want to use multiple tables to complete the data query, but do not want the data set to become too small because the join condition are not met.

For example, on the homepage of the Bookshop app, you want to display a list of new books with average ratings. In this case, the new books may not have been rated by anyone yet. Using inner joins will cause the information of these unrated books to be filtered out, which is not what you expect.

In the following SQL statement, use the `LEFT JOIN` keyword to declare that the left table `books` will be joined to the right table `ratings` in a left outer join, thus ensuring that all rows in the `books` table are returned.

```
SELECT b.id AS book_id, ANY_VALUE(b.title) AS book_title, AVG(r.score) AS
    ↪ average_score
FROM books b
LEFT JOIN ratings r ON b.id = r.book_id
GROUP BY b.id
ORDER BY b.published_at DESC
LIMIT 10;
```

The query results are as follows:

```
+------------+---------------------------------+---------------+
| book_id    | book_title                      | average_score |
+------------+---------------------------------+---------------+
| 3438991610 | The Documentary of lion         |        2.7619 |
| 3897175886 | Torey Kuhn                      |        3.0000 |
| 1256171496 | Elmo Vandervort                 |        2.5500 |
| 1036915727 | The Story of Munchkin           |        2.0000 |
|  270254583 | Tate Kovacek                    |        2.5000 |
| 1280950719 | Carson Damore                   |        3.2105 |
| 1098041838 | The Documentary of grasshopper  |   2.8462 |
| 1476566306 | The Adventures of Vince Sanford | 2.3529 |
| 4036300890 | The Documentary of turtle       |        2.4545 |
| 1299849448 | Antwan Olson                    |        3.0000 |
+------------+---------------------------------+---------------+
10 rows in set (0.30 sec)
```

It seems that the latest published book already has a lot of ratings. To verify the above method, let's delete all the ratings of the book *The Documentary of lion* through the SQL statement:

```
DELETE FROM ratings WHERE book_id = 3438991610;
```

Query again. The book *The Documentary of lion* still appears in the result set, but the `average_score` column calculated from `score` of the right table `ratings` is filled with `NULL`.

```
+------------+---------------------------------+---------------+
| book_id    | book_title                      | average_score |
+------------+---------------------------------+---------------+
| 3438991610 | The Documentary of lion         |          NULL |
| 3897175886 | Torey Kuhn                      |        3.0000 |
| 1256171496 | Elmo Vandervort                 |        2.5500 |
| 1036915727 | The Story of Munchkin           |        2.0000 |
|  270254583 | Tate Kovacek                    |        2.5000 |
| 1280950719 | Carson Damore                   |        3.2105 |
| 1098041838 | The Documentary of grasshopper  |   2.8462 |
| 1476566306 | The Adventures of Vince Sanford | 2.3529 |
| 4036300890 | The Documentary of turtle       |        2.4545 |
| 1299849448 | Antwan Olson                    |        3.0000 |
+------------+---------------------------------+---------------+
10 rows in set (0.30 sec)
```

What happens if you use `INNER JOIN`? It's up to you to have a try.

```
public List<Book> getLatestBooksWithAverageScore() throws SQLException {
    List<Book> books = new ArrayList<>();
```

```
    try (Connection conn = ds.getConnection()) {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("""
        SELECT b.id AS book_id, ANY_VALUE(b.title) AS book_title, AVG(r.score
            ↪ ) AS average_score
        FROM books b
        LEFT JOIN ratings r ON b.id = r.book_id
        GROUP BY b.id
        ORDER BY b.published_at DESC
        LIMIT 10;
        """);
        while (rs.next()) {
            Book book = new Book();
            book.setId(rs.getLong("book_id"));
            book.setTitle(rs.getString("book_title"));
            book.setAverageScore(rs.getFloat("average_score"));
            books.add(book);
        }
    }
    return books;
}
```

4.7.2.1.3 RIGHT OUTER JOIN

A right outer join returns all the records in the right table and the values in the left table that match the join condition. If there is no matching value, it is filled with `NULL`.

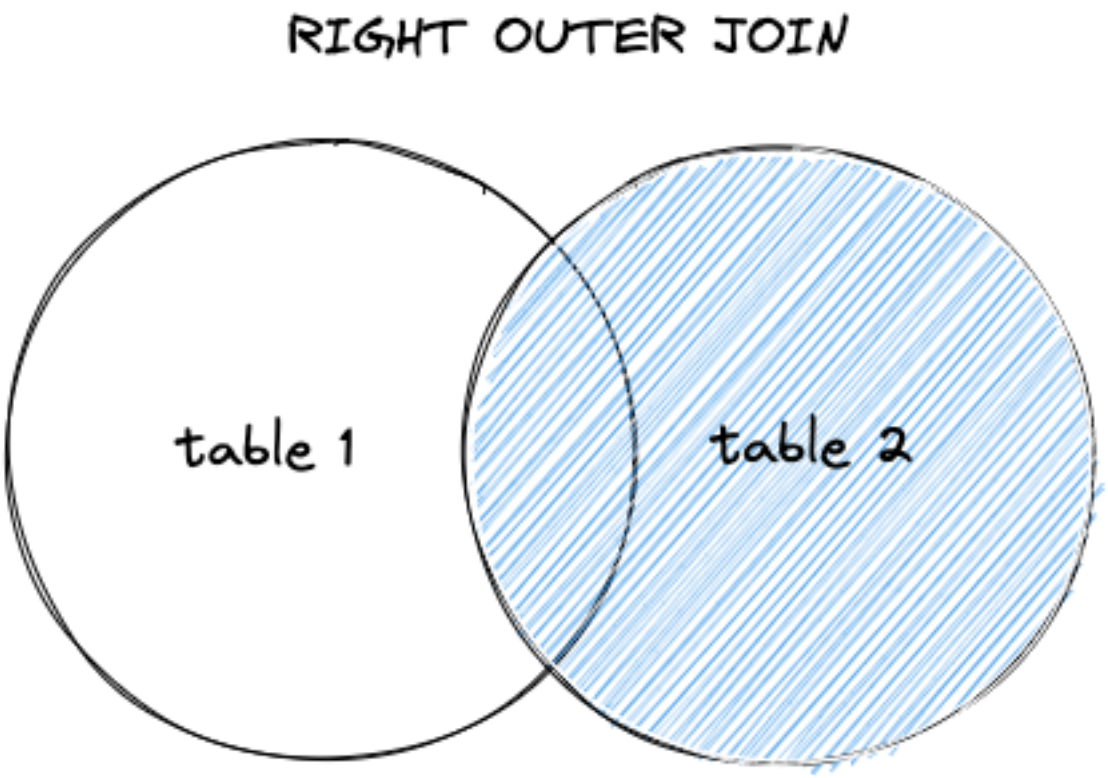


Figure 16: Right Outer Join

4.7.2.1.4 CROSS JOIN

When the join condition is constant, the inner join between the two tables is called a cross join. A cross join joins every record of the left table to all the records of the right table. If the number of records in the left table is `m` and the number of records in the right table is `n`, then `m \* n` records will be generated in the result set.

4.7.2.1.5 LEFT SEMI JOIN

TiDB does not support `LEFT SEMI JOIN table_name` at the SQL syntax level. But at the execution plan level, subquery-related optimizations will use `semi join` as the default join method for rewritten equivalent JOIN queries.

4.7.2.2 Implicit join

Before the `JOIN` statement that explicitly declared a join was added to the SQL standard, it was possible to join two or more tables in a SQL statement using the `FROM t1, t2` clause, and specify the conditions for the join using the `WHERE t1.id = t2.id` clause. You can understand it as an implicit join, which uses the inner join to join tables.

4.7.2.3 Join related algorithms

TiDB supports the following general table join algorithms.

- Index Join
- Hash Join
- Merge Join

The optimizer selects an appropriate join algorithm to execute based on the factors such as the data volume in the joined table. You can see which algorithm the query uses for Join by using the `EXPLAIN` statement.

If the optimizer of TiDB does not execute according to the optimal join algorithm, you can use Optimizer Hints to force TiDB to use a better join algorithm.

For example, assuming the example for the left join query above executes faster using the Hash Join algorithm, which is not chosen by the optimizer, you can append the hint `/*+ HASH_JOIN(b, r)*/` after the `SELECT` keyword. Note that If the table has an alias, use the alias in the hint.

```
EXPLAIN SELECT /*+ HASH_JOIN(b, r) */ b.id AS book_id, ANY_VALUE(b.title)
    ↪ AS book_title, AVG(r.score) AS average_score
FROM books b
LEFT JOIN ratings r ON b.id = r.book_id
GROUP BY b.id
ORDER BY b.published_at DESC
LIMIT 10;
```

Hints related to join algorithms:

- MERGE_JOIN(t1_name [, tl_name ...])
- INL_JOIN(t1_name [, tl_name ...])
- INL_HASH_JOIN(t1_name [, tl_name ...])
- HASH_JOIN(t1_name [, tl_name ...])

4.7.2.4 Join orders

In real business scenarios, join statements of multiple tables are very common. The execution efficiency of join is related to the order of each table in join. TiDB uses the Join Reorder algorithm to determine the order in which multiple tables are joined.

If the join order selected by the optimizer is not optimal as expected, you can use `STRAIGHT_JOIN` to enforce TiDB to join queries in the order of the tables used in the `FROM` clause.

```
EXPLAIN SELECT *
FROM authors a STRAIGHT_JOIN book_authors ba STRAIGHT_JOIN books b
WHERE b.id = ba.book_id AND ba.author_id = a.id;
```

For more information about the implementation details and limitations of this Join Reorder algorithm, see Introduction to Join Reorder Algorithm.

4.7.2.5 See also

- Explain Statements That Use Joins
- Introduction to Join Reorder

4.7.3 Subquery

This document introduces subquery statements and categories in TiDB.

4.7.3.1 Overview

An subquery is a query within another SQL query. With subquery, the query result can be used in another query.

The following takes the Bookshop application as an example to introduce subquery.

4.7.3.2 Subquery statement

In most cases, there are five types of subqueries:

- Scalar Subquery, such as `SELECT (SELECT s1 FROM t2)FROM t1`.
- Derived Tables, such as `SELECT t1.s1 FROM (SELECT s1 FROM t2)t1`.
- Existential Test, such as `WHERE NOT EXISTS(SELECT ... FROM t2)`, `WHERE t1.a IN (SELECT ... FROM t2)`.
- Quantified Comparison, such as `WHERE t1.a = ANY(SELECT ... FROM t2)`, `WHERE t1.a = ANY(SELECT ... FROM t2)`.
- Subquery as a comparison operator operand, such as `WHERE t1.a > (SELECT ... FROM t2)`.

4.7.3.3 Category of subquery

The subquery can be categorized as Correlated Subquery and Self-contained Subquery. TiDB treats these two types differently.

Whether a subquery is correlated or not depends on whether it refers to columns used in its outer query.

4.7.3.3.1 Self-contained subquery

For a self-contained subquery that uses subquery as operand of comparison operators (`>`, `>=`, `<` , `<=` , `=` , or `!=`), the inner subquery queries only once, and TiDB rewrites it as a constant during the execution plan phase.

For example, to query authors in the `authors` table whose age is greater than the average age, you can use a subquery as a comparison operator operand.

```
SELECT * FROM authors a1 WHERE (IFNULL(a1.death_year, YEAR(NOW())) - a1.
    ↪ birth_year) > (
    SELECT
        AVG(IFNULL(a2.death_year, YEAR(NOW())) - a2.birth_year) AS
            ↪ average_age
    FROM
        authors a2
)
```

The inner subquery is executed before TiDB executes the above query:

```
SELECT AVG(IFNULL(a2.death_year, YEAR(NOW())) - a2.birth_year) AS
    ↪ average_age FROM authors a2;
```

Suppose the result of the query is 34, that is, the average age is 34, and 34 will be used as a constant to replace the original subquery.

```
SELECT * FROM authors a1
WHERE (IFNULL(a1.death_year, YEAR(NOW())) - a1.birth_year) > 34;
```

The result is as follows:

```
+--------+------------------+--------+------------+------------+
| id     | name             | gender | birth_year | death_year |
+--------+------------------+--------+------------+------------+
| 13514 | Kennith Kautzer | 1     | 1956      | 2018      |
| 13748 | Dillon Langosh  | 1     | 1985      | NULL      |
| 99184 | Giovanny Emmerich | 1   | 1954      | 2012      |
| 180191 | Myrtie Robel   | 1     | 1958      | 2009      |
| 200969 | Iva Renner     | 0     | 1977      | NULL      |
| 209671 | Abraham Ortiz  | 0     | 1943      | 2016      |
| 229908 | Wellington Wiza | 1    | 1932      | 1969      |
| 306642 | Markus Crona   | 0     | 1969      | NULL      |
| 317018 | Ellis McCullough | 0   | 1969      | 2014      |
| 322369 | Mozelle Hand   | 0     | 1942      | 1977      |
| 325946 | Elta Flatley   | 0     | 1933      | 1986      |
| 361692 | Otho Langosh   | 1     | 1931      | 1997      |
| 421294 | Karelle VonRueden | 0  | 1977      | NULL      |
...
```

For self-contained subqueries such as Existential Test and Quantified Comparison, TiDB rewrites and replaces them with equivalent queries for better performance. For more information, see Subquery Related Optimizations.

4.7.3.3.2 Correlated subquery

For correlated subquery, because the inner subquery references the columns from the outer query, each subquery is executed once for each row of the outer query. That is, assuming that the outer query gets 10 million results, the subquery will also be executed 10 million times, which will consume more time and resources.

Therefore, in the process of processing, TiDB will try to Decorrelate of Correlated Subquery to improve the query efficiency at the execution plan level.

The following statement is to query authors who are older than the average age of other authors of the same gender.

```
SELECT * FROM authors a1 WHERE (IFNULL(a1.death_year, YEAR(NOW())) - a1.
    ↪ birth_year) > (
    SELECT
        AVG(
            IFNULL(a2.death_year, YEAR(NOW())) - IFNULL(a2.birth_year, YEAR(
                ↪ NOW()))
        ) AS average_age
    FROM
        authors a2
    WHERE a1.gender = a2.gender
);
```

TiDB rewrites it to an equivalent `join` query:

```
SELECT *
FROM
    authors a1,
    (
        SELECT
            gender, AVG(
                IFNULL(a2.death_year, YEAR(NOW())) - IFNULL(a2.birth_year,
                    ↪ YEAR(NOW()))
            ) AS average_age
        FROM
            authors a2
        GROUP BY gender
    ) a2
WHERE
    a1.gender = a2.gender
    AND (IFNULL(a1.death_year, YEAR(NOW())) - a1.birth_year) > a2.
        ↪ average_age;
```

As a best practice, in actual development, it is recommended to avoid querying through a correlated subquery if you can write another equivalent query with better performance.

4.7.3.4 Read more

- Subquery Related Optimizations
- Decorrelation of Correlated Subquery
- Subquery Optimization in TiDB

4.7.4 Paginate Results

To page through a large query result, you can get your desired part in a "paginated" manner.

4.7.4.1 Paginate query results

In TiDB, you can paginate query results using the `LIMIT` statement. For example:

```
SELECT * FROM table_a t ORDER BY gmt_modified DESC LIMIT offset, row_count;
```

`offset` indicates the beginning number of records and `row_count` indicates the number of records per page. TiDB also supports `LIMIT row_count OFFSET offset` syntax.

When pagination is used, it is recommended that you sort query results with the `ORDER BY` statement unless there is a need to display data randomly.

For example, to let users of the Bookshop application view the latest published books in a paginated manner, you can use the `LIMIT 0, 10` statement, which returns the first page of the result list, with a maximum of 10 records per page. To get the second page, you can change the statement to `LIMIT 10, 10`.

```
SELECT *
FROM books
ORDER BY published_at DESC
LIMIT 0, 10;
```

In application development, the backend program receives the `page_number` parameter (which means the number of the page being requested) and the `page_size` parameter (which controls how many records per page) from the frontend instead of the `offset` parameter. Therefore, some conversions needed to be done before querying.

```
public List<Book> getLatestBooksPage(Long pageNumber, Long pageSize) throws
    ↪ SQLException {
    pageNumber = pageNumber < 1L ? 1L : pageNumber;
    pageSize = pageSize < 10L ? 10L : pageSize;
    Long offset = (pageNumber - 1) * pageSize;
    Long limit = pageSize;
    List<Book> books = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        PreparedStatement stmt = conn.prepareStatement("""
        SELECT id, title, published_at
        FROM books
        ORDER BY published_at DESC
```

```
            LIMIT ?, ?;
            """);
            stmt.setLong(1, offset);
            stmt.setLong(2, limit);
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Book book = new Book();
                book.setId(rs.getLong("id"));
                book.setTitle(rs.getString("title"));
                book.setPublishedAt(rs.getDate("published_at"));
                books.add(book);
            }
        }
        return books;
    }
```

4.7.4.2 Paging batches for single-field primary key tables

Usually, you can write a pagination SQL statement using a primary key or unique index to sort results and the `offset` keyword in the `LIMIT` clause to split pages by a specified row count. Then the pages are wrapped into independent transactions to achieve flexible paging updates. However, the disadvantage is also obvious. As the primary key or unique index needs to be sorted, a larger offset consumes more computing resources, especially in the case of a large volume of data.

The following introduces a more efficient paging batching method:

First, sort the data by primary key and call the window function `row_number()` to generate a row number for each row. Then, call the aggregation function to group row numbers by the specified page size and calculate the minimum and maximum values of each page.

```
SELECT
    floor((t.row_num - 1) / 1000) + 1 AS page_num,
    min(t.id) AS start_key,
    max(t.id) AS end_key,
    count(*) AS page_size
FROM (
    SELECT id, row_number() OVER (ORDER BY id) AS row_num
    FROM books
) t
GROUP BY page_num
ORDER BY page_num;
```

The result is as follows:

```
+----------+------------+------------+-----------+
```

```
| page_num | start_key | end_key | page_size |
+----------+-----------+-----------+-----------+
|        1 |    268996 | 213168525 |    1000 |
|        2 |  213210359 | 430012226 |    1000 |
|        3 |  430137681 | 647846033 |    1000 |
|        4 |  647998334 | 848878952 |    1000 |
|        5 |  848899254 | 1040978080 |   1000 |
...
|       20 | 4077418867 | 4294004213 |  1000 |
+----------+-----------+-----------+-----------+
20 rows in set (0.01 sec)
```

Next, use the `WHERE id BETWEEN start_key AND end_key` statement to query the data of each slice. To update data more efficiently, you can use the above slice information when modifying the data.

To delete the basic information of all books on page 1, replace the `start_key` and `end_key` with values of page 1 in the above result:

```
DELETE FROM books
WHERE
    id BETWEEN 268996 AND 213168525
ORDER BY id;
```

In Java, define a `PageMeta` class to store page meta information.

```
public class PageMeta<K> {
    private Long pageNum;
    private K startKey;
    private K endKey;
    private Long pageSize;

    // Skip the getters and setters.

}
```

Define a `getPageMetaList()` method to get the page meta information list, and then define a `deleteBooksByPageMeta()` method to delete data in batches according to the page meta information.

```
public class BookDAO {
    public List<PageMeta<Long>> getPageMetaList() throws SQLException {
        List<PageMeta<Long>> pageMetaList = new ArrayList<>();
        try (Connection conn = ds.getConnection()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("""
            SELECT
```

```
                floor((t.row_num - 1) / 1000) + 1 AS page_num,
                min(t.id) AS start_key,
                max(t.id) AS end_key,
                count(*) AS page_size
            FROM (
                SELECT id, row_number() OVER (ORDER BY id) AS row_num
                FROM books
            ) t
            GROUP BY page_num
            ORDER BY page_num;
            """);
            while (rs.next()) {
                PageMeta<Long> pageMeta = new PageMeta<>();
                pageMeta.setPageNum(rs.getLong("page_num"));
                pageMeta.setStartKey(rs.getLong("start_key"));
                pageMeta.setEndKey(rs.getLong("end_key"));
                pageMeta.setPageSize(rs.getLong("page_size"));
                pageMetaList.add(pageMeta);
            }
        }
        return pageMetaList;
    }

    public void deleteBooksByPageMeta(PageMeta<Long> pageMeta) throws
        ↪ SQLException {
        try (Connection conn = ds.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("DELETE FROM books
                ↪  WHERE id >= ? AND id <= ?");
            stmt.setLong(1, pageMeta.getStartKey());
            stmt.setLong(2, pageMeta.getEndKey());
            stmt.executeUpdate();
        }
    }
}
```

The following statement is to delete the data on page 1:

```
List<PageMeta<Long>> pageMetaList = bookDAO.getPageMetaList();
if (pageMetaList.size() > 0) {
    bookDAO.deleteBooksByPageMeta(pageMetaList.get(0));
}
```

The following statement is to delete all book data in batches by paging:

```
List<PageMeta<Long>> pageMetaList = bookDAO.getPageMetaList();
pageMetaList.forEach((pageMeta) -> {
```

```
    try {
        bookDAO.deleteBooksByPageMeta(pageMeta);
    } catch (SQLException e) {
        e.printStackTrace();
    }
});
```

This method significantly improves the efficiency of batch processing by avoiding wasting computing resources caused by frequent data sorting operations.

4.7.4.3 Paging batches for composite primary key tables

4.7.4.3.1 Non-clustered index table

For non-clustered index tables (also known as "non-index-organized tables"), the internal field `_tidb_rowid` can be used as a pagination key, and the pagination method is the same as that of single-field primary key tables.

> **Tip:**
>
> You can use the `SHOW CREATE TABLE users;` statement to check whether the table primary key uses clustered index.

For example:

```
SELECT
    floor((t.row_num - 1) / 1000) + 1 AS page_num,
    min(t._tidb_rowid) AS start_key,
    max(t._tidb_rowid) AS end_key,
    count(*) AS page_size
FROM (
    SELECT _tidb_rowid, row_number () OVER (ORDER BY _tidb_rowid) AS row_num
    FROM users
) t
GROUP BY page_num
ORDER BY page_num;
```

The result is as follows:

```
+----------+-----------+---------+-----------+
| page_num | start_key | end_key | page_size |
+----------+-----------+---------+-----------+
|        1 |         1 |    1000 |      1000 |
```

```
|       2 |     1001 |   2000 |     1000 |
|       3 |     2001 |   3000 |     1000 |
|       4 |     3001 |   4000 |     1000 |
|       5 |     4001 |   5000 |     1000 |
|       6 |     5001 |   6000 |     1000 |
|       7 |     6001 |   7000 |     1000 |
|       8 |     7001 |   8000 |     1000 |
|       9 |     8001 |   9000 |     1000 |
|      10 |     9001 |   9990 |      990 |
+---------+----------+--------+----------+
10 rows in set (0.00 sec)
```

4.7.4.3.2 Clustered index table

For clustered index tables (also known as "index-organized tables"), you can use the `concat` function to concatenate values of multiple columns as a key, and then use a window function to query the paging information.

It should be noted that the key is a string at this time, and you must ensure that the length of the string is always the same, to obtain the correct `start_key` and `end_key` in the slice through the `min` and `max` aggregation function. If the length of the field for string concatenation is not fixed, you can use the `LPAD` function to pad it.

For example, you can implement a paging batch for the data in the `ratings` table as follows:

Create the meta information table by using the following statement. As the key concatenated by `book_id` and `user_id`, which are `bigint` types, is unable to convert to the same length, the `LPAD` function is used to pad the length with `0` according to the maximum bits 19 of `bigint`.

```
SELECT
    floor((t1.row_num - 1) / 10000) + 1 AS page_num,
    min(mvalue) AS start_key,
    max(mvalue) AS end_key,
    count(*) AS page_size
FROM (
    SELECT
        concat('(', LPAD(book_id, 19, 0), ',', LPAD(user_id, 19, 0), ')') AS
            ↪  mvalue,
        row_number() OVER (ORDER BY book_id, user_id) AS row_num
    FROM ratings
) t1
GROUP BY page_num
ORDER BY page_num;
```

> **Note:**
>
> The preceding SQL statement is executed as `TableFullScan`. When the data volume is large, the query will be slow, and you can use TiFlash to speed up it.

The result is as follows:

```
+----------+-----------------------------------------+-----------------------------
    ↪
| page_num | start_key                               | end_key
    ↪                                | page_size |
+----------+-----------------------------------------+-----------------------------
    ↪
|        1 | (0000000000000268996,0000000000092104804) |
    ↪ (0000000000140982742,0000000000374645100) | 10000 |
|        2 | (0000000000140982742,0000000000456757551) |
    ↪ (0000000000287195082,0000000004053200550) | 10000 |
|        3 | (0000000000287196791,0000000000191962769) |
    ↪ (0000000000434010216,0000000000237646714) | 10000 |
|        4 | (0000000000434010216,0000000000375066168) |
    ↪ (0000000000578893327,0000000002167504460) | 10000 |
|        5 | (0000000000578893327,0000000002457322286) |
    ↪ (0000000000718287668,0000000001502744628) | 10000 |
...
|       29 | (0000000004002523918,0000000000902930986) |
    ↪ (0000000004147203315,0000000004090920746) | 10000 |
|       30 | (0000000004147421329,0000000000319181561) |
    ↪ (0000000004294004213,0000000003586311166) | 9972 |
+----------+-----------------------------------------+-----------------------------
    ↪
30 rows in set (0.28 sec)
```

To delete all rating records on page 1, replace the `start_key` and `end_key` with values of page 1 in the above result:

```
SELECT * FROM ratings
WHERE
    (book_id > 268996 AND book_id < 140982742)
    OR (
        book_id = 268996 AND user_id >= 92104804
    )
    OR (
```

```
        book_id = 140982742 AND user_id <= 374645100
    )
ORDER BY book_id, user_id;
```

4.7.5 Views

This document describes how to use views in TiDB.

4.7.5.1 Overview

TiDB supports views. A view acts as a virtual table, whose schema is defined by the `SELECT` statement that creates the view.

- You can create views to expose only safe fields and data to users, which ensures the security of sensitive fields and data in the underlying tables.
- You can create views for complex queries that are frequently used to make complex queries easier and more convenient.

4.7.5.2 Create a view

In TiDB, a complex query can be defined as a view with the `CREATE VIEW` statement. The syntax is as follows:

```
CREATE VIEW view_name AS query;
```

Note that you cannot create a view with the same name as an existing view or table.

For example, the multi-table join query gets a list of books with average ratings by joining the `books` table and the `ratings` table through a `JOIN` statement.

For the convenience of subsequent queries, you can define the query as a view using the following statement:

```
CREATE VIEW book_with_ratings AS
SELECT b.id AS book_id, ANY_VALUE(b.title) AS book_title, AVG(r.score) AS
    ↪ average_score
FROM books b
LEFT JOIN ratings r ON b.id = r.book_id
GROUP BY b.id;
```

4.7.5.3 Query views

Once a view is created, you can use the `SELECT` statement to query the view just like a normal table.

```
SELECT * FROM book_with_ratings LIMIT 10;
```

When TiDB queries a view, it queries the `SELECT` statement associated with the view.

4.7.5.4 Update views

Currently, the view in TiDB does not support the `ALTER VIEW view_name AS query;`, you can "update" a view in the following two ways:

- Delete the old view with the `DROP VIEW view_name;` statement, and then update the view by creating a new view with the `CREATE VIEW view_name AS query;` statement.
- Use the `CREATE OR REPLACE VIEW view_name AS query;` statement to overwrite an existing view with the same name.

```
CREATE OR REPLACE VIEW book_with_ratings AS
SELECT b.id AS book_id, ANY_VALUE(b.title), ANY_VALUE(b.published_at) AS
    ↪ book_title, AVG(r.score) AS average_score
FROM books b
LEFT JOIN ratings r ON b.id = r.book_id
GROUP BY b.id;
```

4.7.5.5 Get view related information

4.7.5.5.1 Using the `SHOW CREATE TABLE|VIEW view_name` statement

```
SHOW CREATE VIEW book_with_ratings\G
```

The result is as follows:

```
*************************** 1. row ***************************
             View: book_with_ratings
      Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`%` SQL
          ↪ SECURITY DEFINER VIEW `book_with_ratings` (`book_id`, `
          ↪ ANY_VALUE(b.title)`, `book_title`, `average_score`) AS SELECT
          ↪  `b`.`id` AS `book_id`,ANY_VALUE(`b`.`title`) AS `ANY_VALUE(b
          ↪ .title)`,ANY_VALUE(`b`.`published_at`) AS `book_title`,AVG(`r
          ↪ `.`score`) AS `average_score` FROM `bookshop`.`books` AS `b`
          ↪ LEFT JOIN `bookshop`.`ratings` AS `r` ON `b`.`id`=`r`.`
          ↪ book_id` GROUP BY `b`.`id`
character_set_client: utf8mb4
collation_connection: utf8mb4_general_ci
1 row in set (0.00 sec)
```

4.7.5.5.2 Query the `INFORMATION_SCHEMA.VIEWS` table

```
SELECT * FROM information_schema.views WHERE TABLE_NAME = '
    ↪ book_with_ratings'\G
```

The result is as follows:

```
*************************** 1. row ***************************
       TABLE_CATALOG: def
        TABLE_SCHEMA: bookshop
          TABLE_NAME: book_with_ratings
     VIEW_DEFINITION: SELECT `b`.`id` AS `book_id`,ANY_VALUE(`b`.`title`) AS
        ↪  `ANY_VALUE(b.title)`,ANY_VALUE(`b`.`published_at`) AS `
        ↪ book_title`,AVG(`r`.`score`) AS `average_score` FROM `bookshop`.`
        ↪ books` AS `b` LEFT JOIN `bookshop`.`ratings` AS `r` ON `b`.`id`=`
        ↪ r`.`book_id` GROUP BY `b`.`id`
        CHECK_OPTION: CASCADED
        IS_UPDATABLE: NO
             DEFINER: root@%
       SECURITY_TYPE: DEFINER
CHARACTER_SET_CLIENT: utf8mb4
COLLATION_CONNECTION: utf8mb4_general_ci
1 row in set (0.00 sec)
```

4.7.5.6 Drop views

Use the `DROP VIEW view_name;` statement to drop a view.

```
DROP VIEW book_with_ratings;
```

4.7.5.7 Limitation

For limitations of views in TiDB, see Limitations of Views.

4.7.5.8 Read More

- Views
- CREATE VIEW Statement
- DROP VIEW Statement
- EXPLAIN Statements Using Views
- TiFlink: Strongly Consistent Materialized Views Using TiKV and Flink

4.7.6 Temporary Tables

Temporary tables can be thought of as a technique for reusing query results.

If you want to know something about the eldest authors in the Bookshop application, you might write multiple queries that use the list of eldest authors.

For example, you can use the following statement to get the top 50 eldest authors from the `authors` table:

```
SELECT a.id, a.name, (IFNULL(a.death_year, YEAR(NOW())) - a.birth_year) AS
    ↪ age
FROM authors a
ORDER BY age DESC
LIMIT 50;
```

The result is as follows:

```
+------------+--------------------+------+
| id         | name               | age |
+------------+--------------------+------+
| 4053452056 | Dessie Thompson |  80 |
| 2773958689 | Pedro Hansen    |  80 |
| 4005636688 | Wyatt Keeling   |  80 |
| 3621155838 | Colby Parker    |  80 |
| 2738876051 | Friedrich Hagenes | 80 |
| 2299112019 | Ray Macejkovic  |  80 |
| 3953661843 | Brandi Williamson | 80 |
...
| 4100546410 | Maida Walsh     |  80 |
+------------+--------------------+------+
50 rows in set (0.01 sec)
```

For the convenience of subsequent queries, you need to cache the result of this query. When using general tables for storage, you should pay attention to how to avoid the table name duplication problem between different sessions, and the need of cleaning up intermediate results in time, as these tables might not be used after a batch query.

4.7.6.1 Create a temporary table

To cache intermediate results, the temporary tables feature is introduced in TiDB v5.3.0. TiDB automatically drops a local temporary table after a session ends, which frees you from worrying about the management trouble caused by increasing intermediate results.

4.7.6.1.1 Types of temporary tables

Temporary tables in TiDB are divided into two types: local temporary tables and global temporary tables.

- For a local temporary table, the table definition and data in the table are visible only to the current session. This type is suitable for temporarily storing intermediate data in the session.
- For a global temporary table, the table definition is visible to the entire TiDB cluster, and the data in the table is visible only to the current transaction. This type is suitable for temporarily storing intermediate data in a transaction.

4.7.6.1.2 Create a local temporary table

Before creating a local temporary table, you need to add `CREATE TEMPORARY TABLES` permission to the current database user.

You can create a temporary table using the `CREATE TEMPORARY TABLE <table_name>` statement. The default type is a local temporary table, which is visible only to the current session.

```
CREATE TEMPORARY TABLE top_50_eldest_authors (
    id BIGINT,
    name VARCHAR(255),
    age INT,
    PRIMARY KEY(id)
);
```

After creating the temporary table, you can use the `INSERT INTO table_name SELECT ...` statement to insert the results of the above query into the temporary table you just created.

```
INSERT INTO top_50_eldest_authors
SELECT a.id, a.name, (IFNULL(a.death_year, YEAR(NOW())) - a.birth_year) AS
    ↪ age
FROM authors a
ORDER BY age DESC
LIMIT 50;
```

The result is as follows:

```
Query OK, 50 rows affected (0.03 sec)
Records: 50 Duplicates: 0 Warnings: 0
```

```
public List<Author> getTop50EldestAuthorInfo() throws SQLException {
    List<Author> authors = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        Statement stmt = conn.createStatement();
        stmt.executeUpdate("""
            CREATE TEMPORARY TABLE top_50_eldest_authors (
                id BIGINT,
                name VARCHAR(255),
                age INT,
                PRIMARY KEY(id)
            );
        """);

        stmt.executeUpdate("""
            INSERT INTO top_50_eldest_authors
```

```
            SELECT a.id, a.name, (IFNULL(a.death_year, YEAR(NOW())) - a.
                ↪ birth_year) AS age
            FROM authors a
            ORDER BY age DESC
            LIMIT 50;
        """);

        ResultSet rs = stmt.executeQuery("""
            SELECT id, name FROM top_50_eldest_authors;
        """);

        while (rs.next()) {
            Author author = new Author();
            author.setId(rs.getLong("id"));
            author.setName(rs.getString("name"));
            authors.add(author);
        }
    }
    return authors;
}
```

4.7.6.1.3 Create a global temporary table

To create a global temporary table, you can add the `GLOBAL` keyword and end with `ON COMMIT DELETE ROWS`, which means the table will be deleted after the current transaction ends.

```
CREATE GLOBAL TEMPORARY TABLE IF NOT EXISTS top_50_eldest_authors_global (
    id BIGINT,
    name VARCHAR(255),
    age INT,
    PRIMARY KEY(id)
) ON COMMIT DELETE ROWS;
```

When inserting data to global temporary tables, you must explicitly declare the start of the transaction via `BEGIN`. Otherwise, the data will be cleared after the `INSERT INTO` statement is executed. Because in the Auto Commit mode, the transaction is automatically committed after the `INSERT INTO` statement is executed, and the global temporary table is cleared when the transaction ends.

When using global temporary tables, you need to turn off Auto Commit mode first. In Java, you can do this with the `conn.setAutoCommit(false);` statement, and you can commit the transaction explicitly with `conn.commit();`. The data added to the global temporary table during the transaction will be cleared after the transaction is committed or canceled.

```
public List<Author> getTop50EldestAuthorInfo() throws SQLException {
    List<Author> authors = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        conn.setAutoCommit(false);

        Statement stmt = conn.createStatement();
        stmt.executeUpdate("""
            CREATE GLOBAL TEMPORARY TABLE IF NOT EXISTS top_50_eldest_authors
                ↪  (
                id BIGINT,
                name VARCHAR(255),
                age INT,
                PRIMARY KEY(id)
            ) ON COMMIT DELETE ROWS;
        """);

        stmt.executeUpdate("""
            INSERT INTO top_50_eldest_authors
            SELECT a.id, a.name, (IFNULL(a.death_year, YEAR(NOW())) - a.
                ↪ birth_year) AS age
            FROM authors a
            ORDER BY age DESC
            LIMIT 50;
        """);

        ResultSet rs = stmt.executeQuery("""
            SELECT id, name FROM top_50_eldest_authors;
        """);

        conn.commit();
        while (rs.next()) {
            Author author = new Author();
            author.setId(rs.getLong("id"));
            author.setName(rs.getString("name"));
            authors.add(author);
        }
    }
    return authors;
}
```

4.7.6.2 View temporary tables

With the `SHOW [FULL] TABLES` statement, you can view a list of existing global temporary tables, but you cannot see any local temporary tables in the list. For now, TiDB does

not have a similar `information_schema.INNODB_TEMP_TABLE_INFO` system table for storing temporary table information.

For example, you can see the global temporary table `top_50_eldest_authors_global` in the table list, but not the `top_50_eldest_authors` table.

```
+------------------------------+------------+
| Tables_in_bookshop           | Table_type |
+------------------------------+------------+
| authors                      | BASE TABLE |
| book_authors                 | BASE TABLE |
| books                        | BASE TABLE |
| orders                       | BASE TABLE |
| ratings                      | BASE TABLE |
| top_50_eldest_authors_global | BASE TABLE |
| users                        | BASE TABLE |
+------------------------------+------------+
9 rows in set (0.00 sec)
```

4.7.6.3 Query a temporary table

Once the temporary table is ready, you can query it as a normal data table:

```
SELECT * FROM top_50_eldest_authors;
```

You can reference data from temporary tables to your query via Multi-table join queries:

```
EXPLAIN SELECT ANY_VALUE(ta.id) AS author_id, ANY_VALUE(ta.age), ANY_VALUE(
    ↪ ta.name), COUNT(*) AS books
FROM top_50_eldest_authors ta
LEFT JOIN book_authors ba ON ta.id = ba.author_id
GROUP BY ta.id;
```

Different from view, querying a temporary table gets data directly from the temporary table instead of executing the original query used in the data insert. In some cases, this can improve the query performance.

4.7.6.4 Drop a temporary table

A local temporary table in a session is automatically dropped after the **session** ends, along with both data and table schema. A global temporary table in a transaction is automatically cleared at the end of the **transaction**, but the table schema remains and needs to be deleted manually.

To manually drop local temporary tables, use the `DROP TABLE` or `DROP TEMPORARY TABLE` syntax. For example:

```
DROP TEMPORARY TABLE top_50_eldest_authors;
```

To manually drop global temporary tables, use the `DROP TABLE` or `DROP GLOBAL TEMPORARY TABLE` syntax. For example:

```
DROP GLOBAL TEMPORARY TABLE top_50_eldest_authors_global;
```

4.7.6.5 Limitation

For limitations of temporary tables in TiDB, see Compatibility restrictions with other TiDB features.

4.7.6.6 Read more

- Temporary Tables

4.7.7 Common Table Expression

In some transaction scenarios, due to application complexity, you might need to write a single SQL statement of up to 2,000 lines. The statement probably contains a lot of aggregations and multi-level subquery nesting. Maintaining such a long SQL statement can be a developer's nightmare.

To avoid such a long SQL statement, you can simplify queries by using Views or cache intermediate query results by using Temporary tables.

This document introduces the Common Table Expression (CTE) syntax in TiDB, which is a more convenient way to reuse query results.

Since TiDB v5.1, TiDB supports the CTE of the ANSI SQL99 standard and recursion. With CTE, you can write SQL statements for complex application logic more efficiently and maintain the code much easier.

4.7.7.1 Basic use

A Common Table Expression (CTE) is a temporary result set that can be referred to multiple times within a SQL statement to improve the statement readability and execution efficiency. You can apply the `WITH` statement to use CTE.

Common Table Expressions can be classified into two types: non-recursive CTE and recursive CTE.

4.7.7.1.1 Non-recursive CTE

Non-recursive CTE can be defined using the following syntax:

```
WITH <query_name> AS (
    <query_definition>
)
SELECT ... FROM <query_name>;
```

For example, if you want to know how many books each of the 50 oldest authors have written, take the following steps:

Change the statement in temporary tables to the following:

```
WITH top_50_eldest_authors_cte AS (
    SELECT a.id, a.name, (IFNULL(a.death_year, YEAR(NOW())) - a.birth_year)
        ↪ AS age
    FROM authors a
    ORDER BY age DESC
    LIMIT 50
)
SELECT
    ANY_VALUE(ta.id) AS author_id,
    ANY_VALUE(ta.age) AS author_age,
    ANY_VALUE(ta.name) AS author_name,
    COUNT(*) AS books
FROM top_50_eldest_authors_cte ta
LEFT JOIN book_authors ba ON ta.id = ba.author_id
GROUP BY ta.id;
```

The result is as follows:

```
+------------+------------+--------------------+-------+
| author_id | author_age | author_name    | books |
+------------+------------+--------------------+-------+
| 1238393239 |      80 | Araceli Purdy    |     1 |
|  817764631 |      80 | Ivory Davis      |     3 |
| 3093759193 |      80 | Lysanne Harris   |     1 |
| 2299112019 |      80 | Ray Macejkovic   |     4 |
...
+------------+------------+--------------------+-------+
50 rows in set (0.01 sec)
```

```
public List<Author> getTop50EldestAuthorInfoByCTE() throws SQLException {
    List<Author> authors = new ArrayList<>();
    try (Connection conn = ds.getConnection()) {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("""
            WITH top_50_eldest_authors_cte AS (
                SELECT a.id, a.name, (IFNULL(a.death_year, YEAR(NOW())) - a.
                    ↪ birth_year) AS age
                FROM authors a
                ORDER BY age DESC
                LIMIT 50
            )
```

```
            SELECT
                ANY_VALUE(ta.id) AS author_id,
                ANY_VALUE(ta.name) AS author_name,
                ANY_VALUE(ta.age) AS author_age,
                COUNT(*) AS books
            FROM top_50_eldest_authors_cte ta
            LEFT JOIN book_authors ba ON ta.id = ba.author_id
            GROUP BY ta.id;
        """);
        while (rs.next()) {
            Author author = new Author();
            author.setId(rs.getLong("author_id"));
            author.setName(rs.getString("author_name"));
            author.setAge(rs.getShort("author_age"));
            author.setBooks(rs.getInt("books"));
            authors.add(author);
        }
    }
    return authors;
}
```

It can be found that the author "Ray Macejkovic" wrote 4 books. With the CTE query, you can further get the order and rating information of these 4 books as follows:

```
WITH books_authored_by_rm AS (
    SELECT *
    FROM books b
    LEFT JOIN book_authors ba ON b.id = ba.book_id
    WHERE author_id = 2299112019
), books_with_average_ratings AS (
    SELECT
        b.id AS book_id,
        AVG(r.score) AS average_rating
    FROM books_authored_by_rm b
    LEFT JOIN ratings r ON b.id = r.book_id
    GROUP BY b.id
), books_with_orders AS (
    SELECT
        b.id AS book_id,
        COUNT(*) AS orders
    FROM books_authored_by_rm b
    LEFT JOIN orders o ON b.id = o.book_id
    GROUP BY b.id
)
SELECT
```

```
    b.id AS `book_id`,
    b.title AS `book_title`,
    br.average_rating AS `average_rating`,
    bo.orders AS `orders`
FROM
    books_authored_by_rm b
    LEFT JOIN books_with_average_ratings br ON b.id = br.book_id
    LEFT JOIN books_with_orders bo ON b.id = bo.book_id
;
```

The result is as follows:

```
+------------+-------------------------+---------------+--------+
| book_id    | book_title              | average_rating | orders |
+------------+-------------------------+---------------+--------+
|  481008467 | The Documentary of goat |    2.0000 |    16 |
| 2224531102 | Brandt Skiles           |        2.7143 |    17 |
| 2641301356 | Sheridan Bashirian      |        2.4211 |    12 |
| 4154439164 | Karson Streich          |        2.5833 |    19 |
+------------+-------------------------+---------------+--------+
4 rows in set (0.06 sec)
```

Three CTE blocks, which are separated by `,`, are defined in this SQL statement.

First, check out the books written by the author (ID is `2299112019`) in the CTE block `books_authored_by_rm`. Then find the average rating and order for these books respectively in `books_with_average_ratings` and `books_with_orders`. Finally, aggregate the results by the `JOIN` statement.

Note that the query in `books_authored_by_rm` executes only once, and then TiDB creates a temporary space to cache its result. When the queries in `books_with_average_ratings` ↪ and `books_with_orders` refer to `books_authored_by_rm`, TiDB gets its result directly from this temporary space.

4.7.7.1.2 Recursive CTE

Recursive CTE can be defined using the following syntax:

```
WITH RECURSIVE <query_name> AS (
    <query_definition>
)
SELECT ... FROM <query_name>;
```

A classic example is to generate a set of Fibonacci numbers with recursive CTE:

```
WITH RECURSIVE fibonacci (n, fib_n, next_fib_n) AS
(
  SELECT 1, 0, 1
```

```
  UNION ALL
  SELECT n + 1, next_fib_n, fib_n + next_fib_n FROM fibonacci WHERE n < 10
)
SELECT * FROM fibonacci;
```

The result is as follows:

```
+------+-------+------------+
| n    | fib_n | next_fib_n |
+------+-------+------------+
|    1 |     0 |          1 |
|    2 |     1 |          1 |
|    3 |     1 |          2 |
|    4 |     2 |          3 |
|    5 |     3 |          5 |
|    6 |     5 |          8 |
|    7 |     8 |         13 |
|    8 |    13 |         21 |
|    9 |    21 |         34 |
|   10 |    34 |         55 |
+------+-------+------------+
10 rows in set (0.00 sec)
```

4.7.7.2 Read more

- WITH

4.7.8 Read Replica Data

4.7.8.1 Follower Read

This document introduces how to use Follower Read to optimize query performance.

4.7.8.1.1 Introduction

TiDB uses Region as the basic unit to distribute data to all nodes in the cluster. A Region can have multiple replicas, and the replicas are divided into a leader and multiple followers. When the data on the leader changes, TiDB will update the data to the followers synchronously.

By default, TiDB only reads and writes data on the leader of the same Region. When a read hotspot occurs in a Region, the Region leader can become a read bottleneck for the entire system. In this situation, enabling the Follower Read feature can significantly reduce the load of the leader and improve the throughput of the whole system by balancing the load among multiple followers.

4.7.8.1.2 When to use

You can visually analyze whether your application has a hotspot Region on the TiDB Dashboard Key Visualizer Page. You can check whether a read hotspot occurs by selecting the "metrics selection box" to `Read (bytes)` or `Read (keys)`.

For more information about handling hotspot, see TiDB Hotspot Problem Handling.

If read hotspots are unavoidable or the changing cost is very high, you can try using the Follower Read feature to better load the balance of reading requests to the follower Region.

4.7.8.1.3 Enable Follower Read

To enable Follower Read, set the variable `tidb_replica_read` (default value is `leader`) to `follower` or `leader-and-follower`:

```
SET [GLOBAL] tidb_replica_read = 'follower';
```

For more details about this variable, see Follower Read Usage.

In Java, to enable Follower Read, define a `FollowerReadHelper` class.

```
public enum FollowReadMode {
    LEADER("leader"),
    FOLLOWER("follower"),
    LEADER_AND_FOLLOWER("leader-and-follower");

    private final String mode;

    FollowReadMode(String mode) {
        this.mode = mode;
    }

    public String getMode() {
        return mode;
    }
}

public class FollowerReadHelper {

    public static void setSessionReplicaRead(Connection conn, FollowReadMode
        ↪  mode) throws SQLException {
        if (mode == null) mode = FollowReadMode.LEADER;
        PreparedStatement stmt = conn.prepareStatement(
            "SET @@tidb_replica_read = ?;"
        );
        stmt.setString(1, mode.getMode());
        stmt.execute();
```

```
    }

    public static void setGlobalReplicaRead(Connection conn, FollowReadMode
        ↪ mode) throws SQLException {
        if (mode == null) mode = FollowReadMode.LEADER;
        PreparedStatement stmt = conn.prepareStatement(
            "SET GLOBAL @@tidb_replica_read = ?;"
        );
        stmt.setString(1, mode.getMode());
        stmt.execute();
    }

}
```

When reading data from the Follower node, use the `setSessionReplicaRead(conn, ↪ FollowReadMode.LEADER_AND_FOLLOWER)` method to enable the Follower Read feature, which can balance the load between the Leader node and the Follower node in the current session. When the connection is disconnected, it will be restored to the original mode.

```
public static class AuthorDAO {

    // Omit initialization of instance variables...

    public void getAuthorsByFollowerRead() throws SQLException {
        try (Connection conn = ds.getConnection()) {
            // Enable the follower read feature.
            FollowerReadHelper.setSessionReplicaRead(conn, FollowReadMode.
                ↪ LEADER_AND_FOLLOWER);

            // Read the authors list for 100000 times.
            Random random = new Random();
            for (int i = 0; i < 100000; i++) {
                Integer birthYear = 1920 + random.nextInt(100);
                List<Author> authors = this.getAuthorsByBirthYear(birthYear);
                System.out.println(authors.size());
            }
        }
    }

    public List<Author> getAuthorsByBirthYear(Integer birthYear) throws
        ↪ SQLException {
        List<Author> authors = new ArrayList<>();
        try (Connection conn = ds.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("SELECT id, name
                ↪ FROM authors WHERE birth_year = ?");
```

```
            stmt.setInt(1, birthYear);
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Author author = new Author();
                author.setId( rs.getLong("id"));
                author.setName(rs.getString("name"));
                authors.add(author);
            }
        }
        return authors;
    }
}
```

4.7.8.1.4 Read more

- Follower Read

- Troubleshoot Hotspot Issues
- TiDB Dashboard - Key Visualizer Page

4.7.8.2 Stale Read

Stale Read is a mechanism that TiDB applies to read historical versions of data stored in TiDB. Using this mechanism, you can read the corresponding historical data at a specific time or within a specified time range, and thus save the latency caused by data replication between storage nodes. When you are using Stale Read, TiDB randomly selects a replica for data reading, which means that all replicas are available for data reading.

In practice, consider carefully whether it is appropriate to enable Stale Read in TiDB based on the usage scenarios. Do not enable Stale Read if your application cannot tolerate reading non-real-time data.

TiDB provides three levels of Stale Read: statement level, transaction level, and session level.

4.7.8.2.1 Introduction

In the Bookshop application, you can query the latest published books and their prices through the following SQL statement:

```
SELECT id, title, type, price FROM books ORDER BY published_at DESC LIMIT
    ↪ 5;
```

The result is as follows:

```
+------------+------------------------------+----------------------+--------+
    ↪
| id         | title                        | type                 | price |
+------------+------------------------------+----------------------+--------+
    ↪
| 3181093216 | The Story of Droolius Caesar | Novel                | 100.00 |
| 1064253862 | Collin Rolfson               | Education & Reference | 92.85 |
| 1748583991 | The Documentary of cat       | Magazine             | 159.75 |
|  893930596 | Myrl Hills                   | Education & Reference | 356.85 |
| 3062833277 | Keven Wyman                  | Life                 | 477.91 |
+------------+------------------------------+----------------------+--------+
    ↪
5 rows in set (0.02 sec)
```

In the list at this time (2022-04-20 15:20:00), the price of *The Story of Droolius Caesar* is 100.0.

At the same time, the seller found that the book was very popular and raised the price of the book to 150.0 through the following SQL statement:

```
UPDATE books SET price = 150 WHERE id = 3181093216;
```

The result is as follows:

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

By querying the latest books list, you can see that the price of this book has increased.

```
+------------+------------------------------+----------------------+--------+
    ↪
| id         | title                        | type                 | price |
+------------+------------------------------+----------------------+--------+
    ↪
| 3181093216 | The Story of Droolius Caesar | Novel                | 150.00 |
| 1064253862 | Collin Rolfson               | Education & Reference | 92.85 |
| 1748583991 | The Documentary of cat       | Magazine             | 159.75 |
|  893930596 | Myrl Hills                   | Education & Reference | 356.85 |
| 3062833277 | Keven Wyman                  | Life                 | 477.91 |
+------------+------------------------------+----------------------+--------+
    ↪
5 rows in set (0.01 sec)
```

If it is not necessary to use the latest data, you can query with Stale Read, which might return outdated data, to avoid the latency caused by data replication during a strongly consistent read.

Assuming that in the Bookshop application, the real-time price of a book is not required on the book lists page but only required on the book details and order pages. Stale Read can be used to improve throughout of the application.

4.7.8.2.2 Statement level

To query the price of a book before a specific time, add an `AS OF TIMESTAMP <datetime > ` clause in the above query statement.

```
SELECT id, title, type, price FROM books AS OF TIMESTAMP '2022-04-20
    ↪ 15:20:00' ORDER BY published_at DESC LIMIT 5;
```

The result is as follows:

```
+------------+------------------------------+----------------------+--------+
    ↪
| id         | title                        | type                 | price |
+------------+------------------------------+----------------------+--------+
    ↪
| 3181093216 | The Story of Droolius Caesar | Novel                | 100.00 |
| 1064253862 | Collin Rolfson               | Education & Reference | 92.85 |
| 1748583991 | The Documentary of cat       | Magazine             | 159.75 |
|  893930596 | Myrl Hills                   | Education & Reference | 356.85 |
| 3062833277 | Keven Wyman                  | Life                 | 477.91 |
+------------+------------------------------+----------------------+--------+
    ↪
5 rows in set (0.01 sec)
```

In addition to specifying an exact time, you can also specify the following:

- `AS OF TIMESTAMP NOW()- INTERVAL 10 SECOND` queries the latest data 10 seconds ago.
- `AS OF TIMESTAMP TIDB_BOUNDED_STALENESS('2016-10-08 16:45:26', '2016-10-08 ↪ 16:45:29')` queries the latest data between `2016-10-08 16:45:26` and `2016-10-08 16:45:29`.
- `AS OF TIMESTAMP TIDB_BOUNDED_STALENESS(NOW()-INTERVAL 20 SECOND, NOW())` queries the latest data within 20 seconds.

Note that the specified timestamp or interval cannot be too early or later than the current time.

Expired data will be recycled by Garbage Collection in TiDB, and the data will be retained for a short period before being cleared. The period is called GC Life Time (default 10 minutes). When a GC starts, the current time minus the time period will be used as the **GC Safe Point**. If you try to read the data before GC Safe Point, TiDB will report the following error:

```
ERROR 9006 (HY000): GC life time is shorter than transaction duration...
```

If the given timestamp is a future time, TiDB will report the following error:

```
ERROR 9006 (HY000): cannot set read timestamp to a future time.
```

```java
public class BookDAO {

    // Omit some code...

    public List<Book> getTop5LatestBooks() throws SQLException {
        List<Book> books = new ArrayList<>();
        try (Connection conn = ds.getConnection()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("""
            SELECT id, title, type, price FROM books ORDER BY published_at
                ↪ DESC LIMIT 5;
            """);
            while (rs.next()) {
                Book book = new Book();
                book.setId(rs.getLong("id"));
                book.setTitle(rs.getString("title"));
                book.setType(rs.getString("type"));
                book.setPrice(rs.getDouble("price"));
                books.add(book);
            }
        }
        return books;
    }

    public void updateBookPriceByID(Long id, Double price) throws
        ↪ SQLException {
        try (Connection conn = ds.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("""
            UPDATE books SET price = ? WHERE id = ?;
            """);
            stmt.setDouble(1, price);
            stmt.setLong(2, id);
            int affects = stmt.executeUpdate();
            if (affects == 0) {
                throw new SQLException("Failed to update the book with id: "
                    ↪ + id);
            }
        }
    }
```

```
    public List<Book> getTop5LatestBooksWithStaleRead(Integer seconds)
        ↪ throws SQLException {
        List<Book> books = new ArrayList<>();
        try (Connection conn = ds.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("""
            SELECT id, title, type, price FROM books AS OF TIMESTAMP NOW() -
                ↪ INTERVAL ? SECOND ORDER BY published_at DESC LIMIT 5;
            """);
            stmt.setInt(1, seconds);
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Book book = new Book();
                book.setId(rs.getLong("id"));
                book.setTitle(rs.getString("title"));
                book.setType(rs.getString("type"));
                book.setPrice(rs.getDouble("price"));
                books.add(book);
            }
        } catch (SQLException e) {
            if ("HY000".equals(e.getSQLState()) && e.getErrorCode() == 1105)
                ↪ {
                System.out.println("WARN: cannot set read timestamp to a
                    ↪ future time.");
            } else if ("HY000".equals(e.getSQLState()) && e.getErrorCode() ==
                ↪  9006) {
                System.out.println("WARN: GC life time is shorter than
                    ↪ transaction duration.");
            } else {
                throw e;
            }
        }
        return books;
    }
}
```

```
List<Book> top5LatestBooks = bookDAO.getTop5LatestBooks();

if (top5LatestBooks.size() > 0) {
    System.out.println("The latest book price (before update): " +
        ↪ top5LatestBooks.get(0).getPrice());

    Book book = top5LatestBooks.get(0);
    bookDAO.updateBookPriceByID(book.getId(), book.price + 10);
```

```
    top5LatestBooks = bookDAO.getTop5LatestBooks();
    System.out.println("The latest book price (after update): " +
        ↪ top5LatestBooks.get(0).getPrice());

    // Use the stale read.
    top5LatestBooks = bookDAO.getTop5LatestBooksWithStaleRead(5);
    System.out.println("The latest book price (maybe stale): " +
        ↪ top5LatestBooks.get(0).getPrice());

    // Try to stale read the data at the future time.
    bookDAO.getTop5LatestBooksWithStaleRead(-5);

    // Try to stale read the data before 20 minutes.
    bookDAO.getTop5LatestBooksWithStaleRead(20 * 60);
}
```

The following result shows that the price returned by Stale Read is 100.00, which is the value before the update.

```
The latest book price (before update): 100.00
The latest book price (after update): 150.00
The latest book price (maybe stale): 100.00
WARN: cannot set read timestamp to a future time.
WARN: GC life time is shorter than transaction duration.
```

4.7.8.2.3 Transaction level

With the `START TRANSACTION READ ONLY AS OF TIMESTAMP` statement, you can start a read-only transaction based on historical time, which reads historical data from a specified historical timestamp.

For example:

```
START TRANSACTION READ ONLY AS OF TIMESTAMP NOW() - INTERVAL 5 SECOND;
```

By querying the latest price of the book, you can see that the price of *The Story of Droolius Caesar* is still 100.0, which is the value before the update.

```
SELECT id, title, type, price FROM books ORDER BY published_at DESC LIMIT
    ↪ 5;
```

The result is as follows:

```
+-----------+-----------------------------+----------------------+--------+
    ↪
| id        | title                       | type                 | price |
```

```
+------------+------------------------------+----------------------+--------+
    ↪
| 3181093216 | The Story of Droolius Caesar | Novel                | 100.00 |
| 1064253862 | Collin Rolfson               | Education & Reference | 92.85 |
| 1748583991 | The Documentary of cat       | Magazine             | 159.75 |
|  893930596 | Myrl Hills                   | Education & Reference | 356.85 |
| 3062833277 | Keven Wyman                  | Life                 | 477.91 |
+------------+------------------------------+----------------------+--------+
    ↪
5 rows in set (0.01 sec)
```

After the transaction with the `COMMIT;` statement is committed, you can read the latest data.

```
+------------+------------------------------+----------------------+--------+
    ↪
| id         | title                        | type                 | price |
+------------+------------------------------+----------------------+--------+
    ↪
| 3181093216 | The Story of Droolius Caesar | Novel                | 150.00 |
| 1064253862 | Collin Rolfson               | Education & Reference | 92.85 |
| 1748583991 | The Documentary of cat       | Magazine             | 159.75 |
|  893930596 | Myrl Hills                   | Education & Reference | 356.85 |
| 3062833277 | Keven Wyman                  | Life                 | 477.91 |
+------------+------------------------------+----------------------+--------+
    ↪
5 rows in set (0.01 sec)
```

You can define a helper class for transactions, which encapsulates the command to enable Stale Read at the transaction level as a helper method.

```
public static class StaleReadHelper {

    public static void startTxnWithStaleRead(Connection conn, Integer
        ↪ seconds) throws SQLException {
        conn.setAutoCommit(false);
        PreparedStatement stmt = conn.prepareStatement(
            "START TRANSACTION READ ONLY AS OF TIMESTAMP NOW() - INTERVAL ?
                ↪ SECOND;"
        );
        stmt.setInt(1, seconds);
        stmt.execute();
    }

}
```

Then define a method to enable the Stale Read feature through a transaction in the `BookDAO` class. Use the method to query instead of adding `AS OF TIMESTAMP` to the query statement.

```
public class BookDAO {

    // Omit some code...

    public List<Book> getTop5LatestBooksWithTxnStaleRead(Integer seconds)
        ↪ throws SQLException {
        List<Book> books = new ArrayList<>();
        try (Connection conn = ds.getConnection()) {
            // Start a read only transaction.
            TxnHelper.startTxnWithStaleRead(conn, seconds);

            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("""
            SELECT id, title, type, price FROM books ORDER BY published_at
                ↪ DESC LIMIT 5;
            """);
            while (rs.next()) {
                Book book = new Book();
                book.setId(rs.getLong("id"));
                book.setTitle(rs.getString("title"));
                book.setType(rs.getString("type"));
                book.setPrice(rs.getDouble("price"));
                books.add(book);
            }

            // Commit transaction.
            conn.commit();
        } catch (SQLException e) {
            if ("HY000".equals(e.getSQLState()) && e.getErrorCode() == 1105)
                ↪ {
                System.out.println("WARN: cannot set read timestamp to a
                    ↪ future time.");
            } else if ("HY000".equals(e.getSQLState()) && e.getErrorCode() ==
                ↪  9006) {
                System.out.println("WARN: GC life time is shorter than
                    ↪ transaction duration.");
            } else {
                throw e;
            }
        }
        return books;
```

```
    }
}
```

```
List<Book> top5LatestBooks = bookDAO.getTop5LatestBooks();

if (top5LatestBooks.size() > 0) {
    System.out.println("The latest book price (before update): " +
        ↪ top5LatestBooks.get(0).getPrice());

    Book book = top5LatestBooks.get(0);
    bookDAO.updateBookPriceByID(book.getId(), book.price + 10);

    top5LatestBooks = bookDAO.getTop5LatestBooks();
    System.out.println("The latest book price (after update): " +
        ↪ top5LatestBooks.get(0).getPrice());

    // Use the stale read.
    top5LatestBooks = bookDAO.getTop5LatestBooksWithTxnStaleRead(5);
    System.out.println("The latest book price (maybe stale): " +
        ↪ top5LatestBooks.get(0).getPrice());

    // After the stale read transaction is committed.
    top5LatestBooks = bookDAO.getTop5LatestBooks();
    System.out.println("The latest book price (after the transaction commit)
        ↪ : " + top5LatestBooks.get(0).getPrice());
}
```

The result is as follows:

```
The latest book price (before update): 100.00
The latest book price (after update): 150.00
The latest book price (maybe stale): 100.00
The latest book price (after the transaction commit): 150
```

With the `SET TRANSACTION READ ONLY AS OF TIMESTAMP` statement, you can set the opened transaction or the next transaction to be a read-only transaction based on a specified historical time. The transaction will read historical data based on the provided historical time.

For example, you can use the following `AS OF TIMESTAMP` statement to switch the ongoing transactions to the read-only mode and read historical data 5 seconds ago.

```
SET TRANSACTION READ ONLY AS OF TIMESTAMP NOW() - INTERVAL 5 SECOND;
```

You can define a helper class for transactions, which encapsulates the command to enable Stale Read at the transaction level as a helper method.

```
public static class TxnHelper {

    public static void setTxnWithStaleRead(Connection conn, Integer seconds)
        ↪ throws SQLException {
        PreparedStatement stmt = conn.prepareStatement(
            "SET TRANSACTION READ ONLY AS OF TIMESTAMP NOW() - INTERVAL ?
                ↪ SECOND;"
        );
        stmt.setInt(1, seconds);
        stmt.execute();
    }

}
```

Then define a method to enable the Stale Read feature through a transaction in the `BookDAO` class. Use the method to query instead of adding `AS OF TIMESTAMP` to the query statement.

```
public class BookDAO {

    // Omit some code...

    public List<Book> getTop5LatestBooksWithTxnStaleRead2(Integer seconds)
        ↪ throws SQLException {
        List<Book> books = new ArrayList<>();
        try (Connection conn = ds.getConnection()) {
            StaleReadHelper.setTxnWithStaleRead(conn, seconds);

            // Start a read only transaction.
            conn.setAutoCommit(false);

            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("""
            SELECT id, title, type, price FROM books ORDER BY published_at
                ↪ DESC LIMIT 5;
            """);
            while (rs.next()) {
                Book book = new Book();
                book.setId(rs.getLong("id"));
                book.setTitle(rs.getString("title"));
                book.setType(rs.getString("type"));
                book.setPrice(rs.getDouble("price"));
                books.add(book);
            }
```

```
            // Commit transaction.
            conn.commit();
        } catch (SQLException e) {
            if ("HY000".equals(e.getSQLState()) && e.getErrorCode() == 1105)
                ↪ {
                System.out.println("WARN: cannot set read timestamp to a
                    ↪ future time.");
            } else if ("HY000".equals(e.getSQLState()) && e.getErrorCode() ==
                ↪  9006) {
                System.out.println("WARN: GC life time is shorter than
                    ↪ transaction duration.");
            } else {
                throw e;
            }
        }
        return books;
    }
}
```

4.7.8.2.4 Session level

To support reading historical data, TiDB has introduced a new system variable `tidb_read_staleness` since v5.4. you can use it to set the range of historical data that the current session is allowed to read. Its data type is `int` and its scope is `SESSION`.

Enable Stale Read in a session:

```
SET @@tidb_read_staleness="-5";
```

For example, if the value is set to `-5` and TiKV has the corresponding historical data, TiDB selects a timestamp as new as possible within a 5-second time range.

Disable Stale Read in the session:

```
set @@tidb_read_staleness="";
```

```
public static class StaleReadHelper{

    public static void enableStaleReadOnSession(Connection conn, Integer
        ↪ seconds) throws SQLException {
        PreparedStatement stmt = conn.prepareStatement(
            "SET @@tidb_read_staleness= ?;"
        );
        stmt.setString(1, String.format("-%d", seconds));
        stmt.execute();
    }
```

```
    public static void disableStaleReadOnSession(Connection conn) throws
        ↪ SQLException {
        PreparedStatement stmt = conn.prepareStatement(
            "SET @@tidb_read_staleness=\"\";"
        );
        stmt.execute();
    }

}
```

4.7.8.2.5 Read more

- Usage Scenarios of Stale Read
- Read Historical Data Using the `AS OF TIMESTAMP` Clause
- Read Historical Data Using the `tidb_read_staleness` System Variable

4.7.9 HTAP Queries

HTAP stands for Hybrid Transactional and Analytical Processing. Traditionally, databases are often designed for transactional or analytical scenarios, so the data platform often needs to be split into Transactional Processing and Analytical Processing, and the data needs to be replicated from the transactional database to the analytical database for quick response to analytical queries. TiDB databases can perform both transactional and analytical tasks, which greatly simplifies the construction of data platforms and allows users to use fresher data for analysis.

TiDB uses TiKV, a row-based storage engine, for Online Transactional Processing (OLTP), and TiFlash, a columnar storage engine, for Online Analytical Processing (OLAP). The row-based storage engine and the columnar storage engine co-exist for HTAP. Both storage engines can replicate data automatically and keep strong consistency. The row-based storage engine optimizes OLTP performance, and the columnar storage engine optimizes OLAP performance.

The Create a table section introduces how to enable the HTAP capability of TiDB. The following describes how to use HTAP to analyze data faster.

4.7.9.1 Data preparation

Before starting, you can import more sample data via the `tiup demo` command. For example:

```
tiup demo bookshop prepare --users=200000 --books=500000 --authors=100000 --
    ↪ ratings=1000000 --orders=1000000 --host 127.0.0.1 --port 4000 --drop-
    ↪ tables
```

Or you can use the Import function of TiDB Cloud to import the pre-prepared sample data.

4.7.9.2 Window functions

When using a database, in addition to storing your data and providing application features (such as ordering and rating books), you might also need to analyze the data in the database to make further operations and decisions.

The Query data from a single table document introduces how to use aggregate queries to analyze data as a whole. In more complex scenarios, you might want to aggregate the results of multiple aggregation queries into a single query. If you want to know the historical trend of the order amount of a particular book, you can aggregate `sum` for all order data of each month, and then aggregate the `sum` results together to get the historical trend.

To facilitate such analysis, since TiDB v3.0, TiDB supports window functions. For each row of data, this function provides the ability to access data across multiple rows. Different from a regular aggregation query, the window function aggregates rows without merging results set into a single row.

Similar to aggregate functions, you also need to follow a fixed set of syntax when using the window function:

```
SELECT
    window_function() OVER ([partition_clause] [order_clause] [frame_clause
        ↪ ]) AS alias
FROM
    table_name
```

4.7.9.2.1 `ORDER BY` clause

With the aggregate window function `sum()`, you can analyze the historical trend of the order amount of a particular book. For example:

```
WITH orders_group_by_month AS (
  SELECT DATE_FORMAT(ordered_at, '%Y-%c') AS month, COUNT(*) AS orders
  FROM orders
  WHERE book_id = 3461722937
  GROUP BY 1
)
SELECT
month,
SUM(orders) OVER(ORDER BY month ASC) as acc
FROM orders_group_by_month
ORDER BY month ASC;
```

The `sum()` function accumulates the data in the order specified by the `ORDER BY` statement in the `OVER` clause. The result is as follows:

```
+---------+-------+
| month   | acc   |
+---------+-------+
| 2011-5  |     1 |
| 2011-8  |     2 |
| 2012-1  |     3 |
| 2012-2  |     4 |
| 2013-1  |     5 |
| 2013-3  |     6 |
| 2015-11 |     7 |
| 2015-4  |     8 |
| 2015-8  |     9 |
| 2017-11 |    10 |
| 2017-5  |    11 |
| 2019-5  |    13 |
| 2020-2  |    14 |
+---------+-------+
13 rows in set (0.01 sec)
```

Visualize the above data through a line chart with time as the horizontal axis and cumulative order amount as the vertical axis. You can easily know the historical ordering trend of the book through the change of the slope.

4.7.9.2.2 `PARTITION BY` clause

Suppose that you want to analyze the historical ordering trend of different types of books, and visualize it in the same line chart with multiple series.

You can use the `PARTITION BY` clause to group books by types and count history orders for each type separately.

```
WITH orders_group_by_month AS (
    SELECT
        b.type AS book_type,
        DATE_FORMAT(ordered_at, '%Y-%c') AS month,
        COUNT(*) AS orders
    FROM orders o
    LEFT JOIN books b ON o.book_id = b.id
    WHERE b.type IS NOT NULL
    GROUP BY book_type, month
), acc AS (
    SELECT
        book_type,
        month,
        SUM(orders) OVER(PARTITION BY book_type ORDER BY book_type, month
            ↪ ASC) as acc
```

```
    FROM orders_group_by_month
    ORDER BY book_type, month ASC
)
SELECT * FROM acc;
```

The result is as follows:

```
+-----------------------------+---------+------+
| book_type                   | month   | acc |
+-----------------------------+---------+------+
| Magazine                    | 2011-10 |   1 |
| Magazine                    | 2011-8 |    2 |
| Magazine                    | 2012-5 |    3 |
| Magazine                    | 2013-1 |    4 |
| Magazine                    | 2013-6 |    5 |
...
| Novel                       | 2011-3 |   13 |
| Novel                       | 2011-4 |   14 |
| Novel                       | 2011-6 |   15 |
| Novel                       | 2011-8 |   17 |
| Novel                       | 2012-1 |   18 |
| Novel                       | 2012-2 |   20 |
...
| Sports                      | 2021-4 |   49 |
| Sports                      | 2021-7 |   50 |
| Sports                      | 2022-4 |   51 |
+-----------------------------+---------+------+
1500 rows in set (1.70 sec)
```

4.7.9.2.3 Non-aggregate window functions

TiDB also provides some non-aggregated window functions for more analysis statements.

For example, the Pagination Query document introduces how to use the `row_number()` function to achieve efficient pagination batch processing.

4.7.9.3 Hybrid workload

When using TiDB for real-time online analytical processing in hybrid load scenarios, you only need to provide an entry point of TiDB to your data. TiDB automatically selects different processing engines based on the specific business.

4.7.9.3.1 Create TiFlash replicas

TiDB uses the row-based storage engine, TiKV, by default. To use the columnar storage engine, TiFlash, see Enable HTAP capability. Before querying data through TiFlash, you need to create TiFlash replicas for `books` and `orders` tables using the following statement:

```
ALTER TABLE books SET TIFLASH REPLICA 1;
ALTER TABLE orders SET TIFLASH REPLICA 1;
```

You can check the progress of the TiFlash replicas using the following statement:

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = '
    ↪ bookshop' and TABLE_NAME = 'books';
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = '
    ↪ bookshop' and TABLE_NAME = 'orders';
```

A `PROGRESS` column of 1 indicates that the progress is 100% complete, and a `AVAILABLE` column of 1 indicates that the replica is currently available.

```
+--------------+------------+----------+---------------+-----------------+-----------+--
    ↪
| TABLE_SCHEMA | TABLE_NAME | TABLE_ID | REPLICA_COUNT | LOCATION_LABELS |
    ↪ AVAILABLE | PROGRESS |
+--------------+------------+----------+---------------+-----------------+-----------+--
    ↪
| bookshop     | books      |      143 |             1 |                 |         1 |
    ↪         1 |
+--------------+------------+----------+---------------+-----------------+-----------+--
    ↪
1 row in set (0.07 sec)
+--------------+------------+----------+---------------+-----------------+-----------+--
    ↪
| TABLE_SCHEMA | TABLE_NAME | TABLE_ID | REPLICA_COUNT | LOCATION_LABELS |
    ↪ AVAILABLE | PROGRESS |
+--------------+------------+----------+---------------+-----------------+-----------+--
    ↪
| bookshop     | orders     |      147 |             1 |                 |         1 |
    ↪         1 |
+--------------+------------+----------+---------------+-----------------+-----------+--
    ↪
1 row in set (0.07 sec)
```

After replicas are added, you can use the `EXPLAIN` statement to check the execution plan of the above window function `PARTITION BY` clause. If `cop[tiflash]` appears in the execution plan, it means that the TiFlash engine has started to work.

Then, execute the sample SQL statement in `PARTITION BY` clause again. The result is as follows:

```
+-----------------------------+---------+------+
| book_type                   | month   | acc |
+-----------------------------+---------+------+
```

```
| Magazine                    | 2011-10 |   1 |
| Magazine                    | 2011-8 |    2 |
| Magazine                    | 2012-5 |    3 |
| Magazine                    | 2013-1 |    4 |
| Magazine                    | 2013-6 |    5 |
...
| Novel                       | 2011-3 |   13 |
| Novel                       | 2011-4 |   14 |
| Novel                       | 2011-6 |   15 |
| Novel                       | 2011-8 |   17 |
| Novel                       | 2012-1 |   18 |
| Novel                       | 2012-2 |   20 |
...
| Sports                      | 2021-4 |   49 |
| Sports                      | 2021-7 |   50 |
| Sports                      | 2022-4 |   51 |
+-----------------------------+---------+------+
1500 rows in set (0.79 sec)
```

By comparing the two execution results, you can find that the query speed is significantly improved with TiFlash (the improvement is more significant with a large volume of data). This is because a window function usually relies on a full table scan for some columns, and columnar TiFlash is more suitable to handle this type of analytical task than row-based TiKV. For TiKV, if you use primary keys or indexes to reduce the number of rows to be queried, the queries can be fast too and consume fewer resources compared with TiFlash.

4.7.9.3.2 Specify a query engine

TiDB uses the Cost Based Optimizer (CBO) to automatically choose whether to use TiFlash replicas based on cost estimates. However, if you are sure whether your query is transactional or analytical, you can specify the query engine to be used with Optimizer Hints.

To specify which engine to be used in a query, you can use the `/*+ read_from_storage (engine_name[table_name])*/` hint as in the following statement.

Note:

- If a table has an alias, use the alias instead of the table name in the hint, otherwise, the hint does not work.
- The `read_from_storage` hint does not work for common table expression.

```
WITH orders_group_by_month AS (
    SELECT
        /*+ read_from_storage(tikv[o]) */
        b.type AS book_type,
        DATE_FORMAT(ordered_at, '%Y-%c') AS month,
        COUNT(*) AS orders
    FROM orders o
    LEFT JOIN books b ON o.book_id = b.id
    WHERE b.type IS NOT NULL
    GROUP BY book_type, month
), acc AS (
    SELECT
        book_type,
        month,
        SUM(orders) OVER(PARTITION BY book_type ORDER BY book_type, month
            ↪ ASC) as acc
    FROM orders_group_by_month mo
    ORDER BY book_type, month ASC
)
SELECT * FROM acc;
```

You can use the `EXPLAIN` statement to check the execution plan of the above SQL statement. If `cop[tiflash]` and `cop[tikv]` appear in the task column at the same time, it means that TiFlash and TiKV are both scheduled to complete this query. Note that TiFlash and TiKV storage engines usually use different TiDB nodes, so the two query types are not affected by each other.

For more information about how TiDB chooses to use TiFlash, see Use TiDB to read TiFlash replicas

4.7.9.4 Read more

- Quick Start with HTAP
- Explore HTAP

- Window Functions
- Use TiFlash

4.8 Transaction

4.8.1 Transaction overview

TiDB supports complete distributed transactions, providing optimistic transactions and pessimistic transactions (introduced in TiDB 3.0). This article mainly introduces transaction

statements, optimistic transactions and pessimistic transactions, transaction isolation levels, and application-side retry and error handling in optimistic transactions.

4.8.1.1 Common statements

This chapter introduces how to use transactions in TiDB. The following example demonstrates the process of a simple transaction:

Bob wants to transfer $20 to Alice. This transaction includes two operations:

- Bob's account is reduced by $20.
- Alice's account is increased by $20.

Transactions can ensure that both of the above operations are executed successfully or both fail.

Insert some sample data into the table using the `users` table in the bookshop database:

```
INSERT INTO users (id, nickname, balance)
  VALUES (2, 'Bob', 200);
INSERT INTO users (id, nickname, balance)
  VALUES (1, 'Alice', 100);
```

Run the following transactions and explain what each statement means:

```
BEGIN;
  UPDATE users SET balance = balance - 20 WHERE nickname = 'Bob';
  UPDATE users SET balance = balance + 20 WHERE nickname= 'Alice';
COMMIT;
```

After the above transaction is executed successfully, the table should look like this:

```
+----+--------------+---------+
| id | account_name | balance |
+----+--------------+---------+
|  1 | Alice        |  120.00 |
|  2 | Bob          |  180.00 |
+----+--------------+---------+
```

4.8.1.1.1 Start a transaction

To explicitly start a new transaction, you can use either `BEGIN` or `START TRANSACTION`.

```
BEGIN;
```

```
START TRANSACTION;
```

The default transaction mode of TiDB is pessimistic. You can also explicitly specify the optimistic transaction model:

```
BEGIN OPTIMISTIC;
```

Enable the pessimistic transaction mode:

```
BEGIN PESSIMISTIC;
```

If the current session is in the middle of a transaction when the above statement is executed, TiDB commits the current transaction first, and then starts a new transaction.

4.8.1.1.2 Commit a transaction

You can use the `COMMIT` statement to commit all modifications made by TiDB in the current transaction.

```
COMMIT;
```

Before enabling optimistic transactions, make sure that your application can properly handle errors that may be returned by a `COMMIT` statement. If you are not sure how your application will handle it, it is recommended to use the pessimistic transaction mode instead.

4.8.1.1.3 Roll back a transaction

You can use the `ROLLBACK` statement to roll back modifications of the current transaction.

```
ROLLBACK;
```

In the previous transfer example, if you roll back the entire transaction, Alice's and Bob's balances will remain unchanged, and all modifications of the current transaction are canceled.

```
TRUNCATE TABLE `users`;

INSERT INTO `users` (`id`, `nickname`, `balance`) VALUES (1, 'Alice', 100),
    ↪  (2, 'Bob', 200);

SELECT * FROM `users`;
+----+--------------+---------+
| id | nickname   | balance |
+----+--------------+---------+
|  1 | Alice      |  100.00 |
|  2 | Bob        |  200.00 |
+----+--------------+---------+

BEGIN;
  UPDATE `users` SET `balance` = `balance` - 20 WHERE `nickname`='Bob';
```

```
  UPDATE `users` SET `balance` = `balance` + 20 WHERE `nickname`='Alice';
ROLLBACK;

SELECT * FROM `users`;
+----+--------------+---------+
| id | nickname   | balance |
+----+--------------+---------+
|  1 | Alice      |  100.00 |
|  2 | Bob        |  200.00 |
+----+--------------+---------+
```

The transaction is also automatically rolled back if the client connection is stopped or closed.

4.8.1.2 Transaction isolation levels

The transaction isolation levels are the basis of database transaction processing. The "I" (Isolation) in **ACID** refers to the isolation of the transactions.

The SQL-92 standard defines four isolation levels:

- read uncommitted (`READ UNCOMMITTED`)
- read committed (`READ COMMITTED`)
- repeatable read (`REPEATABLE READ`)
- serializable (`SERIALIZABLE`).

See the table below for details:

Isolation Level	Dirty Write	Dirty Read	Fuzzy Read	Phantom
READ UNCOMMITTED	Not Possible	Possible	Possible	Possible
READ COMMITTED	Not Possible	Not possible	Possible	Possible
REPEATABLE READ	Not Possible	Not possible	Not possible	Possible
SERIALIZABLE	Not Possible	Not possible	Not possible	Not possible

TiDB supports the following isolation levels: `READ COMMITTED` and `REPEATABLE READ`:

```
mysql> SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
ERROR 8048 (HY000): The isolation level 'READ-UNCOMMITTED' is not supported
    ↪ . Set tidb_skip_isolation_level_check=1 to skip this error
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)

mysql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 8048 (HY000): The isolation level 'SERIALIZABLE' is not supported.
    ↪ Set tidb_skip_isolation_level_check=1 to skip this error
```

TiDB implements Snapshot Isolation (SI) level consistency, also known as "repeatable read" for consistency with MySQL. This isolation level is different from ANSI Repeatable Read Isolation Level and MySQL Repeatable Read Isolation Level. For more details, see TiDB Transaction Isolation Levels.

4.8.2 Optimistic Transactions and Pessimistic Transactions

The optimistic transaction model commits the transaction directly, and rolls back when there is a conflict. By contrast, the pessimistic transaction model tries to lock the resources that need to be modified before actually committing the transaction, and only starts committing after ensuring that the transaction can be successfully executed.

The optimistic transaction model is suitable for scenarios with low conflict rates, because the direct commit has a high probability of success. But once a transaction conflict occurs, the cost of rollback is relatively high.

The advantage of the pessimistic transaction model is that for scenarios with high conflict rates, the cost of locking ahead is less than the cost of rollback afterwards. Moreover, it can solve the problem that multiple concurrent transactions fail to commit due to conflicts. However, the pessimistic transaction model is not as efficient as the optimistic transaction model in scenarios with low conflict rates.

The pessimistic transaction model is more intuitive and easier to implement on the application side. The optimistic transaction model requires complex application-side retry mechanisms.

The following is an example of a bookshop. It uses an example of buying books to show the pros and cons of optimistic and pessimistic transactions. The process of buying books mainly consists of the following:

1. Update the stock quantity
2. Create an order
3. Make the payment

These operations must either all succeed or all fail. You must ensure that overselling does not happen in the case of concurrent transactions.

4.8.2.1 Pessimistic transactions

The following code uses two threads to simulate the process that two users buy the same book in a pessimistic transaction mode. There are 10 books left in the bookstore. Bob buys 6 books, and Alice buys 4 books. They complete the orders at nearly the same time. As a result, all books in inventory are sold out.

Because you use multiple threads to simulate the situation that multiple users insert data simultaneously, you need to use a connection object with safe threads. Here use Java's popular connection pool HikariCP for demo.

`sql.DB` in Golang is concurrency-safe, so there is no need to import a third-party package.

To adapt TiDB transactions, write a toolkit util according to the following code:

```
package util

import (
    "context"
    "database/sql"
)

type TiDBSqlTx struct {
    *sql.Tx
    conn       *sql.Conn
    pessimistic bool
}

func TiDBSqlBegin(db *sql.DB, pessimistic bool) (*TiDBSqlTx, error) {
    ctx := context.Background()
    conn, err := db.Conn(ctx)
    if err != nil {
        return nil, err
    }
    if pessimistic {
        _, err = conn.ExecContext(ctx, "set @@tidb_txn_mode=?", "pessimistic
            ↪ ")
    } else {
        _, err = conn.ExecContext(ctx, "set @@tidb_txn_mode=?", "optimistic")
    }
    if err != nil {
        return nil, err
    }
    tx, err := conn.BeginTx(ctx, nil)
    if err != nil {
        return nil, err
    }
    return &TiDBSqlTx{
        conn:       conn,
        Tx:         tx,
        pessimistic: pessimistic,
    }, nil
}
```

```
func (tx *TiDBSqlTx) Commit() error {
    defer tx.conn.Close()
    return tx.Tx.Commit()
}

func (tx *TiDBSqlTx) Rollback() error {
    defer tx.conn.Close()
    return tx.Tx.Rollback()
}
```

To ensure thread safety, you can use the mysqlclient driver to open multiple connections that are not shared between threads.

4.8.2.1.1 Write a pessimistic transaction example

Configuration file

If you use Maven to manage the package, in the `<dependencies>` node in `pom.xml`, add the following dependencies to import `HikariCP`, and set the packaging target, and the main class of the JAR package startup. The following is an example of `pom.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
    ↪ org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
      ↪ org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.pingcap</groupId>
  <artifactId>plain-java-txn</artifactId>
  <version>0.0.1</version>

  <name>plain-java-jdbc</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
```

```
    <scope>test</scope>
  </dependency>

  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.28</version>
  </dependency>

  <dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>5.0.1</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.3.0</version>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <archive>
          <manifest>
            <mainClass>com.pingcap.txn.TxnExample</mainClass>
          </manifest>
        </archive>

      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
```

```
  </build>

</project>
```

Coding

Then write the code:

```
package com.pingcap.txn;

import com.zaxxer.hikari.HikariDataSource;

import java.math.BigDecimal;
import java.sql.*;
import java.util.Arrays;
import java.util.concurrent.*;

public class TxnExample {
    public static void main(String[] args) throws SQLException,
        ↪ InterruptedException {
        System.out.println(Arrays.toString(args));
        int aliceQuantity = 0;
        int bobQuantity = 0;

        for (String arg: args) {
            if (arg.startsWith("ALICE_NUM")) {
                aliceQuantity = Integer.parseInt(arg.replace("ALICE_NUM=", "")
                    ↪ );
            }

            if (arg.startsWith("BOB_NUM")) {
                bobQuantity = Integer.parseInt(arg.replace("BOB_NUM=", ""));
            }
        }

        HikariDataSource ds = new HikariDataSource();
        ds.setJdbcUrl("jdbc:mysql://localhost:4000/bookshop?
            ↪ useServerPrepStmts=true&cachePrepStmts=true");
        ds.setUsername("root");
        ds.setPassword("");

        // prepare data
        Connection connection = ds.getConnection();
        createBook(connection, 1L, "Designing Data-Intensive Application", "
            ↪ Science & Technology",
                Timestamp.valueOf("2018-09-01 00:00:00"), new BigDecimal(100)
```

```
                ↪ , 10);
        createUser(connection, 1L, "Bob", new BigDecimal(10000));
        createUser(connection, 2L, "Alice", new BigDecimal(10000));

        CountDownLatch countDownLatch = new CountDownLatch(2);
        ExecutorService threadPool = Executors.newFixedThreadPool(2);

        final int finalBobQuantity = bobQuantity;
        threadPool.execute(() -> {
            buy(ds, 1, 1000L, 1L, 1L, finalBobQuantity);
            countDownLatch.countDown();
        });
        final int finalAliceQuantity = aliceQuantity;
        threadPool.execute(() -> {
            buy(ds, 2, 1001L, 1L, 2L, finalAliceQuantity);
            countDownLatch.countDown();
        });

        countDownLatch.await(5, TimeUnit.SECONDS);
    }

    public static void createUser(Connection connection, Long id, String
        ↪ nickname, BigDecimal balance) throws SQLException {
        PreparedStatement insert = connection.prepareStatement(
                "INSERT INTO `users` (`id`, `nickname`, `balance`) VALUES (?,
                    ↪ ?, ?)");
        insert.setLong(1, id);
        insert.setString(2, nickname);
        insert.setBigDecimal(3, balance);
        insert.executeUpdate();
    }

    public static void createBook(Connection connection, Long id, String
        ↪ title, String type, Timestamp publishedAt, BigDecimal price,
        ↪ Integer stock) throws SQLException {
        PreparedStatement insert = connection.prepareStatement(
                "INSERT INTO `books` (`id`, `title`, `type`, `published_at`, `
                    ↪ price`, `stock`) values (?, ?, ?, ?, ?, ?)");
        insert.setLong(1, id);
        insert.setString(2, title);
        insert.setString(3, type);
        insert.setTimestamp(4, publishedAt);
        insert.setBigDecimal(5, price);
        insert.setInt(6, stock);
```

```
        insert.executeUpdate();
    }

    public static void buy (HikariDataSource ds, Integer threadID,
                            Long orderID, Long bookID, Long userID, Integer
                               ↪ quantity) {
        String txnComment = "/* txn " + threadID + " */ ";

        try (Connection connection = ds.getConnection()) {
            try {
                connection.setAutoCommit(false);
                connection.createStatement().executeUpdate(txnComment + "begin
                   ↪  pessimistic");

                // waiting for other thread ran the 'begin pessimistic'
                   ↪ statement
                TimeUnit.SECONDS.sleep(1);

                BigDecimal price = null;

                // read price of book
                PreparedStatement selectBook = connection.prepareStatement(
                   ↪ txnComment + "select price from books where id = ? for
                   ↪ update");
                selectBook.setLong(1, bookID);
                ResultSet res = selectBook.executeQuery();
                if (!res.next()) {
                    throw new RuntimeException("book not exist");
                } else {
                    price = res.getBigDecimal("price");
                }

                // update book
                String updateBookSQL = "update `books` set stock = stock - ?
                   ↪ where id = ? and stock - ? >= 0";
                PreparedStatement updateBook = connection.prepareStatement(
                   ↪ txnComment + updateBookSQL);
                updateBook.setInt(1, quantity);
                updateBook.setLong(2, bookID);
                updateBook.setInt(3, quantity);
                int affectedRows = updateBook.executeUpdate();

                if (affectedRows == 0) {
                    // stock not enough, rollback
                    connection.createStatement().executeUpdate(txnComment + "
```

```
                    ↪ rollback");
                return;
            }

            // insert order
            String insertOrderSQL = "insert into `orders` (`id`, `book_id
                ↪ `, `user_id`, `quality`) values (?, ?, ?, ?)";
            PreparedStatement insertOrder = connection.prepareStatement(
                ↪ txnComment + insertOrderSQL);
            insertOrder.setLong(1, orderID);
            insertOrder.setLong(2, bookID);
            insertOrder.setLong(3, userID);
            insertOrder.setInt(4, quantity);
            insertOrder.executeUpdate();

            // update user
            String updateUserSQL = "update `users` set `balance` = `
                ↪ balance` - ? where id = ?";
            PreparedStatement updateUser = connection.prepareStatement(
                ↪ txnComment + updateUserSQL);
            updateUser.setBigDecimal(1, price.multiply(new BigDecimal(
                ↪ quantity)));
            updateUser.setLong(2, userID);
            updateUser.executeUpdate();

            connection.createStatement().executeUpdate(txnComment + "
                ↪ commit");
        } catch (Exception e) {
            connection.createStatement().executeUpdate(txnComment + "
                ↪ rollback");
            e.printStackTrace();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

Write a `helper.go` file that contains the required database operations:

```
package main

import (
    "context"
    "database/sql"
```

```
    "fmt"
    "time"

    "github.com/go-sql-driver/mysql"
    "github.com/pingcap-inc/tidb-example-golang/util"
    "github.com/shopspring/decimal"
)

type TxnFunc func(txn *util.TiDBSqlTx) error

const (
    ErrWriteConflict    = 9007 // Transactions in TiKV encounter write
        ↪ conflicts.
    ErrInfoSchemaChanged = 8028 // table schema changes
    ErrForUpdateCantRetry = 8002 // "SELECT FOR UPDATE" commit conflict
    ErrTxnRetryable     = 8022 // The transaction commit fails and has been
        ↪ rolled back
)

const retryTimes = 5

var retryErrorCodeSet = map[uint16]interface{}{
    ErrWriteConflict:    nil,
    ErrInfoSchemaChanged: nil,
    ErrForUpdateCantRetry: nil,
    ErrTxnRetryable:     nil,
}

func runTxn(db *sql.DB, optimistic bool, optimisticRetryTimes int, txnFunc
    ↪ TxnFunc) {
    txn, err := util.TiDBSqlBegin(db, !optimistic)
    if err != nil {
        panic(err)
    }

    err = txnFunc(txn)
    if err != nil {
        txn.Rollback()
        if mysqlErr, ok := err.(*mysql.MySQLError); ok && optimistic &&
            ↪ optimisticRetryTimes != 0 {
            if _, retryableError := retryErrorCodeSet[mysqlErr.Number];
                ↪ retryableError {
                fmt.Printf("[runTxn] got a retryable error, rest time: %d\n",
                    ↪ optimisticRetryTimes-1)
                runTxn(db, optimistic, optimisticRetryTimes-1, txnFunc)
```

```
                return
            }
        }

        fmt.Printf("[runTxn] got an error, rollback: %+v\n", err)
    } else {
        err = txn.Commit()
        if mysqlErr, ok := err.(*mysql.MySQLError); ok && optimistic &&
            ↪ optimisticRetryTimes != 0 {
            if _, retryableError := retryErrorCodeSet[mysqlErr.Number];
                ↪ retryableError {
                fmt.Printf("[runTxn] got a retryable error, rest time: %d\n",
                    ↪ optimisticRetryTimes-1)
                runTxn(db, optimistic, optimisticRetryTimes-1, txnFunc)
                return
            }
        }

        if err == nil {
            fmt.Println("[runTxn] commit success")
        }
    }
}

func prepareData(db *sql.DB, optimistic bool) {
    runTxn(db, optimistic, retryTimes, func(txn *util.TiDBSqlTx) error {
        publishedAt, err := time.Parse("2006-01-02 15:04:05", "2018-09-01
            ↪ 00:00:00")
        if err != nil {
            return err
        }

        if err = createBook(txn, 1, "Designing Data-Intensive Application",
            "Science & Technology", publishedAt, decimal.NewFromInt(100), 10)
                ↪ ; err != nil {
            return err
        }

        if err = createUser(txn, 1, "Bob", decimal.NewFromInt(10000)); err !=
            ↪  nil {
            return err
        }

        if err = createUser(txn, 2, "Alice", decimal.NewFromInt(10000)); err
            ↪ != nil {
```

```
            return err
        }

        return nil
    })
}

func buyPessimistic(db *sql.DB, goroutineID, orderID, bookID, userID, amount
    ↪ int) {
    txnComment := fmt.Sprintf("/* txn %d */ ", goroutineID)
    if goroutineID != 1 {
        txnComment = "\t" + txnComment
    }

    fmt.Printf("\nuser %d try to buy %d books(id: %d)\n", userID, amount,
        ↪ bookID)

    runTxn(db, false, retryTimes, func(txn *util.TiDBSqlTx) error {
        time.Sleep(time.Second)

        // read the price of book
        selectBookForUpdate := "select `price` from books where id = ? for
            ↪ update"
        bookRows, err := txn.Query(selectBookForUpdate, bookID)
        if err != nil {
            return err
        }
        fmt.Println(txnComment + selectBookForUpdate + " successful")
        defer bookRows.Close()

        price := decimal.NewFromInt(0)
        if bookRows.Next() {
            err = bookRows.Scan(&price)
            if err != nil {
                return err
            }
        } else {
            return fmt.Errorf("book ID not exist")
        }
        bookRows.Close()

        // update book
        updateStock := "update `books` set stock = stock - ? where id = ? and
            ↪ stock - ? >= 0"
        result, err := txn.Exec(updateStock, amount, bookID, amount)
```

```
        if err != nil {
            return err
        }
        fmt.Println(txnComment + updateStock + " successful")

        affected, err := result.RowsAffected()
        if err != nil {
            return err
        }

        if affected == 0 {
            return fmt.Errorf("stock not enough, rollback")
        }

        // insert order
        insertOrder := "insert into `orders` (`id`, `book_id`, `user_id`, `
            ↪ quality`) values (?, ?, ?, ?)"
        if _, err := txn.Exec(insertOrder,
            orderID, bookID, userID, amount); err != nil {
            return err
        }
        fmt.Println(txnComment + insertOrder + " successful")

        // update user
        updateUser := "update `users` set `balance` = `balance` - ? where id
            ↪ = ?"
        if _, err := txn.Exec(updateUser,
            price.Mul(decimal.NewFromInt(int64(amount))), userID); err != nil
                ↪  {
            return err
        }
        fmt.Println(txnComment + updateUser + " successful")

        return nil
    })
}

func buyOptimistic(db *sql.DB, goroutineID, orderID, bookID, userID, amount
    ↪ int) {
    txnComment := fmt.Sprintf("/* txn %d */ ", goroutineID)
    if goroutineID != 1 {
        txnComment = "\t" + txnComment
    }

    fmt.Printf("\nuser %d try to buy %d books(id: %d)\n", userID, amount,
```

```
    ↪ bookID)

runTxn(db, true, retryTimes, func(txn *util.TiDBSqlTx) error {
    time.Sleep(time.Second)

    // read the price and stock of book
    selectBookForUpdate := "select `price`, `stock` from books where id =
        ↪ ? for update"
    bookRows, err := txn.Query(selectBookForUpdate, bookID)
    if err != nil {
        return err
    }
    fmt.Println(txnComment + selectBookForUpdate + " successful")
    defer bookRows.Close()

    price, stock := decimal.NewFromInt(0), 0
    if bookRows.Next() {
        err = bookRows.Scan(&price, &stock)
        if err != nil {
            return err
        }
    } else {
        return fmt.Errorf("book ID not exist")
    }
    bookRows.Close()

    if stock < amount {
        return fmt.Errorf("book not enough")
    }

    // update book
    updateStock := "update `books` set stock = stock - ? where id = ? and
        ↪ stock - ? >= 0"
    result, err := txn.Exec(updateStock, amount, bookID, amount)
    if err != nil {
        return err
    }
    fmt.Println(txnComment + updateStock + " successful")

    affected, err := result.RowsAffected()
    if err != nil {
        return err
    }

    if affected == 0 {
```

```
            return fmt.Errorf("stock not enough, rollback")
        }

        // insert order
        insertOrder := "insert into `orders` (`id`, `book_id`, `user_id`, `
            ↪ quality`) values (?, ?, ?, ?)"
        if _, err := txn.Exec(insertOrder,
            orderID, bookID, userID, amount); err != nil {
            return err
        }
        fmt.Println(txnComment + insertOrder + " successful")

        // update user
        updateUser := "update `users` set `balance` = `balance` - ? where id
            ↪ = ?"
        if _, err := txn.Exec(updateUser,
            price.Mul(decimal.NewFromInt(int64(amount))), userID); err != nil
                ↪  {
            return err
        }
        fmt.Println(txnComment + updateUser + " successful")

        return nil
    })
}

func createBook(txn *util.TiDBSqlTx, id int, title, bookType string,
    publishedAt time.Time, price decimal.Decimal, stock int) error {
    _, err := txn.ExecContext(context.Background(),
        "INSERT INTO `books` (`id`, `title`, `type`, `published_at`, `price`,
            ↪  `stock`) values (?, ?, ?, ?, ?, ?)",
        id, title, bookType, publishedAt, price, stock)
    return err
}

func createUser(txn *util.TiDBSqlTx, id int, nickname string, balance
    ↪ decimal.Decimal) error {
    _, err := txn.ExecContext(context.Background(),
        "INSERT INTO `users` (`id`, `nickname`, `balance`) VALUES (?, ?, ?)",
        id, nickname, balance)
    return err
}
```

Then write a `txn.go` with a `main` function to call `helper.go` and handle the incoming command line arguments:

```
package main

import (
    "database/sql"
    "flag"
    "fmt"
    "sync"
)

func main() {
    optimistic, alice, bob := parseParams()

    openDB("mysql", "root:@tcp(127.0.0.1:4000)/bookshop?charset=utf8mb4",
        ↪ func(db *sql.DB) {
        prepareData(db, optimistic)
        buy(db, optimistic, alice, bob)
    })
}

func buy(db *sql.DB, optimistic bool, alice, bob int) {
    buyFunc := buyOptimistic
    if !optimistic {
        buyFunc = buyPessimistic
    }

    wg := sync.WaitGroup{}
    wg.Add(1)
    go func() {
        defer wg.Done()
        buyFunc(db, 1, 1000, 1, 1, bob)
    }()

    wg.Add(1)
    go func() {
        defer wg.Done()
        buyFunc(db, 2, 1001, 1, 2, alice)
    }()

    wg.Wait()
}

func openDB(driverName, dataSourceName string, runnable func(db *sql.DB)) {
    db, err := sql.Open(driverName, dataSourceName)
    if err != nil {
```

```
        panic(err)
    }
    defer db.Close()

    runnable(db)
}

func parseParams() (optimistic bool, alice, bob int) {
    flag.BoolVar(&optimistic, "o", false, "transaction is optimistic")
    flag.IntVar(&alice, "a", 4, "Alice bought num")
    flag.IntVar(&bob, "b", 6, "Bob bought num")

    flag.Parse()

    fmt.Println(optimistic, alice, bob)

    return optimistic, alice, bob
}
```

The Golang example already includes optimistic transactions.

```
import time

import MySQLdb
import os
import datetime
from threading import Thread

REPEATABLE_ERROR_CODE_SET = {
    9007, # Transactions in TiKV encounter write conflicts.
    8028, # table schema changes
    8002, # "SELECT FOR UPDATE" commit conflict
    8022  # The transaction commit fails and has been rolled back
}


def create_connection():
    return MySQLdb.connect(
        host="127.0.0.1",
        port=4000,
        user="root",
        password="",
        database="bookshop",
        autocommit=False
    )
```

```
def prepare_data() -> None:
    connection = create_connection()
    with connection:
        with connection.cursor() as cursor:
            cursor.execute("INSERT INTO `books` (`id`, `title`, `type`, `
                ↪ published_at`, `price`, `stock`) "
                           "values (%s, %s, %s, %s, %s, %s)",
                           (1, "Designing Data-Intensive Application", "Science
                              ↪  & Technology",
                            datetime.datetime(2018, 9, 1), 100, 10))

            cursor.executemany("INSERT INTO `users` (`id`, `nickname`, `
                ↪ balance`) VALUES (%s, %s, %s)",
                               [(1, "Bob", 10000), (2, "ALICE", 10000)])
            connection.commit()


def buy_optimistic(thread_id: int, order_id: int, book_id: int, user_id:
    ↪ int, amount: int,
                   optimistic_retry_times: int = 5) -> None:
    connection = create_connection()

    txn_log_header = f"/* txn {thread_id} */"
    if thread_id != 1:
        txn_log_header = "\t" + txn_log_header

    with connection:
        with connection.cursor() as cursor:
            cursor.execute("BEGIN OPTIMISTIC")
            print(f'{txn_log_header} BEGIN OPTIMISTIC')
            time.sleep(1)

            try:
                # read the price of book
                select_book_for_update = "SELECT `price`, `stock` FROM books
                    ↪ WHERE id = %s FOR UPDATE"
                cursor.execute(select_book_for_update, (book_id,))
                book = cursor.fetchone()
                if book is None:
                    raise Exception("book_id not exist")
                price, stock = book
                print(f'{txn_log_header} {select_book_for_update} successful'
                    ↪ )
```

```
            if stock < amount:
                raise Exception("book not enough, rollback")

            # update book
            update_stock = "update `books` set stock = stock - %s where id
                ↪  = %s and stock - %s >= 0"
            rows_affected = cursor.execute(update_stock, (amount, book_id,
                ↪  amount))
            print(f'{txn_log_header} {update_stock} successful')

            if rows_affected == 0:
                raise Exception("stock not enough, rollback")

            # insert order
            insert_order = "insert into `orders` (`id`, `book_id`, `
                ↪ user_id`, `quality`) values (%s, %s, %s, %s)"
            cursor.execute(insert_order, (order_id, book_id, user_id,
                ↪ amount))
            print(f'{txn_log_header} {insert_order} successful')

            # update user
            update_user = "update `users` set `balance` = `balance` - %s
                ↪ where id = %s"
            cursor.execute(update_user, (amount * price, user_id))
            print(f'{txn_log_header} {update_user} successful')

        except Exception as err:
            connection.rollback()

            print(f'something went wrong: {err}')
        else:
            # important here! you need deal the Exception from the TiDB
            try:
                connection.commit()
            except MySQLdb.MySQLError as db_err:
                code, desc = db_err.args
                if code in REPEATABLE_ERROR_CODE_SET and
                    ↪ optimistic_retry_times > 0:
                    print(f'retry, rest {optimistic_retry_times - 1} times
                        ↪ , for {code} {desc}')
                    buy_optimistic(thread_id, order_id, book_id, user_id,
                        ↪ amount, optimistic_retry_times - 1)
```

```
def buy_pessimistic(thread_id: int, order_id: int, book_id: int, user_id:
    ↪ int, amount: int) -> None:
    connection = create_connection()

    txn_log_header = f"/* txn {thread_id} */"
    if thread_id != 1:
        txn_log_header = "\t" + txn_log_header

    with connection:
        with connection.cursor() as cursor:
            cursor.execute("BEGIN PESSIMISTIC")
            print(f'{txn_log_header} BEGIN PESSIMISTIC')
            time.sleep(1)

            try:
                # read the price of book
                select_book_for_update = "SELECT `price` FROM books WHERE id =
                    ↪  %s FOR UPDATE"
                cursor.execute(select_book_for_update, (book_id,))
                book = cursor.fetchone()
                if book is None:
                    raise Exception("book_id not exist")
                price = book[0]
                print(f'{txn_log_header} {select_book_for_update} successful'
                    ↪ )

                # update book
                update_stock = "update `books` set stock = stock - %s where id
                    ↪  = %s and stock - %s >= 0"
                rows_affected = cursor.execute(update_stock, (amount, book_id,
                    ↪  amount))
                print(f'{txn_log_header} {update_stock} successful')

                if rows_affected == 0:
                    raise Exception("stock not enough, rollback")

                # insert order
                insert_order = "insert into `orders` (`id`, `book_id`, `
                    ↪ user_id`, `quality`) values (%s, %s, %s, %s)"
                cursor.execute(insert_order, (order_id, book_id, user_id,
                    ↪ amount))
                print(f'{txn_log_header} {insert_order} successful')

                # update user
                update_user = "update `users` set `balance` = `balance` - %s
```

```
                ↪ where id = %s"
            cursor.execute(update_user, (amount * price, user_id))
            print(f'{txn_log_header} {update_user} successful')

        except Exception as err:
            connection.rollback()
            print(f'something went wrong: {err}')
        else:
            connection.commit()


optimistic = os.environ.get('OPTIMISTIC')
alice = os.environ.get('ALICE')
bob = os.environ.get('BOB')

if not (optimistic and alice and bob):
    raise Exception("please use \"OPTIMISTIC=<is_optimistic> ALICE=<
        ↪ alice_num> "
                    "BOB=<bob_num> python3 txn_example.py\" to start this
                        ↪ script")

prepare_data()

if bool(optimistic) is True:
    buy_func = buy_optimistic
else:
    buy_func = buy_pessimistic

bob_thread = Thread(target=buy_func, kwargs={
    "thread_id": 1, "order_id": 1000, "book_id": 1, "user_id": 1, "amount":
        ↪ int(bob)})
alice_thread = Thread(target=buy_func, kwargs={
    "thread_id": 2, "order_id": 1001, "book_id": 1, "user_id": 2, "amount":
        ↪ int(alice)})

bob_thread.start()
alice_thread.start()
bob_thread.join(timeout=10)
alice_thread.join(timeout=10)
```

The Python example already includes optimistic transactions.

4.8.2.1.2 An example that does not involve overselling

Run the sample program:

```
mvn clean package
java -jar target/plain-java-txn-0.0.1-jar-with-dependencies.jar ALICE_NUM=4
    ↪ BOB_NUM=6
```

```
go build -o bin/txn
./bin/txn -a 4 -b 6
```

```
OPTIMISTIC=False ALICE=4 BOB=6 python3 txn_example.py
```

SQL logs:

```
/* txn 1 */ BEGIN PESSIMISTIC
    /* txn 2 */ BEGIN PESSIMISTIC
    /* txn 2 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
    /* txn 2 */ UPDATE `books` SET `stock` = `stock` - 4 WHERE `id` = 1 AND
       ↪  `stock` - 4 >= 0
    /* txn 2 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
       ↪  VALUES (1001, 1, 1, 4)
    /* txn 2 */ UPDATE `users` SET `balance` = `balance` - 400.0 WHERE `id`
       ↪ = 2
    /* txn 2 */ COMMIT
/* txn 1 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
/* txn 1 */ UPDATE `books` SET `stock` = `stock` - 6 WHERE `id` = 1 AND `
    ↪ stock` - 6 >= 0
/* txn 1 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
    ↪ VALUES (1000, 1, 1, 6)
/* txn 1 */ UPDATE `users` SET `balance` = `balance` - 600.0 WHERE `id` = 1
/* txn 1 */ COMMIT
```

Finally, check that the order is created, the user balance is deducted, and the book inventory is deducted as expected.

```
mysql> SELECT * FROM `books`;
+--
    ↪ --+--------------------------------------+---------------------+-------------
    ↪
| id | title                                | type                | published_at
    ↪      | stock | price |
+--
    ↪ --+--------------------------------------+---------------------+-------------
    ↪
|  1 | Designing Data-Intensive Application | Science & Technology |
    ↪ 2018-09-01 00:00:00 | 0 | 100.00 |
+--
    ↪ --+--------------------------------------+---------------------+-------------
    ↪
```

```
1 row in set (0.00 sec)

mysql> SELECT * FROM orders;
+------+---------+---------+---------+---------------------+
| id   | book_id | user_id | quality | ordered_at          |
+------+---------+---------+---------+---------------------+
| 1000 |       1 |       1 |       6 | 2022-04-19 10:58:12 |
| 1001 |       1 |       1 |       4 | 2022-04-19 10:58:11 |
+------+---------+---------+---------+---------------------+
2 rows in set (0.01 sec)

mysql> SELECT * FROM users;
+----+---------+----------+
| id | balance | nickname |
+----+---------+----------+
|  1 | 9400.00 | Bob      |
|  2 | 9600.00 | Alice    |
+----+---------+----------+
2 rows in set (0.00 sec)
```

4.8.2.1.3 An example that prevents overselling

The task in this example is more challenging. Suppose there are 10 books left in stock. Bob buys 7 books, Alice buys 4 books, and they place orders almost at the same time. What will happen? You can reuse the code from the previous example to solve this challenge, and change Bob's purchase quantity from 6 to 7.

Run the sample program:

```
mvn clean package
java -jar target/plain-java-txn-0.0.1-jar-with-dependencies.jar ALICE_NUM=4
    ↪ BOB_NUM=7
```

```
go build -o bin/txn
./bin/txn -a 4 -b 7
```

```
OPTIMISTIC=False ALICE=4 BOB=7 python3 txn_example.py
```

```
/* txn 1 */ BEGIN PESSIMISTIC
    /* txn 2 */ BEGIN PESSIMISTIC
    /* txn 2 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
    /* txn 2 */ UPDATE `books` SET `stock` = `stock` - 4 WHERE `id` = 1 AND
        ↪ `stock` - 4 >= 0
    /* txn 2 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
        ↪ values (1001, 1, 1, 4)
```

```
    /* txn 2 */ UPDATE `users` SET `balance` = `balance` - 400.0 WHERE `id`
       ↪ = 2
    /* txn 2 */ COMMIT
/* txn 1 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
/* txn 1 */ UPDATE `books` SET `stock` = `stock` - 7 WHERE `id` = 1 AND `
   ↪ stock` - 7 >= 0
/* txn 1 */ ROLLBACK
```

Since `txn 2` preemptively gets the lock resource and updates the stock, the return value of `affected_rows` in `txn 1` is 0, and it enters the `rollback` process.

Let's check the order creation, user balance deduction, and book inventory deduction. Alice successfully ordered 4 books, Bob failed to order 7 books, and the remaining 6 books are in stock as expected.

```
mysql> SELECT * FROM books;
+--
   ↪ --+--------------------------------------+---------------------+-------------
   ↪
| id | title                                | type                | published_at
   ↪      | stock | price |
+--
   ↪ --+--------------------------------------+---------------------+-------------
   ↪
|  1 | Designing Data-Intensive Application | Science & Technology |
   ↪ 2018-09-01 00:00:00 | 6 | 100.00 |
+--
   ↪ --+--------------------------------------+---------------------+-------------
   ↪
1 row in set (0.00 sec)

mysql> SELECT * FROM orders;
+------+---------+---------+---------+---------------------+
| id   | book_id | user_id | quality | ordered_at          |
+------+---------+---------+---------+---------------------+
| 1001 |       1 |       1 |       4 | 2022-04-19 11:03:03 |
+------+---------+---------+---------+---------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM users;
+----+----------+----------+
| id | balance  | nickname |
+----+----------+----------+
|  1 | 10000.00 | Bob      |
|  2 |  9600.00 | Alice    |
+----+----------+----------+
```

```
2 rows in set (0.01 sec)
```

4.8.2.2 Optimistic transactions

The following code uses two threads to simulate the process that two users buy the same book in an optimistic transaction, just like the pessimistic transaction example. There are 10 books left in inventory. Bob buys 6 and Alice buys 4. They complete the order at about the same time. In the end, no books are left in inventory.

4.8.2.2.1 Write an optimistic transaction example

Coding

```
package com.pingcap.txn.optimistic;

import com.zaxxer.hikari.HikariDataSource;

import java.math.BigDecimal;
import java.sql.*;
import java.util.Arrays;
import java.util.concurrent.*;

public class TxnExample {
    public static void main(String[] args) throws SQLException,
        ↪ InterruptedException {
        System.out.println(Arrays.toString(args));
        int aliceQuantity = 0;
        int bobQuantity = 0;

        for (String arg: args) {
            if (arg.startsWith("ALICE_NUM")) {
                aliceQuantity = Integer.parseInt(arg.replace("ALICE_NUM=", "")
                    ↪ );
            }

            if (arg.startsWith("BOB_NUM")) {
                bobQuantity = Integer.parseInt(arg.replace("BOB_NUM=", ""));
            }
        }

        HikariDataSource ds = new HikariDataSource();
        ds.setJdbcUrl("jdbc:mysql://localhost:4000/bookshop?
            ↪ useServerPrepStmts=true&cachePrepStmts=true");
        ds.setUsername("root");
        ds.setPassword("");
```

```
        // prepare data
        Connection connection = ds.getConnection();
        createBook(connection, 1L, "Designing Data-Intensive Application", "
            ↪ Science & Technology",
                Timestamp.valueOf("2018-09-01 00:00:00"), new BigDecimal(100)
                    ↪ , 10);
        createUser(connection, 1L, "Bob", new BigDecimal(10000));
        createUser(connection, 2L, "Alice", new BigDecimal(10000));

        CountDownLatch countDownLatch = new CountDownLatch(2);
        ExecutorService threadPool = Executors.newFixedThreadPool(2);

        final int finalBobQuantity = bobQuantity;
        threadPool.execute(() -> {
            buy(ds, 1, 1000L, 1L, 1L, finalBobQuantity, 5);
            countDownLatch.countDown();
        });
        final int finalAliceQuantity = aliceQuantity;
        threadPool.execute(() -> {
            buy(ds, 2, 1001L, 1L, 2L, finalAliceQuantity, 5);
            countDownLatch.countDown();
        });

        countDownLatch.await(5, TimeUnit.SECONDS);
    }

    public static void createUser(Connection connection, Long id, String
        ↪ nickname, BigDecimal balance) throws SQLException {
        PreparedStatement insert = connection.prepareStatement(
                "INSERT INTO `users` (`id`, `nickname`, `balance`) VALUES (?,
                    ↪ ?, ?)");
        insert.setLong(1, id);
        insert.setString(2, nickname);
        insert.setBigDecimal(3, balance);
        insert.executeUpdate();
    }

    public static void createBook(Connection connection, Long id, String
        ↪ title, String type, Timestamp publishedAt, BigDecimal price,
        ↪ Integer stock) throws SQLException {
        PreparedStatement insert = connection.prepareStatement(
                "INSERT INTO `books` (`id`, `title`, `type`, `published_at`, `
                    ↪ price`, `stock`) values (?, ?, ?, ?, ?, ?)");
        insert.setLong(1, id);
```

```
        insert.setString(2, title);
        insert.setString(3, type);
        insert.setTimestamp(4, publishedAt);
        insert.setBigDecimal(5, price);
        insert.setInt(6, stock);

        insert.executeUpdate();
    }

    public static void buy (HikariDataSource ds, Integer threadID, Long
        ↪ orderID, Long bookID,
                            Long userID, Integer quantity, Integer retryTimes)
                                ↪ {
        String txnComment = "/* txn " + threadID + " */ ";

        try (Connection connection = ds.getConnection()) {
            try {

                connection.setAutoCommit(false);
                connection.createStatement().executeUpdate(txnComment + "begin
                    ↪  optimistic");

                // waiting for other thread ran the 'begin optimistic'
                    ↪ statement
                TimeUnit.SECONDS.sleep(1);

                BigDecimal price = null;

                // read price of book
                PreparedStatement selectBook = connection.prepareStatement(
                    ↪ txnComment + "SELECT * FROM books where id = ? for
                    ↪ update");
                selectBook.setLong(1, bookID);
                ResultSet res = selectBook.executeQuery();
                if (!res.next()) {
                    throw new RuntimeException("book not exist");
                } else {
                    price = res.getBigDecimal("price");
                    int stock = res.getInt("stock");
                    if (stock < quantity) {
                        throw new RuntimeException("book not enough");
                    }
                }

                // update book
```

```
            String updateBookSQL = "update `books` set stock = stock - ?
                ↪ where id = ? and stock - ? >= 0";
            PreparedStatement updateBook = connection.prepareStatement(
                ↪ txnComment + updateBookSQL);
            updateBook.setInt(1, quantity);
            updateBook.setLong(2, bookID);
            updateBook.setInt(3, quantity);
            updateBook.executeUpdate();

            // insert order
            String insertOrderSQL = "insert into `orders` (`id`, `book_id
                ↪ `, `user_id`, `quality`) values (?, ?, ?, ?)";
            PreparedStatement insertOrder = connection.prepareStatement(
                ↪ txnComment + insertOrderSQL);
            insertOrder.setLong(1, orderID);
            insertOrder.setLong(2, bookID);
            insertOrder.setLong(3, userID);
            insertOrder.setInt(4, quantity);
            insertOrder.executeUpdate();

            // update user
            String updateUserSQL = "update `users` set `balance` = `
                ↪ balance` - ? where id = ?";
            PreparedStatement updateUser = connection.prepareStatement(
                ↪ txnComment + updateUserSQL);
            updateUser.setBigDecimal(1, price.multiply(new BigDecimal(
                ↪ quantity)));
            updateUser.setLong(2, userID);
            updateUser.executeUpdate();

            connection.createStatement().executeUpdate(txnComment + "
                ↪ commit");
        } catch (Exception e) {
            connection.createStatement().executeUpdate(txnComment + "
                ↪ rollback");
            System.out.println("error occurred: " + e.getMessage());

            if (e instanceof SQLException sqlException) {
                switch (sqlException.getErrorCode()) {
                    // You can get all error codes at https://docs.
                        ↪ pingcap.com/tidb/stable/error-codes
                    case 9007: // Transactions in TiKV encounter write
                        ↪ conflicts.
                    case 8028: // table schema changes
                    case 8002: // "SELECT FOR UPDATE" commit conflict
```

```
                    case 8022: // The transaction commit fails and has
                        ↪ been rolled back
                        if (retryTimes != 0) {
                            System.out.println("rest " + retryTimes + "
                                ↪ times. retry for " + e.getMessage());
                            buy(ds, threadID, orderID, bookID, userID,
                                ↪ quantity, retryTimes - 1);
                        }
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

Configuration changes

Change the startup class in `pom.xml`:

```
<mainClass>com.pingcap.txn.TxnExample</mainClass>
```

Change it to the following to point to the optimistic transaction example.

```
<mainClass>com.pingcap.txn.optimistic.TxnExample</mainClass>
```

The Golang example in the Write a pessimistic transaction example section already supports optimistic transactions and can be used directly without changes.

The Python example in the Write a pessimistic transaction example section already supports optimistic transactions and can be used directly without changes.

4.8.2.2.2 An example that does not involve overselling

Run the sample program:

```
mvn clean package
java -jar target/plain-java-txn-0.0.1-jar-with-dependencies.jar ALICE_NUM=4
    ↪ BOB_NUM=6
```

```
go build -o bin/txn
./bin/txn -a 4 -b 6 -o true
```

```
OPTIMISTIC=True ALICE=4 BOB=6 python3 txn_example.py
```

SQL statement execution process:

```
    /* txn 2 */ BEGIN OPTIMISTIC
/* txn 1 */ BEGIN OPTIMISTIC
    /* txn 2 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
    /* txn 2 */ UPDATE `books` SET `stock` = `stock` - 4 WHERE `id` = 1 AND
       ↪  `stock` - 4 >= 0
    /* txn 2 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
       ↪  VALUES (1001, 1, 1, 4)
    /* txn 2 */ UPDATE `users` SET `balance` = `balance` - 400.0 WHERE `id`
       ↪ = 2
    /* txn 2 */ COMMIT
/* txn 1 */ SELECT * FROM `books` WHERE `id` = 1 for UPDATE
/* txn 1 */ UPDATE `books` SET `stock` = `stock` - 6 WHERE `id` = 1 AND `
    ↪ stock` - 6 >= 0
/* txn 1 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
    ↪ VALUES (1000, 1, 1, 6)
/* txn 1 */ UPDATE `users` SET `balance` = `balance` - 600.0 WHERE `id` = 1
retry 1 times for 9007 Write conflict, txnStartTS=432618733006225412,
    ↪ conflictStartTS=432618733006225411, conflictCommitTS
    ↪ =432618733006225414, key={tableID=126, handle=1} primary={tableID
    ↪ =114, indexID=1, indexValues={1, 1000, }} [try again later]
/* txn 1 */ BEGIN OPTIMISTIC
/* txn 1 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
/* txn 1 */ UPDATE `books` SET `stock` = `stock` - 6 WHERE `id` = 1 AND `
    ↪ stock` - 6 >= 0
/* txn 1 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
    ↪ VALUES (1000, 1, 1, 6)
/* txn 1 */ UPDATE `users` SET `balance` = `balance` - 600.0 WHERE `id` = 1
/* txn 1 */ COMMIT
```

In the optimistic transaction mode, because the intermediate state is not necessarily correct, it is not possible to judge whether a statement is successfully executed through `affected_rows` as in the pessimistic transaction mode. You need to regard the transaction as a whole, and judge whether the current transaction has a write conflict by checking whether the final `COMMIT` statement returns an exception.

As you can see from the above SQL log, because two transactions are executed concurrently and the same record is modified, a `9007 Write conflict` exception is thrown after `txn 1` COMMIT. For write conflicts in the optimistic transaction mode, you can safely retry on the application side. After one retry, the data is committed successfully. The final execution result is as expected:

```
mysql> SELECT * FROM books;
+--
    ↪ --+-------------------------------------+--------------------+-------------
    ↪
```

```
| id | title                                | type                | published_at
    ↪      | stock | price |
+--
    ↪ --+--------------------------------------+---------------------+--------------
    ↪
|  1 | Designing Data-Intensive Application | Science & Technology |
    ↪ 2018-09-01 00:00:00 | 0 | 100.00 |
+--
    ↪ --+--------------------------------------+---------------------+--------------
    ↪
1 row in set (0.01 sec)

mysql> SELECT * FROM orders;
+------+---------+---------+---------+---------------------+
| id   | book_id | user_id | quality | ordered_at    |
+------+---------+---------+---------+---------------------+
| 1000 |       1 |       1 |       6 | 2022-04-19 03:18:19 |
| 1001 |       1 |       1 |       4 | 2022-04-19 03:18:17 |
+------+---------+---------+---------+---------------------+
2 rows in set (0.01 sec)

mysql> SELECT * FROM users;
+----+---------+----------+
| id | balance | nickname |
+----+---------+----------+
|  1 | 9400.00 | Bob    |
|  2 | 9600.00 | Alice  |
+----+---------+----------+
2 rows in set (0.00 sec)
```

4.8.2.2.3 An example that prevents overselling

This section describes an optimistic transaction example that prevents overselling. Suppose there are 10 books left in inventory. Bob buys 7 books,and Alice buys 4 books. They place orders almost at the same time. What will happen? You can reuse the code from the optimistic transaction example to address this requirement. Change Bob's purchases from 6 to 7.

Run the sample program:

```
mvn clean package
java -jar target/plain-java-txn-0.0.1-jar-with-dependencies.jar ALICE_NUM=4
    ↪ BOB_NUM=7
```

```
go build -o bin/txn
```

```
./bin/txn -a 4 -b 7 -o true
```

```
OPTIMISTIC=True ALICE=4 BOB=7 python3 txn_example.py
```

```
/* txn 1 */ BEGIN OPTIMISTIC
    /* txn 2 */ BEGIN OPTIMISTIC
    /* txn 2 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
    /* txn 2 */ UPDATE `books` SET `stock` = `stock` - 4 WHERE `id` = 1 AND
       ↪  `stock` - 4 >= 0
    /* txn 2 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
       ↪  VALUES (1001, 1, 1, 4)
    /* txn 2 */ UPDATE `users` SET `balance` = `balance` - 400.0 WHERE `id`
       ↪ = 2
    /* txn 2 */ COMMIT
/* txn 1 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
/* txn 1 */ UPDATE `books` SET `stock` = `stock` - 7 WHERE `id` = 1 AND `
    ↪ stock` - 7 >= 0
/* txn 1 */ INSERT INTO `orders` (`id`, `book_id`, `user_id`, `quality`)
    ↪ VALUES (1000, 1, 1, 7)
/* txn 1 */ UPDATE `users` SET `balance` = `balance` - 700.0 WHERE `id` = 1
retry 1 times for 9007 Write conflict, txnStartTS=432619094333980675,
    ↪ conflictStartTS=432619094333980676, conflictCommitTS
    ↪ =432619094333980678, key={tableID=126, handle=1} primary={tableID
    ↪ =114, indexID=1, indexValues={1, 1000, }} [try again later]
/* txn 1 */ BEGIN OPTIMISTIC
/* txn 1 */ SELECT * FROM `books` WHERE `id` = 1 FOR UPDATE
Fail -> out of stock
/* txn 1 */ ROLLBACK
```

You can see from the above SQL log that `txn 1` is retried on the application side due to a write conflict in the first execution. By comparing the latest snapshots, you can find that the stock is running out. The application side throws `out of stock`, and ends abnormally.

```
mysql> SELECT * FROM books;
+--
    ↪ --+--------------------------------------+---------------------+-------------
    ↪
| id | title                                | type                | published_at
    ↪      | stock | price |
+--
    ↪ --+--------------------------------------+---------------------+-------------
    ↪
|  1 | Designing Data-Intensive Application | Science & Technology |
    ↪ 2018-09-01 00:00:00 | 6 | 100.00 |
```

```
+--
    ↪ --+-------------------------------------+---------------------+---------------
    ↪
1 row in set (0.00 sec)

mysql> SELECT * FROM orders;
+------+---------+---------+---------+---------------------+
| id   | book_id | user_id | quality | ordered_at    |
+------+---------+---------+---------+---------------------+
| 1001 |      1 |      1 |      4 | 2022-04-19 03:41:16 |
+------+---------+---------+---------+---------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM users;
+----+----------+----------+
| id | balance | nickname |
+----+----------+----------+
|  1 | 10000.00 | Bob    |
|  2 |  9600.00 | Alice  |
+----+----------+----------+
2 rows in set (0.00 sec)
```

4.8.3 Transaction Restraints

This document briefly introduces the transaction restraints in TiDB.

4.8.3.1 Isolation levels

The isolation levels supported by TiDB are **RC (Read Committed)** and **SI (Snapshot Isolation)**, where **SI** is basically equivalent to the **RR (Repeatable Read)** isolation level.

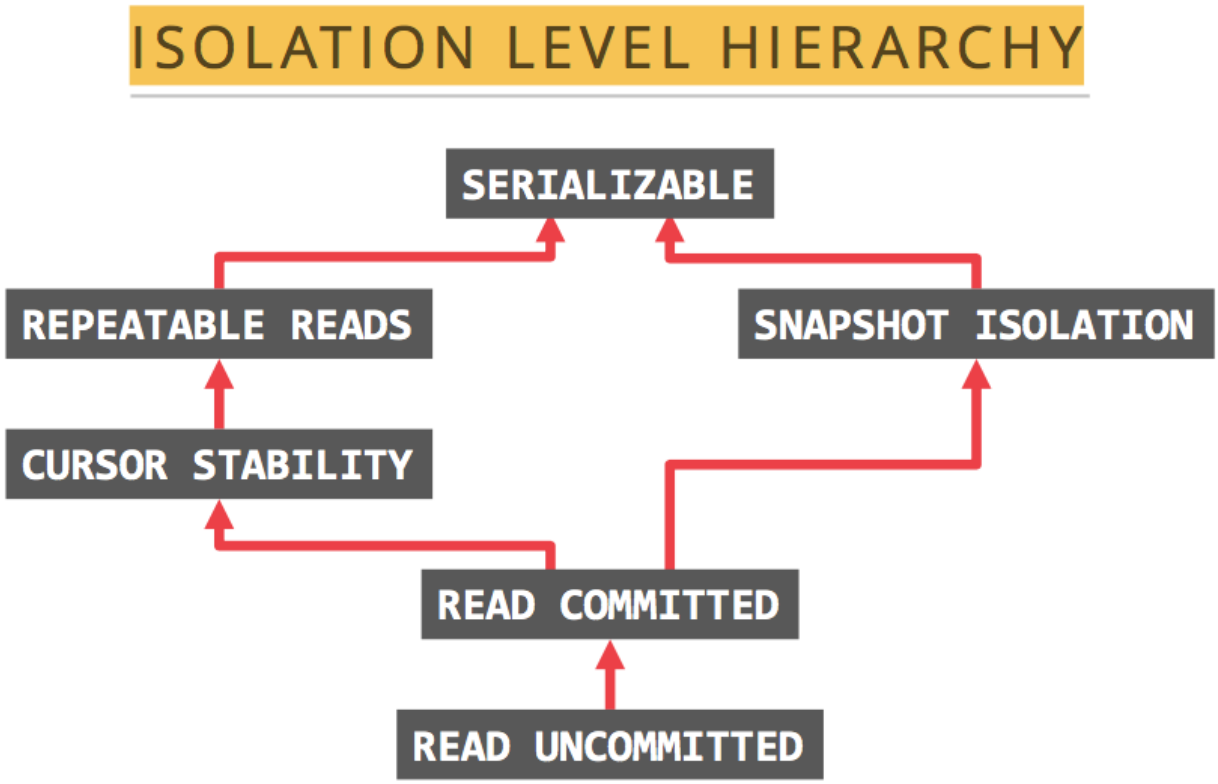


Figure 17: isolation level

4.8.3.2 Snapshot Isolation can avoid phantom reads

The `SI` isolation level of TiDB can avoid **Phantom Reads**, but the `RR` in ANSI/ISO SQL standard cannot.

The following two examples show what **phantom reads** is.

- Example 1: **Transaction A** first gets `n` rows according to the query, and then **Transaction B** changes `m` rows other than these `n` rows or adds `m` rows that match the query of **Transaction A**. When **Transaction A** runs the query again, it finds that there are `n+m` rows that match the condition. It is like a phantom, so it is called a **phantom read**.
- Example 2: **Admin A** changes the grades of all students in the database from specific scores to ABCDE grades, but **Admin B** inserts a record with a specific score at this time. When **Admin A** finishes changing and finds that there is still a record (the one inserted by **Admin B**) that has not been changed yet. That is a **phantom read**.

4.8.3.3 SI cannot avoid write skew

TiDB's SI isolation level cannot avoid **write skew** exceptions. You can use the `SELECT FOR UPDATE` syntax to avoid **write skew** exceptions.

A **write skew** exception occurs when two concurrent transactions read different but related records, and then each transaction updates the data it reads and eventually commits the transaction. If there is a constraint between these related records that cannot be modified concurrently by multiple transactions, then the end result will violate the constraint.

For example, suppose you are writing a doctor shift management program for a hospital. Hospitals typically require several doctors to be on call at the same time, but the minimum requirement is that at least one doctor is on call. Doctors can drop their shifts (for example, if they are feeling sick) as long as at least one doctor is on call during that shift.

Now there is a situation where doctors `Alice` and `Bob` are on call. Both are feeling sick, so they decide to take sick leave. They happen to click the button at the same time. Let's simulate this process with the following program:

```
package com.pingcap.txn.write.skew;

import com.zaxxer.hikari.HikariDataSource;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Semaphore;

public class EffectWriteSkew {
    public static void main(String[] args) throws SQLException,
        ↪ InterruptedException {
        HikariDataSource ds = new HikariDataSource();
        ds.setJdbcUrl("jdbc:mysql://localhost:4000/test?useServerPrepStmts=
            ↪ true&cachePrepStmts=true");
        ds.setUsername("root");

        // prepare data
        Connection connection = ds.getConnection();
        createDoctorTable(connection);
        createDoctor(connection, 1, "Alice", true, 123);
        createDoctor(connection, 2, "Bob", true, 123);
        createDoctor(connection, 3, "Carol", false, 123);

        Semaphore txn1Pass = new Semaphore(0);
        CountDownLatch countDownLatch = new CountDownLatch(2);
        ExecutorService threadPool = Executors.newFixedThreadPool(2);

        threadPool.execute(() -> {
```

```
            askForLeave(ds, txn1Pass, 1, 1);
            countDownLatch.countDown();
        });

        threadPool.execute(() -> {
            askForLeave(ds, txn1Pass, 2, 2);
            countDownLatch.countDown();
        });

        countDownLatch.await();
    }

    public static void createDoctorTable(Connection connection) throws
        ↪ SQLException {
        connection.createStatement().executeUpdate("CREATE TABLE `doctors` ("
            ↪ +
                "    `id` int(11) NOT NULL," +
                "    `name` varchar(255) DEFAULT NULL," +
                "    `on_call` tinyint(1) DEFAULT NULL," +
                "    `shift_id` int(11) DEFAULT NULL," +
                "    PRIMARY KEY (`id`)," +
                "    KEY `idx_shift_id` (`shift_id`)" +
                "  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
                    ↪ ");
    }

    public static void createDoctor(Connection connection, Integer id,
        ↪ String name, Boolean onCall, Integer shiftID) throws SQLException
        ↪ {
        PreparedStatement insert = connection.prepareStatement(
                "INSERT INTO `doctors` (`id`, `name`, `on_call`, `shift_id`)
                    ↪ VALUES (?, ?, ?, ?)");
        insert.setInt(1, id);
        insert.setString(2, name);
        insert.setBoolean(3, onCall);
        insert.setInt(4, shiftID);
        insert.executeUpdate();
    }

    public static void askForLeave(HikariDataSource ds, Semaphore txn1Pass,
        ↪ Integer txnID, Integer doctorID) {
        try(Connection connection = ds.getConnection()) {
            try {
                connection.setAutoCommit(false);
```

```
            String comment = txnID == 2 ? " " : "" + "/* txn #{txn_id} */
                ↪ ";
            connection.createStatement().executeUpdate(comment + "BEGIN");

            // Txn 1 should be waiting for txn 2 done
            if (txnID == 1) {
                txn1Pass.acquire();
            }

            PreparedStatement currentOnCallQuery = connection.
                ↪ prepareStatement(comment +
                    "SELECT COUNT(*) AS `count` FROM `doctors` WHERE `
                        ↪ on_call` = ? AND `shift_id` = ?");
            currentOnCallQuery.setBoolean(1, true);
            currentOnCallQuery.setInt(2, 123);
            ResultSet res = currentOnCallQuery.executeQuery();

            if (!res.next()) {
                throw new RuntimeException("error query");
            } else {
                int count = res.getInt("count");
                if (count >= 2) {
                    // If current on-call doctor has 2 or more, this
                        ↪ doctor can leave
                    PreparedStatement insert = connection.prepareStatement(
                        ↪  comment +
                            "UPDATE `doctors` SET `on_call` = ? WHERE `id` =
                                ↪  ? AND `shift_id` = ?");
                    insert.setBoolean(1, false);
                    insert.setInt(2, doctorID);
                    insert.setInt(3, 123);
                    insert.executeUpdate();

                    connection.commit();
                } else {
                    throw new RuntimeException("At least one doctor is on
                        ↪ call");
                }
            }

            // Txn 2 done, let txn 1 run again
            if (txnID == 2) {
                txn1Pass.release();
            }
        } catch (Exception e) {
```

```
                // If got any error, you should roll back, data is priceless
                connection.rollback();
                e.printStackTrace();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

To adapt TiDB transactions, write a util according to the following code:

```
package main

import (
    "database/sql"
    "fmt"
    "sync"

    "github.com/pingcap-inc/tidb-example-golang/util"

    _ "github.com/go-sql-driver/mysql"
)

func main() {
    openDB("mysql", "root:@tcp(127.0.0.1:4000)/test", func(db *sql.DB) {
        writeSkew(db)
    })
}

func openDB(driverName, dataSourceName string, runnable func(db *sql.DB)) {
    db, err := sql.Open(driverName, dataSourceName)
    if err != nil {
        panic(err)
    }
    defer db.Close()

    runnable(db)
}

func writeSkew(db *sql.DB) {
    err := prepareData(db)
    if err != nil {
        panic(err)
    }
```

```
    waitingChan, waitGroup := make(chan bool), sync.WaitGroup{}

    waitGroup.Add(1)
    go func() {
        defer waitGroup.Done()
        err = askForLeave(db, waitingChan, 1, 1)
        if err != nil {
            panic(err)
        }
    }()

    waitGroup.Add(1)
    go func() {
        defer waitGroup.Done()
        err = askForLeave(db, waitingChan, 2, 2)
        if err != nil {
            panic(err)
        }
    }()

    waitGroup.Wait()
}

func askForLeave(db *sql.DB, waitingChan chan bool, goroutineID, doctorID
    ↪ int) error {
    txnComment := fmt.Sprintf("/* txn %d */ ", goroutineID)
    if goroutineID != 1 {
        txnComment = "\t" + txnComment
    }

    txn, err := util.TiDBSqlBegin(db, true)
    if err != nil {
        return err
    }
    fmt.Println(txnComment + "start txn")

    // Txn 1 should be waiting until txn 2 is done.
    if goroutineID == 1 {
        <-waitingChan
    }

    txnFunc := func() error {
        queryCurrentOnCall := "SELECT COUNT(*) AS `count` FROM `doctors`
            ↪ WHERE `on_call` = ? AND `shift_id` = ?"
```

```
        rows, err := txn.Query(queryCurrentOnCall, true, 123)
        if err != nil {
            return err
        }
        defer rows.Close()
        fmt.Println(txnComment + queryCurrentOnCall + " successful")

        count := 0
        if rows.Next() {
            err = rows.Scan(&count)
            if err != nil {
                return err
            }
        }
        rows.Close()

        if count < 2 {
            return fmt.Errorf("at least one doctor is on call")
        }

        shift := "UPDATE `doctors` SET `on_call` = ? WHERE `id` = ? AND `
            ↪ shift_id` = ?"
        _, err = txn.Exec(shift, false, doctorID, 123)
        if err == nil {
            fmt.Println(txnComment + shift + " successful")
        }
        return err
    }

    err = txnFunc()
    if err == nil {
        txn.Commit()
        fmt.Println("[runTxn] commit success")
    } else {
        txn.Rollback()
        fmt.Printf("[runTxn] got an error, rollback: %+v\n", err)
    }

    // Txn 2 is done. Let txn 1 run again.
    if goroutineID == 2 {
        waitingChan <- true
    }

    return nil
}
```

```
func prepareData(db *sql.DB) error {
    err := createDoctorTable(db)
    if err != nil {
        return err
    }

    err = createDoctor(db, 1, "Alice", true, 123)
    if err != nil {
        return err
    }
    err = createDoctor(db, 2, "Bob", true, 123)
    if err != nil {
        return err
    }
    err = createDoctor(db, 3, "Carol", false, 123)
    if err != nil {
        return err
    }
    return nil
}

func createDoctorTable(db *sql.DB) error {
    _, err := db.Exec("CREATE TABLE IF NOT EXISTS `doctors` (" +
        "    `id` int(11) NOT NULL," +
        "    `name` varchar(255) DEFAULT NULL," +
        "    `on_call` tinyint(1) DEFAULT NULL," +
        "    `shift_id` int(11) DEFAULT NULL," +
        "    PRIMARY KEY (`id`)," +
        "    KEY `idx_shift_id` (`shift_id`)" +
        "  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin")
    return err
}

func createDoctor(db *sql.DB, id int, name string, onCall bool, shiftID int)
    ↪  error {
    _, err := db.Exec("INSERT INTO `doctors` (`id`, `name`, `on_call`, `
        ↪ shift_id`) VALUES (?, ?, ?, ?)",
        id, name, onCall, shiftID)
    return err
}
```

SQL log:

```
/* txn 1 */ BEGIN
```

```
    /* txn 2 */ BEGIN
    /* txn 2 */ SELECT COUNT(*) as `count` FROM `doctors` WHERE `on_call` =
       ↪  1 AND `shift_id` = 123
    /* txn 2 */ UPDATE `doctors` SET `on_call` = 0 WHERE `id` = 2 AND `
       ↪ shift_id` = 123
    /* txn 2 */ COMMIT
/* txn 1 */ SELECT COUNT(*) AS `count` FROM `doctors` WHERE `on_call` = 1
    ↪ and `shift_id` = 123
/* txn 1 */ UPDATE `doctors` SET `on_call` = 0 WHERE `id` = 1 AND `shift_id
    ↪ ` = 123
/* txn 1 */ COMMIT
```

Running result:

```
mysql> SELECT * FROM doctors;
+----+-------+---------+----------+
| id | name | on_call | shift_id |
+----+-------+---------+----------+
|  1 | Alice |       0 |      123 |
|  2 | Bob   |       0 |      123 |
|  3 | Carol |       0 |      123 |
+----+-------+---------+----------+
```

In both transactions, the application first checks if two or more doctors are on call; if so, it assumes that one doctor can safely take leave. Since the database uses the snapshot isolation, both checks return `2`, so both transactions move on to the next stage. `Alice` updates her record to be off duty, and so does `Bob`. Both transactions are successfully committed. Now there are no doctors on duty which violates the requirement that at least one doctor should be on call. The following diagram (quoted from ***Designing Data-Intensive Applications***) illustrates what actually happens.

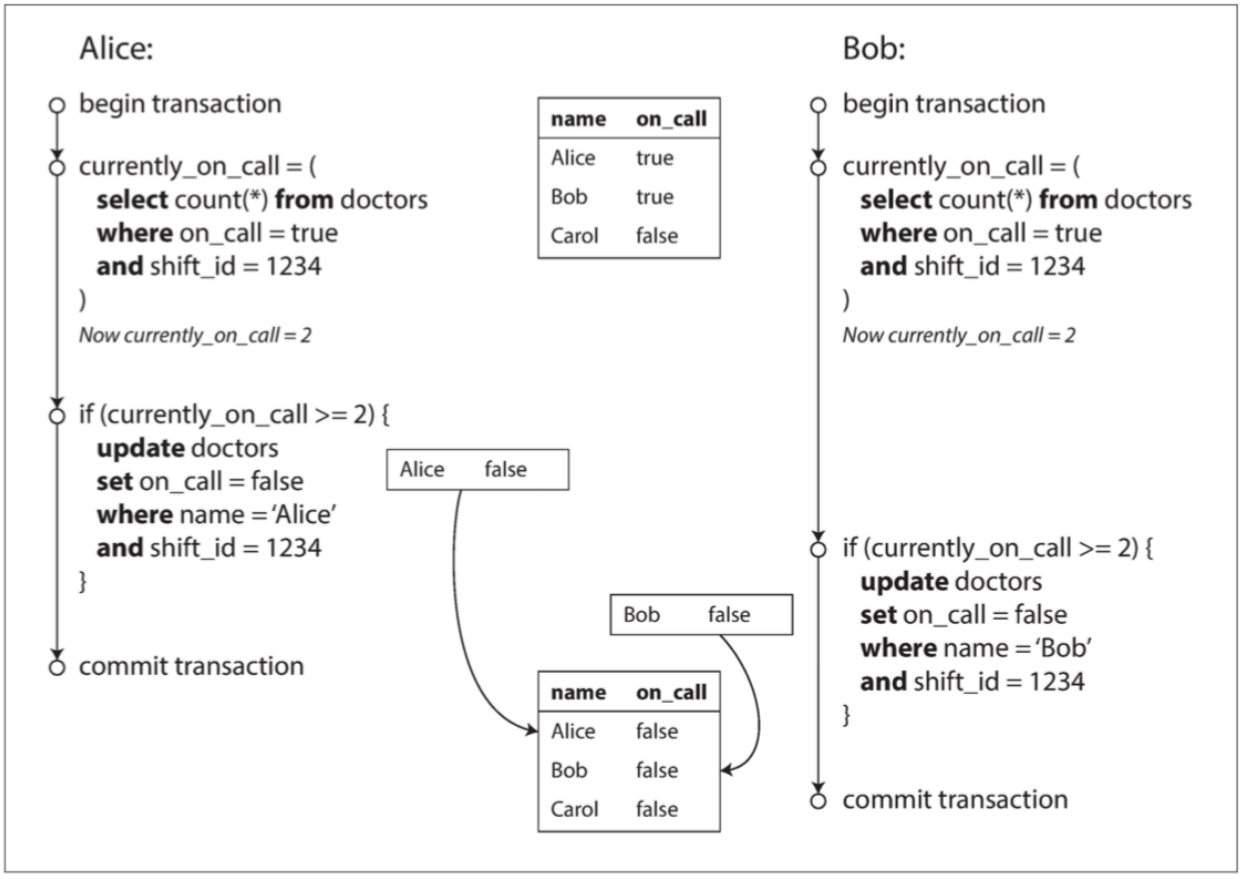


Figure 18: Write Skew

Now let's change the sample program to use `SELECT FOR UPDATE` to avoid the write skew problem:

```
package com.pingcap.txn.write.skew;

import com.zaxxer.hikari.HikariDataSource;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Semaphore;

public class EffectWriteSkew {
    public static void main(String[] args) throws SQLException,
```

```
        ↪ InterruptedException {
        HikariDataSource ds = new HikariDataSource();
        ds.setJdbcUrl("jdbc:mysql://localhost:4000/test?useServerPrepStmts=
            ↪ true&cachePrepStmts=true");
        ds.setUsername("root");

        // prepare data
        Connection connection = ds.getConnection();
        createDoctorTable(connection);
        createDoctor(connection, 1, "Alice", true, 123);
        createDoctor(connection, 2, "Bob", true, 123);
        createDoctor(connection, 3, "Carol", false, 123);

        Semaphore txn1Pass = new Semaphore(0);
        CountDownLatch countDownLatch = new CountDownLatch(2);
        ExecutorService threadPool = Executors.newFixedThreadPool(2);

        threadPool.execute(() -> {
            askForLeave(ds, txn1Pass, 1, 1);
            countDownLatch.countDown();
        });

        threadPool.execute(() -> {
            askForLeave(ds, txn1Pass, 2, 2);
            countDownLatch.countDown();
        });

        countDownLatch.await();
    }

    public static void createDoctorTable(Connection connection) throws
        ↪ SQLException {
        connection.createStatement().executeUpdate("CREATE TABLE `doctors` ("
            ↪  +
                "    `id` int(11) NOT NULL," +
                "    `name` varchar(255) DEFAULT NULL," +
                "    `on_call` tinyint(1) DEFAULT NULL," +
                "    `shift_id` int(11) DEFAULT NULL," +
                "    PRIMARY KEY (`id`)," +
                "    KEY `idx_shift_id` (`shift_id`)" +
                "  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
                    ↪ ");
    }

    public static void createDoctor(Connection connection, Integer id,
```

```
    ↪ String name, Boolean onCall, Integer shiftID) throws SQLException
    ↪ {
    PreparedStatement insert = connection.prepareStatement(
            "INSERT INTO `doctors` (`id`, `name`, `on_call`, `shift_id`)
               ↪ VALUES (?, ?, ?, ?)");
    insert.setInt(1, id);
    insert.setString(2, name);
    insert.setBoolean(3, onCall);
    insert.setInt(4, shiftID);
    insert.executeUpdate();
}

public static void askForLeave(HikariDataSource ds, Semaphore txn1Pass,
    ↪ Integer txnID, Integer doctorID) {
    try(Connection connection = ds.getConnection()) {
        try {
            connection.setAutoCommit(false);

            String comment = txnID == 2 ? " " : "" + "/* txn #{txn_id} */
               ↪ ";
            connection.createStatement().executeUpdate(comment + "BEGIN");

            // Txn 1 should be waiting for txn 2 done
            if (txnID == 1) {
                txn1Pass.acquire();
            }

            PreparedStatement currentOnCallQuery = connection.
               ↪ prepareStatement(comment +
                   "SELECT COUNT(*) AS `count` FROM `doctors` WHERE `
                      ↪ on_call` = ? AND `shift_id` = ? FOR UPDATE");
            currentOnCallQuery.setBoolean(1, true);
            currentOnCallQuery.setInt(2, 123);
            ResultSet res = currentOnCallQuery.executeQuery();

            if (!res.next()) {
                throw new RuntimeException("error query");
            } else {
                int count = res.getInt("count");
                if (count >= 2) {
                    // If current on-call doctor has 2 or more, this
                       ↪ doctor can leave
                    PreparedStatement insert = connection.prepareStatement(
                       ↪  comment +
                           "UPDATE `doctors` SET `on_call` = ? WHERE `id` =
```

```
                                ↪  ? AND `shift_id` = ?");
                        insert.setBoolean(1, false);
                        insert.setInt(2, doctorID);
                        insert.setInt(3, 123);
                        insert.executeUpdate();

                        connection.commit();
                    } else {
                        throw new RuntimeException("At least one doctor is on
                            ↪ call");
                    }
                }

                // Txn 2 done, let txn 1 run again
                if (txnID == 2) {
                    txn1Pass.release();
                }
            } catch (Exception e) {
                // If got any error, you should roll back, data is priceless
                connection.rollback();
                e.printStackTrace();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
package main

import (
    "database/sql"
    "fmt"
    "sync"

    "github.com/pingcap-inc/tidb-example-golang/util"

    _ "github.com/go-sql-driver/mysql"
)

func main() {
    openDB("mysql", "root:@tcp(127.0.0.1:4000)/test", func(db *sql.DB) {
        writeSkew(db)
    })
}
```

```
func openDB(driverName, dataSourceName string, runnable func(db *sql.DB)) {
    db, err := sql.Open(driverName, dataSourceName)
    if err != nil {
        panic(err)
    }
    defer db.Close()

    runnable(db)
}

func writeSkew(db *sql.DB) {
    err := prepareData(db)
    if err != nil {
        panic(err)
    }

    waitingChan, waitGroup := make(chan bool), sync.WaitGroup{}

    waitGroup.Add(1)
    go func() {
        defer waitGroup.Done()
        err = askForLeave(db, waitingChan, 1, 1)
        if err != nil {
            panic(err)
        }
    }()

    waitGroup.Add(1)
    go func() {
        defer waitGroup.Done()
        err = askForLeave(db, waitingChan, 2, 2)
        if err != nil {
            panic(err)
        }
    }()

    waitGroup.Wait()
}

func askForLeave(db *sql.DB, waitingChan chan bool, goroutineID, doctorID
    ↪ int) error {
    txnComment := fmt.Sprintf("/* txn %d */ ", goroutineID)
    if goroutineID != 1 {
        txnComment = "\t" + txnComment
```

```
}

txn, err := util.TiDBSqlBegin(db, true)
if err != nil {
    return err
}
fmt.Println(txnComment + "start txn")

// Txn 1 should be waiting until txn 2 is done.
if goroutineID == 1 {
    <-waitingChan
}

txnFunc := func() error {
    queryCurrentOnCall := "SELECT COUNT(*) AS `count` FROM `doctors`
        ↪ WHERE `on_call` = ? AND `shift_id` = ?"
    rows, err := txn.Query(queryCurrentOnCall, true, 123)
    if err != nil {
        return err
    }
    defer rows.Close()
    fmt.Println(txnComment + queryCurrentOnCall + " successful")

    count := 0
    if rows.Next() {
        err = rows.Scan(&count)
        if err != nil {
            return err
        }
    }
    rows.Close()

    if count < 2 {
        return fmt.Errorf("at least one doctor is on call")
    }

    shift := "UPDATE `doctors` SET `on_call` = ? WHERE `id` = ? AND `
        ↪ shift_id` = ?"
    _, err = txn.Exec(shift, false, doctorID, 123)
    if err == nil {
        fmt.Println(txnComment + shift + " successful")
    }
    return err
}
```

```
    err = txnFunc()
    if err == nil {
        txn.Commit()
        fmt.Println("[runTxn] commit success")
    } else {
        txn.Rollback()
        fmt.Printf("[runTxn] got an error, rollback: %+v\n", err)
    }

    // Txn 2 is done. Let txn 1 run again.
    if goroutineID == 2 {
        waitingChan <- true
    }

    return nil
}

func prepareData(db *sql.DB) error {
    err := createDoctorTable(db)
    if err != nil {
        return err
    }

    err = createDoctor(db, 1, "Alice", true, 123)
    if err != nil {
        return err
    }
    err = createDoctor(db, 2, "Bob", true, 123)
    if err != nil {
        return err
    }
    err = createDoctor(db, 3, "Carol", false, 123)
    if err != nil {
        return err
    }
    return nil
}

func createDoctorTable(db *sql.DB) error {
    _, err := db.Exec("CREATE TABLE IF NOT EXISTS `doctors` (" +
        "    `id` int(11) NOT NULL," +
        "    `name` varchar(255) DEFAULT NULL," +
        "    `on_call` tinyint(1) DEFAULT NULL," +
        "    `shift_id` int(11) DEFAULT NULL," +
        "    PRIMARY KEY (`id`)," +
```

```
        "    KEY `idx_shift_id` (`shift_id`)" +
        "  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin")
    return err
}

func createDoctor(db *sql.DB, id int, name string, onCall bool, shiftID int)
    ↪  error {
    _, err := db.Exec("INSERT INTO `doctors` (`id`, `name`, `on_call`, `
        ↪ shift_id`) VALUES (?, ?, ?, ?)",
        id, name, onCall, shiftID)
    return err
}
```

SQL log:

```
/* txn 1 */ BEGIN
    /* txn 2 */ BEGIN
    /* txn 2 */ SELECT COUNT(*) AS `count` FROM `doctors` WHERE on_call = 1
        ↪  AND `shift_id` = 123 FOR UPDATE
    /* txn 2 */ UPDATE `doctors` SET on_call = 0 WHERE `id` = 2 AND `
        ↪ shift_id` = 123
    /* txn 2 */ COMMIT
/* txn 1 */ SELECT COUNT(*) AS `count` FROM `doctors` WHERE `on_call` = 1
    ↪ FOR UPDATE
At least one doctor is on call
/* txn 1 */ ROLLBACK
```

Running result:

```
mysql> SELECT * FROM doctors;
+----+-------+---------+----------+
| id | name | on_call | shift_id |
+----+-------+---------+----------+
|  1 | Alice |       1 |      123 |
|  2 | Bob   |       0 |      123 |
|  3 | Carol |       0 |      123 |
+----+-------+---------+----------+
```

4.8.3.4 savepoint and nested transactions are not supported

TiDB does ***NOT*** support the `savepoint` mechanism and therefore does not support the `PROPAGATION_NESTED` propagation behavior. If your applications are based on the **Java Spring** framework that use the `PROPAGATION_NESTED` propagation behavior, you need to adapt it on the application side to remove the logic for nested transactions.

The `PROPAGATION_NESTED` propagation behavior supported by **Spring** triggers a nested transaction, which is a child transaction that is started independently of the current transaction. A `savepoint` is recorded when the nested transaction starts. If the nested transaction fails, the transaction will roll back to the `savepoint` state. The nested transaction is part of the outer transaction and will be committed together with the outer transaction.

The following example demonstrates the `savepoint` mechanism:

```
mysql> BEGIN;
mysql> INSERT INTO T2 VALUES(100);
mysql> SAVEPOINT svp1;
mysql> INSERT INTO T2 VALUES(200);
mysql> ROLLBACK TO SAVEPOINT svp1;
mysql> RELEASE SAVEPOINT svp1;
mysql> COMMIT;
mysql> SELECT * FROM T2;
+------+
|  ID  |
+------+
|  100 |
+------+
```

4.8.3.5 Large transaction restrictions

The basic principle is to limit the size of the transaction. At the KV level, TiDB has a restriction on the size of a single transaction. At the SQL level, one row of data is mapped to one KV entry, and each additional index will add one KV entry. The restriction is as follows at the SQL level:

- The maximum single row record size is `120 MB`. You can adjust it by using the `performance.txn-entry-size-limit` configuration parameter of tidb-server for TiDB v4.0.10 and later v4.0.x versions, TiDB v5.0.0 and later versions. The value is `6 MB` for versions earlier than v4.0.10.
- The maximum single transaction size supported is `10 GB`. You can configure it by `performance.txn-total-size-limit` for TiDB v4.0 and later versions. The value is `100 MB` for earlier versions.

Note that for both the size restrictions and row restrictions, you should also consider the overhead of encoding and additional keys for the transaction during the transaction execution. To achieve optimal performance, it is recommended to write one transaction every 100 ~ 500 rows.

4.8.3.6 Auto-committed `SELECT FOR UPDATE` statements do NOT wait for locks

Currently locks are not added to auto-committed `SELECT FOR UPDATE` statements. The effect is shown in the following figure:

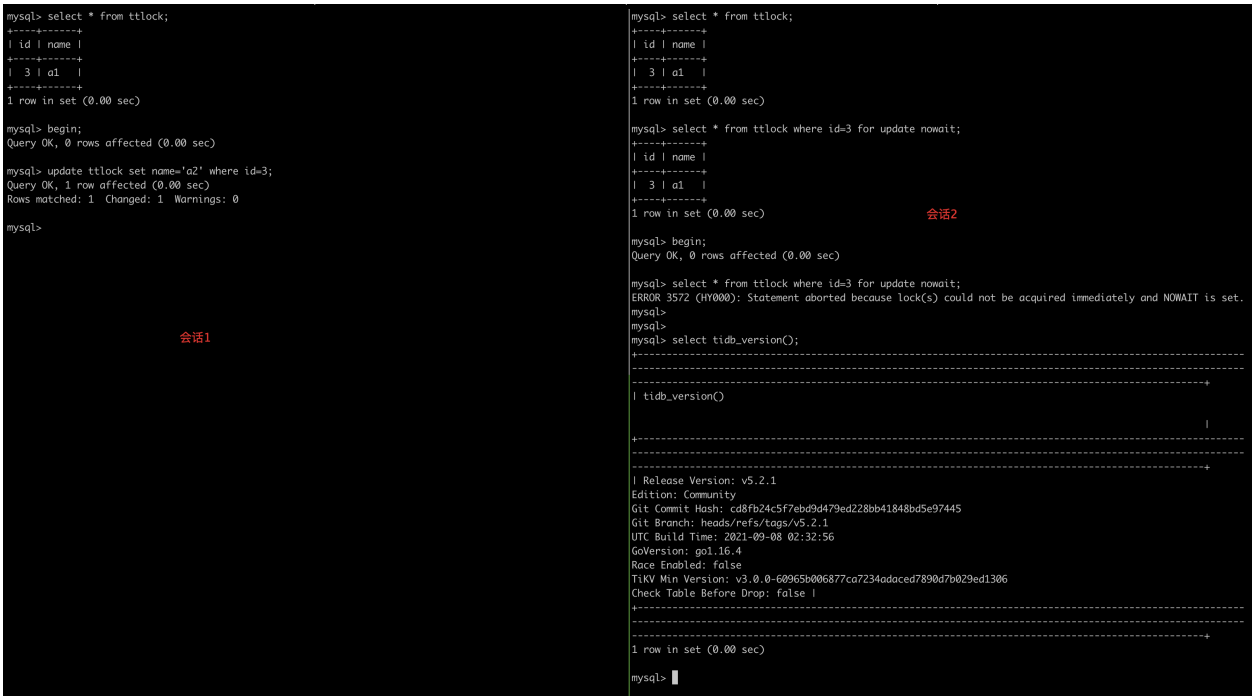


Figure 19: The situation in TiDB

This is a known incompatibility issue with MySQL. You can solve this issue by using the explicit `BEGIN;COMMIT;` statements.

4.8.4 Handle Transaction Errors

This document introduces how to handle transaction errors, such as deadlocks and application retry errors.

4.8.4.1 Deadlocks

The following error in your application indicates a deadlock issue:

```
ERROR 1213: Deadlock found when trying to get lock; try restarting
    ↪ transaction
```

A deadlock occurs when two or more transactions are waiting for each other to release the lock they already hold, or the inconsistent lock order results in a loop waiting for the lock resources.

The following is an example of a deadlock using the table `books` in the `bookshop` database:

First, insert 2 rows into the table `books`:

```
INSERT INTO books (id, title, stock, published_at) VALUES (1, 'book-1', 10,
    ↪ now()), (2, 'book-2', 10, now());
```

In TiDB pessimistic transaction mode, if two clients execute the following statements respectively, a deadlock will occur:

Client-A	Client-B
BEGIN;	
	BEGIN;
UPDATE books SET stock=stock-1 WHERE id=1;	
	UPDATE books SET s
UPDATE books SET stock=stock-1 WHERE id=2; – execution will be blocked	
	UPDATE books SET s

After client-B encounters a deadlock error, TiDB automatically rolls back the transaction in client-B. Updating `id=2` in client-A will be executed successfully. You can then run `COMMIT` to finish the transaction.

4.8.4.1.1 Solution 1: avoid deadlocks

To get better performance, you can avoid deadlocks at the application level by adjusting the business logic or schema design. In the example above, if client-B also uses the same update order as client-A, that is, they update books with `id=1` first, and then update books with `id=2`. The deadlock can then be avoided:

Client-A	Client-B
BEGIN;	
	BEGIN;
UPDATE books SET stock=stock-1 WHERE id=1;	
	UPDATE books SET stock=stock-1 WHERE id=1
UPDATE books SET stock=stock-1 WHERE id=2;	
COMMIT;	
	UPDATE books SET stock=stock-1 WHERE id=2
	COMMIT;

Alternatively, you can update 2 books with 1 SQL statement, which can also avoid the deadlock and execute more efficiently:

```
UPDATE books SET stock=stock-1 WHERE id IN (1, 2);
```

4.8.4.1.2 Solution 2: reduce transaction granularity

If you only update 1 book in each transaction, you can also avoid deadlocks. However, the trade-off is that too small transaction granularity may affect performance.

4.8.4.1.3 Solution 3: use optimistic transactions

There are no deadlocks in the optimistic transaction model. But in your application, you need to add the optimistic transaction retry logic in case of failure. For details, see Application retry and error handling.

4.8.4.1.4 Solution 4: retry

Add the retry logic in the application as suggested in the error message. For details, see Application retry and error handling.

4.8.4.2 Application retry and error handling

Although TiDB is as compatible as possible with MySQL, the nature of its distributed system leads to certain differences. One of them is the transaction model.

The Adapters and ORMs that developers use to connect with databases are tailored for traditional databases such as MySQL and Oracle. In these databases, transactions rarely fail to commit at the default isolation level, so retry mechanisms are not required. When a transaction fails to commit, these clients abort due to an error, as it is treated as an exception in these databases.

Different from traditional databases such as MySQL, in TiDB, if you use the optimistic transaction model and want to avoid commit failure, you need to add a mechanism to handle related exceptions in your applications.

The following Python pseudocode shows how to implement application-level retries. It does not require your driver or ORM to implement advanced retry logic. It can be used in any programming language or environment.

Your retry logic must follow the following rules:

- Throws an error if the number of failed retries reaches the `max_retries` limit.
- Use `try ... catch ...` to catch SQL execution exceptions. Retry when encountering the following errors. Roll back when encountering other errors.
    - `Error 8002: can not retry select for update statement`: SELECT FOR UPDATE write conflict error
    - `Error 8022: Error: KV error safe to retry`: transaction commit failed error.
    - `Error 8028: Information schema is changed during the execution of the statement`: Table schema has been changed by DDL operation, resulting in an error in the transaction commit.
    - `Error 9007: Write conflict`: Write conflict error, usually caused by multiple transactions modifying the same row of data when the optimistic transaction mode is used.

- `COMMIT` the transaction at the end of the try block.

For more information about error codes, see Error Codes and Troubleshooting.

```
while True:
    n++
    if n == max_retries:
        raise("did not succeed within #{n} retries")
    try:
        connection.execute("your sql statement here")
        connection.exec('COMMIT')
        break
    catch error:
        if (error.code != "9007" && error.code != "8028" && error.code != "
            ↪ 8002" && error.code != "8022"):
            raise error
        else:
            connection.exec('ROLLBACK')

            # Capture the error types that require application-side retry,
            # wait for a short period of time,
            # and exponentially increase the wait time for each transaction
                ↪ failure
            sleep_ms = int(((1.5 ** n) + rand) * 100)
            sleep(sleep_ms) # make sure your sleep() takes milliseconds
```

> **Note:**
>
> If you frequently encounter `Error 9007: Write conflict`, you may need to check your schema design and the data access patterns of your workload to find the root cause of the conflict and try to avoid conflicts by a better design.

For information about how to troubleshoot and resolve transaction conflicts, see Troubleshoot Lock Conflicts.

4.8.4.3 See also

- Troubleshoot Write Conflicts in Optimistic Transactions
- Troubleshoot Write Conflicts in Optimistic Transactions

4.9 Optimize

4.9.1 Overview of Optimizing SQL Performance

This document introduces how to optimize the performance of SQL statements in TiDB. To get good performance, you can start with the following aspects:

- SQL performance tuning
- Schema design: Based on your application workload patterns, you might need to change the table schema to avoid transaction contention or hot spots.

4.9.1.1 SQL performance tuning

To get good SQL statement performance, you can follow these guidelines:

- Scan as few rows as possible. It is recommended to scan only the data you need and avoid scanning excess data.
- Use the right index. Ensure that there is a corresponding index for the column in the `WHERE` clause in SQL. If not, the statement entails a full table scan and thus causes poor performance.
- Use the right join type. It is important to choose the right join type based on the relative size of the tables involved in the query. In general, TiDB's cost-based optimizer picks the best-performing join type. However, in a few cases, you might need to manually specify a better join type.
- Use the right storage engine. For hybrid OLTP and OLAP workloads, the TiFlash engine is recommended. For details, see HTAP Query.

4.9.1.2 Schema design

After tuning SQL performance, if your application still cannot get good performance, you might need to check your schema design and data access patterns to avoid the following issues:

- Transaction contention. For how to diagnose and resolve transaction contention, see Troubleshoot Lock Conflicts.
- Hot spots. For how to diagnose and resolve hot spots, see Troubleshoot Hotspot Issues.

4.9.1.2.1 See also

- SQL Performance Tuning

4.9.2 SQL Performance Tuning

This document introduces some common reasons for slow SQL statements and techniques for tuning SQL performance.

4.9.2.1 Before you begin

You can use `tiup demo import` to prepare data:

```
tiup demo bookshop prepare --host 127.0.0.1 --port 4000 --books 1000000
```

Or using the Import feature of TiDB Cloud to import the pre-prepared sample data.

4.9.2.2 Issue: Full table scan

The most common reason for slow SQL queries is that the `SELECT` statements perform full table scan or use incorrect indexes.

When TiDB retrieves a small number of rows from a large table based on a column that is not the primary key or in the secondary index, the performance is usually poor:

```
SELECT * FROM books WHERE title = 'Marian Yost';
```

```
+--
    ↪ ----------+-------------+----------------------+--------------------+-------+-
    ↪
| id         | title      | type                 | published_at       | stock |
    ↪ price |
+--
    ↪ ----------+-------------+----------------------+--------------------+-------+-
    ↪
| 65670536   | Marian Yost | Arts                 | 1950-04-09 06:28:58 | 542 |
    ↪ 435.01 |
| 1164070689 | Marian Yost | Education & Reference | 1916-05-27 12:15:35 |
    ↪ 216 | 328.18 |
| 1414277591 | Marian Yost | Arts                 | 1932-06-15 09:18:14 | 303 |
    ↪ 496.52 |
| 2305318593 | Marian Yost | Arts                 | 2000-08-15 19:40:58 | 398 |
    ↪ 402.90 |
| 2638226326 | Marian Yost | Sports               | 1952-04-02 12:40:37 | 191 |
    ↪ 174.64 |
+--
    ↪ ----------+-------------+----------------------+--------------------+-------+-
    ↪
5 rows in set
Time: 0.582s
```

To understand why this query is slow, you can use `EXPLAIN` to see the execution plan:

```
EXPLAIN SELECT * FROM books WHERE title = 'Marian Yost';
```

```
+--
   ↪ ------------------+-----------+----------+--------------+-------------------
   ↪
| id                | estRows   | task     | access object | operator info
   ↪                     |
+--
   ↪ ------------------+-----------+----------+--------------+-------------------
   ↪
| TableReader_7     | 1.27      | root     |               | data:Selection_6
   ↪                     |
|  -Selection_6     | 1.27      | cop[tikv] |              | eq(bookshop.books.
   ↪ title, "Marian Yost") |
|    -TableFullScan_5 | 1000000.00 | cop[tikv] | table:books | keep order:
   ↪ false               |
+--
   ↪ ------------------+-----------+----------+--------------+-------------------
   ↪
```

As can be seen from `TableFullScan_5` in the execution plan, TiDB performs a full table scan on the `books` table and checks whether `title` satisfies the condition for each row. The `estRows` value of `TableFullScan_5` is `1000000.00`, which means the optimizer estimates that this full table scan takes `1000000.00` rows of data.

For more information about the usage of `EXPLAIN`, see EXPLAIN Walkthrough.

4.9.2.2.1 Solution: Use secondary index

To speed up this query above, add a secondary index on the `books.title` column:

```
CREATE INDEX title_idx ON books (title);
```

The query execution is much faster:

```
SELECT * FROM books WHERE title = 'Marian Yost';
```

```
+--
   ↪ ----------+------------+----------------------+-------------------+-------+-
   ↪
| id         | title      | type                 | published_at      | stock |
   ↪ price |
+--
   ↪ ----------+------------+----------------------+-------------------+-------+-
   ↪
| 1164070689 | Marian Yost | Education & Reference | 1916-05-27 12:15:35 |
   ↪ 216 | 328.18 |
```

```
| 1414277591 | Marian Yost | Arts            | 1932-06-15 09:18:14 | 303 |
    ↪ 496.52 |
| 2305318593 | Marian Yost | Arts            | 2000-08-15 19:40:58 | 398 |
    ↪ 402.90 |
| 2638226326 | Marian Yost | Sports          | 1952-04-02 12:40:37 | 191 |
    ↪ 174.64 |
| 65670536   | Marian Yost | Arts            | 1950-04-09 06:28:58 | 542 |
    ↪ 435.01 |
+--
    ↪ ----------+-------------+----------------------+---------------------+-------+-
    ↪
5 rows in set
Time: 0.007s
```

To understand why the performance is improved, use `EXPLAIN` to see the new execution plan:

```
EXPLAIN SELECT * FROM books WHERE title = 'Marian Yost';
```

```
+--
    ↪ -----------------------+---------+-----------+---------------------------------
    ↪
| id                       | estRows | task      | access object
    ↪                      | operator info                                    |
+--
    ↪ -----------------------+---------+-----------+---------------------------------
    ↪
| IndexLookUp_10           | 1.27    | root      |
    ↪                                  |
    ↪                                                        |
|  ─IndexRangeScan_8(Build) | 1.27 | cop[tikv] | table:books, index:
    ↪ title_idx(title) | range:["Marian Yost","Marian Yost"], keep order:
    ↪ false |
|  ─TableRowIDScan_9(Probe) | 1.27 | cop[tikv] | table:books
    ↪                    | keep order:false                                  |
+--
    ↪ -----------------------+---------+-----------+---------------------------------
    ↪
```

As can be seen from `IndexLookup_10` in the execution plan, TiDB queries the data by the `title_idx` index. Its `estRows` value is `1.27`, which means that the optimizer estimates that only `1.27` rows are scanned. The estimated rows scanned are much fewer than the `1000000.00` rows of data in the full table scan.

The `IndexLookup_10` execution plan is to first use the `IndexRangeScan_8` operator to read the index data that meets the condition through the `title_idx` index, and then use the

`TableLookup_9` operator to query the corresponding rows according to the Row ID stored in the index data.

For more information on the TiDB execution plan, see TiDB Query Execution Plan Overview.

4.9.2.2.2 Solution: Use covering index

If the index is a covering index, which contains all the columns queried by the SQL statements, scanning the index data is sufficient for the query.

For example, in the following query, you only need to query the corresponding `price` based on `title`:

```
SELECT title, price FROM books WHERE title = 'Marian Yost';
```

```
+-------------+--------+
| title      | price |
+-------------+--------+
| Marian Yost | 435.01 |
| Marian Yost | 328.18 |
| Marian Yost | 496.52 |
| Marian Yost | 402.90 |
| Marian Yost | 174.64 |
+-------------+--------+
5 rows in set
Time: 0.007s
```

Because the `title_idx` index only contains data in the `title` column, TiDB still needs to first scan the index data and then query the `price` column from the table.

```
EXPLAIN SELECT title, price FROM books WHERE title = 'Marian Yost';
```

```
+--
    ↪ ------------------------+---------+-----------+-----------------------------
    ↪
| id                     | estRows | task    | access object
    ↪                   | operator info                                  |
+--
    ↪ ------------------------+---------+-----------+-----------------------------
    ↪
| IndexLookUp_10         | 1.27    | root      |
    ↪                                  |
    ↪                                                 |
|  ─IndexRangeScan_8(Build) | 1.27 | cop[tikv] | table:books, index:
    ↪ title_idx(title) | range:["Marian Yost","Marian Yost"], keep order:
    ↪ false |
```

```
|  ─TableRowIDScan_9(Probe) | 1.27 | cop[tikv] | table:books
   ↪                  | keep order:false                              |
+--
   ↪ -------------------------+---------+-----------+-------------------------------
   ↪
```

To optimize the performance, drop the `title_idx` index and create a new covering index `title_price_idx`:

```
ALTER TABLE books DROP INDEX title_idx;
```

```
CREATE INDEX title_price_idx ON books (title, price);
```

Because the `price` data is stored in the `title_price_idx` index, the following query only needs to scan the index data:

```
EXPLAIN SELECT title, price FROM books WHERE title = 'Marian Yost';
```

```
--
   ↪ ------------------+---------+-----------+-----------------------------------
   ↪
| id              | estRows | task    | access object
   ↪                              | operator info
   ↪                                  |
+--
   ↪ ------------------+---------+-----------+-----------------------------------
   ↪
| IndexReader_6   | 1.27   | root     |
   ↪                                          | index:IndexRangeScan_5
   ↪                          |
|  ─IndexRangeScan_5 | 1.27 | cop[tikv] | table:books, index:
   ↪ title_price_idx(title, price) | range:["Marian Yost","Marian Yost"],
   ↪ keep order:false |
+--
   ↪ ------------------+---------+-----------+-----------------------------------
   ↪
```

Now this query runs faster:

```
SELECT title, price FROM books WHERE title = 'Marian Yost';
```

```
+-------------+--------+
| title      | price |
+-------------+--------+
| Marian Yost | 174.64 |
| Marian Yost | 328.18 |
```

```
| Marian Yost | 402.90 |
| Marian Yost | 435.01 |
| Marian Yost | 496.52 |
+-------------+--------+
5 rows in set
Time: 0.004s
```

Since the `books` table will be used in later examples, drop the `title_price_idx` index:

```
ALTER TABLE books DROP INDEX title_price_idx;
```

4.9.2.2.3 Solution: Use primary index

If a query uses the primary key to filter data, the query runs fast. For example, the primary key of the `books` table is the `id` column, so you can use the `id` column to query data:

```
SELECT * FROM books WHERE id = 896;
```

```
+--
    ↪ ---+---------------+---------------------+---------------------+-------+------
    ↪
| id  | title         | type                | published_at        | stock | price
    ↪ |
+--
    ↪ ---+---------------+---------------------+---------------------+-------+------
    ↪
| 896 | Kathryne Doyle | Science & Technology | 1969-03-18 01:34:15 | 468 |
    ↪ 281.32 |
+--
    ↪ ---+---------------+---------------------+---------------------+-------+------
    ↪
1 row in set
Time: 0.004s
```

Use `EXPLAIN` to see the execution plan:

```
EXPLAIN SELECT * FROM books WHERE id = 896;
```

```
+-------------+---------+------+--------------+---------------+
| id          | estRows | task | access object | operator info |
+-------------+---------+------+--------------+---------------+
| Point_Get_1 | 1.00 | root | table:books | handle:896 |
+-------------+---------+------+--------------+---------------+
```

`Point_Get` is a very fast execute plan.

4.9.2.3 Use the right join type

See JOIN Execution Plan.

4.9.2.3.1 See also

- EXPLAIN Walkthrough
- Explain Statements That Use Indexes

4.9.3 Performance Tuning Best Practices

This document introduces some best practices for using TiDB databases.

4.9.3.1 DML best practices

This section describes the best practices involved when you use DML with TiDB.

4.9.3.1.1 Use multi-row statements

When you need to modify multiple rows of table, it is recommended to use multi-row statements:

```
INSERT INTO t VALUES (1, 'a'), (2, 'b'), (3, 'c');

DELETE FROM t WHERE id IN (1, 2, 3);
```

It is not recommended to use multiple single-row statements:

```
INSERT INTO t VALUES (1, 'a');
INSERT INTO t VALUES (2, 'b');
INSERT INTO t VALUES (3, 'c');

DELETE FROM t WHERE id = 1;
DELETE FROM t WHERE id = 2;
DELETE FROM t WHERE id = 3;
```

4.9.3.1.2 Use `PREPARE`

When you need to execute a SQL statement for multiple times, it is recommended to use the `PREPARE` statement to avoid the overhead of repeatedly parsing the SQL syntax.

```
func BatchInsert(db *sql.DB) error {
    stmt, err := db.Prepare("INSERT INTO t (id) VALUES (?), (?), (?), (?),
        ↪ (?)")
    if err != nil {
        return err
```

```
    }
    for i := 0; i < 1000; i += 5 {
        values := []interface{}{i, i + 1, i + 2, i + 3, i + 4}
        _, err = stmt.Exec(values...)
        if err != nil {
            return err
        }
    }
    return nil
}
```

```
public void batchInsert(Connection connection) throws SQLException {
    PreparedStatement statement = connection.prepareStatement(
            "INSERT INTO `t` (`id`) VALUES (?), (?), (?), (?), (?)");
    for (int i = 0; i < 1000; i ++) {
        statement.setInt(i % 5 + 1, i);

        if (i % 5 == 4) {
            statement.executeUpdate();
        }
    }
}
```

Do not execute the `PREPARE` statement repeatedly. Otherwise, the execution efficiency cannot be improved.

4.9.3.1.3 Only query the columns you need

If you do not need data from all columns, do not use `SELECT *` to return all columns data. The following query is inefficient:

```
SELECT * FROM books WHERE title = 'Marian Yost';
```

You should only query the columns you need. For example:

```
SELECT title, price FROM books WHERE title = 'Marian Yost';
```

4.9.3.1.4 Use bulk delete

When you delete a large amount of data, it is recommended to use bulk delete.

4.9.3.1.5 Use bulk update

When you update a large amount of data, it is recommended to use bulk update.

4.9.3.1.6 Use `TRUNCATE` instead of `DELETE` for full table data

When you need to delete all data from a table, it is recommended to use the `TRUNCATE` statement:

```
TRUNCATE TABLE t;
```

It is not recommended to use `DELETE` for full table data:

```
DELETE FROM t;
```

4.9.3.2 DDL best practices

This section describes the best practices involved when using TiDB's DDL.

4.9.3.2.1 Primary key best practices

See the rules to follow when selecting the primary key.

4.9.3.3 Index best practices

See Index Best Practices.

4.9.3.3.1 Add index best practices

TiDB supports the online index add operation. You can use ADD INDEX or CREATE INDEX statement to add an index. It does not block data reads and writes in the table. You can adjust the concurrency and the batch size during the `re-organize` phase of the index add operation by modifying the following system variables:

- `tidb_ddl_reorg_worker_cnt`
- `tidb_ddl_reorg_batch_size`

To reduce the impact on the online application, the default speed of add index operation is slow. When the target column of add index operation only involves read load or is not directly related to online workload, you can appropriately increase the value of the above variables to speed up the add index operation:

```
SET @@global.tidb_ddl_reorg_worker_cnt = 16;
SET @@global.tidb_ddl_reorg_batch_size = 4096;
```

When the target column of the add index operation is updated frequently (including `UPDATE`, `INSERT` and `DELETE`), increasing the above variables causes more write conflicts, which impacts the online workload. Accordingly, the add index operation might take a long time to complete due to constant retries. In this case, it is recommended to decrease the value of the above variables to avoid write conflicts with the online application:

```
SET @@global.tidb_ddl_reorg_worker_cnt = 4;
SET @@global.tidb_ddl_reorg_batch_size = 128;
```

4.9.3.4 Transaction conflicts

For how to locate and resolve transaction conflicts, see Troubleshoot Lock Conflicts.

4.9.3.5 Best practices for developing Java applications with TiDB

See Best Practices for Developing Java Applications with TiDB.

4.9.3.5.1 See also

- Highly Concurrent Write Best Practices

4.9.4 Best Practices for Indexing

This document introduces some best practices for creating and using indexes in TiDB.

4.9.4.1 Before you begin

This section takes the `books` table in the bookshop database as an example.

```
CREATE TABLE `books` (
  `id` bigint(20) AUTO_RANDOM NOT NULL,
  `title` varchar(100) NOT NULL,
  `type` enum('Magazine', 'Novel', 'Life', 'Arts', 'Comics', 'Education &
    ↪ Reference', 'Humanities & Social Sciences', 'Science & Technology',
    ↪ 'Kids', 'Sports') NOT NULL,
  `published_at` datetime NOT NULL,
  `stock` int(11) DEFAULT '0',
  `price` decimal(15,2) DEFAULT '0.0',
  PRIMARY KEY (`id`) CLUSTERED
) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

4.9.4.2 Best practices for creating indexes

- Creating a combined index with multiple columns, which is an optimization called covering index optimization. **Covering index optimization** allows TiDB to query data directly on indexes, which helps improve performance.
- Avoid creating a secondary index on columns that you do not query often. A useful secondary index can speed up queries, but be aware that it also has side effects. Each time you add an index, an additional Key-Value is added when you insert a row. The more indexes you have, the slower you write, and the more space it consumes. In addition, too many indexes affect optimizer runtime, and inappropriate indexes can mislead the optimizer. So, more indexes do not always mean better performance.

- Create an appropriate index based on your application. In principle, create indexes only on the columns to be used in queries to improve performance. The following cases are suitable for creating an index:

    - Columns with a high distinction degree can significantly reduce the number of filtered rows. For example, it is recommended to create an index on the personal ID number, but not on the gender.
    - Use combined indexes when querying with multiple conditions. Note that columns with equivalent conditions need to be placed in the front of the combined index. Here is an example: if the `select* from t where c1 = 10 and c2 = 100 and c3 > 10` query is frequently used, consider creating a combined index `Index cidx (c1, c2, c3)`, so that a index prefix can be constructed to scan by query conditions.

- Name your secondary index meaningfully, and it is recommended to follow the table naming conventions of your company or organization. If such naming conventions do not exist, follow the rules in Index Naming Specification.

4.9.4.3 Best practices for using indexes

- Indexes are to speed up queries, so make sure that the existing indexes are actually used by some queries. If an index is not used by any query, the index is meaningless, and you need to drop it.

- When using a combined index, follow the left-prefix rule.

    Suppose that you create a new combined index on the `title` and `published_at` columns:

    ```
    CREATE INDEX title_published_at_idx ON books (title, published_at);
    ```

    The following query can still use the combined index:

    ```
    SELECT * FROM books WHERE title = 'database';
    ```

    However, the following query cannot use the combined index because the condition for the leftmost first column in the index is not specified:

    ```
    SELECT * FROM books WHERE published_at = '2018-08-18 21:42:08';
    ```

- When using an index column as a condition in a query, do not use calculation, function, or type conversion on it, which will prevent the TiDB optimizer from using the index.

    Suppose that you create a new index on the time type column `published_at`:

    ```
    CREATE INDEX published_at_idx ON books (published_at);
    ```

    However, the following query cannot use the index on `published_at`:

```
SELECT * FROM books WHERE YEAR(published_at)=2022;
```

To use the index on `published_at`, you can rewrite the query as follows, which avoids using any function on the index column:

```
SELECT * FROM books WHERE published_at >= '2022-01-01' AND
    ↪ published_at < '2023-01-01';
```

You can also use an expression index to create an expression index for `YEAR(Published at)` in the query condition:

```
CREATE INDEX published_year_idx ON books ((YEAR(published_at)));
```

Now, if you execute the `SELECT * FROM books WHERE YEAR(published_at)=2022;` query, the query can use the `published_year_idx` index to speed up the execution.

> **Warning:**
>
> Currently, expression index is an experimental feature, and it needs to be enabled in the TiDB configuration file. For more details, see expression index.

- Try to use a covering index, in which the columns in the index contain the columns to be queried, and avoid querying all columns with `SELECT *` statements.

  The following query only needs to scan the index `title_published_at_idx` to get the data:

  ```
  SELECT title, published_at FROM books WHERE title = 'database';
  ```

  Although the following query statement can use the combined index `(title, published_at)`, it causes an extra cost to query the non-indexed column, which requires TiDB to query row data according to the reference stored in the index data (usually the primary key information).

  ```
  SELECT * FROM books WHERE title = 'database';
  ```

- A query cannot use indexes when the query condition contains `!=` or `NOT IN`. For example, the following query cannot use any indexes:

  ```
  SELECT * FROM books WHERE title != 'database';
  ```

- A query cannot use indexes if the `LIKE` condition starts with wildcard `%` in the query. For example, the following query cannot use any indexes:

  ```
  SELECT * FROM books WHERE title LIKE '%database';
  ```

- When the query condition has multiple indexes available, and you know which index is the best in practice, it is recommended to use Optimizer Hint to force the TiDB optimizer to use this index. This can prevent the TiDB optimizer from selecting the wrong index due to inaccurate statistics or other problems.

  In the following query, assuming that indexes `id_idx` and `title_idx` are available on the column `id` and `title` respectively, if you know that `id_idx` is better, you can use `USE INDEX` hint in SQL to force the TiDB optimizer to use the `id_idx` index.

```
SELECT * FROM t USE INDEX(id_idx) WHERE id = 1 and title = 'database';
```

- When using the `IN` expression in a query condition, it is recommended that the number of value matched after it does not exceed 300, otherwise the execution efficiency will be poor.

4.9.5 Other Optimization Methods

4.9.5.1 Avoid Implicit Type Conversions

This document introduces the rules and possible consequences of implicit type conversions in TiDB and how to avoid implicit type conversions.

4.9.5.1.1 Conversion rules

When the data types on the two sides of the predicate in a SQL statement do not match, TiDB implicitly convert the data types on one or both sides to a compatible data type for predicate operations.

The rules for implicit type conversion in TiDB are as follows:

- If one or both arguments are `NULL`, the result of the comparison is `NULL`. The NULL-safe `<=>` equivalent comparison operator does not require conversion, because NULL `<=>` NULL results in `true`.
- If both arguments in the comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- If no comparison is made with numbers, the hexadecimal value is treated as a binary string.
- If one of the arguments is a decimal value, the comparison depends on the other argument. If the other argument is a decimal or integer value, the argument is compared with the decimal value. If the other argument is a floating-point value, the argument is compared with the floating-point value.
- If one of the arguments is a `TIMESTAMP` or `DATETIME` column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed.
- In all other cases, the arguments are compared as floating-point numbers (the `DOUBLE` type).

4.9.5.1.2 Consequences caused by implicit type conversion

Implicit type conversions increase the usability of human-computer interaction. However, avoid using implicit type conversions in application code, because they might lead to the following issues:

- Index invalidity
- Loss of precision

Index invalidity

In the following case, `account_id` is the primary key and its data type is `varchar`. In the execution plan, this SQL statement has an implicit type conversion and cannot use the index.

```
DESC SELECT * FROM `account` WHERE `account_id`=6010000000009801;
+--
   ↪ ----------------------+---------------+----------+--------------+----------
   ↪
| id                    | estRows       | task     | access object | operator
   ↪ info                                       |
+--
   ↪ ----------------------+---------------+----------+--------------+----------
   ↪
| TableReader_7         | 8000628000.00 | root     |               | data:
   ↪ Selection_6                                |
| ─Selection_6          | 8000628000.00 | cop[tikv] |              | eq(cast(
   ↪ findpt.account.account_id), 6.010000000009801e+15) |
|   ─TableFullScan_5    | 10000785000.00 | cop[tikv] | table:account | keep
   ↪ order:false                                |
+--
   ↪ ----------------------+---------------+----------+--------------+----------
   ↪
3 rows in set (0.00 sec)
```

Brief description of run results: From the above execution plan, the `Cast` operator is visible.

Loss of precision

In the following case, the data type of the `a` field is `decimal(32,0)`. In the execution plan, an implicit type conversion occurs, and both the decimal field and the string constant are converted to the double type. Because the precision of the double type is not as high as decimal, there is a loss of precision. In this case, the SQL statement incorrectly filters the result set out of range.

```
DESC SELECT * FROM `t1` WHERE `a` BETWEEN '12123123' AND '
   ↪ 1111222211111111200000';
```

```
+--
   ↪ ----------------------+---------+-----------+--------------+------------------
   ↪
| id                    | estRows | task    | access object | operator info
   ↪                                                                |
+--
   ↪ ----------------------+---------+-----------+--------------+------------------
   ↪
| TableReader_7         | 0.80    | root      |               | data:Selection_6
   ↪                                                                |
| └─Selection_6         | 0.80    | cop[tikv] |               | ge(cast(findpt.t1
   ↪ .a), 1.2123123e+07), le(cast(findpt.t1.a), 1.1112222111111112e+21) |
|   └─TableFullScan_5   | 1.00    | cop[tikv] | table:t1    | keep order:false,
   ↪  stats:pseudo                                                  |
+--
   ↪ ----------------------+---------+-----------+--------------+------------------
   ↪
3 rows in set (0.00 sec)
```

Brief description of run results: From the above execution plan, the `Cast` operator is visible.

```
SELECT * FROM `t1` WHERE `a` BETWEEN '12123123' AND '1111222211111111200000
   ↪ ';
+-----------------------+
| a                     |
+-----------------------+
| 1111222211111111222211 |
+-----------------------+
1 row in set (0.01 sec)
```

Brief description of run results: The above execution gives a wrong result.

4.9.5.2 Unique Serial Number Generation

This document introduces the unique serial number generation scheme to help developers who generate their own unique IDs.

4.9.5.2.1 Auto-increment column

`AUTO_INCREMENT` is a column attribute of many RDBMSs that are compatible with the MySQL protocol. With the `AUTO_INCREMENT` attribute, a database can automatically assign values to this column without user intervention. As the number of records in the table increases, the value of this column automatically increments and are guaranteed to be unique. In most scenarios, `AUTO_INCREMENT` columns are used as proxy primary keys with no actual meaning.

The limitation of `AUTO_INCREMENT` columns is that the column must be of the integer type and the values assigned to them must be integer. If the serial numbers required by the application are sliced by letters, numbers, and other characters, it is difficult for the user to get the auto-increment numbers required in the serial number through the `AUTO_INCREMENT` column.

4.9.5.2.2 Sequence

A **Sequence** is a database object that an application can invoke to produce incremental sequence values. Applications can flexibly use the sequence values to assign values to one or more tables. Applications can also use the sequence values for more complex processing to produce a combination of text and numbers. This approach gives some tracking and classification meaning to proxy keys.

Sequence is available starting with TiDB v4.0. For details, refer to sequence documentation.

4.9.5.2.3 Snowflake-like solutions

Snowflake is a distributed ID generation solution proposed by Twitter. There are several implementations, the more popular ones are Baidu's **uid-generator** and Meituan's **leaf**. This section uses `uid-generator` as an example.

The 64-bit ID structure generated by `uid-generator` is as follows:

```
| sign | delta seconds | worker node id | sequencs |
|------|---------------|----------------|----------|
| 1bit |    28bits     | 22bits         | 13bits   |
```

- sign: Fixed length of 1 bit. Fixed to `0` to indicate that the generated ID is always a positive number.
- delta seconds: 28 bits by default. The current time, presented as an incremental value in seconds relative to a preset time base (defaults to `2016-05-20`). 28 bits can support up to about 8.7 years.
- worker node id: 22 bits by default. Represents the machine ID, usually obtained from a centralized ID generator when the application process is started. Common centralized ID generators include auto-increment columns and ZooKeeper. The default allocation policy is discard-as-you-go, and the process re-acquires a new worker node ID on restart. 22 bits can support up to about 4.2 million starts.
- sequence: 13 bits by default. The sequence of concurrency per second. 13 bits can support 8192 concurrent sequences per second.

4.9.5.2.4 Number allocation solution

The number allocation solution can be understood as bulk acquisition of auto-increment IDs from the database. This scheme requires a sequence number generation table, with each row representing a sequence object. An example of table definition is as follows:

Field Name	Field Type	Field Description
`SEQ_NAME` ↪	varchar(128)	The name of the sequence, used to distinguish different applications.
`MAX_ID`	bigint(20)	The maximum value of the current sequence that has been allocated.
`STEP`	int(11)	The step, which indicates the length of each assigned segment.

Every time, the application gets a segment of sequence numbers at the configured step. It updates the database at the same time to persist the maximum value of the current sequence that has been allocated. The processing and allocation of sequence numbers are completed in the application's memory. After a segment of sequence numbers is used up, the application gets a new segment of sequence numbers, which effectively alleviates the pressure on the database write. In practice, you can also adjust the step to control the frequency of database updates.

Finally, note that the IDs generated by the above two solutions are not random enough to be directly used as **primary keys** for TiDB tables. In practice, you can perform bit-reverse on the generated IDs to get more random new IDs.

4.10 Troubleshoot

4.10.1 SQL or Transaction Issues

This document introduces problems that may occur during application development and related documents.

4.10.1.1 Troubleshoot SQL query problems

If you want to improve SQL query performance, follow the instructions in SQL Performance Tuning to solve performance problems such as full table scans and missing indexes.

If you still have performance issues, see the following documents:

- Analyze Slow Queries
- Identify Expensive Queries Using Top SQL

If you have questions about SQL operations, see SQL FAQs.

4.10.1.2 Troubleshoot transaction issues

See Handle transaction errors.

4.10.1.3 See also

- Unsupported features

- Cluster Management FAQs
- TiDB FAQs

4.10.2 Unstable Result Set

This document describes how to solve unstable result set errors.

4.10.2.1 GROUP BY

For convenience, MySQL "extends" the `GROUP BY` syntax to allow the `SELECT` clause to refer to non-aggregated fields not declared in the `GROUP BY` clause, that is, the `NON-FULL GROUP BY` syntax. In other databases, this is considered a syntax ***ERROR*** because it causes unstable result sets.

For example, you have two tables:

- `stu_info` stores the student information
- `stu_score` stores the student test scores.

Then you can write a SQL query statement like this:

```
SELECT
    `a`.`class`,
    `a`.`stuname`,
    max( `b`.`courscore` )
FROM
    `stu_info` `a`
    JOIN `stu_score` `b` ON `a`.`stuno` = `b`.`stuno`
GROUP BY
    `a`.`class`,
    `a`.`stuname`
ORDER BY
    `a`.`class`,
    `a`.`stuname`;
```

Result:

```
+-----------+-------------+-----------------+
| class     | stuname     | max(b.courscore) |
+-----------+-------------+-----------------+
| 2018_CS_01 | MonkeyDLuffy |        95.5 |
| 2018_CS_03 | PatrickStar |        99.0 |
| 2018_CS_03 | SpongeBob |          95.0 |
+-----------+-------------+-----------------+
3 rows in set (0.00 sec)
```

The `a.class` and `a.stuname` fields are specified in the `GROUP BY` statement, and the selected columns are `a.class`, `a.stuname` and `b.courscore`. The only column that is not in the `GROUP BY` condition, `b.courscore`, is also specified with a unique value using the `max()` function. There is ***ONLY ONE*** result that satisfies this SQL statement without any ambiguity, which is called the `FULL GROUP BY` syntax.

A counterexample is the `NON-FULL GROUP BY` syntax. For example, in these two tables, write the following SQL query (delete `a.stuname` in `GROUP BY`).

```
SELECT
    `a`.`class`,
    `a`.`stuname`,
    max( `b`.`courscore` )
FROM
    `stu_info` `a`
    JOIN `stu_score` `b` ON `a`.`stuno` = `b`.`stuno`
GROUP BY
    `a`.`class`
ORDER BY
    `a`.`class`,
    `a`.`stuname`;
```

Then two values that match this SQL are returned.

The first returned value:

```
    sql    +-----------+-------------+-----------------------+ | class |
↪ stuname    | max( `b`.`courscore` )| +-----------+-------------+-------------------
↪  | 2018_CS_01 | MonkeyDLuffy |         95.5 |    | 2018_CS_03 | PatrickStar
↪  |              99.0 |   +-----------+-------------+-----------------------+
↪
```

The second returned value:

```
    sql    +-----------+-------------+-----------------+ | class | stuname
↪     | max(b.courscore)|  +-----------+-------------+-----------------+
↪  | 2018_CS_01 | MonkeyDLuffy |     95.5 |    | 2018_CS_03 | SpongeBob |
↪           99.0 |   +-----------+-------------+-----------------+
```

There are two results because you did ***NOT*** specify how to get the value of the `a.stuname` field in SQL, and two results are both satisfied by SQL semantics. It results in an unstable result set. Therefore, if you want to guarantee the stability of the result set of the `GROUP BY` statement, use the `FULL GROUP BY` syntax.

MySQL provides a `sql_mode` switch `ONLY_FULL_GROUP_BY` to control whether to check the `FULL GROUP BY` syntax or not. TiDB is also compatible with this `sql_mode` switch.

```
mysql> select a.class, a.stuname, max(b.courscore) from stu_info a join
    ↪ stu_score b on a.stuno=b.stuno group by a.class order by a.class, a.
    ↪ stuname;
+-----------+-------------+------------------+
| class     | stuname     | max(b.courscore) |
+-----------+-------------+------------------+
| 2018_CS_01 | MonkeyDLuffy |         95.5 |
| 2018_CS_03 | PatrickStar |         99.0 |
+-----------+-------------+------------------+
2 rows in set (0.01 sec)

mysql> set @@sql_mode='STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION,
    ↪ ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.01 sec)

mysql> select a.class, a.stuname, max(b.courscore) from stu_info a join
    ↪ stu_score b on a.stuno=b.stuno group by a.class order by a.class, a.
    ↪ stuname;
ERROR 1055 (42000): Expression #2 of ORDER BY is not in GROUP BY clause and
    ↪  contains nonaggregated column '' which is not functionally dependent
    ↪  on columns in GROUP BY clause; this is incompatible with sql_mode=
    ↪ only_full_group_by
```

Run results: The above example shows the effect when you set `ONLY_FULL_GROUP_BY` for `sql_mode`.

4.10.2.2 ORDER BY

In SQL semantics, the result set is output in order only if the `ORDER BY` syntax is used. For a single-instance database, since the data is stored on one server, the results of multiple executions are often stable without data reorganization. Some databases (especially the MySQL InnoDB storage engine) can even output the result sets in order of the primary key or index.

As a distributed database, TiDB stores data on multiple servers. In addition, the TiDB layer does not cache data pages, so the result set order of SQL statements without `ORDER BY` is easily perceived as unstable. To output a sequential result set, you need to explicitly add the order field into the `ORDER BY` clause, which conforms to SQL semantics.

In the following example, only one field is added to the `ORDER BY` clause, and TiDB only sorts the results by that one field.

```
mysql> select a.class, a.stuname, b.course, b.courscore from stu_info a
    ↪ join stu_score b on a.stuno=b.stuno order by a.class;
+------------+-------------+------------------------+-----------+
| class      | stuname     | course                 | courscore |
+------------+-------------+------------------------+-----------+
| 2018_CS_01 | MonkeyDLuffy | PrinciplesofDatabase | 60.5 |
| 2018_CS_01 | MonkeyDLuffy | English              |     43.0 |
| 2018_CS_01 | MonkeyDLuffy | OpSwimming           |     67.0 |
| 2018_CS_01 | MonkeyDLuffy | OpFencing            |     76.0 |
| 2018_CS_01 | MonkeyDLuffy | FundamentalsofCompiling | 88.0 |
| 2018_CS_01 | MonkeyDLuffy | OperatingSystem    |     90.5 |
| 2018_CS_01 | MonkeyDLuffy | PrincipleofStatistics | 69.0 |
| 2018_CS_01 | MonkeyDLuffy | ProbabilityTheory |     76.0 |
| 2018_CS_01 | MonkeyDLuffy | Physics              |     63.5 |
| 2018_CS_01 | MonkeyDLuffy | AdvancedMathematics |   95.5 |
| 2018_CS_01 | MonkeyDLuffy | LinearAlgebra      |     92.5 |
| 2018_CS_01 | MonkeyDLuffy | DiscreteMathematics |   89.0 |
| 2018_CS_03 | SpongeBob | PrinciplesofDatabase |     88.0 |
| 2018_CS_03 | SpongeBob | English              |     79.0 |
| 2018_CS_03 | SpongeBob | OpBasketball         |     92.0 |
| 2018_CS_03 | SpongeBob | OpTennis             |     94.0 |
| 2018_CS_03 | PatrickStar | LinearAlgebra      |      6.5 |
| 2018_CS_03 | PatrickStar | AdvancedMathematics |    5.0 |
| 2018_CS_03 | SpongeBob | DiscreteMathematics |     72.0 |
| 2018_CS_03 | PatrickStar | ProbabilityTheory |     12.0 |
| 2018_CS_03 | PatrickStar | PrincipleofStatistics | 20.0 |
| 2018_CS_03 | PatrickStar | OperatingSystem    |     36.0 |
| 2018_CS_03 | PatrickStar | FundamentalsofCompiling | 2.0 |
| 2018_CS_03 | PatrickStar | DiscreteMathematics |   14.0 |
| 2018_CS_03 | PatrickStar | PrinciplesofDatabase |    9.0 |
| 2018_CS_03 | PatrickStar | English              |     60.0 |
| 2018_CS_03 | PatrickStar | OpTableTennis      |     12.0 |
| 2018_CS_03 | PatrickStar | OpPiano              |     99.0 |
| 2018_CS_03 | SpongeBob | FundamentalsofCompiling | 43.0 |
| 2018_CS_03 | SpongeBob | OperatingSystem      |     95.0 |
| 2018_CS_03 | SpongeBob | PrincipleofStatistics |   90.0 |
| 2018_CS_03 | SpongeBob | ProbabilityTheory    |     87.0 |
| 2018_CS_03 | SpongeBob | Physics              |     65.0 |
| 2018_CS_03 | SpongeBob | AdvancedMathematics |     55.0 |
| 2018_CS_03 | SpongeBob | LinearAlgebra        |     60.5 |
| 2018_CS_03 | PatrickStar | Physics            |      6.0 |
+------------+-------------+------------------------+-----------+
```

```
36 rows in set (0.01 sec)
```

Results are unstable when the `ORDER BY` values are the same. To reduce randomness, `ORDER BY` values should be unique. If you can't guarantee the uniqueness, you need to add more `ORDER BY` fields until the combination of the `ORDER BY` fields in `ORDER BY` is unique, then the result will be stable.

4.10.2.3 The result set is unstable because order by is not used in `GROUP_CONCAT()`

The result set is unstable because TiDB reads data from the storage layer in parallel, so the result set order returned by `GROUP_CONCAT()` without `ORDER BY` is easily perceived as unstable.

To let `GROUP_CONCAT()` get the result set output in order, you need to add the sorting fields to the `ORDER BY` clause, which conforms to the SQL semantics. In the following example, `GROUP_CONCAT()` that splices `customer_id` without `ORDER BY` causes an unstable result set.

1. Excluded `ORDER BY`

   First query:

```
sql   mysql> select GROUP_CONCAT( customer_id SEPARATOR ',' )FROM customer
↪  where customer_id like '200002%'; +-----------------------------------------------
↪  | GROUP_CONCAT(customer_id SEPARATOR ',')        |   +-----------------------------
↪  | 20000200992,20000200993,20000200994,20000200995,20000200996,20000200...
↪  | +-------------------------------------------------------------------------+
↪
```

   Second query:

```
sql   mysql> select GROUP_CONCAT( customer_id SEPARATOR ',' )FROM customer
↪  where customer_id like '200002%'; +-----------------------------------------------
↪  | GROUP_CONCAT(customer_id SEPARATOR ',')        |   +-----------------------------
↪  | 20000203040,20000203041,20000203042,20000203043,20000203044,20000203...
↪  | +-------------------------------------------------------------------------+
↪
```

2. Include `ORDER BY`

   First query:

```
sql   mysql> select GROUP_CONCAT( customer_id order by customer_id
↪ SEPARATOR ',' )FROM customer where customer_id like '200002%'; +--------------------
↪  | GROUP_CONCAT(customer_id SEPARATOR ',')     |   +-------------------------------
↪  | 20000200000,20000200001,20000200002,20000200003,20000200004,20000200...
↪  | +-------------------------------------------------------------------------+
↪
```

Second query:

```
sql   mysql> select GROUP_CONCAT( customer_id order by customer_id
↪ SEPARATOR ',' )FROM customer where customer_id like '200002%'; +--------------------
↪  | GROUP_CONCAT(customer_id SEPARATOR ',')      |   +-------------------------------
↪  | 20000200000,20000200001,20000200002,20000200003,20000200004,20000200...
↪  | +-------------------------------------------------------------------------+
↪
```

4.10.2.4 Unstable results in `SELECT * FROM T LIMIT N`

The returned result is related to the distribution of data on the storage node (TiKV). If multiple queries are performed, different storage units (Regions) of the storage nodes (TiKV) return results at different speeds, which can cause unstable results.

4.10.3 Timeouts in TiDB

This document describes various timeouts in TiDB to help you troubleshoot errors.

4.10.3.1 GC timeout

TiDB's transaction implementation uses the MVCC (Multiple Version Concurrency Control) mechanism. When the newly written data overwrites the old data, the old data will not be replaced, but kept together with the newly written data. The versions are distinguished by the timestamp. TiDB uses the mechanism of periodic Garbage Collection (GC) to clean up the old data that is no longer needed.

- For TiDB versions earlier than v4.0:

  By default, each MVCC version (consistency snapshots) is kept for 10 minutes. Transactions that take longer than 10 minutes to read will receive an error `GC life time ↪ is shorter than transaction duration`.

- For TiDB v4.0 and later versions:

  For running transactions that do not exceed a duration of 24 hours, garbage collection (GC) are blocked during the transaction execution. The error `GC life time is ↪ shorter than transaction duration` does not occur.

If you need longer read time temporarily in some cases, you can increase the retention time of MVCC versions:

- For TiDB versions earlier than v5.0: adjust `tikv_gc_life_time` in the `mysql.tidb` table in TiDB.
- For TiDB v5.0 and later versions: adjust the system variable `tidb_gc_life_time`.

Note that the system variable configuration takes effect globally and immediately. Increasing its value will increase the life time of all existing snapshots, and decreasing it will immediately shorten the life time of all snapshots. Too many MVCC versions will impact the performance of the TiDB cluster. So you need to change this variable back to the previous setting in time.

> **Tip:**
>
> Specifically, when Dumpling is exporting data from TiDB (less than 1 TB), if the TiDB version is v4.0.0 or later and Dumpling can access the PD address and the `INFORMATION_SCHEMA.CLUSTER_INFO` table of the TiDB cluster, Dumpling automatically adjusts the GC safe point to block GC without affecting the original cluster.
>
> However, in either of the following scenarios, Dumpling cannot automatically adjust the GC time:
>
> - The data size is very large (more than 1 TB).
> - Dumpling cannot connect directly to PD, for example, the TiDB cluster is on TiDB Cloud or on Kubernetes that is separated from Dumpling.
>
> In such scenarios, you must manually extend the GC time in advance to avoid export failure due to GC during the export process.
>
> For more details, see Manually set the TiDB GC time.

For more information about GC, see GC Overview.

4.10.3.2 Transaction timeout

GC does not affect ongoing transactions. However, there is still an upper limit to the number of pessimistic transactions that can run, with a limit on the transaction timeout and a limit on the memory used by the transaction. You can modify the transaction timeout by `max-txn-ttl` under the `[performance]` category of the TiDB profile, `60` minutes by default.

SQL statements such as `INSERT INTO t10 SELECT * FROM t1` are not affected by GC, but will be rolled back due to timeout after exceeding `max-txn-ttl`.

4.10.3.3 SQL execution timeout

TiDB also provides a system variable (`max_execution_time`, `0` by default, indicating no limit) to limit the execution time of a single SQL statement. `max_execution_time` currently takes effect for all types of statements, not just the `SELECT` statements. The unit is `ms`, but the actual precision is at the `100ms` level instead of the millisecond level.

4.10.3.4 JDBC query timeout

Starting from v6.1.0, when the `enable-global-kill` configuration item is set to its default value `true`, you can use the `setQueryTimeout()` method provided by MySQL JDBC to control the query timeout.

> **Note:**
>
> When your TiDB version is earlier than v6.1.0 or `enable-global-kill` is set to `false`, `setQueryTimeout()` does not work for TiDB. This is because the client sends a `KILL` command to the database when it detects the query timeout. However, because the TiDB service is load balanced, TiDB does not execute the `KILL` command to avoid termination of the connection on a wrong TiDB node. In such cases, you can use `max_execution_time` to control query timeout.

TiDB provides the following MySQL-compatible timeout control parameters.

- **wait_timeout**, controls the non-interactive idle timeout for the connection to Java applications. Since TiDB v5.4, the default value of `wait_timeout` is `28800` seconds, which is 8 hours. For TiDB versions earlier than v5.4, the default value is `0`, which means the timeout is unlimited.
- **interactive_timeout**, controls the interactive idle timeout for the connection to Java applications. The value is `8 hours` by default.
- **max_execution_time**, controls the timeout for SQL execution in the connection. The value is `0` by default, which allows the connection to be infinitely busy, that is, an SQL statement is executed for an infinitely long time.

However, in a real production environment, idle connections and indefinitely executing SQL statements have a negative effect on both the database and the application. You can avoid idle connections and indefinitely executing SQL statements by configuring these two session-level variables in your application's connection string. For example, set the following:

- `sessionVariables=wait_timeout=3600` (1 hour)
- `sessionVariables=max_execution_time=300000` (5 minutes)

4.11 Reference

4.11.1 Bookshop Example Application

Bookshop is a virtual online bookstore application through which you can buy books of various categories and rate the books you have read.

To make your reading on the application developer guide more smoothly, we present the example SQL statements based on the table structures and data of the Bookshop application. This document focuses on the methods of importing the table structures and data as well as the definitions of the table structures.

4.11.1.1 Import table structures and data

You can import Bookshop table structures and data either via TiUP or via the import feature of TiDB Cloud.

4.11.1.1.1 Method 1: Via `tiup demo`

If your TiDB cluster is deployed using TiUP or you can connect to your TiDB server, you can quickly generate and import sample data for the Bookshop application by running the following command:

```
tiup demo bookshop prepare
```

By default, this command enables your application to connect to port `4000` on address `127.0.0.1`, enables you to log in as the `root` user without a password, and creates a table structure in the database named `bookshop`.

Configure connection information

The following table lists the connection parameters. You can change their default settings to match your environment.

Parameter	Abbreviation	Default value	Description
`--password`	`-p`	None	Database user password
`--host`	`-H`	`127.0.0.1`	Database address
`--port`	`-P`	`4000`	Database port
`--db`	`-D`	`bookshop`	Database name
`--user`	`-U`	`root`	Database user

For example, if you want to connect to a database on TiDB Cloud, you can specify the connection information as follows:

```
tiup demo bookshop prepare -U <username> -H <endpoint> -P 4000 -p <password>
```

Set the data volume

You can specify the volume of data to be generated in each database table by configuring the following parameters:

Parameter	Default value	Description
`--users`	`10000`	The number of rows of data to be generated in the `users` table
`--authors`	`20000`	The number of rows to be generated in the `authors` table

Parameter	Default value	Description
`--books`	`20000`	The number of rows of data to be generated in the `books` table
`--orders`	`300000`	The number of rows of data to be generated in the `orders` table
`--ratings`	`300000`	The number of rows of data to be generated in the `ratings` table

For example, the following command is executed to generate:

- 200,000 rows of user information via the `--users` parameter
- 500,000 rows of book information via the `--books` parameter
- 100,000 rows of author information via the `--authors` parameter
- 1,000,000 rows of rating records via the `--ratings` parameter
- 1,000,000 rows of order records via the `--orders` parameter

```
tiup demo bookshop prepare --users=200000 --books=500000 --authors=100000 --
    ↪ ratings=1000000 --orders=1000000 --drop-tables
```

You can delete the original table structure through the `--drop-tables` parameter. For more parameter descriptions, run the `tiup demo bookshop --help` command.

4.11.1.1.2 Method 2: Via TiDB Cloud Import

Perform the following steps to import the Bookshop sample data from AWS S3 to TiDB Cloud.

1. In the TiDB Cloud console, go to the page for data import.

    1. Log in to the TiDB Cloud console and navigate to the **Clusters** page.
    2. Click the name of your target cluster to go to its cluster overview page, and then click **Import** in the left navigation pane.
    3. Click **Import Data** in the upper-right corner, and then select **From S3**.

2. Select **SQL File** for **Data format**, and copy the following **Bucket URI** and **Role ARN** to the corresponding input boxes:

    Bucket URI:

    ```
    s3://developer.pingcap.com/bookshop/
    ```

    Role ARN:

    ```
    ```

    arn:aws:iam::494090988690:role/s3-tidb-cloud-developer-access
    ```
    ```

3. Click **Next** to go to the **Choose the tables to be imported** step to confirm the information of the files to be imported.

4. Click **Next** again to go to the **Preview** step to confirm the preview of the data to be imported.

   In this example, the following data is generated in advance:

   - 200,000 rows of user information
   - 500,000 rows of book information
   - 100,000 rows of author information
   - 1,000,000 rows of rating records
   - 1,000,000 rows of order records

5. Click **Start Import** to start the import process and wait for TiDB Cloud to complete the import.

For more information about how to import or migrate data to TiDB Cloud, see TiDB Cloud Migration Overview.

4.11.1.1.3 View data import status

After the import is completed, you can view the data volume information of each table by executing the following SQL statement:

```
SELECT
    CONCAT(table_schema,'.',table_name) AS 'Table Name',
    table_rows AS 'Number of Rows',
    CONCAT(ROUND(data_length/(1024*1024*1024),4),'G') AS 'Data Size',
    CONCAT(ROUND(index_length/(1024*1024*1024),4),'G') AS 'Index Size',
    CONCAT(ROUND((data_length+index_length)/(1024*1024*1024),4),'G') AS '
        ↪ Total'
FROM
    information_schema.TABLES
WHERE table_schema LIKE 'bookshop';
```

The result is as follows:

```
+----------------------+---------------+-----------+------------+---------+
    ↪
| Table Name         | Number of Rows | Data Size | Index Size | Total |
+----------------------+---------------+-----------+------------+---------+
    ↪
| bookshop.orders    |       1000000 | 0.0373G | 0.0075G  | 0.0447G |
| bookshop.book_authors |    1000000 | 0.0149G | 0.0149G  | 0.0298G |
| bookshop.ratings   |       4000000 | 0.1192G | 0.1192G  | 0.2384G |
| bookshop.authors   |        100000 | 0.0043G | 0.0000G  | 0.0043G |
```

```
| bookshop.users      |         195348 | 0.0048G | 0.0021G  | 0.0069G |
| bookshop.books      |        1000000 | 0.0546G | 0.0000G  | 0.0546G |
+---------------------+----------------+---------+----------+---------+
    ↪
6 rows in set (0.03 sec)
```

4.11.1.2 Description of the tables

This section describes the database tables of the Bookshop application in detail.

4.11.1.2.1 `books` table

This table stores the basic information of books.

Field name	Type	Description
id	bigint(20)	Unique ID of a book
title	varchar(100)	Title of a book
type	enum	Type of a book (for example, magazine, animation, or teaching aids)
stock	bigint(20)	Stock
price	decimal(15,2)	Price
published_at	datetime	Date of publish

4.11.1.2.2 `authors` table

This table stores basic information of authors.

Field name	Type	Description
id	bigint(20)	Unique ID of an author
name	varchar(100)	Name of an author
gender	tinyint(1)	Biological gender (0: female, 1: male, NULL: unknown)
birth_year	smallint(6)	Year of birth
death_year	smallint(6)	Year of death

4.11.1.2.3 `users` table

This table stores information of Bookshop users.

Field name	Type	Description
id	bigint(20)	Unique ID of a user
balance	decimal(15,2)	Balance
nickname	varchar(100)	Nickname

4.11.1.2.4 `ratings` table

This table stores records of user ratings on books.

Field name	Type	Description
book_id	bigint	Unique ID of a book (linked to books)
user_id	bigint	User's unique identifier (linked to users)
score	tinyint	User rating (1-5)
rated_at	datetime	Rating time

4.11.1.2.5 `book_authors` table

An author may write multiple books, and a book may involve more than one author. This table stores the correspondence between books and authors.

Field name	Type	Description
book_id	bigint(20)	Unique ID of a book (linked to books)
author_id	bigint(20)	Unique ID of an author (Link to authors)

4.11.1.2.6 `orders` table

This table stores user purchase information.

Field name	Type	Description
id	bigint(20)	Unique ID of an order
book_id	bigint(20)	Unique ID of a book (linked to books)
user_id	bigint(20)	User unique identifier (associated with users)
quantity	tinyint(4)	Purchase quantity
ordered_at	datetime	Purchase time

4.11.1.3 Database initialization script `dbinit.sql`

If you want to manually create database table structures in the Bookshop application, run the following SQL statements:

```
CREATE DATABASE IF NOT EXISTS `bookshop`;

DROP TABLE IF EXISTS `bookshop`.`books`;
CREATE TABLE `bookshop`.`books` (
  `id` bigint(20) AUTO_RANDOM NOT NULL,
  `title` varchar(100) NOT NULL,
  `type` enum('Magazine', 'Novel', 'Life', 'Arts', 'Comics', 'Education &
    ↪ Reference', 'Humanities & Social Sciences', 'Science & Technology',
    ↪ 'Kids', 'Sports') NOT NULL,
```

```
  `published_at` datetime NOT NULL,
  `stock` int(11) DEFAULT '0',
  `price` decimal(15,2) DEFAULT '0.0',
  PRIMARY KEY (`id`) CLUSTERED
) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;

DROP TABLE IF EXISTS `bookshop`.`authors`;
CREATE TABLE `bookshop`.`authors` (
  `id` bigint(20) AUTO_RANDOM NOT NULL,
  `name` varchar(100) NOT NULL,
  `gender` tinyint(1) DEFAULT NULL,
  `birth_year` smallint(6) DEFAULT NULL,
  `death_year` smallint(6) DEFAULT NULL,
  PRIMARY KEY (`id`) CLUSTERED
) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;

DROP TABLE IF EXISTS `bookshop`.`book_authors`;
CREATE TABLE `bookshop`.`book_authors` (
  `book_id` bigint(20) NOT NULL,
  `author_id` bigint(20) NOT NULL,
  PRIMARY KEY (`book_id`,`author_id`) CLUSTERED
) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;

DROP TABLE IF EXISTS `bookshop`.`ratings`;
CREATE TABLE `bookshop`.`ratings` (
  `book_id` bigint NOT NULL,
  `user_id` bigint NOT NULL,
  `score` tinyint NOT NULL,
  `rated_at` datetime NOT NULL DEFAULT NOW() ON UPDATE NOW(),
  PRIMARY KEY (`book_id`,`user_id`) CLUSTERED,
  UNIQUE KEY `uniq_book_user_idx` (`book_id`,`user_id`)
) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
ALTER TABLE `bookshop`.`ratings` SET TIFLASH REPLICA 1;

DROP TABLE IF EXISTS `bookshop`.`users`;
CREATE TABLE `bookshop`.`users` (
  `id` bigint AUTO_RANDOM NOT NULL,
  `balance` decimal(15,2) DEFAULT '0.0',
  `nickname` varchar(100) UNIQUE NOT NULL,
  PRIMARY KEY (`id`)
) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;

DROP TABLE IF EXISTS `bookshop`.`orders`;
CREATE TABLE `bookshop`.`orders` (
  `id` bigint(20) AUTO_RANDOM NOT NULL,
```

```
  `book_id` bigint(20) NOT NULL,
  `user_id` bigint(20) NOT NULL,
  `quality` tinyint(4) NOT NULL,
  `ordered_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
    ↪ CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`) CLUSTERED,
  KEY `orders_book_id_idx` (`book_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

4.11.2 Guidelines

4.11.2.1 Object Naming Convention

This document introduces the rules to name database objects, such as database, table, index, and user.

4.11.2.1.1 General rules

- It is recommended to use meaningful English words separated by underscores.
- Use only letters, numbers, and underscores in a name.
- Avoid using TiDB reserved words, such as `group` and `order`, as column names.
- It is recommended to use lowercase letters for all database objects.

4.11.2.1.2 Database naming convention

It is recommended to differentiate database names by business, product, or other metrics and use no more than 20 characters in a database name. For example, you can name a temporary library as `tmp_crm` or a test library as `test_crm`.

4.11.2.1.3 Table naming convention

- Use the same prefix for tables of the same business or module, and make sure that the table name is self-explanatory as much as possible.
- Separate words in a name by underscores. It is recommended to use no more than 32 characters in a table name.
- It is recommended to annotate the purpose of the table for a better understanding. For example:
  - Temporary table: `tmp_t_crm_relation_0425`
  - Backup table: `bak_t_crm_relation_20170425`
  - Temporary table of business operations: `tmp_st_{business code}_{creator abbreviation}_{date}`
  - Record table of accounts period: `t_crm_ec_record_YYYY{MM}{dd}`
- Create separate databases for tables of different business modules and add annotations accordingly.

4.11.2.1.4 Column naming convention

- The column naming is the actual meaning or abbreviation of the column.
- It is recommended to use the same column name between tables with the same meaning.
- It is recommended to add annotations to columns and specify named values for enumerated types, such as "0: offline, 1: online".
- It is recommended to name the boolean column as `is_{description}`. For example, the column of a `member` table that indicates whether the member is enabled, can be named as `is_enabled`.
- It is not recommended to name a column with more than 30 characters, and the number of columns should be less than 60.
- Avoid using TiDB reserved words as column names, such as `order`, `from`, and `desc`. To check whether a keyword is reserved, see TiDB keywords.

4.11.2.1.5 Index naming convention

- Primary key index: `pk_{table_name_abbreviation}_{field_name_abbreviation}`
- Unique index: `uk_{table_name_abbreviation}_{field_name_abbreviation}`
- Common index: `idx_{table_name_abbreviation}_{field_name_abbreviation}`
- Column name with multiple words: use meaningful abbreviations

4.11.2.2 SQL Development Specifications

This document introduces some general development specifications for using SQL.

4.11.2.2.1 Create and delete tables

- Basic principle: under the premise of following the table naming convention, it is recommended that the application internally packages the table creation and deletion statements and adds judgment logic to prevent abnormal interruption of business processes.
- Details: `create table if not exists table_name` or `drop table if exists table_name` statements are recommended to add `if` judgments to avoid abnormal interruptions caused by SQL commands running abnormally on the application side.

4.11.2.2.2 `SELECT *` usage

- Basic principle: avoid using `SELECT *` for queries.
- Details: select the appropriate columns as required and avoid using `SELECT *` to read all fields because such operations consume network bandwidth. Consider adding the queried fields to the index to make effective use of the covering index.

4.11.2.2.3 Use functions on fields

- Basic principle: You can use related functions on the queried fields. To avoid index failure, do not use any functions on the filtered fields in the `WHERE` clause, including data type conversion functions. You may consider using the expression index.
- Detailed description:

  NOT recommended:

```
SELECT gmt_create
FROM ...
WHERE DATE_FORMAT(gmt_create, '%Y%m%d %H:%i:%s') = '20090101 00:00:00'
```

  Recommended:

```
SELECT DATE_FORMAT(gmt_create, '%Y%m%d %H:%i:%s')
FROM ...
WHERE gmt_create = str_to_date('20090101 00:00:00', '%Y%m%d %H:%i:%s')
```

4.11.2.2.4 Other specifications

- Do not perform mathematical operations or functions on the index column in the `WHERE` condition.
- Replace `OR` with `IN` or `UNION`. The number of `IN` must be less than `300`.
- Avoid using the `%` prefix for fuzzy prefix queries.
- If the application uses **Multi Statements** to execute SQL, that is, multiple SQLs are joined with semicolons and sent to the client for execution at once, TiDB only returns the result of the first SQL execution.
- When you use expressions, check if the expressions support computing push-down to the storage layer (TiKV or TiFlash). If not, you should expect more memory consumption and even OOM at the TiDB layer. Computing that can be pushe down the storage layer is as follows:
  - TiFlash supported push-down calculations.
  - TiKV - List of Expressions for Pushdown.
  - Predicate push down.

4.11.3 Legacy Docs

4.11.3.1 App Development for Django

> **Note:**
>
> This legacy document is outdated and will not be updated thereafter. You can see Developer Guide Overview for more details.

This tutorial shows you how to build a simple Python application based on TiDB and Django. The sample application to build here is a simple CRM tool where you can add, query, and update customer and order information.

4.11.3.1.1 Step 1. Start a TiDB cluster

Start a pseudo TiDB cluster on your local storage:

```
docker run -p 127.0.0.1:$LOCAL_PORT:4000 pingcap/tidb:v5.1.0
```

The above command starts a temporary and single-node cluster with mock TiKV. The cluster listens on the port `$LOCAL_PORT`. After the cluster is stopped, any changes already made to the database are not persisted.

> **Note:**
>
> To deploy a "real" TiDB cluster for production, see the following guides:
>
> - Deploy TiDB using TiUP for Self-Hosted Environment
> - Deploy TiDB on Kubernetes
>
> You can also use TiDB Cloud, a fully-managed Database-as-a-Service (DBaaS) of TiDB.

4.11.3.1.2 Step 2. Create a database

1. In the SQL shell, create the `django` database that your application will use:

   ```
   CREATE DATABASE django;
   ```

2. Create a SQL user for your application:

   ```
   CREATE USER <username> IDENTIFIED BY <password>;
   ```

   Take note of the username and password. You will use them in your application code when initializing the project.

3. Grant necessary permissions to the SQL user you have just created:

   ```
   GRANT ALL ON django.* TO <username>;
   ```

4.11.3.1.3 Step 3. Set virtual environments and initialize the project

1. Use Poetry, a dependency and package manager in Python, to set virtual environments and initialize the project.

   Poetry can isolate system dependencies from other dependencies and avoid dependency pollution. Use the following command to install Poetry.

```
pip install --user poetry
```

2. Initialize the development environment using Poetry:

```
poetry init --no-interaction --dependency django
poetry run django-admin startproject tidb_example

mv pyproject.toml ./tidb_example
cd tidb_example

poetry add django-tidb

poetry shell
```

3. Modify the configuration file. The configuration in `tidb_example/settings.py` is as follows.

```
USE_TZ = True

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

   Modify the configuration above as follows. This is used for connection to TiDB.

```
USE_TZ = False

DATABASES = {
    'default': {
        'ENGINE': 'django_tidb',
        'NAME': 'django',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
```

```
        'PORT': 4000,
    },
}

DEFAULT_AUTO_FIELD = 'django.db.models.AutoField'
```

4.11.3.1.4 Step 4. Write the application logic

After you have configured the application's database connection, you can start building out the application. To write the application logic, you need to build the models, build the controller, and define the URL routes.

1. Build models that are defined in a file called `models.py`. You can copy the sample code below and paste it into a new file.

```
from django.db import models

class Orders(models.Model):
    id = models.AutoField(primary_key=True)
    username = models.CharField(max_length=250)
    price = models.FloatField()
```

2. Build class-based views in a file called `views.py`. You can copy the sample code below and paste it into a new file.

```
from django.http import JsonResponse, HttpResponse
from django.utils.decorators import method_decorator
from django.views.generic import View
from django.views.decorators.csrf import csrf_exempt
from django.db import Error, OperationalError
from django.db.transaction import atomic
from functools import wraps
import json
import sys
import time

from .models import *

def retry_on_exception(view, num_retries=3, on_failure=HttpResponse(
    ↪ status=500), delay_=0.5, backoff_=1.5):
    @wraps(view)
    def retry(*args, **kwargs):
        delay = delay_
        for i in range(num_retries):
            try:
```

```
                return view(*args, **kwargs)
            except Exception as e:
                return on_failure
    return retry


class PingView(View):
    def get(self, request, *args, **kwargs):
        return HttpResponse("python/django", status=200)


@method_decorator(csrf_exempt, name='dispatch')
class OrderView(View):
    def get(self, request, id=None, *args, **kwargs):
        if id is None:
            orders = list(Orders.objects.values())
        else:
            orders = list(Orders.objects.filter(id=id).values())
        return JsonResponse(orders, safe=False)


    @retry_on_exception
    @atomic
    def post(self, request, *args, **kwargs):
        form_data = json.loads(request.body.decode())
        username = form_data['username']
        price = form_data['price']
        c = Orders(username=username, price=price)
        c.save()
        return HttpResponse(status=200)

    @retry_on_exception
    @atomic
    def delete(self, request, id=None, *args, **kwargs):
        if id is None:
            return HttpResponse(status=404)
        Orders.objects.filter(id=id).delete()
        return HttpResponse(status=200)
```

3. Define URL routes in a file called `urls.py`. The `django-admin` command-line tool has generated this file when you create the Django project, so the file should already exist in `tidb_example/tidb_example`. You can copy the sample code below and paste it into the existing `urls.py` file.

```
from django.contrib import admin
```

```
from django.urls import path
from django.conf.urls import url

from .views import OrderView, PingView

urlpatterns = [
    path('admin/', admin.site.urls),

    url('ping/', PingView.as_view()),

    url('order/', OrderView.as_view(), name='order'),
    url('order/<int:id>/', OrderView.as_view(), name='order'),
]
```

4.11.3.1.5 Step 5. Set up and run the Django application

In the top `tidb_example` directory, use the `manage.py` script to create Django migrations that initialize the database for the application:

```
python manage.py makemigrations tidb_example
python manage.py migrate tidb_example
python manage.py migrate
```

Then start the application:

```
python3 manage.py runserver 0.0.0.0:8000
```

To test the application by inserting some example data, run the following commands:

```
curl --request POST '127.0.0.1:8000/order/' \
--data-raw '{
    "uid": 1,
    "price": 3.12
}'

curl --request PATCH '127.0.0.1:8000/order/' --data-raw '{ "oid": 1, "price"
    ↪ : 312 }'

curl --request GET '127.0.0.1:8000/order/' --data-raw '{ "oid": 1 }'
```

To verify whether the data insertion is successful, open the terminal with the SQL shell to check:

```
MySQL root@127.0.0.1:(none)> select * from django.tidb_example_orders;
+-----+-----+-------+
| oid | uid | price |
```

```
+-----+-----+-------+
| 1   | 1   | 312.0 |
+-----+-----+-------+
1 row in set
Time: 0.008s
```

The result above shows that the data insertion is successful. Then you can delete the inserted data:

```
curl --request DELETE '127.0.0.1:8000/order/' --data-raw '{ "oid": 1 }'
```

4.12 Cloud Native Development Environment

4.12.1 Gitpod

With Gitpod, you can get a full development environment in your browser with the click of a button or link, and you can write code right away.

Gitpod is an open-source Kubernetes application (GitHub repository address: https://github.com/gitpod-io/gitpod) for direct-to-code development environments, which spins up fresh, automated development environments for each task, in the cloud, in seconds. It enables you to describe your development environment as code and start instant, remote and cloud-based development environments directly from your browser or your Desktop IDE.

4.12.1.1 Quick start

1. Fork the example code repository pingcap-inc/tidb-example-java for TiDB application development.
2. Start your Gitpod workspace by prefixing the URL of the sample code repository with `https://gitpod.io/#` in the address bar of your browser.

- For example, `https://gitpod.io/#https://github.com/pingcap-inc/tidb-example-java`.
- You can configure environment variables in the URL. For example, `https://gitpod.io/#targetFile=spring-jpa-hibernate_Makefile,targetMode=spring-jpa-hibernate/https://github.com/pingcap-inc/tidb-example-java`.

3. Log in and start the workspace using one of the providers listed. For example, `Github`.

4.12.1.2 Use the default Gitpod configuration and environment

After completing the quick-start steps, it will take a while for Gitpod to set up your workspace.

Take the Spring Boot Web application as an example. You can create a new workspace by the `https://gitpod.io/#targetFile=spring-jpa-hibernate_Makefile,targetMode=spring-jpa-hibernate/https://github.com/pingcap-inc/tidb-example-java` URL.

After that, you will see a page similar to the following:

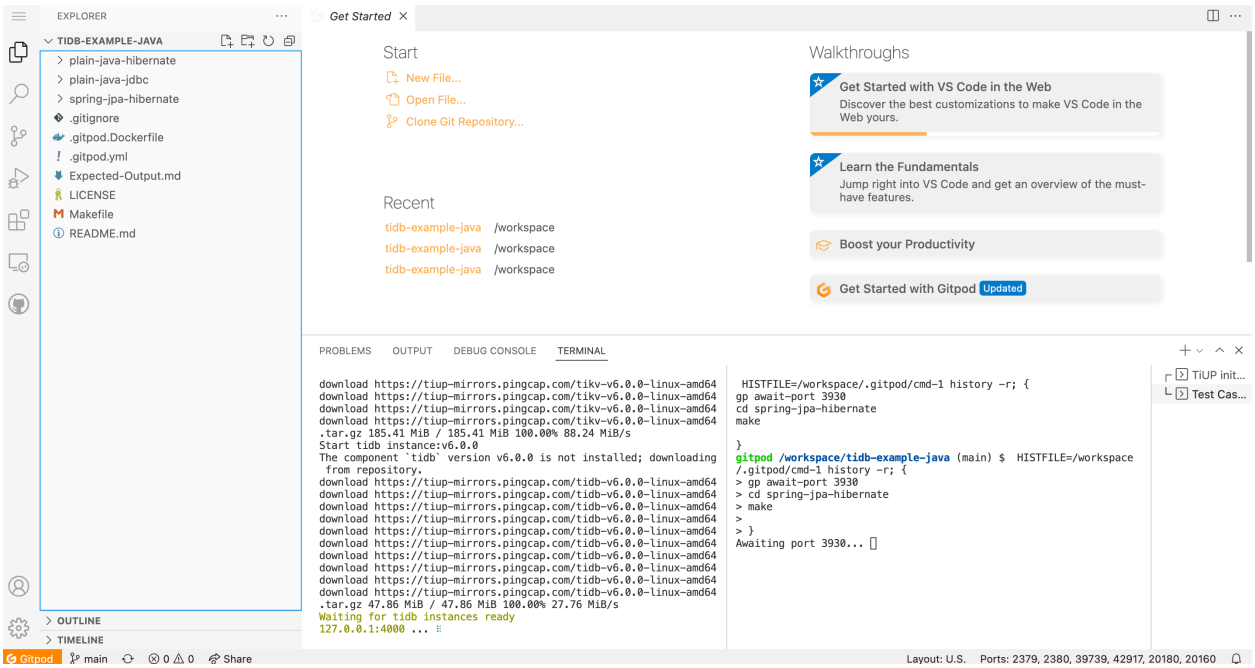


Figure 20: playground gitpod workspace init

This scenario in the page uses TiUP to build a TiDB Playground. You can check the progress on the left side of the terminal area.

Once the TiDB Playground is ready, another `Spring JPA Hibernate` task will run. You can check the progress on the right side of the terminal area.

After all these tasks are finished, you will see a page similar to the following. On this page, check the `REMOTE EXPLORER` area in the left navigation pane (Gitpod supports URL-based port forwarding) and find the URL of your port `8080`.

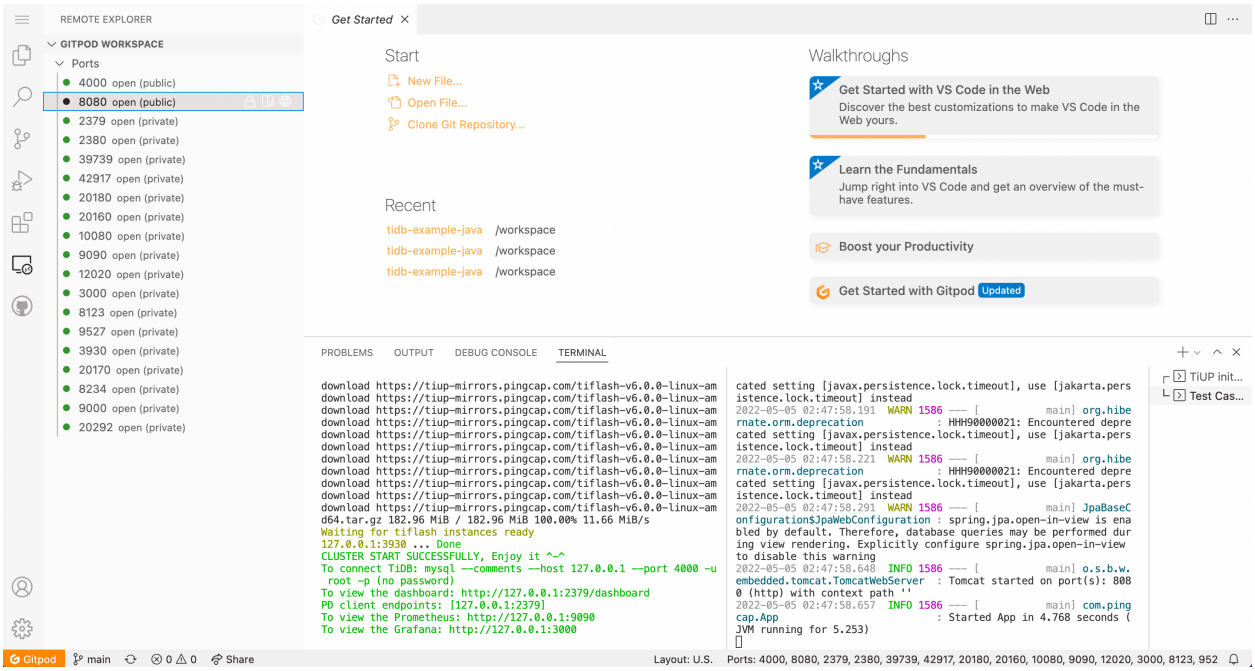


Figure 21: playground gitpod workspace ready

You can test the API by sending an HTTP request. Make sure to replace the `http://localhost:8080` URL with the one you found in the `REMOTE EXPLORER` area.

4.12.1.3 Using custom Gitpod configuration and Docker image

4.12.1.3.1 Customize Gitpod configurations

Referring to example.gitpod.yml, create a `.gitpod. yml` file in the root directory of your project to configure the Gitpod workspace.

```
### This configuration file was automatically generated by Gitpod.
### Please adjust to your needs (see https://www.gitpod.io/docs/config-gitpod-file)
### and commit this file to your remote git repository to share the goodness with others.

### image:
###   file: .gitpod.Dockerfile

tasks:
  - name: Open Target File
    command: |
      if [ -n "$targetFile" ]; then code ${targetFile//[_]//}; fi
  - name: TiUP init playground
```

```
    command: |
      $HOME/.tiup/bin/tiup playground
  - name: Test Case
    openMode: split-right
    init: echo "*** Waiting for TiUP Playground Ready! ***"
    command: |
      gp await-port 3930
      if [ "$targetMode" == "plain-java-jdbc" ]
      then
        cd plain-java-jdbc
        code src/main/resources/dbinit.sql
        code src/main/java/com/pingcap/JDBCExample.java
        make mysql
      elif [ "$targetMode" == "plain-java-hibernate" ]
      then
        cd plain-java-hibernate
        make
      elif [ "$targetMode" == "spring-jpa-hibernate" ]
      then
        cd spring-jpa-hibernate
        make
      fi
ports:
  - port: 8080
    visibility: public
  - port: 4000
    visibility: public
  - port: 2379-36663
    onOpen: ignore
```

4.12.1.3.2 Customize Gitpod Docker images

By default, Gitpod uses a standard Docker image named Workspace-Full as the basis for the workspace. Workspaces launched from this default image are pre-installed with Docker, Go, Java, Node.js, C/C++, Python, Ruby, Rust, PHP, and tools such as Homebrew, Tailscale, and Nginx.

You can use a public Docker image or a Dockerfile and also install any required dependencies for your project.

For example, you can use a Dockerfile (see also Example `.gitpod.Dockerfile`) as follows:

```
FROM gitpod/workspace-java-17

RUN sudo apt install mysql-client -y
```

```
RUN curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

Then, you need to update `.gitpod.yml`:

```
### This configuration file was automatically generated by Gitpod.
### Please adjust to your needs (see https://www.gitpod.io/docs/config-
    ↪ gitpod-file)
### and commit this file to your remote git repository to share the goodness
    ↪  with others.

image:
  # Import your Dockerfile here.
  file: .gitpod.Dockerfile

tasks:
  - name: Open Target File
    command: |
      if [ -n "$targetFile" ]; then code ${targetFile//[_]//}; fi
  - name: TiUP init playground
    command: |
      $HOME/.tiup/bin/tiup playground
  - name: Test Case
    openMode: split-right
    init: echo "*** Waiting for TiUP Playground Ready! ***"
    command: |
      gp await-port 3930
      if [ "$targetMode" == "plain-java-jdbc" ]
      then
        cd plain-java-jdbc
        code src/main/resources/dbinit.sql
        code src/main/java/com/pingcap/JDBCExample.java
        make mysql
      elif [ "$targetMode" == "plain-java-hibernate" ]
      then
        cd plain-java-hibernate
        make
      elif [ "$targetMode" == "spring-jpa-hibernate" ]
      then
        cd spring-jpa-hibernate
        make
      fi
ports:
  - port: 8080
    visibility: public
```

```
- port: 4000
  visibility: public
- port: 2379-36663
  onOpen: ignore
```

4.12.1.3.3 Apply changes

After completing the configuration of the `.gitpod.yml` file, make sure that the latest code is available in your corresponding GitHub repository.

Visit `https://gitpod.io/#<YOUR_REPO_URL>` to create a new Gitpod workspace with the latest code applied.

Visit `https://gitpod.io/workspaces` for all established workspaces.

4.12.1.4 Summary

Gitpod provides a complete, automated, and pre-configured cloud-native development environment. You can develop, run, and test code directly in the browser without any local configurations.

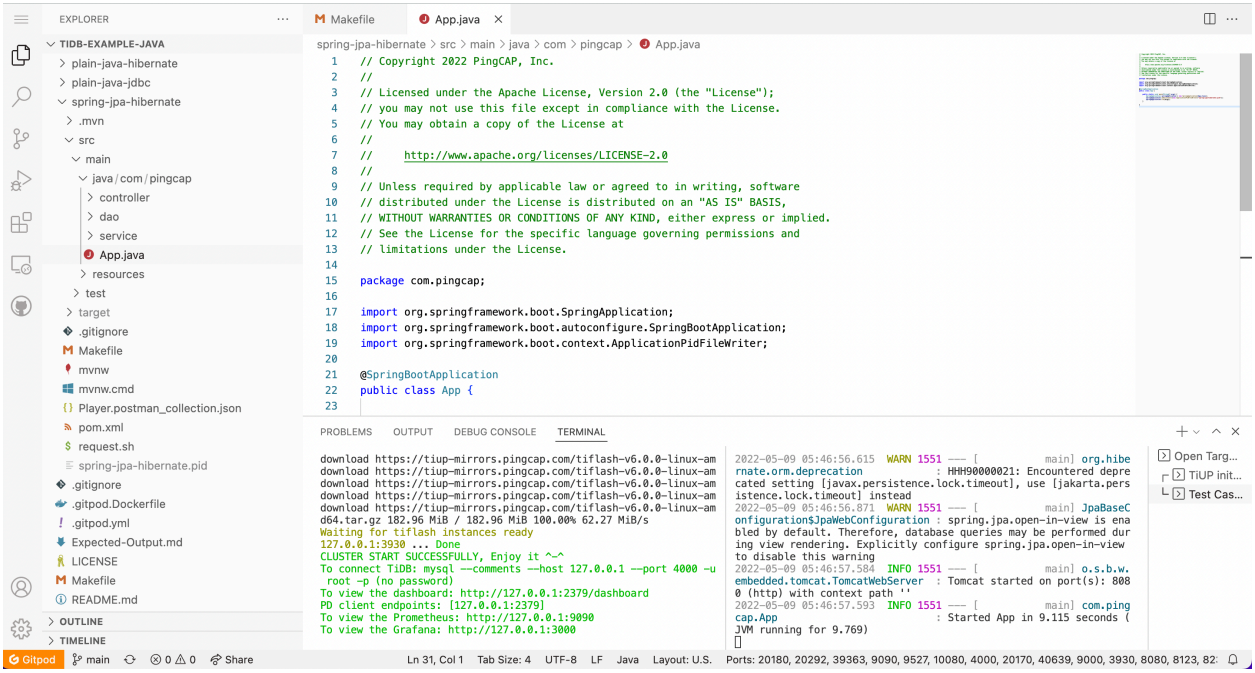


Figure 22: playground gitpod summary

4.13 Third-Party Support

4.13.1 Third-Party Tools Supported by TiDB

> **Note:**
>
> This document only lists common third-party tools supported by TiDB. Some other third-party tools are not listed, not because they are not supported, but because PingCAP is not sure whether they use features that are incompatible with TiDB.

TiDB is highly compatible with the MySQL protocol, so most of the MySQL drivers, ORM frameworks, and other tools that adapt to MySQL are compatible with TiDB. This document focuses on these tools and their support levels for TiDB.

4.13.1.1 Support Level

PingCAP works with the community and provides the following support levels for third-party tools:

- ***Full***: Indicates that TiDB is already compatible with most functionalities of the corresponding third-party tool, and maintains compatibility with its newer versions. PingCAP will periodically conduct compatibility tests with the latest version of the tool.
- ***Compatible***: Indicates that because the corresponding third-party tool is adapted to MySQL and TiDB is highly compatible with the MySQL protocol, so TiDB can use most features of the tool. However, PingCAP has not completed a full test on all features of the tool, which might lead to some unexpected behaviors.

> **Note:**
>
> Unless specified, support for Application retry and error handling is not included for **Driver** or **ORM frameworks**.

If you encounter problems when connecting to TiDB using the tools listed in this document, please submit an issue on GitHub with details to promote support on this tool.

4.13.1.2 Driver

```
<tr>
   <th>Language</th>
   <th>Driver</th>
   <th>Latest tested version</th>
   <th>Support level</th>
   <th>TiDB adapter</th>
   <th>Tutorial</th>
</tr>
```

```
<tr>
   <td>Go</td>
   <td><a href="https://github.com/go-sql-driver/mysql" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">Go-MySQL-Driver</a></
      ↪ td>
   <td>v1.6.0</td>
   <td>Full</td>
   <td>N/A</td>
   <td><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-sample-
      ↪ application-golang">Build a Simple CRUD App with TiDB and Golang
      ↪ </a></td>
</tr>
<tr>
   <td>Java</td>
   <td><a href="https://dev.mysql.com/downloads/connector/j/" target="
      ↪ _blank" referrerpolicy="no-referrer-when-downgrade">JDBC</a></td>
   <td>8.0</td>
   <td>Full</td>
   <td>
      <ul>
         <li><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-choose-
            ↪ driver-or-orm#java-drivers" data-href="https://docs.pingcap.
            ↪ com/tidb/v6.1/dev-guide-choose-driver-or-orm#java-drivers">
            ↪ pingcap/mysql-connector-j</a></li>
         <li><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-choose-
            ↪ driver-or-orm#tidb-loadbalance" data-href="https://docs.
            ↪ pingcap.com/tidb/v6.1/dev-guide-choose-driver-or-orm#tidb-
            ↪ loadbalance">pingcap/tidb-loadbalance</a></li>
      </ul>
   </td>
   <td><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-sample-
      ↪ application-java">Build a Simple CRUD App with TiDB and Java</a
      ↪ ></td>
</tr>
```

4.13.1.3 ORM

```
<tr>
   <th>Language</th>
   <th>ORM framework</th>
   <th>Latest tested version</th>
   <th>Support level</th>
   <th>TiDB adapter</th>
   <th>Tutorial</th>
</tr>
```

```
<tr>
   <td rowspan="4">Go</td>
   <td><a href="https://github.com/go-gorm/gorm" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">gorm</a></td>
   <td>v1.23.5</td>
   <td>Full</td>
   <td>N/A</td>
   <td><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-sample-
      ↪ application-golang">Build a Simple CRUD App with TiDB and Golang
      ↪ </a></td>
</tr>
<tr>
   <td><a href="https://github.com/beego/beego" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">beego</a></td>
   <td>v2.0.3</td>
   <td>Full</td>
   <td>N/A</td>
   <td>N/A</td>
</tr>
<tr>
   <td><a href="https://github.com/upper/db" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">upper/db</a></td>
   <td>v4.5.2</td>
   <td>Full</td>
   <td>N/A</td>
   <td>N/A</td>
</tr>
<tr>
   <td><a href="https://gitea.com/xorm/xorm" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">xorm</a></td>
   <td>v1.3.1</td>
   <td>Full</td>
   <td>N/A</td>
   <td>N/A</td>
```

```
</tr>
<tr>
   <td rowspan="4">Java</td>
   <td><a href="https://hibernate.org/orm/" target="_blank" referrerpolicy
      ↪ ="no-referrer-when-downgrade">Hibernate</a></td>
   <td>6.1.0.Final</td>
   <td>Full</td>
   <td>N/A</td>
   <td><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-sample-
      ↪ application-java">Build a Simple CRUD App with TiDB and Java</a
      ↪ ></td>
</tr>
<tr>
   <td><a href="https://mybatis.org/mybatis-3/" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">MyBatis</a></td>
   <td>v3.5.10</td>
   <td>Full</td>
   <td>N/A</td>
   <td><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-sample-
      ↪ application-java">Build a Simple CRUD App with TiDB and Java</a
      ↪ ></td>
</tr>
<tr>
   <td><a href="https://spring.io/projects/spring-data-jpa/" target="
      ↪ _blank" referrerpolicy="no-referrer-when-downgrade">Spring Data
      ↪ JPA</a></td>
   <td>2.7.2</td>
   <td>Full</td>
   <td>N/A</td>
   <td><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-sample-
      ↪ application-spring-boot">Build a Simple CRUD App with TiDB and
      ↪ Spring Boot</a></td>
</tr>
<tr>
   <td><a href="https://github.com/jOOQ/jOOQ" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">jOOQ</a></td>
   <td>v3.16.7 (Open Source)</td>
   <td>Full</td>
   <td>N/A</td>
   <td>N/A</td>
</tr>
<tr>
   <td>Ruby</td>
   <td><a href="https://guides.rubyonrails.org/active_record_basics.html"
      ↪ target="_blank" referrerpolicy="no-referrer-when-downgrade">
```

```
      ↪ Active Record</a></td>
   <td>v7.0</td>
   <td>Full</td>
   <td>N/A</td>
   <td>N/A</td>
</tr>
<tr>
   <td rowspan="2">JavaScript / TypeScript</td>
   <td><a href="https://www.npmjs.com/package/sequelize" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">sequelize</a></td>
   <td>v6.20.1</td>
   <td>Full</td>
   <td>N/A</td>
   <td>N/A</td>
</tr>
<tr>
   <td><a href="https://www.prisma.io/" target="_blank" referrerpolicy="no
      ↪ -referrer-when-downgrade">Prisma Client</a></td>
   <td>4.16.2</td>
   <td>Full</td>
   <td>N/A</td>
   <td>N/A</td>
</tr>
<tr>
   <td rowspan="2">Python</td>
   <td><a href="https://pypi.org/project/Django/" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">Django</a></td>
   <td>v4.1</td>
   <td>Full</td>
   <td><a href="https://github.com/pingcap/django-tidb" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">django-tidb</a></td>
   <td>N/A</td>
</tr>
<tr>
   <td><a href="https://www.sqlalchemy.org/" target="_blank"
      ↪ referrerpolicy="no-referrer-when-downgrade">SQLAlchemy</a></td>
   <td>v1.4.37</td>
   <td>Full</td>
   <td>N/A</td>
   <td><a href="https://docs.pingcap.com/tidb/v6.1/dev-guide-sample-
      ↪ application-python">Build a Simple CRUD App with TiDB and Python
      ↪ </a></td>
</tr>
```

4.13.1.4 GUI

GUI	Latest tested version	Support level	Tutorial
DBeaver	23.0.1	Full	N/A

4.13.2 Known Incompatibility Issues with Third-Party Tools

> **Note:**
>
> The Unsupported features section lists the unsupported features in TiDB, including:
>
> - Stored procedures and functions
> - Triggers
> - Events
> - User-defined functions
> - `FOREIGN KEY` constraints
> - `SPATIAL` functions, data types and indexes
> - `XA` syntax
>
> The preceding unsupported features are expected behavior and are not listed in this document. For more details, see MySQL Compatibility.

The incompatibility issues listed in this document are found in some third-party tools supported by TiDB.

4.13.2.1 General incompatibility

4.13.2.1.1 `SELECT CONNECTION_ID()` returns a 64-bit integer in TiDB

Description

The `SELECT CONNECTION_ID()` function returns a 64-bit integer in TiDB, such as `2199023260887`, while it returns a 32-bit integer in MySQL, such as `391650`.

Way to avoid

In a TiDB application, to avoid data overflow, you should use a 64-bit integer or string type to store the result of `SELECT CONNECTION_ID()`. For example, you can use `Long` or `String` in Java and use `string` in JavaScript or TypeScript.

4.13.2.1.2 TiDB does not maintain `Com_*` counters

Description

MySQL maintains a series of server status variables starting with `Com_` to keep track of the total number of operations you have performed on the database. For example, `Com_select` records the total number of `SELECT` statements initiated by MySQL since it was last started (even if the statements were not queried successfully). TiDB does not maintain these variables. You can use the statement SHOW GLOBAL STATUS LIKE 'Com_%' to see the difference between TiDB and MySQL.

Way to avoid

Do not use these variables. One common scenario is monitoring. TiDB is well observable and does not require querying from server status variables. For custom monitoring tools, refer to TiDB Monitoring Framework Overview.

4.13.2.1.3 TiDB distinguishes between `TIMESTAMP` and `DATETIME` in error messages

Description

TiDB error messages distinguish between `TIMESTAMP` and `DATETIME`, while MySQL does not, and returns them all as `DATETIME`. That is, MySQL incorrectly converts `TIMESTAMP` type error messages to `DATETIME` type.

Way to avoid

Do not use the error messages for string matching. Instead, use Error Codes for troubleshooting.

4.13.2.1.4 TiDB does not support the `CHECK TABLE` statement

Description

The `CHECK TABLE` statement is not supported in TiDB.

Way to avoid

To check the consistency of data and corresponding indexes, you can use the `ADMIN CHECK [TABLE|INDEX]` statement in TiDB.

4.13.2.2 Compatibility with MySQL JDBC

The test version is MySQL Connector/J 8.0.29.

4.13.2.2.1 The default collation is inconsistent

Description

The collations of MySQL Connector/J are stored on the client side and distinguished by the server version.

The following table lists known client-side and server-side collation inconsistencies in character sets:

Character	Client-side default collation	Server-side default collation
ascii	ascii_general_ci	ascii_bin
latin1	latin1_swedish_ci	latin1_bin
utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_bin

Way to avoid

Set the collation manually, and do not rely on the client-side default collation. The client-side default collation is stored by the MySQL Connector/J configuration file.

4.13.2.2.2 The NO_BACKSLASH_ESCAPES parameter does not take effect

Description

In TiDB, you cannot use the NO_BACKSLASH_ESCAPES parameter without escaping the \ character. For more details, track this issue.

Way to avoid

Do not use NO_BACKSLASH_ESCAPES with \ in TiDB, but use \\ in SQL statements.

4.13.2.2.3 The INDEX_USED related parameters are not supported

Description

TiDB does not set the SERVER_QUERY_NO_GOOD_INDEX_USED and SERVER_QUERY_NO_INDEX_USED parameters in the protocol. This will cause the following parameters to be returned inconsistently with the actual situation:

- com.mysql.cj.protocol.ServerSession.noIndexUsed()
- com.mysql.cj.protocol.ServerSession.noGoodIndexUsed()

Way to avoid

Do not use the noIndexUsed() and noGoodIndexUsed() functions in TiDB.

4.13.2.2.4 The enablePacketDebug parameter is not supported

Description

TiDB does not support the enablePacketDebug parameter. It is a MySQL Connector/J parameter used for debugging that will keep the buffer of the data packet. This might cause the connection to close unexpectedly. **DO NOT** turn it on.

Way to avoid

Do not set the enablePacketDebug parameter in TiDB.

4.13.2.2.5 The UpdatableResultSet is not supported

Description

TiDB does not support `UpdatableResultSet`. **DO NOT** specify the `ResultSet.CONCUR_UPDATABLE` parameter and **DO NOT** update data inside the `ResultSet`.

Way to avoid

To ensure data consistency by transaction, you can use `UPDATE` statements to update data.

4.13.2.3 MySQL JDBC bugs

4.13.2.3.1 `useLocalTransactionState` and `rewriteBatchedStatements` are true at the same time will cause the transaction to fail to commit or roll back

Description

When using MySQL Connector/J 8.0.32 or an earlier version, if the `useLocalTransactionState` and `rewriteBatchedStatements` parameters are set to `true` at the same time, the transaction might fail to commit. You can reproduce with this code.

Way to avoid

> **Note:**
>
> `useConfigs=maxPerformance` includes a group of configurations. For detailed configurations in MySQL Connector/J 8.0 and MySQL Connector/J 5.1, see mysql-connector-j 8.0 and mysql-connector-j 5.1 respectively. You need to disable `useLocalTransactionState` when using `maxPerformance`. That is, use `useConfigs=maxPerformance&useLocalTransactionState=false`.

This bug has been fixed in MySQL Connector/J 8.0.33. Considering updates for the 8.0.x series have ceased, it is strongly recommended to upgrade your MySQL Connector/J to the latest General Availability (GA) version for improved stability and performance.

4.13.2.3.2 Connector is incompatible with the server version earlier than 5.7.5

Description

The database connection might hang under certain conditions when using MySQL Connector/J 8.0.31 or an earlier version with a MySQL server $< 5.7.5$ or a database using the

MySQL server < 5.7.5 protocol (such as TiDB earlier than v6.3.0). For more details, see the Bug Report.

Way to avoid

This bug has been fixed in MySQL Connector/J 8.0.32. Considering updates for the 8.0.x series have ceased, it is strongly recommended to upgrade your MySQL Connector/J to the latest General Availability (GA) version for improved stability and performance.

TiDB also fixes it in the following ways:

- Client side: This bug has been fixed in **pingcap/mysql-connector-j** and you can use the pingcap/mysql-connector-j instead of the official MySQL Connector/J.
- Server side: This compatibility issue has been fixed since TiDB v6.3.0 and you can upgrade the server to v6.3.0 or later versions.

4.13.2.4 Compatibility with Sequelize

The compatibility information described in this section is based on Sequelize v6.32.1.

According to the test results, TiDB supports most of the Sequelize features (using `MySQL` as the dialect).

Unsupported features are:

- Foreign key constraints (including many-to-many relationships) are not supported.
- `GEOMETRY` is not supported.
- Modification of integer primary key is not supported.
- `PROCEDURE` is not supported.
- The `READ-UNCOMMITTED` and `SERIALIZABLE` isolation levels are not supported.
- Modification of a column's `AUTO_INCREMENT` attribute is not allowed by default.
- `FULLTEXT`, `HASH`, and `SPATIAL` indexes are not supported.
- `sequelize.queryInterface.showIndex(Model.tableName);` is not supported.
- `sequelize.options.databaseVersion` is not supported.
- Adding a foreign key reference using `queryInterface.addColumn` is not supported.

4.13.2.4.1 Modification of integer primary key is not supported

Description

Modification of integer primary key is not supported. TiDB uses primary key as an index for data organization if the primary key is integer type. Refer to Issue #18090 and Clustered Indexes for more details.

4.13.2.4.2 The `READ-UNCOMMITTED` and `SERIALIZABLE` isolation levels are not supported

Description

TiDB does not support the `READ-UNCOMMITTED` and `SERIALIZABLE` isolation levels. If the isolation level is set to `READ-UNCOMMITTED` or `SERIALIZABLE`, TiDB throws an error.

Way to avoid

Use only the isolation level that TiDB supports: `REPEATABLE-READ` or `READ-COMMITTED`.

If you want TiDB to be compatible with other applications that set the `SERIALIZABLE` isolation level but not depend on `SERIALIZABLE`, you can set `tidb_skip_isolation_level_check` to `1`. In such case, TiDB ignores the unsupported isolation level error.

4.13.2.4.3 Modification of a column's `AUTO_INCREMENT` attribute is not allowed by default

Description

Adding or removing the `AUTO_INCREMENT` attribute of a column via `ALTER TABLE MODIFY` or `ALTER TABLE CHANGE` command is not allowed by default.

Way to avoid

Refer to the restrictions of `AUTO_INCREMENT`.

To allow the removal of the `AUTO_INCREMENT` attribute, set `@@tidb_allow_remove_auto_inc` to `true`.

4.13.2.4.4 `FULLTEXT`, `HASH`, and `SPATIAL` indexes are not supported

Description

`FULLTEXT`, `HASH`, and `SPATIAL` indexes are not supported.

4.13.3 Integrate TiDB with ProxySQL

This document provides a high-level introduction to ProxySQL, describes how to integrate ProxySQL with TiDB in a development environment and a production environment, and demonstrates the key integration benefits through the scenario of query routing.

If you are interested in learning more about TiDB and ProxySQL, you can find some useful links as follows:

- TiDB Cloud
- TiDB Developer Guide
- ProxySQL Documentation

4.13.3.1 What is ProxySQL?

ProxySQL is a high-performance, open-source SQL proxy. It has a flexible architecture and can be deployed in several different ways, making it ideal for a variety of use cases. For example, ProxySQL can be used to improve performance by caching frequently-accessed data.

ProxySQL is designed from the ground up to be fast, efficient, and easy to use. It is fully compatible with MySQL, and supports all of the features you would expect from a high quality SQL proxy. In addition, ProxySQL comes with a number of unique features that make it an ideal choice for a wide range of applications.

4.13.3.2 Why ProxySQL integration?

- ProxySQL can help boost application performance by reducing latency when interacting with TiDB. Irrespective of what you are building, whether it is a scalable application using serverless functions like Lambda, where the workload is nondeterministic and can spike, or if you are building an application to execute queries that load tons of data. By leveraging powerful capabilities of ProxySQL such as connection pooling and caching frequently-used queries, applications can gain immediate benefits.
- ProxySQL can act as an additional layer of application security protection against SQL vulnerabilities such as SQL injection with the help of query rules, an easy-to-configure feature available in ProxySQL.
- As both ProxySQL and TiDB are open-source projects, you can get the benefits of zero vendor lock-in.

4.13.3.3 Deployment architecture

The most obvious way to deploy ProxySQL with TiDB is to add ProxySQL as a standalone intermediary between the application layer and TiDB. However, the scalability and failure tolerance are not guaranteed, and it also adds additional latency due to network hop. To avoid these problems, an alternate deployment architecture is to deploy ProxySQL as a sidecar as below:

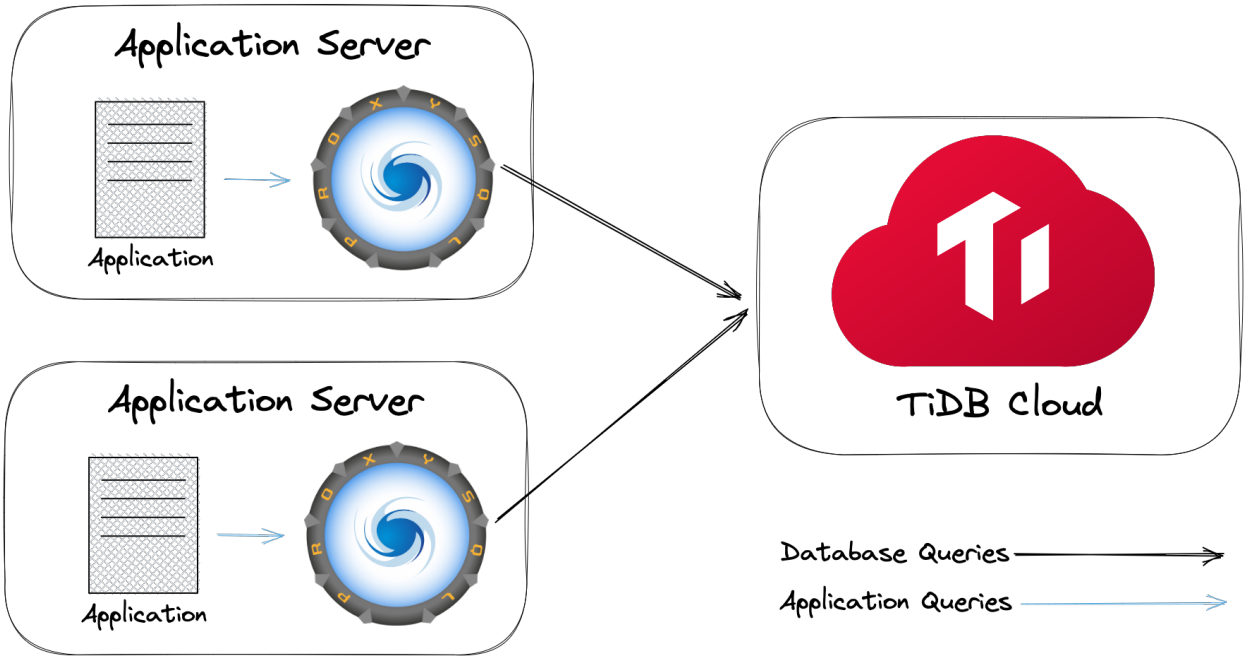


Figure 23: proxysql-client-side-tidb-cloud

Note:

The preceding illustration is only for reference. You must adapt it according to your actual deployment architecture.

4.13.3.4 Development environment

This section describes how to integrate TiDB with ProxySQL in a development environment. To get started with the ProxySQL integration, you can choose either of the following options depending on your TiDB cluster type after you have all the prerequisites in place.

- Option 1: Integrate TiDB Serverless with ProxySQL
- Option 2: Integrate TiDB (self-hosted) with ProxySQL

4.13.3.4.1 Prerequisites

Depending on the option you choose, you might need the following packages:

- Git
- Docker

- Python 3
- Docker Compose
- MySQL Client

You can follow the installation instructions as below:

1. Download and start Docker (the Docker Desktop already includes the Docker Compose).
2. Run the following command to install Python and `mysql-client`:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
    ↪ install/HEAD/install.sh)"
brew install python mysql-client
```

```
curl -fsSL https://get.docker.com | bash -s docker
yum install -y git python39 docker-ce docker-ce-cli containerd.io docker-
    ↪ compose-plugin mysql
systemctl start docker
```

- Download and install Git.

    1. Download the **64-bit Git for Windows Setup** package from the Git Windows Download page.
    2. Install the Git package by following the setup wizard. You can click **Next** for a few times to use the default installation settings.

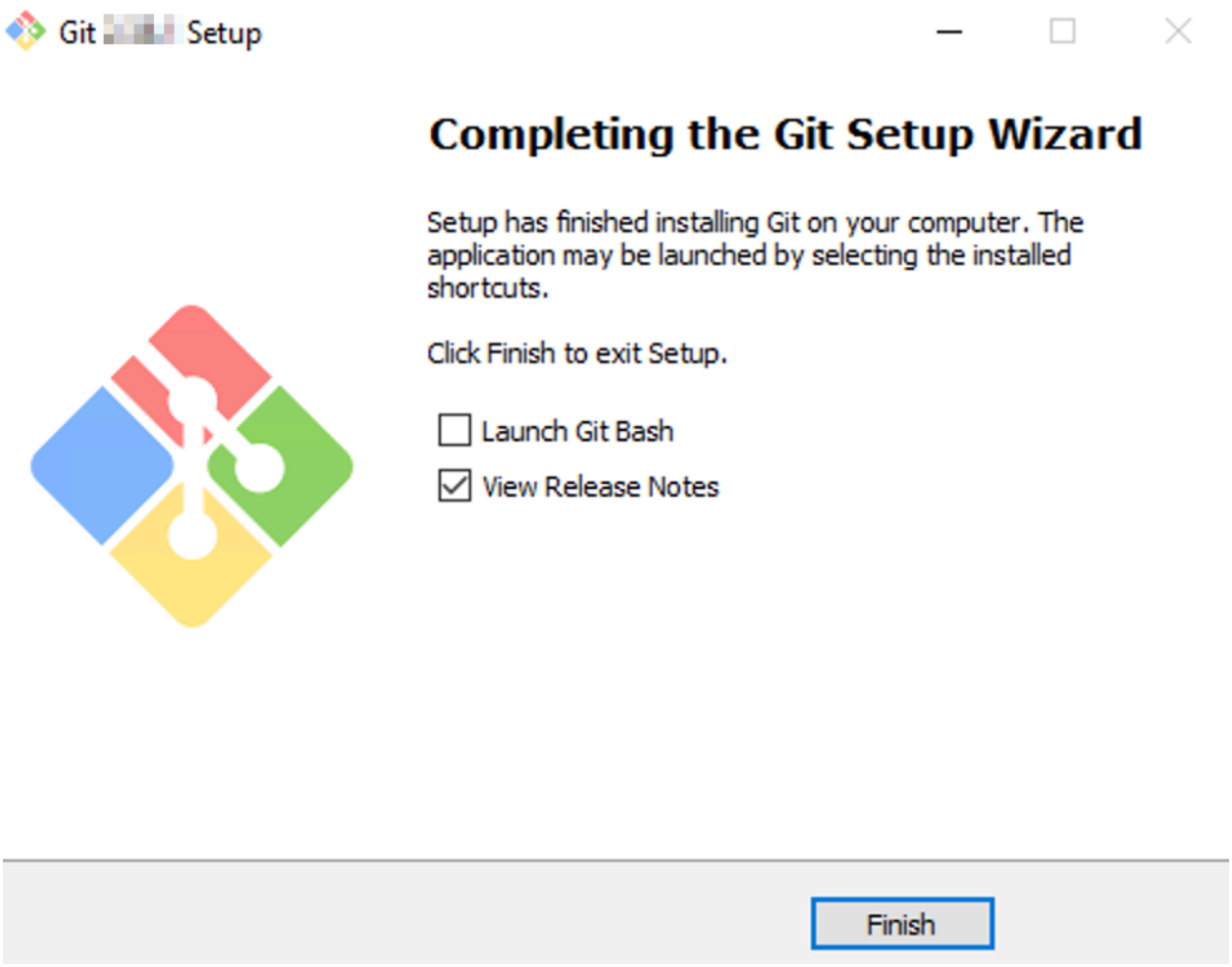


Figure 24: proxysql-windows-git-install

- Download and install MySQL Shell.

    1. Download the ZIP file of MySQL Installer from the MySQL Community Server Download page.
    2. Unzip the file, and locate `mysql.exe` in the `bin` folder. You need to add the path of the `bin` folder to the system variable and set it into the `PATH` variable at Git Bash:

    ```
    echo 'export PATH="(your bin folder)":$PATH' >>~/.bash_profile
    source ~/.bash_profile
    ```

    For example:

    ```
    echo 'export PATH="/c/Program Files (x86)/mysql-8.0.31-winx64/bin"
        ↪ :$PATH' >>~/.bash_profile
    source ~/.bash_profile
    ```

- Download and install Docker.

    1. Download Docker Desktop installer from the Docker Download page.
    2. Double-click the installer to run it. After the installation is completed, you will be prompted for a restart.

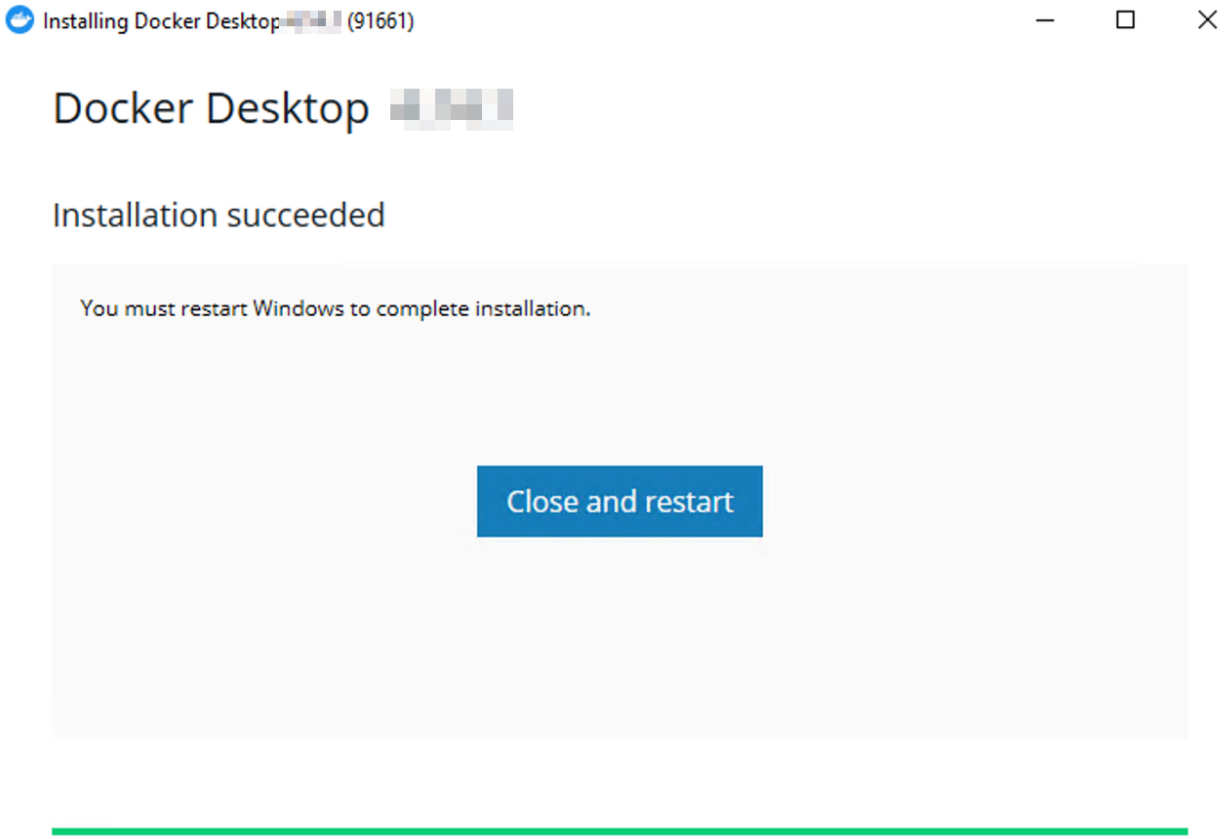


Figure 25: proxysql-windows-docker-install

- Download the latest Python 3 installer from the Python Download page and run it.

4.13.3.4.2 Option 1: Integrate TiDB Serverless with ProxySQL

For this integration, you will be using the ProxySQL Docker image along with a TiDB Serverless cluster. The following steps will set up ProxySQL on port `16033`, so make sure this port is available.

Step 1. Create a TiDB Serverless cluster

1. Create a TiDB Serverless cluster.
2. Follow the steps in Connect via Standard Connection to get the connection string and set a password for your cluster.

3. In the connection string, locate your cluster endpoint after `-h`, your user name after `-u`, and your cluster port after `-P`.

Step 2. Generate ProxySQL configuration files

1. Clone the integration example code repository for TiDB and ProxySQL:

```
git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
```


```
git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
```

```
git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
```


2. Change to the `tidb-cloud-connect` folder:

```
cd tidb-proxysql-integration/example/tidb-cloud-connect
```


```
cd tidb-proxysql-integration/example/tidb-cloud-connect
```

```
cd tidb-proxysql-integration/example/tidb-cloud-connect
```


3. Generate ProxySQL configuration files by running `proxysql-config.py`:

```
python3 proxysql-config.py
```


```
python3 proxysql-config.py
```


```
python proxysql-config.py
```

When prompted, enter the endpoint of your cluster for `Serverless Tier Host`, and then enter the username and the password of your cluster.

The following is an example output. You will see that three configuration files are generated under the current `tidb-cloud-connect` folder.

```
[Begin] generating configuration files..
tidb-cloud-connect.cnf generated successfully.
proxysql-prepare.sql generated successfully.
proxysql-connect.py generated successfully.
[End] all files generated successfully and placed in the current folder
    ↪ .
```

Step 3. Configure ProxySQL

1. Start Docker. If Docker has already started, skip this step:

   Double-click the icon of the installed Docker to start it.

   ```
   systemctl start docker
   ```

   Double-click the icon of the installed Docker to start it.

2. Pull the ProxySQL image and start a ProxySQL container in the background:

   ```
   docker compose up -d
   ```


   ```
   docker compose up -d
   ```

   ```
   docker compose up -d
   ```


3. Integrate with ProxySQL by running the following command, which executes `proxysql -prepare.sql` inside **ProxySQL Admin Interface**:

   ```
   docker compose exec proxysql sh -c "mysql -uadmin -padmin -h127.0.0.1 -P6032 < ./proxysql-prepare.sql"
   ```


   ```
   docker compose exec proxysql sh -c "mysql -uadmin -padmin -h127.0.0.1 -P6032 < ./proxysql-prepare.sql"
   ```

   ```
   docker compose exec proxysql sh -c "mysql -uadmin -padmin -h127.0.0.1 -P6032 < ./proxysql-prepare.sql"
   ```


Note:

The `proxysql-prepare.sql` script does the following:

1. Adds a user using the username and password of your cluster.
2. Assigns the user to the monitoring account.
3. Adds your TiDB Serverless cluster to the list of hosts.
4. Enables a secure connection between ProxySQL and the TiDB Serverless cluster.

To have a better understanding, it is strongly recommended that you check the `proxysql-prepare.sql` file. To learn more about ProxySQL configuration, see ProxySQL documentation.

The following is an example output. You will see that the hostname of your cluster is shown in the output, which means that the connectivity between ProxySQL and the TiDB Serverless cluster is established.

```
*************************** 1. row ***************************
   hostgroup_id: 0
       hostname: gateway01.us-west-2.prod.aws.tidbcloud.com
           port: 4000
       gtid_port: 0
           status: ONLINE
           weight: 1
       compression: 0
   max_connections: 1000
max_replication_lag: 0
           use_ssl: 1
   max_latency_ms: 0
           comment:
```

Step 4. Connect to your TiDB cluster through ProxySQL

1. To connect to your TiDB cluster, run `proxysql-connect.py`. The script will automatically launch the MySQL client and use the username and password you specified in Step 2 for connection.

```
python3 proxysql-connect.py
```

```
python3 proxysql-connect.py
```

```
python proxysql-connect.py
```

2. After connecting to your TiDB cluster, you can use the following SQL statement to validate the connection:

```
SELECT VERSION();
```

If the TiDB version is displayed, you are successfully connected to your TiDB Serverless cluster through ProxySQL. To exit from the MySQL client anytime, enter `quit` and press enter.

> **Note:**
>
> ***For Debugging:*** If you are unable to connect to the cluster, check the files `tidb-cloud-connect.cnf`, `proxysql-prepare.sql`, and `proxysql-connect.py`. Make sure that the server information you provided is available and correct.

3. To stop and remove containers, and go to the previous directory, run the following command:

```
docker compose down
cd -
```


```
docker compose down
cd -
```

```
docker compose down
cd -
```


4.13.3.4.3 Option 2: Integrate TiDB (self-hosted) with ProxySQL

For this integration, you will set up an environment using Docker images of TiDB and ProxySQL. You are encouraged to try other ways of installing TiDB (self-hosted) in your own interest.

The following steps will set up ProxySQL and TiDB on ports `6033` and `4000` respectively, so make sure these ports are available.

1. Start Docker. If Docker has already started, skip this step:

   Double-click the icon of the installed Docker to start it.

   ```
   systemctl start docker
   ```

   Double-click the icon of the installed Docker to start it.

2. Clone the integration example code repository for TiDB and ProxySQL:

   ```
   git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
   ```


   ```
   git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
   ```

   ```
   git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
   ```


3. Pull the latest images of ProxySQL and TiDB:

   ```
   cd tidb-proxysql-integration && docker compose pull
   ```


   ```
   cd tidb-proxysql-integration && docker compose pull
   ```

   ```
   cd tidb-proxysql-integration && docker compose pull
   ```


4. Start an integrated environment using both TiDB and ProxySQL running as containers:

   ```
   docker compose up -d
   ```

```
docker compose up -d
```


```
docker compose up -d
```


To log in to the ProxySQL `6033` port, you can use the `root` username with an empty password.

5. Connect to TiDB via ProxySQL:

```
mysql -u root -h 127.0.0.1 -P 6033
```


```
mysql -u root -h 127.0.0.1 -P 6033
```

```
mysql -u root -h 127.0.0.1 -P 6033
```


6. After connecting to your TiDB cluster, you can use the following SQL statement to validate the connection:

```
SELECT VERSION();
```

If the TiDB version is displayed, you are successfully connected to your TiDB containers through ProxySQL.

7. To stop and remove containers, and go to the previous directory, run the following command:

```
docker compose down
cd -
```


```
docker compose down
cd -
```

```
docker compose down
cd -
```


4.13.3.5 Production environment

For a production environment, it is recommended that you use TiDB Dedicated directly for a fully-managed experience.

4.13.3.5.1 Prerequisite

Download and install a MySQL client. For example, MySQL Shell.

4.13.3.5.2 Integrate TiDB Cloud with ProxySQL on CentOS

ProxySQL can be installed on many different platforms. The following takes CentOS as an example.

For a full list of supported platforms and the corresponding version requirements, see ProxySQL documentation.

Step 1. Create a TiDB Dedicated cluster

For detailed steps, see Create a TiDB Cluster.

Step 2. Install ProxySQL

1. Add ProxySQL to the YUM repository:

```
cat > /etc/yum.repos.d/proxysql.repo << EOF
[proxysql]
name=ProxySQL YUM repository
baseurl=https://repo.proxysql.com/ProxySQL/proxysql-2.4.x/centos/\
    ↪ $releasever
gpgcheck=1
gpgkey=https://repo.proxysql.com/ProxySQL/proxysql-2.4.x/repo_pub_key
EOF
```

2. Install ProxySQL:

```
yum install -y proxysql
```

3. Start ProxySQL:

```
systemctl start proxysql
```

To learn more about the supported platforms of ProxySQL and their installation, refer to ProxySQL README or ProxySQL installation documentation.

Step 3. Configure ProxySQL

To use ProxySQL as a proxy for TiDB, you need to configure ProxySQL. To do so, you can either execute SQL statements inside ProxySQL Admin Interface (recommended) or use the configuration file.

> **Note:**
>
> The following sections list only the required configuration items of ProxySQL. For a comprehensive list of configurations, see ProxySQL documentation.

Option 1: Configure ProxySQL using the Admin Interface

1. Reconfigure ProxySQL's internals using the standard ProxySQL Admin interface, accessible via any MySQL command line client (available by default on port `6032`):

```
mysql -u admin -padmin -h 127.0.0.1 -P6032 --prompt 'ProxySQL Admin> '
```

   The above step will take you to the ProxySQL admin prompt.

2. Configure the TiDB clusters to be used, where you can add one or multiple TiDB clusters to ProxySQL. The following statement will add one TiDB Dedicated cluster for example. You need to replace `<tidb cloud dedicated cluster host>` and `<tidb cloud dedicated cluster port>` with your TiDB Cloud endpoint and port (the default port is `4000`).

```
INSERT INTO mysql_servers(hostgroup_id, hostname, port)
VALUES
  (
    0,
    '<tidb cloud dedicated cluster host>',
    <tidb cloud dedicated cluster port>
  );
LOAD mysql servers TO runtime;
SAVE mysql servers TO DISK;
```

Note:

- `hostgroup_id`: specify an ID of the hostgroup. ProxySQL manages clusters using hostgroup. To distribute SQL traffic to these clusters evenly, you can configure several clusters that need load balancing to the same hostgroup. To distinguish the clusters, such as for read and write purposes, you can configure them to use different hostgroups.
- `hostname`: the endpoint of the TiDB cluster.
- `port`: the port of the TiDB cluster.

3. Configure Proxy login users to make sure that the users have appropriate permissions on the TiDB cluster. In the following statements, you need to replace '*tidb cloud dedicated cluster username*' and '*tidb cloud dedicated cluster password*' with the actual username and password of your cluster.

```
INSERT INTO mysql_users(
  username, password, active, default_hostgroup,
  transaction_persistent
)
VALUES
  (
    '<tidb cloud dedicated cluster username>',
```

```
    '<tidb cloud dedicated cluster password>',
    1, 0, 1
  );
LOAD mysql users TO runtime;
SAVE mysql users TO DISK;
```

Note:

- `username`: TiDB username.
- `password`: TiDB password.
- `active`: controls whether the user is active. `1` indicates that the user is **active** and can be used for login, while `0` indicates that the user is inactive.
- `default_hostgroup`: the default hostgroup used by the user, where SQL traffic is distributed unless the query rule overrides the traffic to a specific hostgroup.
- `transaction_persistent`: `1` indicates a persistent transaction. When a user starts a transaction within a connection, all query statements are routed to the same hostgroup until the transaction is committed or rolled back.

Option 2: Configure ProxySQL using a configuration file

This option should only be considered as an alternate method for configuring ProxySQL. For more information, see Configuring ProxySQL through the config file.

1. Delete any existing SQLite database (where configurations are stored internally):

```
rm /var/lib/proxysql/proxysql.db
```

Warning:

If you delete the SQLite database file, any configuration changes made using ProxySQL Admin interface will be lost.

2. Modify the configuration file `/etc/proxysql.cnf` according to your need. For example:

```
mysql_servers:
(
    {
        address="<tidb cloud dedicated cluster host>"
        port=<tidb cloud dedicated cluster port>
        hostgroup=0
```

```
        max_connections=2000
    }
)

mysql_users:
(
    {
        username = "<tidb cloud dedicated cluster username>"
        password = "<tidb cloud dedicated cluster password>"
        default_hostgroup = 0
        max_connections = 1000
        default_schema = "test"
        active = 1
        transaction_persistent = 1
    }
)
```

In the preceding example:

- `address` and `port`: specify the endpoint and port of your TiDB Cloud cluster.
- `username` and `password`: specify the username and password of your TiDB Cloud cluster.

3. Restart ProxySQL:

```
systemctl restart proxysql
```

After the restart, the SQLite database will be created automatically.

Warning:

Do not run ProxySQL with default credentials in production. Before starting the `proxysql` service, you can change the defaults in the `/etc/proxysql.cnf` file by changing the `admin_credentials` variable.

4.13.3.6 Typical scenario

This section takes query routing as an example to show some of the benefits that you can leverage by integrating ProxySQL with TiDB.

4.13.3.6.1 Query rules

Databases can be overloaded by high traffic, faulty code, or malicious spam. With query rules of ProxySQL, you can respond to these issues quickly and effectively by rerouting, rewriting, or rejecting queries.

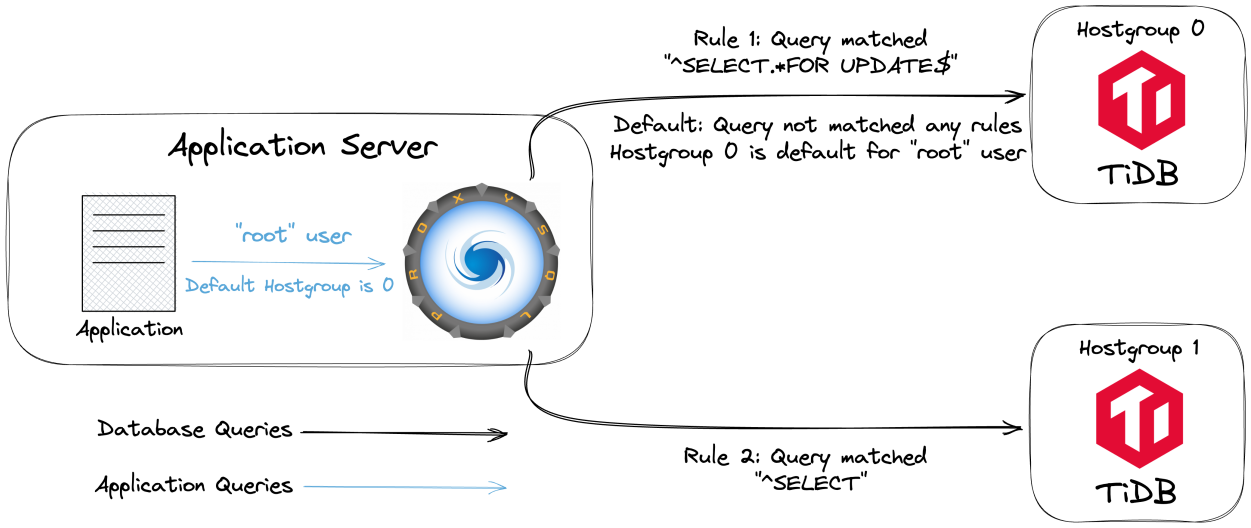


Figure 26: proxysql-client-side-rules

> **Note:**
>
> In the following steps, you will be using the container images of TiDB and ProxySQL to configure query rules. If you have not pulled them, you can check the integration section for detailed steps.

1. Clone the integration example code repository for TiDB and ProxySQL. Skip this step if you have already cloned it in the previous steps.

```
git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
```


```
git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
```

```
git clone https://github.com/pingcap-inc/tidb-proxysql-integration.git
```


2. Change to the example directory for ProxySQL rules:

```
cd tidb-proxysql-integration/example/proxy-rule-admin-interface
```

```
cd tidb-proxysql-integration/example/proxy-rule-admin-interface
```


```
cd tidb-proxysql-integration/example/proxy-rule-admin-interface
```


3. Run the following command to start two TiDB containers and a ProxySQL container:

```
docker compose up -d
```


```
docker compose up -d
```

```
docker compose up -d
```


If everything goes well, the following containers are started:

- Two Docker containers of TiDB clusters exposed via ports 4001, 4002
- One ProxySQL Docker container exposed via port 6034.

4. In the two TiDB containers, using `mysql` to create a table with a similar schema definition and then insert different data (`'tidb-server01-port-4001'`, `'tidb-server02-port-4002'`) to identify these containers.

```
mysql -u root -h 127.0.0.1 -P 4001 << EOF
DROP TABLE IF EXISTS test.tidb_server;
CREATE TABLE test.tidb_server (server_name VARCHAR(255));
INSERT INTO test.tidb_server (server_name) VALUES ('tidb-server01-port
    ↪ -4001');
EOF

mysql -u root -h 127.0.0.1 -P 4002 << EOF
DROP TABLE IF EXISTS test.tidb_server;
CREATE TABLE test.tidb_server (server_name VARCHAR(255));
INSERT INTO test.tidb_server (server_name) VALUES ('tidb-server02-port
    ↪ -4002');
EOF
```


```
mysql -u root -h 127.0.0.1 -P 4001 << EOF
DROP TABLE IF EXISTS test.tidb_server;
CREATE TABLE test.tidb_server (server_name VARCHAR(255));
INSERT INTO test.tidb_server (server_name) VALUES ('tidb-server01-port
    ↪ -4001');
EOF

mysql -u root -h 127.0.0.1 -P 4002 << EOF
DROP TABLE IF EXISTS test.tidb_server;
CREATE TABLE test.tidb_server (server_name VARCHAR(255));
```

```
INSERT INTO test.tidb_server (server_name) VALUES ('tidb-server02-port
    ↪ -4002');
EOF
```

```
mysql -u root -h 127.0.0.1 -P 4001 << EOF
DROP TABLE IF EXISTS test.tidb_server;
CREATE TABLE test.tidb_server (server_name VARCHAR(255));
INSERT INTO test.tidb_server (server_name) VALUES ('tidb-server01-port
    ↪ -4001');
EOF

mysql -u root -h 127.0.0.1 -P 4002 << EOF
DROP TABLE IF EXISTS test.tidb_server;
CREATE TABLE test.tidb_server (server_name VARCHAR(255));
INSERT INTO test.tidb_server (server_name) VALUES ('tidb-server02-port
    ↪ -4002');
EOF
```

5. Configure ProxySQL by running the following command, which executes `proxysql-prepare.sql` inside ProxySQL Admin Interface to establish a proxy connection between the TiDB containers and ProxySQL.

```
docker compose exec proxysql sh -c "mysql -uadmin -padmin -h127.0.0.1 -
    ↪ P6032 < ./proxysql-prepare.sql"
```

```
docker compose exec proxysql sh -c "mysql -uadmin -padmin -h127.0.0.1 -
    ↪ P6032 < ./proxysql-prepare.sql"
```

```
docker compose exec proxysql sh -c "mysql -uadmin -padmin -h127.0.0.1 -
    ↪ P6032 < ./proxysql-prepare.sql"
```

Note:

The `proxysql-prepare.sql` does the following:

- Adds the TiDB clusters in ProxySQL with `hostgroup_id` as `0` and `1`.
- Adds a user `root` with an empty password and sets `default_hostgroup` as `0`.
- Adds the rule `^SELECT.*FOR UPDATE$` with `rule_id` as `1` and `destination_hostgroup` as `0`. If a SQL statement matches this rule, the request will be forwarded to the TiDB cluster with `hostgroup` as `0`.

- Adds the rule `^SELECT` with `rule_id` as `2` and `destination_hostgroup` as `1`. If a SQL statement matches this rule, the request will be forwarded to the TiDB cluster with `hostgroup` as `1`.

To have a better understanding, it is strongly recommended that you check the `proxysql-prepare.sql` file. To learn more about ProxySQL configuration, see ProxySQL documentation.

The following is some additional information about how ProxySQL patterns match query rules:

- ProxySQL tries to match the rules one by one in the ascending order of `rule_id`.
- `^` symbol matches the beginning of a SQL statement and `$` matches the end.

For more information about ProxySQL regular expression and pattern matching, see mysql-query_processor_regex in ProxySQL documentation.

For a full list of parameters, see mysql_query_rules in ProxySQL documentation.

6. Verify the configuration and check whether the query rules work.

    1. Log into ProxySQL MySQL Interface as the `root` user:

    ```
    mysql -u root -h 127.0.0.1 -P 6034
    ```


    ```
    mysql -u root -h 127.0.0.1 -P 6034
    ```

    ```
    mysql -u root -h 127.0.0.1 -P 6034
    ```


    2. Execute the following SQL statements:

        - Execute a `SELECT` statement:

        ```
        SELECT * FROM test.tidb_server;
        ```

        This statement will match rule_id `2` and forward the statement to the TiDB cluster on `hostgroup 1`.

        - Execute a `SELECT ... FOR UPDATE` statement:

        ```
        SELECT * FROM test.tidb_server FOR UPDATE;
        ```

        This statement will match rule_id `1` and forward the statement to the TiDB cluster on `hostgroup 0`.

        - Start a transaction:

```
BEGIN;
INSERT INTO test.tidb_server (server_name) VALUES ('insert this
    ↪ and rollback later');
SELECT * FROM test.tidb_server;
ROLLBACK;
```

In this transaction, the `BEGIN` statement will not match any rules. It uses the default hostgroup (`hostgroup 0` in this example). Because ProxySQL enables user transaction_persistent by default, which will execute all statements within the same transaction in the same hostgroup, the `INSERT` and `SELECT * FROM test.tidb_server;` statements will also be forwarded to the TiDB cluster `hostgroup 0`.

The following is an example output. If you get a similar output, you have successfully configured the query rules with ProxySQL.

```
+-------------------------+
| server_name             |
+-------------------------+
| tidb-server02-port-4002 |
+-------------------------+
+-------------------------+
| server_name             |
+-------------------------+
| tidb-server01-port-4001 |
+-------------------------+
+--------------------------------+
| server_name                    |
+--------------------------------+
| tidb-server01-port-4001        |
| insert this and rollback later |
+--------------------------------+
```

3. To exit from the MySQL client anytime, enter `quit` and press enter.

7. To stop and remove containers, and go to the previous directory, run the following command:

```
docker compose down
cd -
```


```
docker compose down
cd -
```

```
docker compose down
cd -
```

4.13.4 Integrate TiDB with Amazon AppFlow

Amazon AppFlow is a fully managed API integration service that you use to connect your software as a service (SaaS) applications to AWS services, and securely transfer data. With Amazon AppFlow, you can import and export data from and to TiDB into many types of data providers, such as Salesforce, Amazon S3, LinkedIn, and GitHub. For more information, see Supported source and destination applications in AWS documentation.

This document describes how to integrate TiDB with Amazon AppFlow and takes integrating a TiDB Serverless cluster as an example.

If you do not have a TiDB cluster, you can create a TiDB Serverless cluster, which is free and can be created in approximately 30 seconds.

4.13.4.1 Prerequisites

- Git
- JDK 11 or above
- Maven 3.8 or above
- AWS CLI version 2
- AWS Serverless Application Model Command Line Interface (AWS SAM CLI) 1.58.0 or above
- An AWS Identity and Access Management (IAM) user with the following requirements:
    - The user can access AWS using an access key.
    - The user has the following permissions:
        - `AWSCertificateManagerFullAccess`: used for reading and writing the AWS Secrets Manager.
        - `AWSCloudFormationFullAccess`: SAM CLI uses AWS CloudFormation to proclaim the AWS resources.
        - `AmazonS3FullAccess`: AWS CloudFormation uses Amazon S3 to publish.
        - `AWSLambda_FullAccess`: currently, AWS Lambda is the only way to implement a new connector for Amazon AppFlow.
        - `IAMFullAccess`: SAM CLI needs to create a `ConnectorFunctionRole` for the connector.
- A SalesForce account.

4.13.4.2 Step 1. Register a TiDB connector

4.13.4.2.1 Clone the code

Clone the integration example code repository for TiDB and Amazon AppFlow:

```
git clone https://github.com/pingcap-inc/tidb-appflow-integration
```

4.13.4.2.2 Build and upload a Lambda

1. Build the package:

```
cd tidb-appflow-integration
mvn clean package
```

2. (Optional) Configure your AWS access key ID and secret access key if you have not.

```
aws configure
```

3. Upload your JAR package as a Lambda:

```
sam deploy --guided
```

Note:

- The `--guided` option uses prompts to guide you through the deployment. Your input will be stored in a configuration file, which is `samconfig.toml` by default.
- `stack_name` specifies the name of AWS Lambda that you are deploying.
- This prompted guide uses AWS as the cloud provider of TiDB Serverless. To use Amazon S3 as the source or destination, you need to set the `region` of AWS Lambda as the same as that of Amazon S3.
- If you have already run `sam deploy --guided` before, you can just run `sam deploy` instead, and SAM CLI will use the configuration file `samconfig.toml` to simplify the interaction.

If you see a similar output as follows, this Lambda is successfully deployed.

```
Successfully created/updated stack - <stack_name> in <region>
```

4. Go to the AWS Lambda console, and you can see the Lambda that you just uploaded. Note that you need to select the correct region in the upper-right corner of the window.

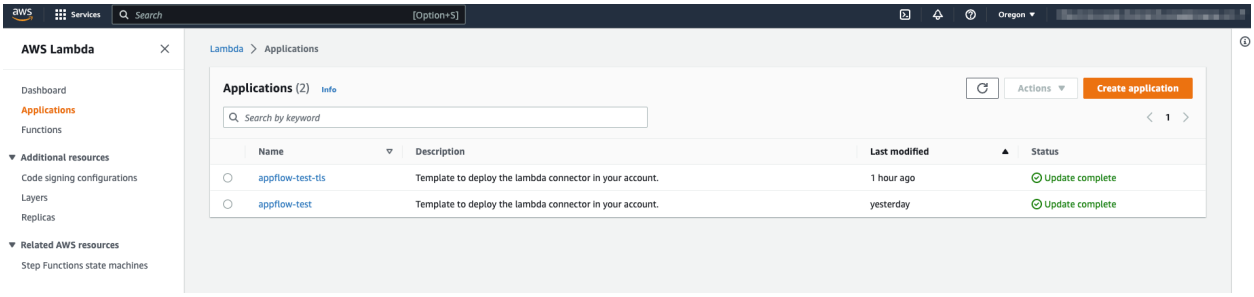


Figure 27: lambda dashboard

4.13.4.2.3 Use Lambda to register a connector

1. In the AWS Management Console, navigate to Amazon AppFlow > Connectors and click **Register new connector**.

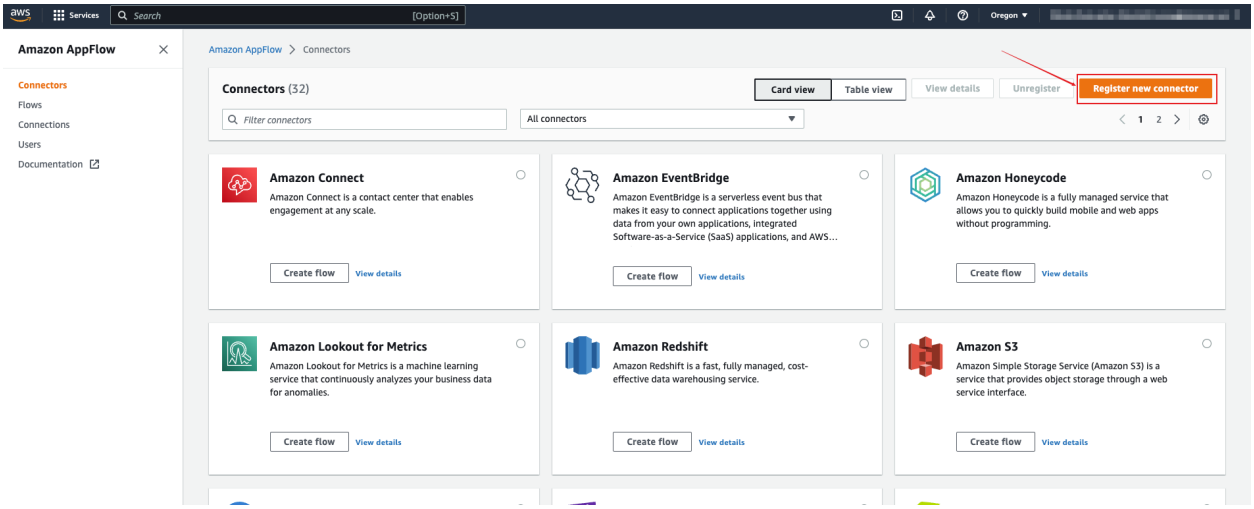


Figure 28: register connector

2. In the **Register a new connector** dialog, choose the Lambda function you uploaded and specify the connector label using the connector name.

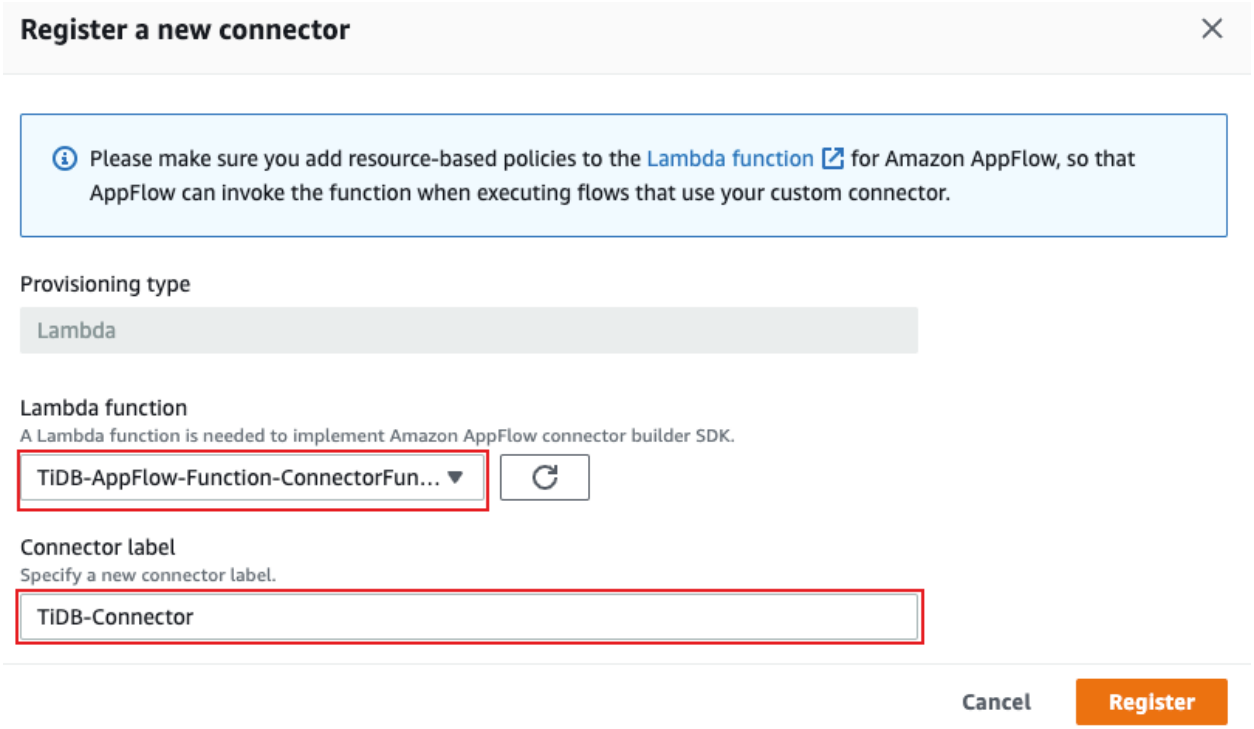


Figure 29: register connector dialog

3. Click **Register**. Then, a TiDB connector is registered successfully.

4.13.4.3 Step 2. Create a flow

Navigate to Amazon AppFlow > Flows and click **Create flow**.

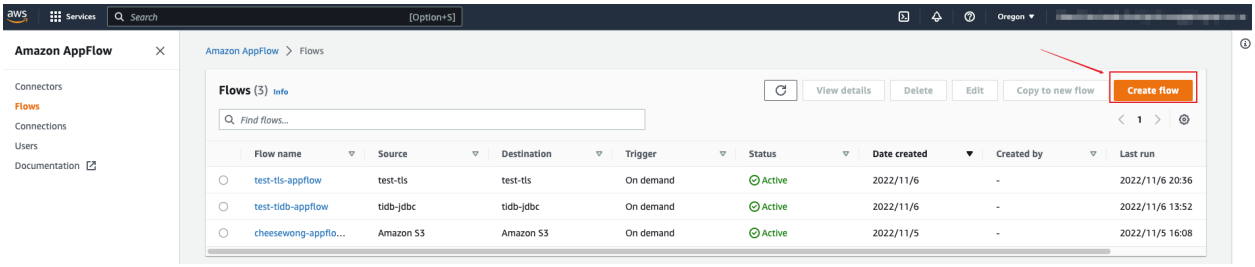


Figure 30: create flow

4.13.4.3.1 Set the flow name

Enter the flow name, and then click **Next**.

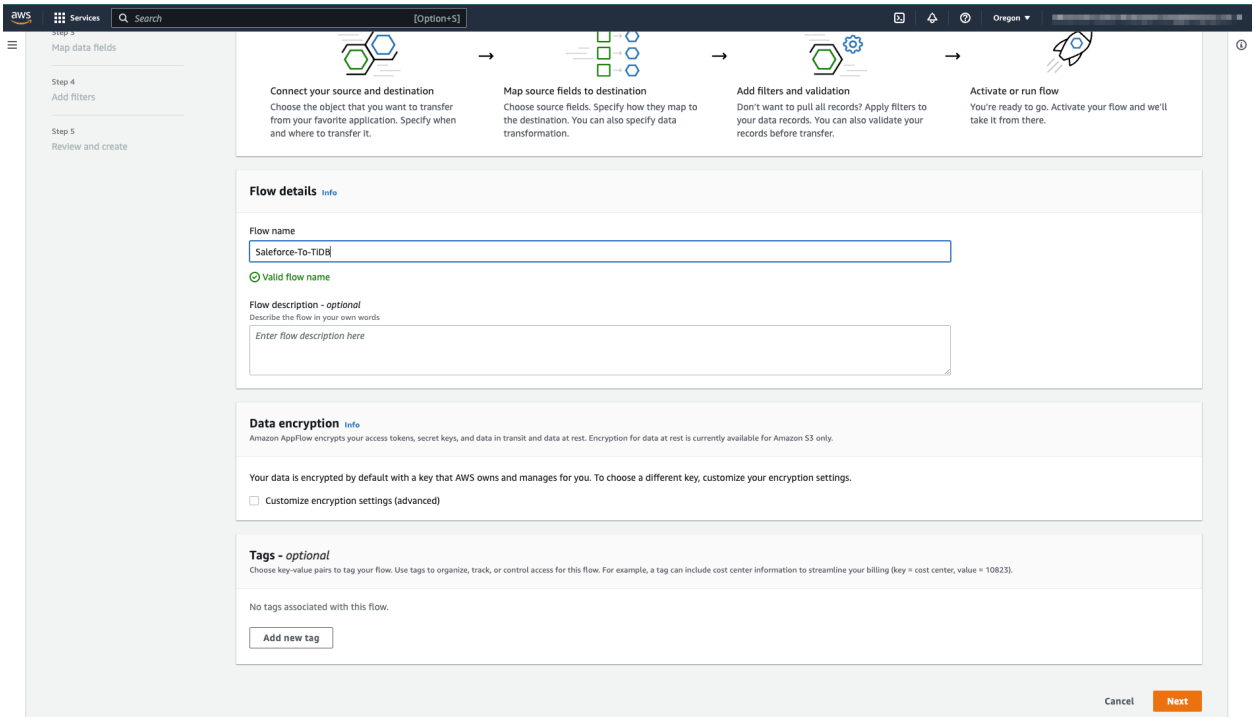


Figure 31: name flow

4.13.4.3.2 Set the source and destination tables

Choose the **Source details** and **Destination details**. TiDB connector can be used in both of them.

1. Choose the source name. This document uses **Salesforce** as an example source.

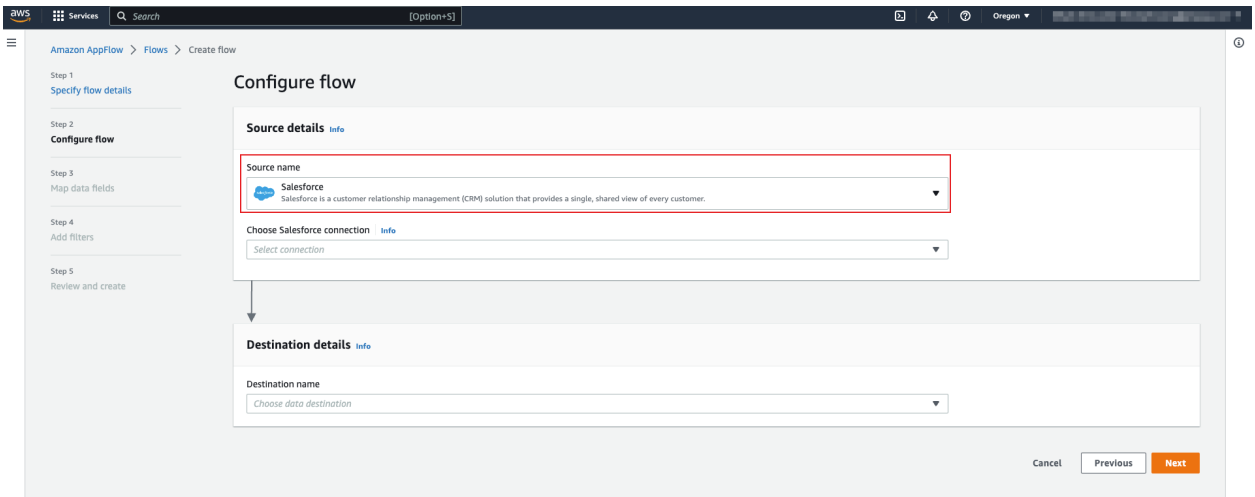
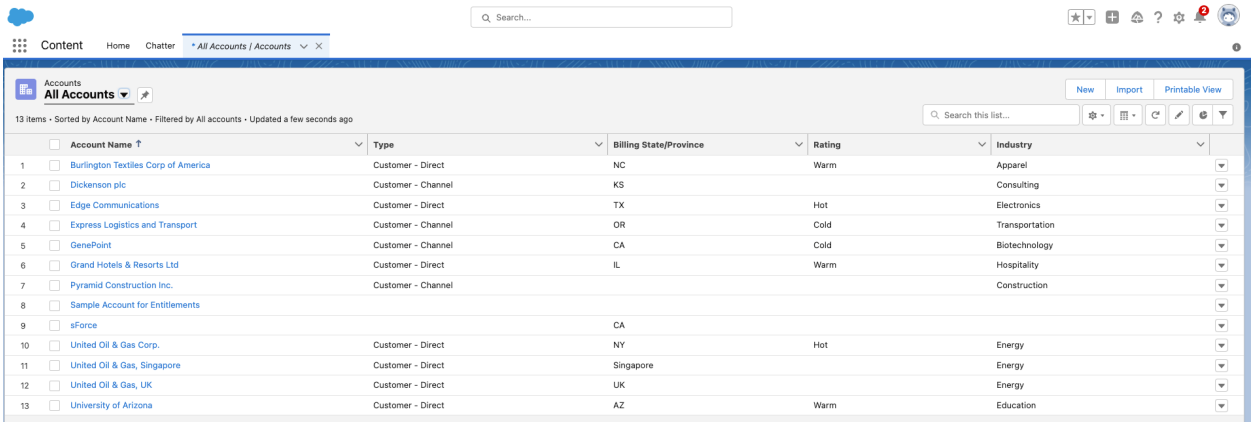


Figure 32: salesforce source

After you register to Salesforce, Salesforce will add some example data to your platform. The following steps will use the **Account** object as an example source object.



	Account Name	Type	Billing State/Province	Rating	Industry
1	Burlington Textiles Corp of America	Customer - Direct	NC	Warm	Apparel
2	Dickenson plc	Customer - Channel	KS		Consulting
3	Edge Communications	Customer - Direct	TX	Hot	Electronics
4	Express Logistics and Transport	Customer - Channel	OR	Cold	Transportation
5	GenePoint	Customer - Channel	CA	Cold	Biotechnology
6	Grand Hotels & Resorts Ltd	Customer - Direct	IL	Warm	Hospitality
7	Pyramid Construction Inc.	Customer - Channel			Construction
8	Sample Account for Entitlements				
9	sForce		CA		
10	United Oil & Gas Corp.	Customer - Direct	NY	Hot	Energy
11	United Oil & Gas, Singapore	Customer - Direct	Singapore		Energy
12	United Oil & Gas, UK	Customer - Direct	UK		Energy
13	University of Arizona	Customer - Direct	AZ	Warm	Education

Figure 33: salesforce data

2. Click **Connect**.

    1. In the **Connect to Salesforce** dialog, specify the name of this connection, and then click **Continue**.

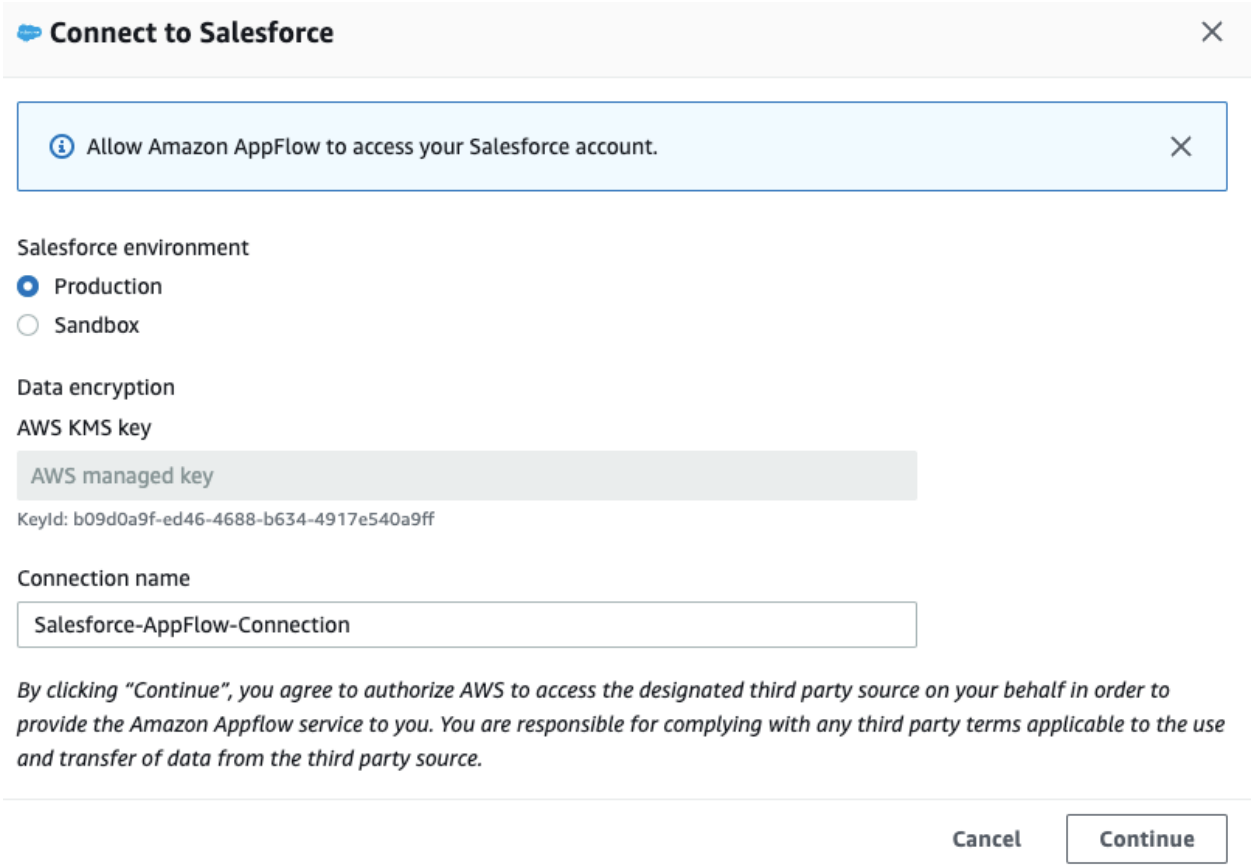


Figure 34: connect to salesforce

2. Click **Allow** to confirm that AWS can read your Salesforce data.

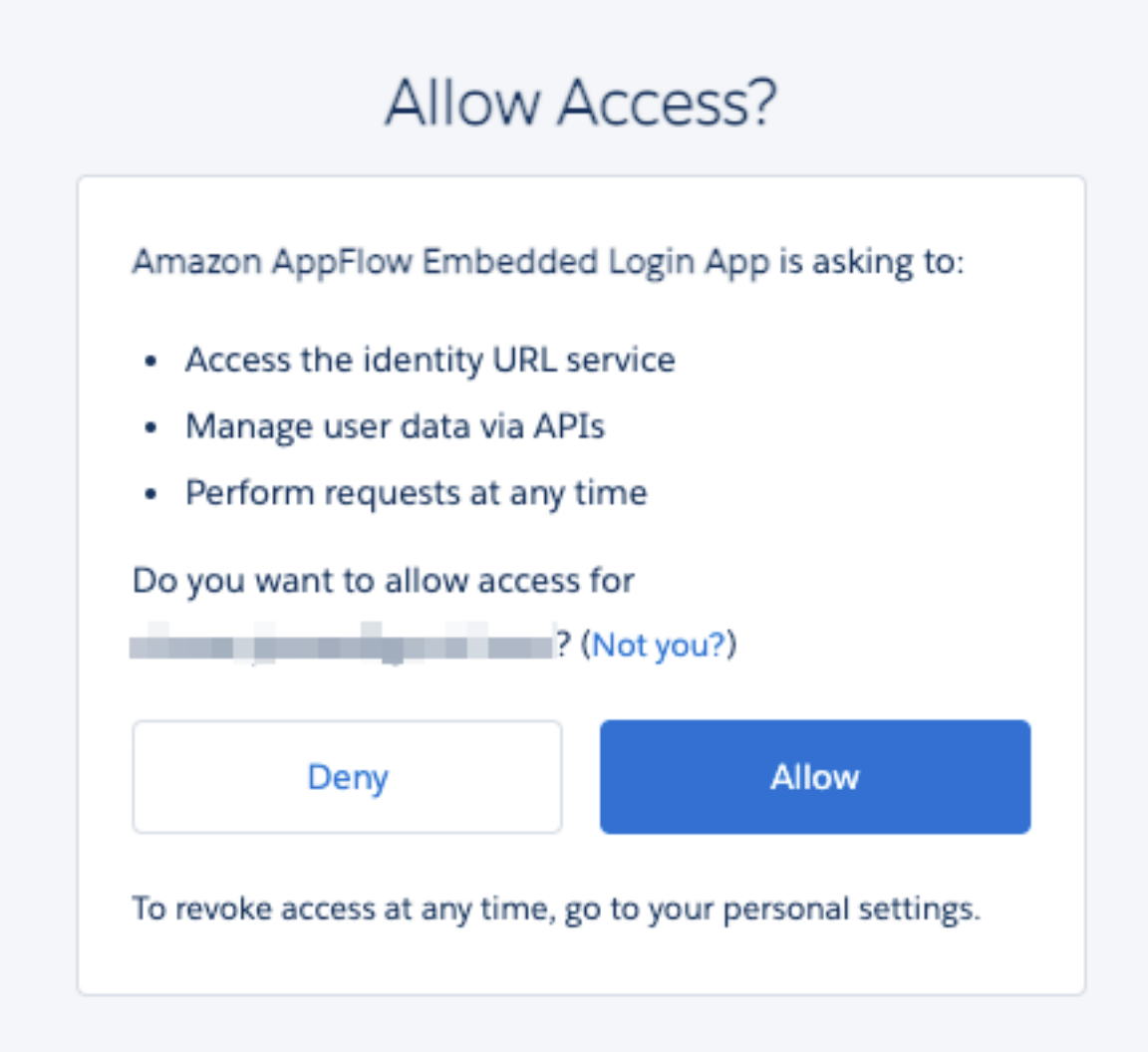


Figure 35: allow salesforce

Note:
If your company has already used the Professional Edition of Salesforce, the REST API is not enabled by default. You might need to register a new Developer Edition to use the REST API. For more information, refer to Salesforce Forum Topic.

3. In the **Destination details** area, choose **TiDB-Connector** as the destination. The **Connect** button is displayed.

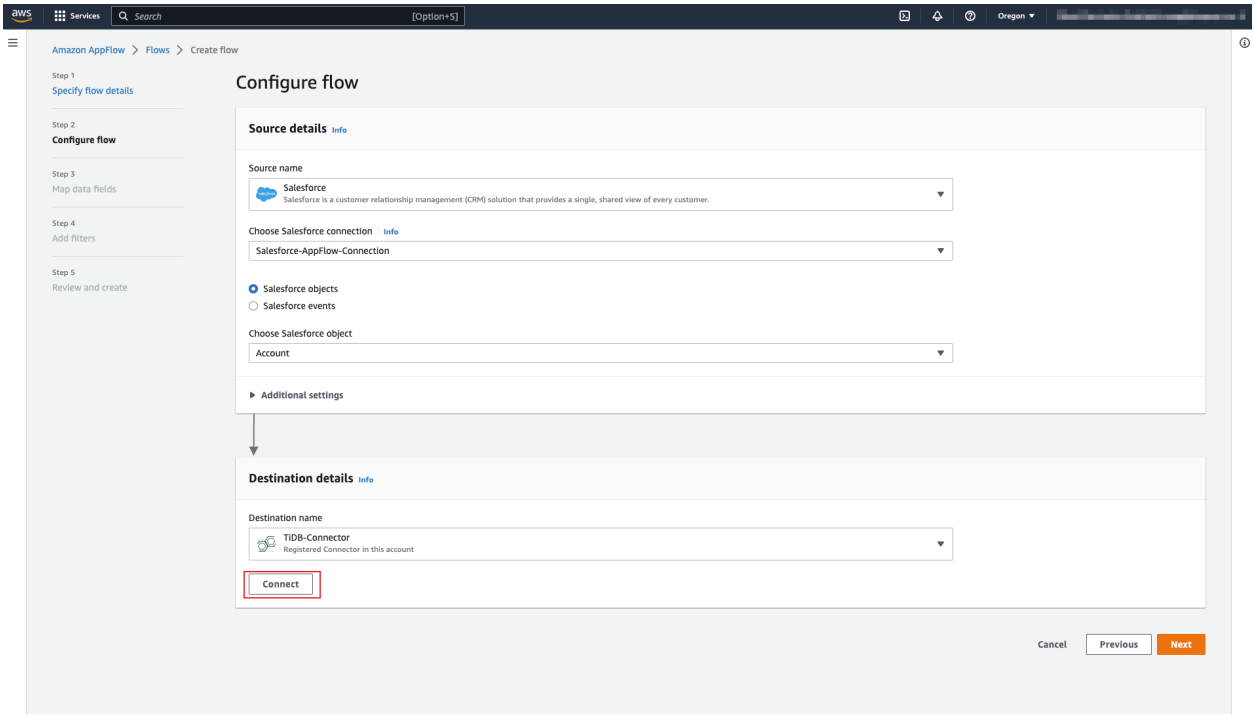


Figure 36: tidb dest

4. Before clicking **Connect**, you need to create a `sf_account` table in TiDB for the Salesforce **Account** object. Note that this table schema is different from the sample data in Tutorial of Amazon AppFlow.

```
CREATE TABLE `sf_account` (
    `id` varchar(255) NOT NULL,
    `name` varchar(150) NOT NULL DEFAULT '',
    `type` varchar(150) NOT NULL DEFAULT '',
    `billing_state` varchar(255) NOT NULL DEFAULT '',
    `rating` varchar(255) NOT NULL DEFAULT '',
    `industry` varchar(255) NOT NULL DEFAULT '',
    PRIMARY KEY (`id`)
);
```

5. After the `sf_account` table is created, click **Connect**. A connection dialog is displayed.

6. In the **Connect to TiDB-Connector** dialog, enter the connection properties of the TiDB cluster. If you use a TiDB Serverless cluster, you need to set the **TLS** option to `Yes`, which lets the TiDB connector use the TLS connection. Then, click **Connect**.

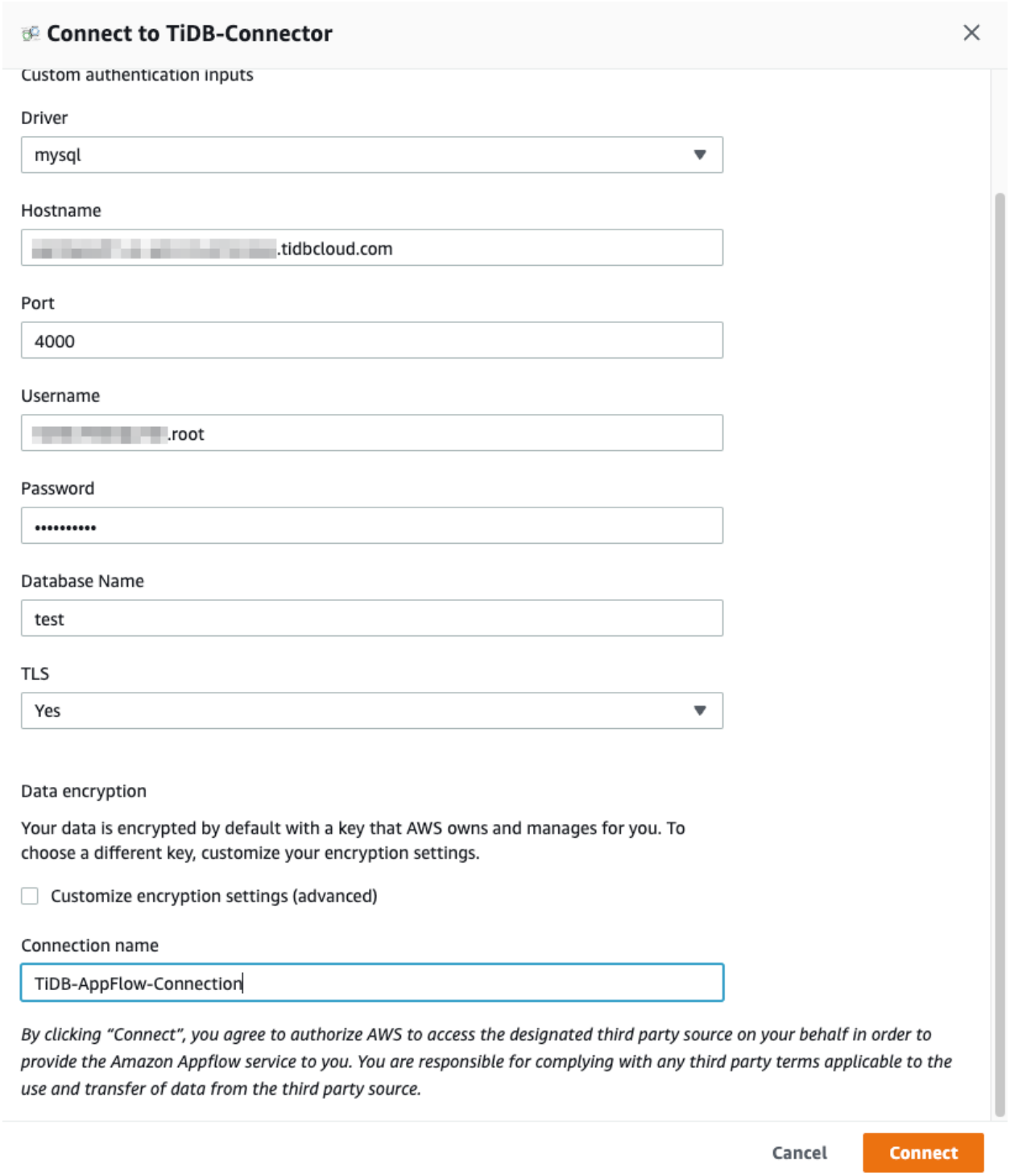


Figure 37: tidb connection message

7. Now you can get all tables in the database that you specified for connection. Choose the **sf_account** table from the drop-down list.

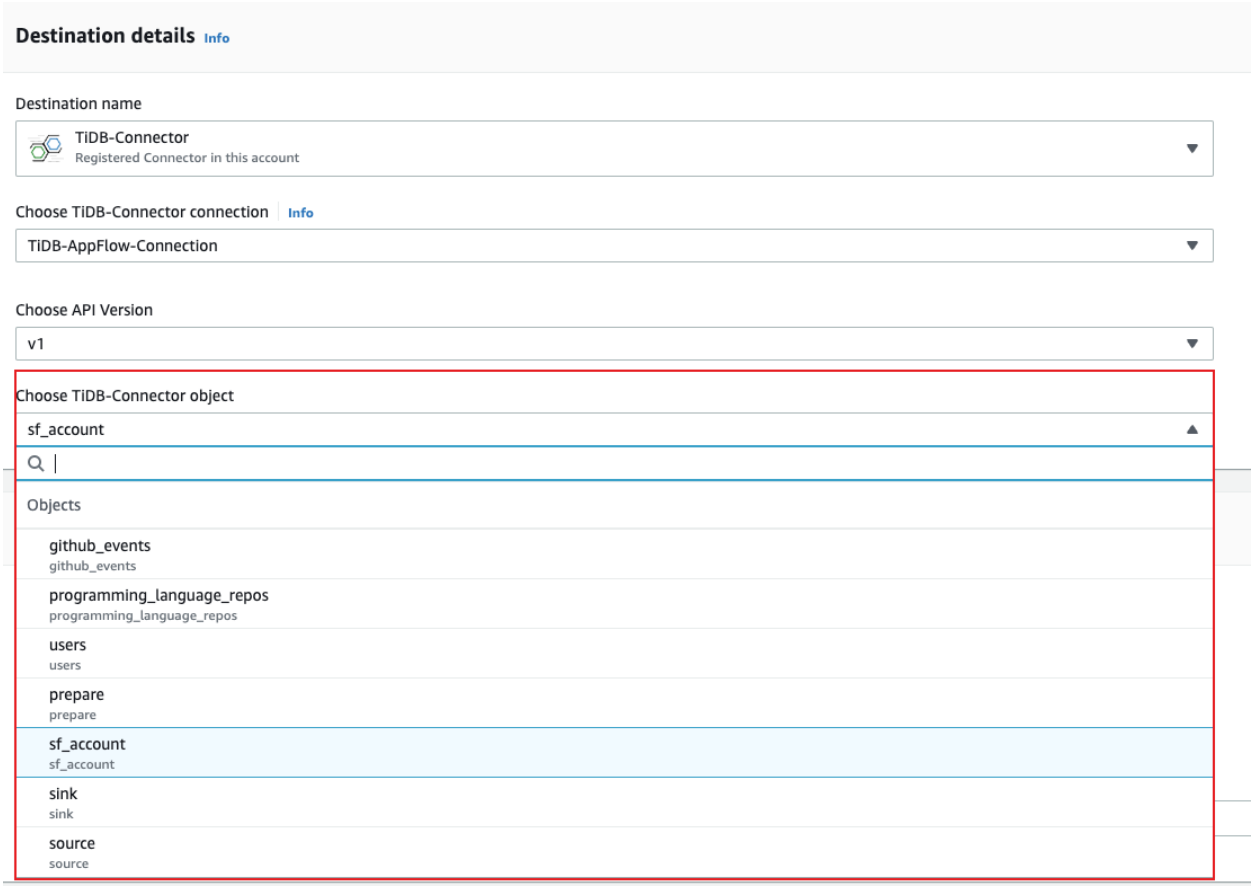


Figure 38: database

The following screenshot shows the configurations to transfer data from the Salesforce **Account** object to the `sf_account` table in TiDB:

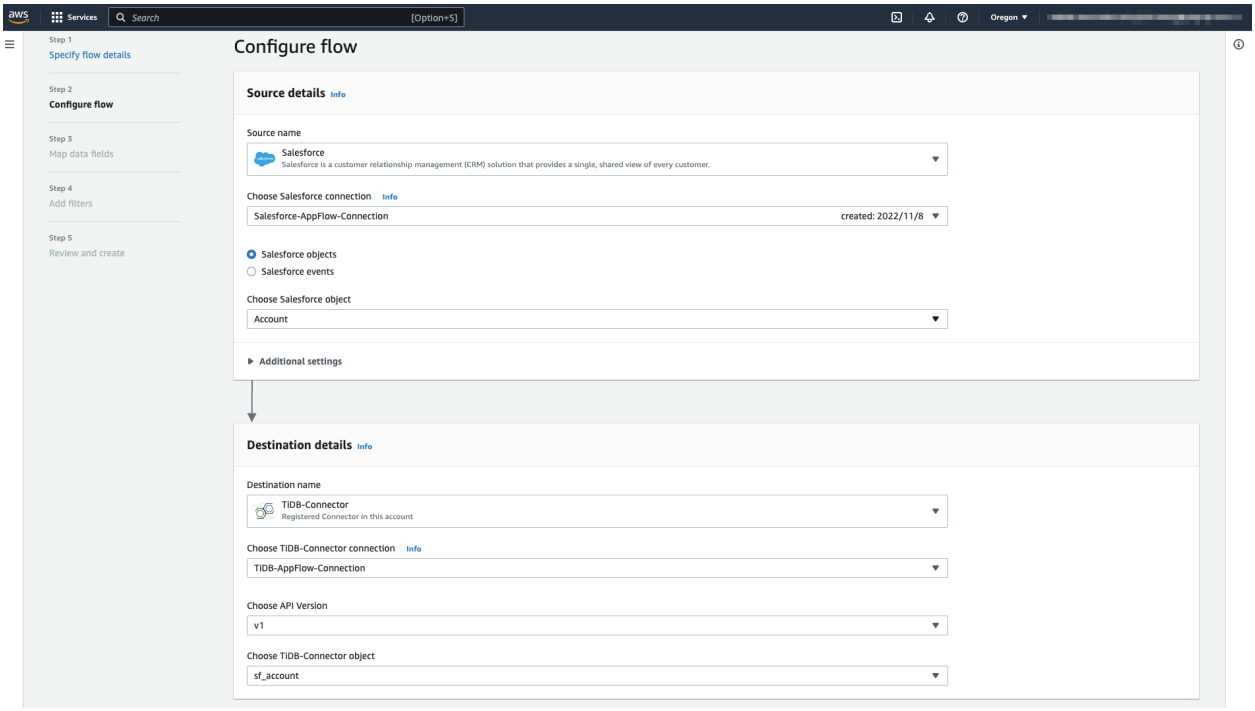


Figure 39: complete flow

8. In the **Error handling** area, choose **Stop the current flow run**. In the **Flow trigger** area, choose the **Run on demand** trigger type, which means you need to run the flow manually. Then, click **Next**.

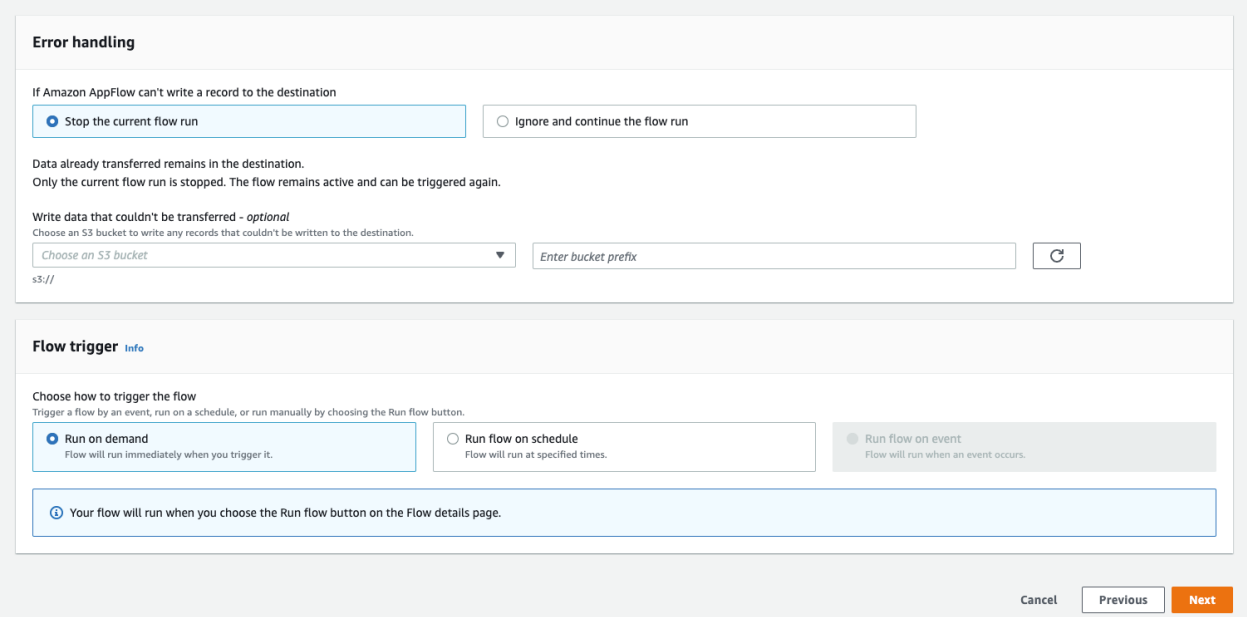


Figure 40: complete step1

4.13.4.3.3 Set mapping rules

Map the fields of the **Account** object in Salesforce to the `sf_account` table in TiDB, and then click **Next**.

- The `sf_account` table is newly created in TiDB and it is empty.

```
test> SELECT * FROM sf_account;
+----+------+------+---------------+--------+----------+
| id | name | type | billing_state | rating | industry |
+----+------+------+---------------+--------+----------+
+----+------+------+---------------+--------+----------+
```

- To set a mapping rule, you can select a source field name on the left, and select a destination field name on the right. Then, click **Map fields**, and a rule is set.

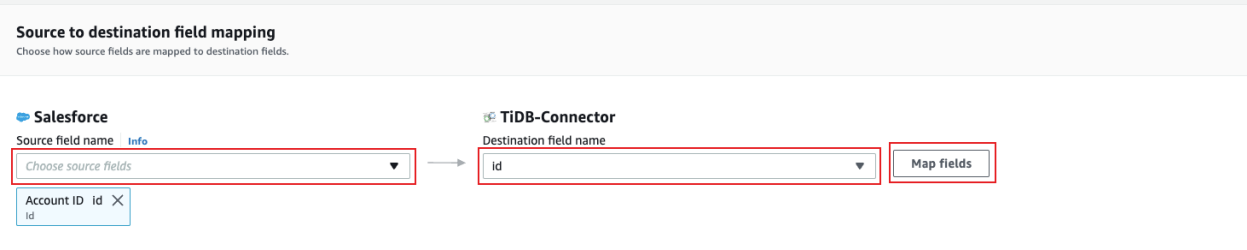


Figure 41: add mapping rule

- The following mapping rules (Source field name -> Destination field name) are needed in this document:

    - Account ID -> id
    - Account Name -> name
    - Account Type -> type
    - Billing State/Province -> billing_state
    - Account Rating -> rating
    - Industry -> industry

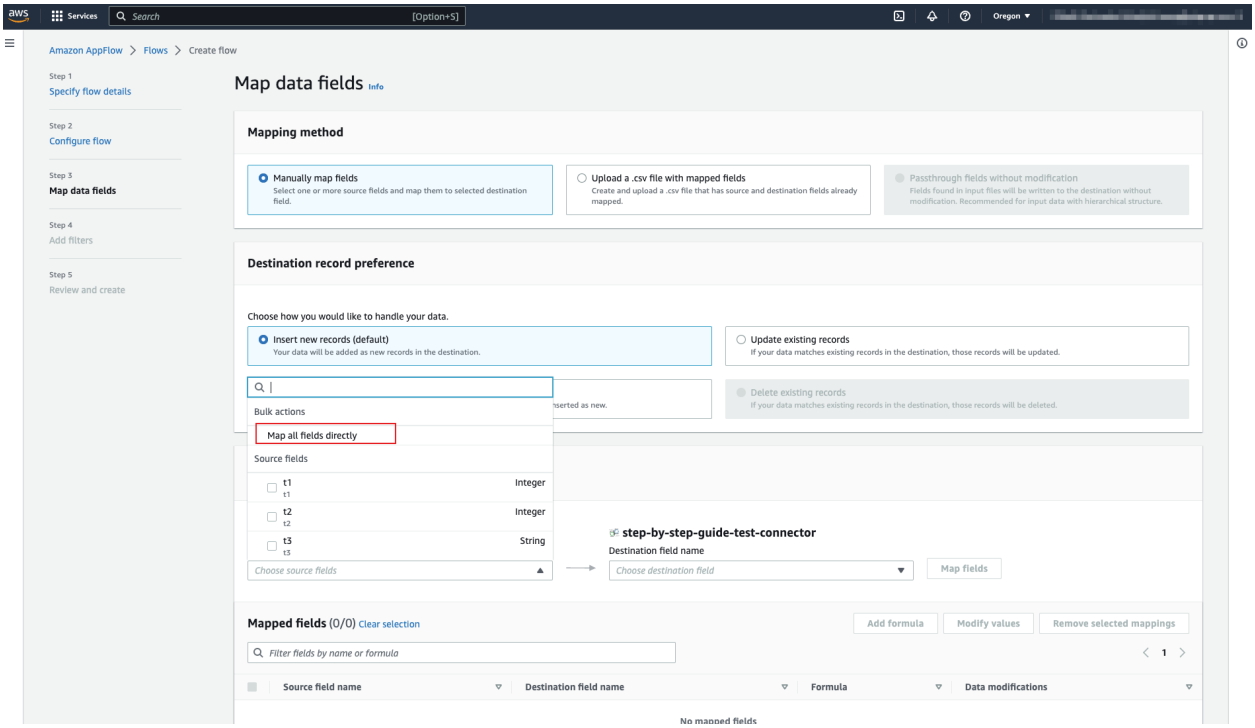


Figure 42: mapping a rule

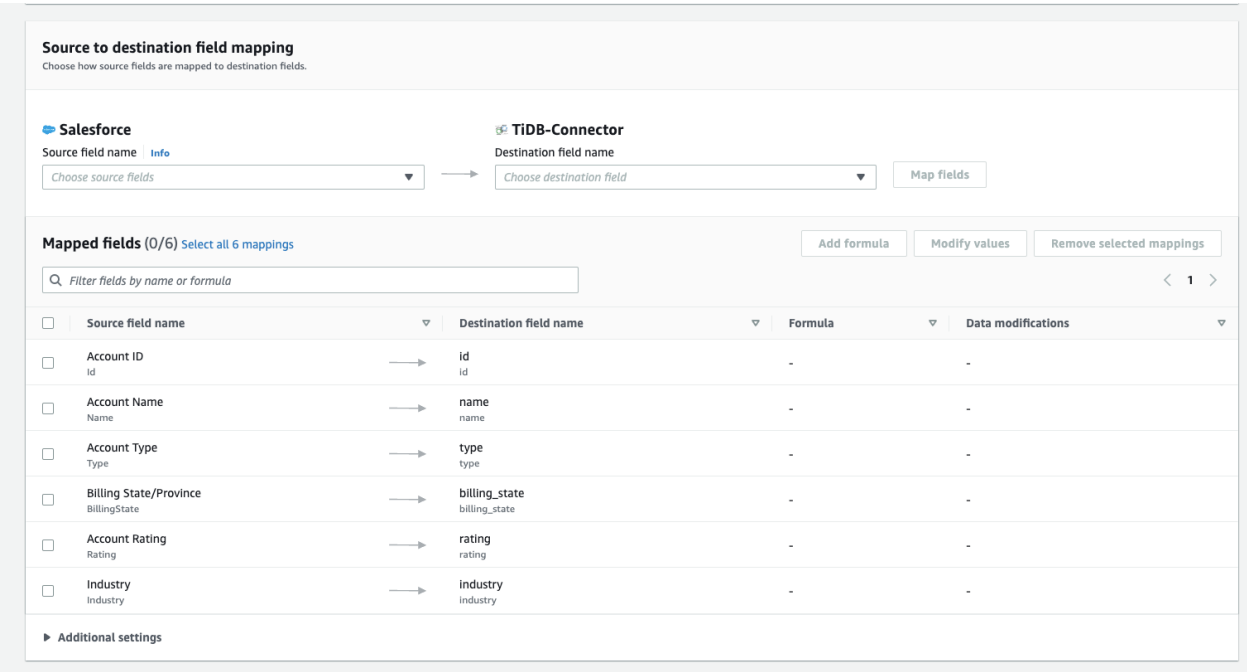


Figure 43: show all mapping rules

4.13.4.3.4 (Optional) Set filters

If you want to add some filters to your data fields, you can set them here. Otherwise, skip this step and click **Next**.

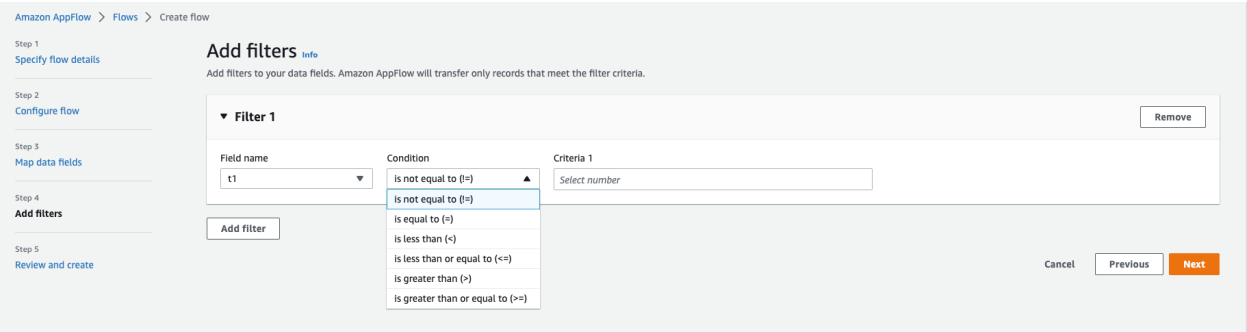


Figure 44: filters

4.13.4.3.5 Confirm and create the flow

Confirm the information of the flow to be created. If everything looks fine, click **Create flow**.

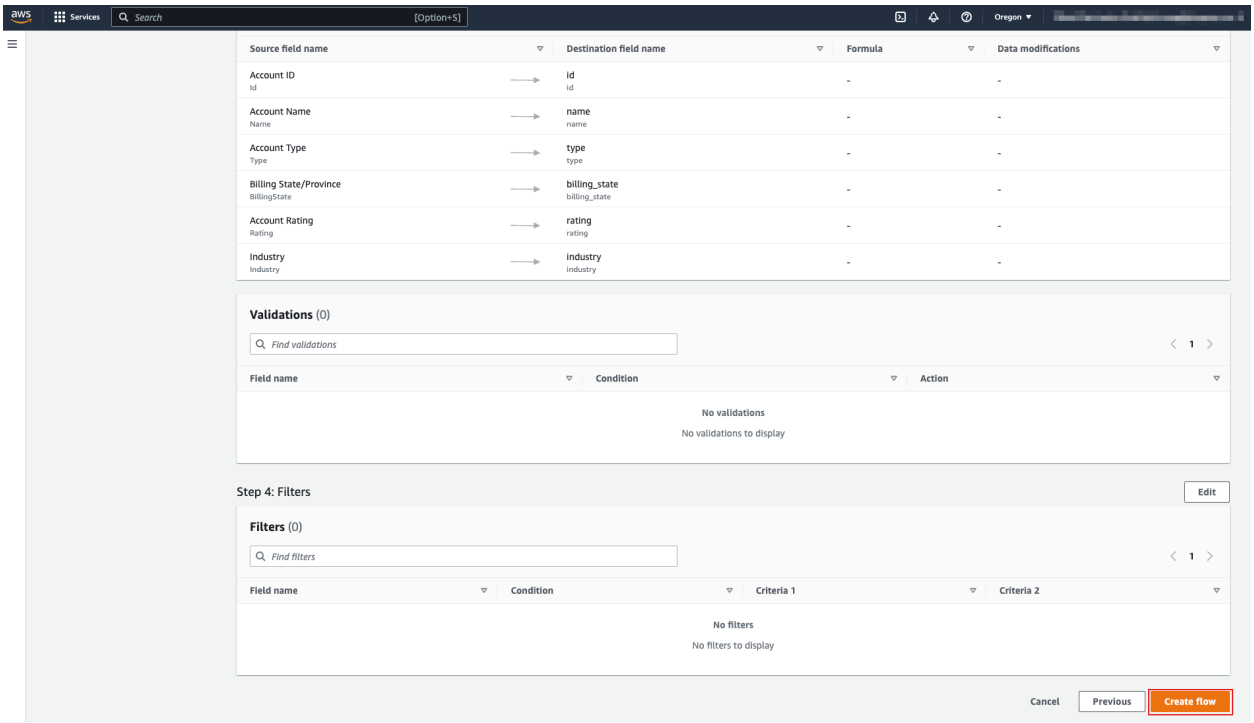


Figure 45: review

4.13.4.4 Step 3. Run the flow

On the page of the newly created flow, click **Run flow** in the upper-right corner.

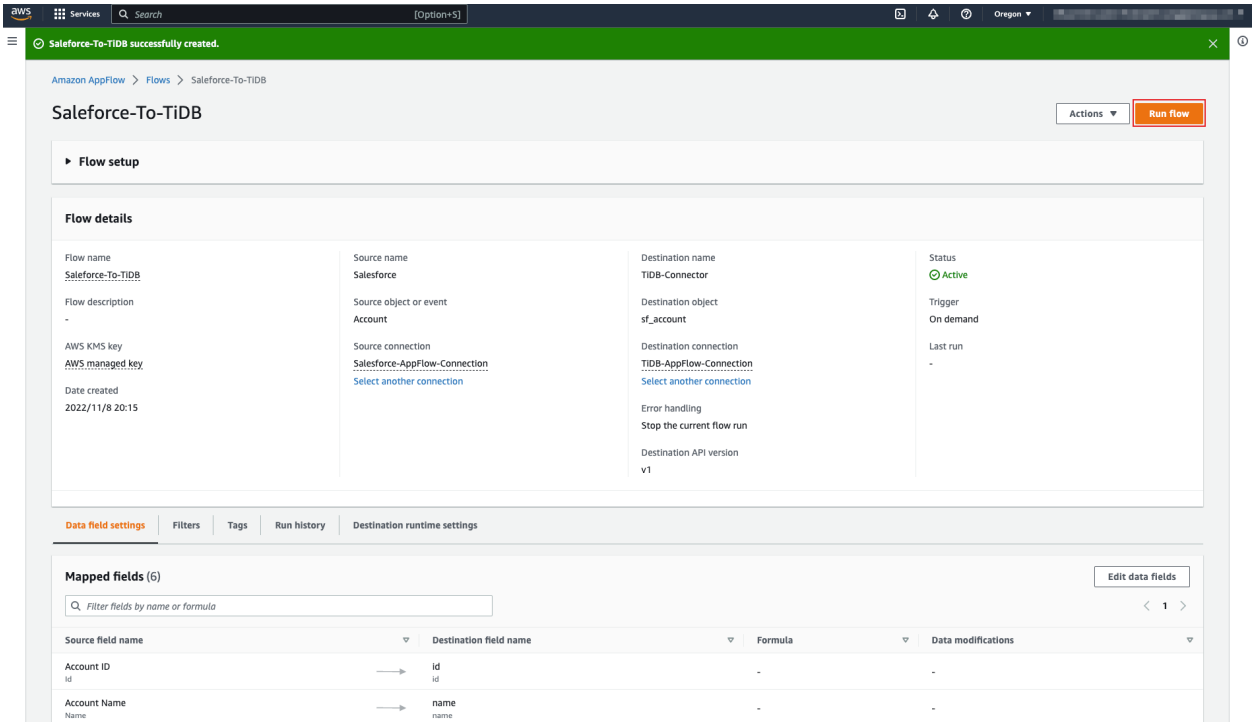


Figure 46: run flow

The following screenshot shows an example that the flow runs successfully:

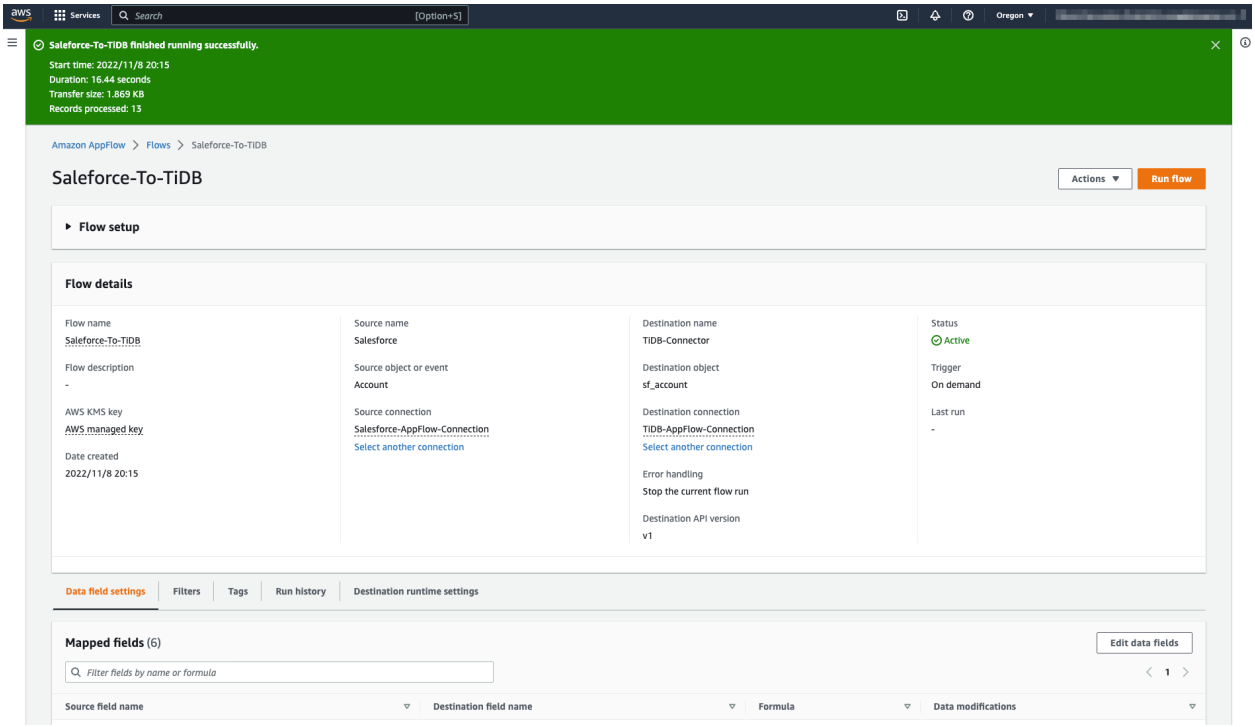


Figure 47: run success

Query the `sf_account` table, and you can see that the records from the Salesforce **Account** object have been written to it:

```
test> SELECT * FROM sf_account;
+--
    ↪ ------------------+----------------------------------------+-------------------+--
    ↪
| id                 | name                                   | type              |
    ↪ billing_state | rating | industry  |
+--
    ↪ ------------------+----------------------------------------+-------------------+--
    ↪
| 001Do000003EDTlIAO | Sample Account for Entitlements | null          | null
    ↪          | null  | null          |
| 001Do000003EDTZIA4 | Edge Communications             | Customer - Direct | TX
    ↪          | Hot   | Electronics   |
| 001Do000003EDTaIAO | Burlington Textiles Corp of America | Customer -
    ↪ Direct | NC | Warm  | Apparel       |
| 001Do000003EDTbIAO | Pyramid Construction Inc.       | Customer - Channel |
    ↪ null      | null  | Construction |
| 001Do000003EDTcIAO | Dickenson plc                   | Customer - Channel |
    ↪ KS        | null  | Consulting    |
```

```
| 001Do000003EDTdIAO | Grand Hotels & Resorts Ltd   | Customer - Direct | IL
   ↪          | Warm  | Hospitality  |
| 001Do000003EDTeIAO | United Oil & Gas Corp.       | Customer - Direct | NY
   ↪          | Hot   | Energy       |
| 001Do000003EDTfIAO | Express Logistics and Transport | Customer - Channel
   ↪  | OR    | Cold  | Transportation |
| 001Do000003EDTgIAO | University of Arizona        | Customer - Direct | AZ
   ↪          | Warm  | Education    |
| 001Do000003EDThIAO | United Oil & Gas, UK         | Customer - Direct | UK
   ↪          | null  | Energy       |
| 001Do000003EDTiIAO | United Oil & Gas, Singapore  | Customer - Direct |
   ↪ Singapore | null  | Energy       |
| 001Do000003EDTjIAO | GenePoint                    | Customer - Channel |
   ↪ CA        | Cold  | Biotechnology |
| 001Do000003EDTkIAO | sForce                       | null              | CA
   ↪          | null  | null         |
+--
   ↪ -----------------+-------------------------------------+-------------------+--
   ↪
```

4.13.4.5 Noteworthy things

- If anything goes wrong, you can navigate to the CloudWatch page on the AWS Management Console to get logs.
- The steps in this document are based on Building custom connectors using the Amazon AppFlow Custom Connector SDK.
- TiDB Serverless is **NOT** a production environment.
- To prevent excessive length, the examples in this document only show the `Insert` strategy, but `Update` and `Upsert` strategies are also tested and can be used.

5 Deploy

5.1 Software and Hardware Recommendations

As an open-source distributed SQL database with high performance, TiDB can be deployed in the Intel architecture server, ARM architecture server, and major virtualization environments and runs well. TiDB supports most of the major hardware networks and Linux operating systems.

5.1.1 OS and platform requirements

Starting from v6.1.1, TiDB provides multi-level support for different quality standards on the combination of operating systems and CPU architectures.

- For the following combinations of operating systems and CPU architectures, TiDB **provides enterprise-level production quality**, and the product features have been comprehensively and systematically verified:

Operating systems	Supported CPU architectures
Red Hat Enterprise Linux 8.4 or a later 8.x version	x86_64 ARM 64
Red Hat Enterprise Linux 7.3 or a later 7.x version CentOS 7.3 or a later 7.x version	x86_64 ARM 64
Amazon Linux 2	x86_64 ARM 64
Kylin V10 SP1/SP2	x86_64 ARM 64
UnionTech OS (UOS) V20	x86_64 ARM 64

> **Note:**
> According to CentOS Linux EOL, the upstream support for CentOS Linux 8 ended on December 31, 2021. CentOS Stream 8 continues to be supported by the CentOS organization.

- For the following combinations of operating systems and CPU architectures, you can compile, build, and deploy TiDB. In addition, you can also use the basic features of OLTP, OLAP, and the data tools. However, TiDB **does not guarantee enterprise-level production quality**:

Operating systems	Supported CPU architectures

| macOS Catalina or later (For v6.1.2 and later versions, only macOS 12 (Monterey) or later is supported) |

x86_64

ARM 64

| | Oracle Enterprise Linux 7.3 or a later 7.x version | x86_64 | | Ubuntu LTS 18.04 or later | x86_64 | | CentOS 8 Stream |

x86_64

ARM 64

| | Debian 9 (Stretch) or later | x86_64 | | Fedora 35 or later | x86_64 | | openSUSE Leap later than v15.3 (not including Tumbleweed) | x86_64 | | SUSE Linux Enterprise Server 15 | x86_64 |

Note:

- For Oracle Enterprise Linux, TiDB supports the Red Hat Compatible Kernel (RHCK) and does not support the Unbreakable Enterprise Kernel provided by Oracle Enterprise Linux.
- Support for Ubuntu 16.04 will be removed in future versions of TiDB. Upgrading to Ubuntu 18.04 or later is strongly recommended.

- If you are using the 32-bit version of an operating system listed in the preceding two tables, TiDB **is not guaranteed** to be compilable, buildable or deployable on the 32-bit operating system and the corresponding CPU architecture, or TiDB does not actively adapt to the 32-bit operating system.
- Other operating system versions not mentioned above might work but are not officially supported.

Linux OS	Version
Red Hat Enterprise Linux	7.3 or later 7.x versions
CentOS	7.3 or later 7.x versions
Oracle Enterprise Linux	7.3 or later 7.x versions
Amazon Linux	2
Ubuntu LTS	16.04 or later

Note:

- For Oracle Enterprise Linux, TiDB supports the Red Hat Compatible Kernel (RHCK) and does not support the Unbreakable Enterprise Kernel provided by Oracle Enterprise Linux.
- A large number of TiDB tests have been run on the CentOS 7.3 system, and in our community there are a lot of best practices in which TiDB is deployed on the Linux operating system. Therefore, it is recommended to deploy TiDB on CentOS 7.3 or later.
- The support for the Linux operating systems above includes the deployment and operation in physical servers as well as in major virtualized environments like VMware, KVM and XEN.
- Red Hat Enterprise Linux 8.0, CentOS 8 Stream, and Oracle Enterprise Linux 8.0 are not supported yet as the testing of these platforms is in progress.
- According to CentOS Linux EOL, the upstream support for CentOS ended on December 31, 2021.
- Support for Ubuntu 16.04 will be removed in future versions of TiDB. Upgrading to Ubuntu 18.04 or later is strongly recommended.

Other Linux OS versions such as Debian Linux and Fedora Linux might work but are not officially supported.

5.1.1.1 Libraries required for compiling and running TiDB

Libraries required for compiling and building TiDB	Version
Golang	For TiDB v6.1.0, v6.1.1, and v6.1.2: 1.18 or later For TiDB v6.1.3: 1.19.3 or later
Rust	nightly-2022-07-31 or later
GCC	7.x
LLVM	13.0 or later

Library required for running TiDB: glibc (2.28-151.el8 version)

5.1.2 Software recommendations

5.1.2.1 Control machine

Software	Version
sshpass	1.06 or later
TiUP	1.5.0 or later

Note:

It is required that you deploy TiUP on the control machine to operate and manage TiDB clusters.

5.1.2.2 Target machines

Software	Version
sshpass	1.06 or later
numa	2.0.12 or later
tar	any

5.1.3 Server recommendations

You can deploy and run TiDB on the 64-bit generic hardware server platform in the Intel x86-64 architecture or on the hardware server platform in the ARM architecture. The requirements and recommendations about server hardware configuration (ignoring the resources occupied by the operating system itself) for development, test, and production environments are as follows:

5.1.3.1 Development and test environments

Component	CPU	Memory	Local Storage	Network	Instance Number (Minimum Requirement)
TiDB	8 core+	16 GB+	No special requirements	Gigabit network card	1 (can be deployed on the same machine with PD)
PD	4 core+	8 GB+	SAS, 200 GB+	Gigabit network card	1 (can be deployed on the same machine with TiDB)

Component	CPU	Memory	Local Storage	Network	Instance Number (Minimum Requirement)
TiKV	8 core+	32 GB+	SAS, 200 GB+	Gigabit network card	3
TiFlash	32 core+	64 GB+	SSD, 200 GB+	Gigabit network card	1
TiCDC	8 core+	16 GB+	SAS, 200 GB+	Gigabit network card	1

Note:

- In the test environment, the TiDB and PD instances can be deployed on the same server.
- For performance-related test, do not use low-performance storage and network hardware configuration, in order to guarantee the correctness of the test result.
- For the TiKV server, it is recommended to use NVMe SSDs to ensure faster reads and writes.
- If you only want to test and verify the features, follow Quick Start Guide for TiDB to deploy TiDB on a single machine.
- The TiDB server uses the disk to store server logs, so there are no special requirements for the disk type and capacity in the test environment.

5.1.3.2 Production environment

Component	CPU	Memory	Hard Disk Type	Network	Instance N
TiDB	16 core+	48 GB+	SAS	10 Gigabit network card (2 preferred)	
PD	8 core+	16 GB+	SSD	10 Gigabit network card (2 preferred)	
TiKV	16 core+	64 GB+	SSD	10 Gigabit network card (2 preferred)	
TiFlash	48 core+	128 GB+	1 or more SSDs	10 Gigabit network card (2 preferred)	
TiCDC	16 core+	64 GB+	SSD	10 Gigabit network card (2 preferred)	
Monitor	8 core+	16 GB+	SAS	Gigabit network card	

Note:

- In the production environment, the TiDB and PD instances can be deployed on the same server. If you have a higher requirement for performance and reliability, try to deploy them separately.
- It is strongly recommended to use higher configuration in the production environment.
- It is recommended to keep the size of TiKV hard disk within 2 TB if you are using PCIe SSDs or within 1.5 TB if you are using regular SSDs.

Before you deploy TiFlash, note the following items:

- TiFlash can be deployed on multiple disks.
- It is recommended to use a high-performance SSD as the first disk of the TiFlash data directory to buffer the real-time replication of TiKV data. The performance of this disk should not be lower than that of TiKV, such as PCI-E SSD. The disk capacity should be no less than 10% of the total capacity; otherwise, it might become the bottleneck of this node. You can deploy ordinary SSDs for other disks, but note that a better PCI-E SSD brings better performance.
- It is recommended to deploy TiFlash on different nodes from TiKV. If you must deploy TiFlash and TiKV on the same node, increase the number of CPU cores and memory, and try to deploy TiFlash and TiKV on different disks to avoid interfering each other.
- The total capacity of the TiFlash disks is calculated in this way: `the data volume of the entire TiKV cluster to be replicated / the number of TiKV replicas * the number of TiFlash replicas`. For example, if the overall planned capacity of TiKV is 1 TB, the number of TiKV replicas is 3, and the number of TiFlash replicas is 2, then the recommended total capacity of TiFlash is `1024 GB / 3 * 2`. You can replicate only the data of some tables. In such case, determine the TiFlash capacity according to the data volume of the tables to be replicated.

Before you deploy TiCDC, note that it is recommended to deploy TiCDC on PCIe-SSD disks larger than 1 TB.

5.1.4 Network requirements

As an open-source distributed SQL database, TiDB requires the following network port configuration to run. Based on the TiDB deployment in actual environments, the administrator can open relevant ports in the network side and host side.

Component	Default Port	Description
TiDB	4000	the communication port for the application and DBA tools
TiDB	10080	the communication port to report TiDB status
TiKV	20160	the TiKV communication port

Component	Default Port	Description
TiKV	20180	the communication port to report TiKV status
PD	2379	the communication port between TiDB and PD
PD	2380	the internode communication port within the PD cluster

Component	Default Port	Description
TiFlash	9000	the TiFlash TCP service port
TiFlash	8123	the TiFlash HTTP service port
TiFlash	3930	the TiFlash RAFT and Coprocessor service port
TiFlash	20170	the TiFlash Proxy service port

Component	Default Port	Description
TiFlash	20292	the port for Prometheus to pull TiFlash Proxy metrics
TiFlash	8234	the port for Prometheus to pull TiFlash metrics
Pump	8250	the Pump communication port
Drainer	8249	the Drainer communication port
TiCDC	8300	the TiCDC communication port

Component	Default Port	Description
Monitoring	9090	the communication port for the Prometheus service
Monitoring	12020	the communication port for the Ng-Monitoring service

Component	Default Port	Description
Node_exporter	9100	the communication port to report the system information of every TiDB cluster node
Blackbox_exporter	9115	the Blackbox_exporter communication port, used to monitor the ports in the TiDB cluster

Component	Default Port	Description
Grafana	3000	the port for the external Web monitoring service and client (Browser) access
Alertmanager	9093	the port for the alert web service
Alertmanager	9094	the alert communication port

5.1.5 Disk space requirements

Component	Disk space requirement	Healthy disk usage
TiDB	At least 30 GB for the log disk	Lower than 90%
PD	At least 20 GB for the data disk and for the log disk, respectively	Lower than 90%

Component	Disk space requirement	Healthy disk usage
TiKV	At least 100 GB for the data disk and for the log disk, respectively	Lower than 80%
TiFlash	At least 100 GB for the data disk and at least 30 GB for the log disk, respectively	Lower than 80%

| TiUP |

Control machine: No more than 1 GB space is required for deploying a TiDB cluster of a single version. The space required increases if TiDB clusters of multiple versions are deployed.

Deployment servers (machines where the TiDB components run): TiFlash occupies about 700 MB space and other components (such as PD, TiDB, and TiKV) occupy about 200 MB space respectively. During the cluster deployment process, the TiUP cluster requires less than 1 MB of temporary space (`/tmp` directory) to store temporary files.

| N/A | | Ngmonitoring |

Conprof: 3 x 1 GB x Number of components (each component occupies about 1 GB per day, 3 days in total) + 20 GB reserved space

Top SQL: 30 x 50 MB x Number of components (each component occupies about 50 MB per day, 30 days in total)

Conprof and Top SQL share the reserved space

| N/A |

5.1.6 Web browser requirements

TiDB relies on Grafana to provide visualization of database metrics. A recent version of Internet Explorer, Chrome or Firefox with Javascript enabled is sufficient.

5.2 TiDB Environment and System Configuration Check

This document describes the environment check operations before deploying TiDB. The following steps are ordered by priorities.

5.2.1 Mount the data disk ext4 filesystem with options on the target machines that deploy TiKV

For production deployments, it is recommended to use NVMe SSD of EXT4 filesystem to store TiKV data. This configuration is the best practice, whose reliability, security, and stability have been proven in a large number of online scenarios.

Log in to the target machines using the `root` user account.

Format your data disks to the ext4 filesystem and add the `nodelalloc` and `noatime` mount options to the filesystem. It is required to add the `nodelalloc` option, or else the TiUP deployment cannot pass the precheck. The `noatime` option is optional.

Note:

If your data disks have been formatted to ext4 and have added the mount options, you can uninstall it by running the `umount /dev/nvme0n1p1` command, skip directly to the fifth step below to edit the `/etc/fstab` file, and add the options again to the filesystem.

Take the `/dev/nvme0n1` data disk as an example:

1. View the data disk.

```
fdisk -l
```

```
Disk /dev/nvme0n1: 1000 GB
```

2. Create the partition.

```
parted -s -a optimal /dev/nvme0n1 mklabel gpt -- mkpart primary ext4 1
    ↪ -1
```

Note:

Use the `lsblk` command to view the device number of the partition: for a NVMe disk, the generated device number is usually `nvme0n1p1`; for a regular disk (for example, `/dev/sdb`), the generated device number is usually `sdb1`.

3. Format the data disk to the ext4 filesystem.

```
mkfs.ext4 /dev/nvme0n1p1
```

4. View the partition UUID of the data disk.

   In this example, the UUID of nvme0n1p1 is `c51eb23b-195c-4061-92a9-3
   ↪ fad812cc12f`.

```
lsblk -f
```

```
NAME   FSTYPE LABEL UUID                                 MOUNTPOINT
sda
 ─sda1 ext4        237b634b-a565-477b-8371-6dff0c41f5ab /boot
 ─sda2 swap        f414c5c0-f823-4bb1-8fdf-e531173a72ed
 ─sda3 ext4        547909c1-398d-4696-94c6-03e43e317b60 /
sr0
nvme0n1
 ─nvme0n1p1 ext4       c51eb23b-195c-4061-92a9-3fad812cc12f
```

5. Edit the `/etc/fstab` file and add the `nodelalloc` mount options.

```
vi /etc/fstab
```

```
UUID=c51eb23b-195c-4061-92a9-3fad812cc12f /data1 ext4 defaults,
    ↪ nodelalloc,noatime 0 2
```

6. Mount the data disk.

```
mkdir /data1 && \
mount -a
```

7. Check using the following command.

```
mount -t ext4
```

```
/dev/nvme0n1p1 on /data1 type ext4 (rw,noatime,nodelalloc,data=ordered)
```

If the filesystem is ext4 and `nodelalloc` is included in the mount options, you have successfully mount the data disk ext4 filesystem with options on the target machines.

5.2.2 Check and disable system swap

TiDB needs sufficient memory space for operation. When memory is insufficient, using swap as a buffer might degrade performance. Therefore, it is recommended to disable the system swap permanently by executing the following commands:

```
echo "vm.swappiness = 0">> /etc/sysctl.conf
swapoff -a && swapon -a
sysctl -p
```

Note:

- Executing `swapoff -a` and then `swapon -a` is to refresh swap by dumping data to memory and cleaning up swap. If you drop the swappiness change and execute only `swapoff -a`, swap will be enabled again after you restart the system.
- `sysctl -p` is to make the configuration effective without restarting the system.

5.2.3 Check and stop the firewall service of target machines

In TiDB clusters, the access ports between nodes must be open to ensure the transmission of information such as read and write requests and data heartbeats. In common online scenarios, the data interaction between the database and the application service and between the database nodes are all made within a secure network. Therefore, if there are no special security requirements, it is recommended to stop the firewall of the target machine. Otherwise, refer to the port usage and add the needed port information to the allowlist of the firewall service.

The rest of this section describes how to stop the firewall service of a target machine.

1. Check the firewall status. Take CentOS Linux release 7.7.1908 (Core) as an example.

```
sudo firewall-cmd --state
sudo systemctl status firewalld.service
```

2. Stop the firewall service.

```
sudo systemctl stop firewalld.service
```

3. Disable automatic start of the firewall service.

```
sudo systemctl disable firewalld.service
```

4. Check the firewall status.

```
sudo systemctl status firewalld.service
```

5.2.4 Check and install the NTP service

TiDB is a distributed database system that requires clock synchronization between nodes to guarantee linear consistency of transactions in the ACID model.

At present, the common solution to clock synchronization is to use the Network Time Protocol (NTP) services. You can use the `pool.ntp.org` timing service on the Internet, or build your own NTP service in an offline environment.

To check whether the NTP service is installed and whether it synchronizes with the NTP server normally, take the following steps:

1. Run the following command. If it returns `running`, then the NTP service is running.

```
sudo systemctl status ntpd.service
```

```
ntpd.service - Network Time Service
Loaded: loaded (/usr/lib/systemd/system/ntpd.service; disabled; vendor
    ↪ preset: disabled)
Active: active (running) since — 2017-12-18 13:13:19 CST; 3s ago
```

- If it returns `Unit ntpd.service could not be found.`, then try the following command to see whether your system is configured to use `chronyd` instead of `ntpd` to perform clock synchronization with NTP:

```
sudo systemctl status chronyd.service
```

```
chronyd.service - NTP client/server
Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled;
    ↪ vendor preset: enabled)
Active: active (running) since Mon 2021-04-05 09:55:29 EDT; 3 days
    ↪ ago
```

  If the result shows that neither `chronyd` nor `ntpd` is configured, it means that neither of them is installed in your system. You should first install `chronyd` or `ntpd` and ensure that it can be automatically started. By default, `ntpd` is used.

  If your system is configured to use `chronyd`, proceed to step 3.

2. Run the `ntpstat` command to check whether the NTP service synchronizes with the NTP server.

> **Note:**
> For the Ubuntu system, you need to install the `ntpstat` package.

```
ntpstat
```

- If it returns `synchronised to NTP server` (synchronizing with the NTP server), then the synchronization process is normal.

```
synchronised to NTP server (85.199.214.101) at stratum 2
time correct to within 91 ms
polling server every 1024 s
```

- The following situation indicates the NTP service is not synchronizing normally:

```
unsynchronised
```

- The following situation indicates the NTP service is not running normally:

```
Unable to talk to NTP daemon. Is it running?
```

3. Run the `chronyc tracking` command to check wheter the Chrony service synchronizes with the NTP server.

> **Note:**
> This only applies to systems that use Chrony instead of NTPd.

```
chronyc tracking
```

- If the command returns `Leap status     : Normal`, the synchronization process is normal.

```
Reference ID    : 5EC69F0A (ntp1.time.nl)
Stratum         : 2
Ref time (UTC)  : Thu May 20 15:19:08 2021
System time     : 0.000022151 seconds slow of NTP time
Last offset     : -0.000041040 seconds
RMS offset      : 0.000053422 seconds
Frequency       : 2.286 ppm slow
Residual freq   : -0.000 ppm
Skew            : 0.012 ppm
Root delay      : 0.012706812 seconds
Root dispersion : 0.000430042 seconds
Update interval : 1029.8 seconds
Leap status     : Normal
```

- If the command returns the following result, an error occurs in the synchronization:

```
Leap status     : Not synchronised
```

- If the command returns the following result, the `chronyd` service is not running normally:

```
506 Cannot talk to daemon
```

To make the NTP service start synchronizing as soon as possible, run the following command. Replace `pool.ntp.org` with your NTP server.

```
sudo systemctl stop ntpd.service && \
sudo ntpdate pool.ntp.org && \
sudo systemctl start ntpd.service
```

To install the NTP service manually on the CentOS 7 system, run the following command:

```
sudo yum install ntp ntpdate && \
sudo systemctl start ntpd.service && \
sudo systemctl enable ntpd.service
```

5.2.5 Check and configure the optimal parameters of the operating system

For TiDB in the production environment, it is recommended to optimize the operating system configuration in the following ways:

1. Disable THP (Transparent Huge Pages). The memory access pattern of databases tends to be sparse rather than consecutive. If the high-level memory fragmentation is serious, higher latency will occur when THP pages are allocated.
2. Set the I/O Scheduler of the storage media to `noop`. For the high-speed SSD storage media, the kernel's I/O scheduling operations can cause performance loss. After the Scheduler is set to `noop`, the performance is better because the kernel directly sends I/O requests to the hardware without other operations. Also, the noop Scheduler is better applicable.
3. Choose the `performance` mode for the cpufrequ module which controls the CPU frequency. The performance is maximized when the CPU frequency is fixed at its highest supported operating frequency without dynamic adjustment.

Take the following steps to check the current operating system configuration and configure optimal parameters:

1. Execute the following command to see whether THP is enabled or disabled:

```
cat /sys/kernel/mm/transparent_hugepage/enabled
```

```
[always] madvise never
```

> **Note:**
>
> If `[always] madvise never` is output, THP is enabled. You need to disable it.

2. Execute the following command to see the I/O Scheduler of the disk where the data directory is located. Assume that you create data directories on both sdb and sdc disks:

```
cat /sys/block/sd[bc]/queue/scheduler
```

```
noop [deadline] cfq
noop [deadline] cfq
```

> **Note:**
> If `noop [deadline] cfq` is output, the I/O Scheduler for the disk is in the `deadline` mode. You need to change it to `noop`.

3. Execute the following command to see the `ID_SERIAL` of the disk:

```
udevadm info --name=/dev/sdb | grep ID_SERIAL
```

```
E: ID_SERIAL=36d0946606d79f90025f3e09a0c1f9e81
E: ID_SERIAL_SHORT=6d0946606d79f90025f3e09a0c1f9e81
```

> **Note:**
> If multiple disks are allocated with data directories, you need to execute the above command several times to record the `ID_SERIAL` of each disk.

4. Execute the following command to see the power policy of the cpufreq module:

```
cpupower frequency-info --policy
```

```
analyzing CPU 0:
current policy: frequency should be within 1.20 GHz and 3.10 GHz.
            The governor "powersave" may decide which speed to use
               ↪ within this range.
```

> **Note:**
> If `The governor "powersave"` is output, the power policy of the cpufreq module is `powersave`. You need to modify it to `performance`. If you use a virtual machine or a cloud host, the output is usually `Unable to ↪ determine current policy`, and you do not need to change anything.

5. Configure optimal parameters of the operating system:

    - Method one: Use tuned (Recommended)

1. Execute the `tuned-adm list` command to see the tuned profile of the current operating system:

```
tuned-adm list
```

```
Available profiles:
- balanced                  - General non-specialized tuned
    ↪ profile
- desktop                   - Optimize for the desktop use-case
- hpc-compute               - Optimize for HPC compute workloads
- latency-performance       - Optimize for deterministic
    ↪ performance at the cost of increased power consumption
- network-latency           - Optimize for deterministic
    ↪ performance at the cost of increased power consumption,
    ↪ focused on low latency network performance
- network-throughput        - Optimize for streaming network
    ↪ throughput, generally only necessary on older CPUs or 40G
    ↪ + networks
- powersave                 - Optimize for low power consumption
- throughput-performance    - Broadly applicable tuning that
    ↪ provides excellent performance across a variety of common
    ↪  server workloads
- virtual-guest             - Optimize for running inside a
    ↪ virtual guest
- virtual-host              - Optimize for running KVM guests
Current active profile: balanced
```

   The output `Current active profile: balanced` means that the tuned profile of the current operating system is `balanced`. It is recommended to optimize the configuration of the operating system based on the current profile.

2. Create a new tuned profile:

```
mkdir /etc/tuned/balanced-tidb-optimal/
vi /etc/tuned/balanced-tidb-optimal/tuned.conf
```

```
[main]
include=balanced

[cpu]
governor=performance

[vm]
transparent_hugepages=never

[disk]
devices_udev_regex=(ID_SERIAL=36d0946606d79f90025f3e09a0c1fc035
    ↪ )|(ID_SERIAL=36d0946606d79f90025f3e09a0c1f9e81)
```

```
elevator=noop
```

The output `include=balanced` means to add the optimization configuration of the operating system to the current `balanced` profile.

3. Apply the new tuned profile:

```
tuned-adm profile balanced-tidb-optimal
```

- Method two: Configure using scripts. Skip this method if you already use method one.

1. Execute the `grubby` command to see the default kernel version:

> **Note:**
> Install the `grubby` package first before you execute `grubby`.

```
grubby --default-kernel
```

```
/boot/vmlinuz-3.10.0-957.el7.x86_64
```

2. Execute `grubby --update-kernel` to modify the kernel configuration:

```
grubby --args="transparent_hugepage=never" --update-kernel /
    ↪ boot/vmlinuz-3.10.0-957.el7.x86_64
```

> **Note:**
> `--update-kernel` is followed by the actual default kernel version.

3. Execute `grubby --info` to see the modified default kernel configuration:

```
grubby --info /boot/vmlinuz-3.10.0-957.el7.x86_64
```

> **Note:**
> `--info` is followed by the actual default kernel version.

```
index=0
kernel=/boot/vmlinuz-3.10.0-957.el7.x86_64
args="ro crashkernel=auto rd.lvm.lv=centos/root rd.lvm.lv=
    ↪ centos/swap rhgb quiet LANG=en_US.UTF-8
    ↪ transparent_hugepage=never"
root=/dev/mapper/centos-root
initrd=/boot/initramfs-3.10.0-957.el7.x86_64.img
title=CentOS Linux (3.10.0-957.el7.x86_64) 7 (Core)
```

4. Modify the current kernel configuration to immediately disable THP:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

5. Configure the I/O Scheduler in the udev script:

```
vi /etc/udev/rules.d/60-tidb-schedulers.rules
```

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{ID_SERIAL}=="36
    ↪ d0946606d79f90025f3e09a0c1fc035", ATTR{queue/scheduler}="
    ↪ noop"
ACTION=="add|change", SUBSYSTEM=="block", ENV{ID_SERIAL}=="36
    ↪ d0946606d79f90025f3e09a0c1f9e81", ATTR{queue/scheduler}="
    ↪ noop"
```

6. Apply the udev script:

```
udevadm control --reload-rules
udevadm trigger --type=devices --action=change
```

7. Create a service to configure the CPU power policy:

```
cat >> /etc/systemd/system/cpupower.service << EOF
[Unit]
Description=CPU performance
[Service]
Type=oneshot
ExecStart=/usr/bin/cpupower frequency-set --governor
    ↪ performance
[Install]
WantedBy=multi-user.target
EOF
```

8. Apply the CPU power policy configuration service:

```
systemctl daemon-reload
systemctl enable cpupower.service
systemctl start cpupower.service
```

6. Execute the following command to verify the THP status:

```
cat /sys/kernel/mm/transparent_hugepage/enabled
```

```
always madvise [never]
```

7. Execute the following command to verify the I/O Scheduler of the disk where the data directory is located:

```
cat /sys/block/sd[bc]/queue/scheduler
```

```
[noop] deadline cfq
[noop] deadline cfq
```

8. Execute the following command to see the power policy of the cpufreq module:

```
cpupower frequency-info --policy
  ```
```

analyzing CPU 0: current policy: frequency should be within 1.20 GHz and 3.10 GHz. The governor "performance" may decide which speed to use within this range. "'

9. Execute the following commands to modify the `sysctl` parameters:

```
echo "fs.file-max = 1000000">> /etc/sysctl.conf
echo "net.core.somaxconn = 32768">> /etc/sysctl.conf
echo "net.ipv4.tcp_tw_recycle = 0">> /etc/sysctl.conf
echo "net.ipv4.tcp_syncookies = 0">> /etc/sysctl.conf
echo "vm.overcommit_memory = 1">> /etc/sysctl.conf
echo "vm.min_free_kbytes = 1048576">> /etc/sysctl.conf
sysctl -p
```

Note:

- `vm.min_free_kbytes` is a Linux kernel parameter that controls the minimum amount of free memory reserved by the system, measured in KiB.
- The setting of `vm.min_free_kbytes` affects the memory reclaim mechanism. Setting it too large reduces the available memory, while setting it too small might cause memory request speeds to exceed background reclaim speeds, leading to memory reclamation and consequent delays in memory allocation.
- It is recommended to set `vm.min_free_kbytes` to `1048576` KiB (1 GiB) at least. If NUMA is installed, it is recommended to set it to `number of NUMA nodes * 1048576` KiB.
- For servers with memory sizes less than 16 GiB, it is recommended to keep the default value of `vm.min_free_kbytes` unchanged.

10. Execute the following command to configure the user's `limits.conf` file:

```
cat << EOF >>/etc/security/limits.conf
tidb           soft    nofile          1000000
tidb           hard    nofile          1000000
```
```

```
tidb          soft    stack         32768
tidb          hard    stack         32768
EOF
```

5.2.6 Manually configure the SSH mutual trust and sudo without password

This section describes how to manually configure the SSH mutual trust and sudo without password. It is recommended to use TiUP for deployment, which automatically configure SSH mutual trust and login without password. If you deploy TiDB clusters using TiUP, ignore this section.

1. Log in to the target machine respectively using the `root` user account, create the `tidb` user and set the login password.

```
useradd tidb && \
passwd tidb
```

2. To configure sudo without password, run the following command, and add `tidb ALL =(ALL)NOPASSWD: ALL` to the end of the file:

```
visudo
```

```
tidb ALL=(ALL) NOPASSWD: ALL
```

3. Use the `tidb` user to log in to the control machine, and run the following command. Replace `10.0.1.1` with the IP of your target machine, and enter the `tidb` user password of the target machine as prompted. After the command is executed, SSH mutual trust is already created. This applies to other machines as well. Newly created `tidb` users do not have the `.ssh` directory. To create such a directory, execute the command that generates the RSA key. To deploy TiDB components on the control machine, configure mutual trust for the control machine and the control machine itself.

```
ssh-keygen -t rsa
ssh-copy-id -i ~/.ssh/id_rsa.pub 10.0.1.1
```

4. Log in to the control machine using the `tidb` user account, and log in to the IP of the target machine using `ssh`. If you do not need to enter the password and can successfully log in, then the SSH mutual trust is successfully configured.

```
ssh 10.0.1.1
```

```
[tidb@10.0.1.1 ~]$
```

5. After you log in to the target machine using the `tidb` user, run the following command. If you do not need to enter the password and can switch to the `root` user, then sudo without password of the `tidb` user is successfully configured.

```
sudo -su root
```

```
[root@10.0.1.1 tidb]#
```

5.2.7 Install the `numactl` tool

This section describes how to install the NUMA tool. In online environments, because the hardware configuration is usually higher than required, to better plan the hardware resources, multiple instances of TiDB or TiKV can be deployed on a single machine. In such scenarios, you can use NUMA tools to prevent the competition for CPU resources which might cause reduced performance.

> **Note:**
>
> - Binding cores using NUMA is a method to isolate CPU resources and is suitable for deploying multiple instances on highly configured physical machines.
> - After completing deployment using `tiup cluster deploy`, you can use the `exec` command to perform cluster level management operations.

To install the NUMA tool, take either of the following two methods:

Method 1: Log in to the target node to install NUMA. Take CentOS Linux release 7.7.1908 (Core) as an example.

```
sudo yum -y install numactl
```

Method 2: Install NUMA on an existing cluster in batches by running the `tiup cluster exec` command.

1. Follow Deploy a TiDB Cluster Using TiUP to deploy a cluster `tidb-test`. If you have installed a TiDB cluster, you can skip this step.

```
tiup cluster deploy tidb-test v6.1.7 ./topology.yaml --user root [-p]
    ↪ [-i /home/root/.ssh/gcp_rsa]
```

2. Run the `tiup cluster exec` command using the `sudo` privilege to install NUMA on all the target machines in the `tidb-test` cluster:

```
tiup cluster exec tidb-test --sudo --command "yum -y install numactl"
```

To get help information of the `tiup cluster exec` command, run the `tiup cluster exec --help` command.

5.3 Plan Cluster Topology

5.3.1 Minimal Deployment Topology

This document describes the minimal deployment topology of TiDB clusters.

5.3.1.1 Topology information

Instance	Count	Physical machine configuration	IP	Configuration
TiDB	2	16 VCore 32 GiB 100 GiB for storage	10.0.1.1 10.0.1.2	Default port Global directory configuration
PD	3	4 VCore 8 GiB 100 GiB for storage	10.0.1.4 10.0.1.5 10.0.1.6	Default port Global directory configuration

Instance	Count	Physical machine configuration	IP	Configuration
TiKV	3	16 VCore 32 GiB 2 TiB (NVMe SSD) for storage	10.0.1.1 10.0.1.2 10.0.1.3	Default port Global directory configuration
Monitoring & Grafana	1	4 VCore 8 GiB 500 GiB (SSD) for storage	10.0.1.10	Default port Global directory configuration

5.3.1.1.1 Topology templates

- The simple template for the minimal topology
- The complex template for the minimal topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see Topology Configuration File for Deploying TiDB Using TiUP.

Note:

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

5.3.2 TiFlash Deployment Topology

This document describes the deployment topology of TiFlash based on the minimal TiDB topology.

TiFlash is a columnar storage engine, and gradually becomes the standard cluster topology. It is suitable for real-time HTAP applications.

5.3.2.1 Topology information

Instance	Count	Physical machine configuration	IP	Configuration
TiDB	3	16 VCore 32GB * 1	10.0.1.7 10.0.1.8 10.0.1.9	Default port Global directory configuration

Instance	Count	Physical machine configuration	IP	Configuration
PD	3	4 VCore 8GB * 1	10.0.1.4 10.0.1.5 10.0.1.6	Default port Global directory configuration
TiKV	3	16 VCore 32GB 2TB (nvme ssd) * 1	10.0.1.1 10.0.1.2 10.0.1.3	Default port Global directory configuration
TiFlash	1	32 VCore 64 GB 2TB (nvme ssd) * 1	10.0.1.11	Default port Global directory configuration

Instance	Count	Physical machine configuration	IP	Configuration
Monitoring & Grafana	1	4 VCore 8GB * 1 500GB (ssd)	10.0.1.10	Default port Global directory configuration

5.3.2.1.1 Topology templates

- The simple template for the TiFlash topology
- The complex template for the TiFlash topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see Topology Configuration File for Deploying TiDB Using TiUP.

5.3.2.1.2 Key parameters

- To enable the Placement Rules feature of PD, set the value of `replication.enable-placement-rules` in the configuration template to `true`.
- The instance level `"-host"` configuration in `tiflash_servers` only supports IP, not domain name.
- For detailed TiFlash parameter description, see TiFlash Configuration.

Note:

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.

- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

5.3.3 TiCDC Deployment Topology

Note:

TiCDC is a feature for general availability (GA) since v4.0.6. You can use it in the production environment.

This document describes the deployment topology of TiCDC based on the minimal cluster topology.

TiCDC is a tool for replicating the incremental data of TiDB, introduced in TiDB 4.0. It supports multiple downstream platforms, such as TiDB, MySQL, and MQ. Compared with TiDB Binlog, TiCDC has lower latency and native high availability.

5.3.3.1 Topology information

Instance	Count	Physical machine configuration	IP	Configuration
TiDB	3	16 VCore 32GB * 1	10.0.1.1 10.0.1.2 10.0.1.3	Default port Global directory configuration

Instance	Count	Physical machine configuration	IP	Configuration
PD	3	4 VCore 8GB * 1	10.0.1.4 10.0.1.5 10.0.1.6	Default port Global directory configuration
TiKV	3	16 VCore 32GB 2TB (nvme ssd) * 1	10.0.1.7 10.0.1.8 10.0.1.9	Default port Global directory configuration
CDC	3	8 VCore 16GB * 1	10.0.1.10 10.0.1.11 10.0.1.12	Default port Global directory configuration

Instance	Count	Physical machine configuration	IP	Configuration
Monitoring & Grafana	1	4 VCore 8GB * 1 500GB (ssd)	10.0.1.11	Default port Global directory configuration

5.3.3.1.1 Topology templates

- The simple template for the TiCDC topology
- The complex template for the TiCDC topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see Topology Configuration File for Deploying TiDB Using TiUP.

Note:

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

5.3.4 TiDB Binlog Deployment Topology

This document describes the deployment topology of TiDB Binlog based on the minimal TiDB topology.

TiDB Binlog is the widely used component for replicating incremental data. It provides near real-time backup and replication.

5.3.4.1 Topology information

Instance	Count	Physical machine configuration	IP	Configuration
TiDB	3	16 VCore 32 GB	10.0.1.1 10.0.1.2 10.0.1.3	Default port configuration; Enable `enable_binlog` ↪ ; Enable `ignore` ↪ - ↪ `error` ↪
PD	3	4 VCore 8 GB	10.0.1.4 10.0.1.5 10.0.1.6	Default port configuration
TiKV	3	16 VCore 32 GB	10.0.1.7 10.0.1.8 10.0.1.9	Default port configuration

Instance	Count	Physical machine configuration	IP	Configuration
Pump	3	8 VCore 16GB	10.0.1.1 10.0.1.7 10.0.1.8	Default port configuration; Set GC time to 7 days

Instance	Count	Physical machine configuration	IP	Configuration
Drainer	1	8 VCore 16GB	10.0.1.12	Default port configuration; Set the default initialization commitTS -1 as the latest timestamp; Configure the downstream target TiDB as `10.0.1.12:4000` ↪

5.3.4.1.1 Topology templates

- The simple template for the TiDB Binlog topology (with `mysql` as the downstream type)
- The simple template for the TiDB Binlog topology (with `file` as the downstream type)
- The complex template for the TiDB Binlog topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see Topology Configuration File for Deploying TiDB Using TiUP.

5.3.4.1.2 Key parameters

The key parameters in the topology configuration templates are as follows:

- `server_configs.tidb.binlog.enable: true`
  - Enables the binlog service.
  - Default value: `false`.
- `server_configs.tidb.binlog.ignore-error: true`
  - It is recommended to enable this configuration in high availability scenarios.
  - If set to `true`, when an error occurs, TiDB stops writing data into binlog, and adds `1` to the value of the `tidb_server_critical_error_total` monitoring metric.
  - If set to `false`, when TiDB fails to write data into binlog, the whole TiDB service is stopped.
- `drainer_servers.config.syncer.db-type`

  The downstream type of TiDB Binlog. Currently, `mysql`, `tidb`, `kafka`, and `file` are supported.
- `drainer_servers.config.syncer.to`

  The downstream configuration of TiDB Binlog. Depending on different `db-type`s, you can use this configuration item to configure the connection parameters of the downstream database, the connection parameters of Kafka, and the file save path. For details, refer to TiDB Binlog Configuration File.

Note:

- When editing the configuration file template, if you do not need custom ports or directories, modify the IP only.

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

5.3.5 TiSpark Deployment Topology

Warning:

TiSpark support in the TiUP cluster is still an experimental feature. It is **NOT** recommended to use it in the production environment.

This document introduces the TiSpark deployment topology and how to deploy TiSpark based on the minimum cluster topology.

TiSpark is a component built for running Apache Spark on top of TiDB/TiKV to answer complex OLAP queries. It brings benefits of both the Spark platform and the distributed TiKV cluster to TiDB and makes TiDB a one-stop solution for both online transactions and analytics.

For more information about the TiSpark architecture and how to use it, see TiSpark User Guide.

5.3.5.1 Topology information

Instance	Count	Physical machine configuration	IP	Configuration
TiDB	3	16 VCore 32GB * 1	10.0.1.1 10.0.1.2 10.0.1.3	Default port Global directory configuration
PD	3	4 VCore 8GB * 1	10.0.1.4 10.0.1.5 10.0.1.6	Default port Global directory configuration
TiKV	3	16 VCore 32GB 2TB (nvme ssd) * 1	10.0.1.7 10.0.1.8 10.0.1.9	Default port Global directory configuration

Instance	Count	Physical machine configuration	IP	Configuration
TiSpark	3	8 VCore 16GB * 1	10.0.1.21 (master) 10.0.1.22 (worker) 10.0.1.23 (worker)	Default port Global directory configuration
Monitoring & Grafana	1	4 VCore 8GB * 1 500GB (ssd)	10.0.1.11	Default port Global directory configuration

5.3.5.2 Topology templates

- Simple TiSpark topology template
- Complex TiSpark topology template

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see Topology Configuration File for Deploying TiDB Using TiUP.

Note:

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

5.3.5.3 Prerequisites

TiSpark is based on the Apache Spark cluster, so before you start the TiDB cluster that contains TiSpark, you must ensure that Java Runtime Environment (JRE) 8 is installed on the server that deploys TiSpark. Otherwise, TiSpark cannot be started.

TiUP does not support installing JRE automatically. You need to install it on your own. For detailed installation instruction, see How to download and install prebuilt OpenJDK packages.

If JRE 8 has already been installed on the deployment server but is not in the path of the system's default package management tool, you can specify the path of the JRE environment to be used by setting the `java_home` parameter in the topology configuration. This parameter corresponds to the `JAVA_HOME` system environment variable.

5.3.6 Geo-Distributed Deployment Topology

This document takes the typical architecture of three data centers (DC) in two cities as an example, and introduces the geo-distributed deployment architecture and the key configuration. The cities used in this example are Shanghai (referred to as `sha`) and Beijing (referred to as `bja` and `bjb`).

5.3.6.1 Topology information

Instance	Count	Physical machine configuration	BJ IP	SH IP	Configuration
TiDB	5	16 VCore 32GB * 1	10.0.1.1 10.0.1.2 10.0.1.3 10.0.1.4	10.0.1.5	Default port Global directory configuration
PD	5	4 VCore 8GB * 1	10.0.1.6 10.0.1.7 10.0.1.8 10.0.1.9	10.0.1.10	Default port Global directory configuration
TiKV	5	16 VCore 32GB 2TB (nvme ssd) * 1	10.0.1.11 10.0.1.12 10.0.1.13 10.0.1.14	10.0.1.15	Default port Global directory configuration

Instance	Count	Physical machine configuration	BJ IP	SH IP	Configuration
Monitoring & Grafana	1	4 VCore 8GB * 1 500GB (ssd)	10.0.1.16		Default port Global directory configuration

5.3.6.1.1 Topology templates

- The geo-distributed topology template

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see Topology Configuration File for Deploying TiDB Using TiUP.

5.3.6.1.2 Key parameters

This section describes the key parameter configuration of the TiDB geo-distributed deployment.

TiKV parameters

- The gRPC compression format (`none` by default):

  To increase the transmission speed of gRPC packages between geo-distributed target nodes, set this parameter to `gzip`.

```
server.grpc-compression-type: gzip
```

- The label configuration:

  Since TiKV is deployed across different data centers, if the physical machines go down, the Raft Group might lose three of the default five replicas, which causes the cluster

unavailability. To address this issue, you can configure the labels to enable the smart scheduling of PD, which ensures that the Raft Group does not allow three replicas to be located in TiKV instances on the same machine in the same cabinet of the same data center.

- The TiKV configuration:

  The same host-level label information is configured for the same physical machine.

```
config:
  server.labels:
    zone: bj
    dc: bja
    rack: rack1
    host: host2
```

- To prevent remote TiKV nodes from launching unnecessary Raft elections, it is required to increase the minimum and maximum number of ticks that the remote TiKV nodes need to launch an election. The two parameters are set to `0` by default.

```
raftstore.raft-min-election-timeout-ticks: 1000
raftstore.raft-max-election-timeout-ticks: 1020
```

PD parameters

- The PD metadata information records the topology of the TiKV cluster. PD schedules the Raft Group replicas on the following four dimensions:

```
replication.location-labels: ["zone","dc","rack","host"]
```

- To ensure high availability of the cluster, adjust the number of Raft Group replicas to be `5`:

```
replication.max-replicas: 5
```

- Forbid the remote TiKV Raft replica being elected as Leader:

```
label-property:
     reject-leader:
       - key: "dc"
         value: "sha"
```

> **Note:**
>
> Since TiDB 5.2, the `label-property` configuration is not supported by default. To set the replica policy, use the placement rules.

For the further information about labels and the number of Raft Group replicas, see Schedule Replicas by Topology Labels.

Note:

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

5.3.7 Hybrid Deployment Topology

This document describes the topology and key parameters of the TiKV and TiDB hybrid deployment.

The hybrid deployment is usually used in the following scenario:

The deployment machine has multiple CPU processors with sufficient memory. To improve the utilization rate of the physical machine resources, multiple instances can be deployed on a single machine, that is, TiDB and TiKV's CPU resources are isolated through NUMA node bindings. PD and Prometheus are deployed together, but their data directories need to use separate file systems.

5.3.7.1 Topology information

Instance	Count	Physical machine configuration	IP	Configuration
TiDB	6	32 VCore 64GB	10.0.1.1 10.0.1.2 10.0.1.3	Configure NUMA to bind CPU cores

Instance	Count	Physical machine configuration	IP	Configuration
PD	3	16 VCore 32 GB	10.0.1.4 10.0.1.5 10.0.1.6	Configure the `location_labels` parameter

Instance	Count	Physical machine configuration	IP	Configuration
TiKV	6	32 VCore 64GB	10.0.1.7 10.0.1.8 10.0.1.9	1. Separate the instance-level port and status_port; 2. Configure the global parameters `readpool` ↪ , `storage` ↪ and `raftstore` ↪ ; 3. Configure labels of the instance-level host; 4. Con-
		586		

Instance	Count	Physical machine configuration	IP	Configuration
Monitoring & Grafana	1	4 VCore 8GB * 1 500GB (ssd)	10.0.1.10	Default configuration

5.3.7.1.1 Topology templates

- The simple template for the hybrid deployment
- The complex template for the hybrid deployment

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see Topology Configuration File for Deploying TiDB Using TiUP.

5.3.7.1.2 Key parameters

This section introduces the key parameters when you deploy multiple instances on a single machine, which is mainly used in scenarios when multiple instances of TiDB and TiKV are deployed on a single machine. You need to fill in the results into the configuration template according to the calculation methods provided below.

- Optimize the configuration of TiKV
  - To configure `readpool` to be self-adaptive to the thread pool. By configuring the `readpool.unified.max-thread-count` parameter, you can make `readpool.storage` and `readpool.coprocessor` share a unified thread pool, and set the self-adaptive switch respectively.
    - Enable `readpool.storage` and `readpool.coprocessor`:

```
readpool.storage.use-unified-pool: true
readpool.coprocessor.use-unified-pool: true
```

    - The calculation method:

```
readpool.unified.max-thread-count = cores * 0.8 / the number of
    ↪ TiKV instances
```

- To configure the storage CF (all RocksDB column families) to be self-adaptive to memory. By configuring the `storage.block-cache.capacity` parameter, you can make CF automatically balance the memory usage.
  - `storage.block-cache` enables the CF self-adaptation by default. You do not need to modify it.

    ```
    storage.block-cache.shared: true
    ```

  - The calculation method:

    ```
    storage.block-cache.capacity = (MEM_TOTAL * 0.5 / the number of
        ↪ TiKV instances)
    ```

- If multiple TiKV instances are deployed on the same physical disk, add the `capacity` parameter in the TiKV configuration:

  ```
  raftstore.capacity = disk total capacity / the number of TiKV
      ↪ instances
  ```

- The label scheduling configuration

  Since multiple instances of TiKV are deployed on a single machine, if the physical machines go down, the Raft Group might lose two of the default three replicas, which causes the cluster unavailability. To address this issue, you can use the label to enable the smart scheduling of PD, which ensures that the Raft Group has more than two replicas in multiple TiKV instances on the same machine.

  - The TiKV configuration

    The same host-level label information is configured for the same physical machine:

    ```
    config:
      server.labels:
        host: tikv1
    ```

  - The PD configuration

    To enable PD to identify and scheduling Regions, configure the labels type for PD:

    ```
    pd:
      replication.location-labels: ["host"]
    ```

- `numa_node` core binding

- In the instance parameter module, configure the corresponding `numa_node` parameter and add the number of CPU cores.
- Before using NUMA to bind cores, make sure that the numactl tool is installed, and confirm the information of CPUs in the physical machines. After that, configure the parameters.
- The `numa_node` parameter corresponds to the `numactl --membind` configuration.

Note:

- When editing the configuration file template, modify the required parameter, IP, port, and directory.
- Each component uses the global `<deploy_dir>/<components_name>-<port>` as their `deploy_dir` by default. For example, if TiDB specifies the `4001` port, its `deploy_dir` is `/tidb-deploy/tidb-4001` by default. Therefore, in multi-instance scenarios, when specifying a non-default port, you do not need to specify the directory again.
- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

5.4 Install and Start

5.4.1 Deploy a TiDB Cluster Using TiUP

TiUP is a cluster operation and maintenance tool introduced in TiDB 4.0. TiUP provides TiUP cluster, a cluster management component written in Golang. By using TiUP cluster, you can easily perform daily database operations, including deploying, starting, stopping, destroying, scaling, and upgrading a TiDB cluster, and manage TiDB cluster parameters.

TiUP supports deploying TiDB, TiFlash, TiDB Binlog, TiCDC, and the monitoring system. This document introduces how to deploy TiDB clusters of different topologies.

5.4.1.1 Step 1. Prerequisites and precheck

Make sure that you have read the following documents:

- Hardware and software requirements
- Environment and system configuration check

5.4.1.2 Step 2. Deploy TiUP on the control machine

You can deploy TiUP on the control machine in either of the two ways: online deployment and offline deployment.

5.4.1.2.1 Deploy TiUP online

Log in to the control machine using a regular user account (take the `tidb` user as an example). Subsequent TiUP installation and cluster management can be performed by the `tidb` user.

1. Install TiUP by running the following command:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

2. Set TiUP environment variables:

    1. Redeclare the global environment variables:

    ```
    source .bash_profile
    ```

    2. Confirm whether TiUP is installed:

    ```
    which tiup
    ```

3. Install the TiUP cluster component:

```
tiup cluster
```

4. If TiUP is already installed, update the TiUP cluster component to the latest version:

```
tiup update --self && tiup update cluster
```

    If `Update successfully!` is displayed, the TiUP cluster is updated successfully.

5. Verify the current version of your TiUP cluster:

```
tiup --binary cluster
```

5.4.1.2.2 Deploy TiUP offline

Perform the following steps in this section to deploy a TiDB cluster offline using TiUP:

Prepare the TiUP offline component package

Method 1: Download the offline binary packages (TiUP offline package included) of the target TiDB version using the following links. You need to download both the server and toolkit packages. Note that your downloading means you agree to the Privacy Policy.

```
https://download.pingcap.org/tidb-community-server-{version}-linux-{arch}.
    ↪ tar.gz
```

```
https://download.pingcap.org/tidb-community-toolkit-{version}-linux-{arch}.
    ↪ tar.gz
```

Tip:

`{version}` in the link indicates the version number of TiDB and `{arch ↪ }` indicates the architecture of the system, which can be `amd64` or `arm64`. For example, the download link for `v6.1.0` in the `amd64` architecture is `https://download.pingcap.org/tidb-community-toolkit- ↪ v6.1.0-linux-amd64.tar.gz`.

Method 2: Manually pack an offline component package using `tiup mirror clone`. The detailed steps are as follows:

1. Install the TiUP package manager online.

    1. Install the TiUP tool:

    ```
    curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.
        ↪ com/install.sh | sh
    ```

    2. Redeclare the global environment variables:

    ```
    source .bash_profile
    ```

    3. Confirm whether TiUP is installed:

    ```
    which tiup
    ```

2. Pull the mirror using TiUP.

    1. Pull the needed components on a machine that has access to the Internet:

    ```
    tiup mirror clone tidb-community-server-${version}-linux-amd64 ${
        ↪ version} --os=linux --arch=amd64
    ```

    The command above creates a directory named `tidb-community-server- ↪ ${version}-linux-amd64` in the current directory, which contains the component package necessary for starting a cluster.

2. Pack the component package by using the `tar` command and send the package to the control machine in the isolated environment:

```
tar czvf tidb-community-server-${version}-linux-amd64.tar.gz tidb-
    ↪ community-server-${version}-linux-amd64
```

`tidb-community-server-${version}-linux-amd64.tar.gz` is an independent offline environment package.

3. Customize the offline mirror, or adjust the contents of an existing offline mirror.

   If you want to adjust an existing offline mirror (such as adding a new version of a component), take the following steps:

   1. When pulling an offline mirror, you can get an incomplete offline mirror by specifying specific information via parameters, such as the component and version information. For example, you can pull an offline mirror that includes only the offline mirror of TiUP v1.10.0 and TiUP Cluster v1.10.0 by running the following command:

```
tiup mirror clone tiup-custom-mirror-v1.10.0 --tiup v1.10.0 --
    ↪ cluster v1.10.0
```

      If you only need the components for a particular platform, you can specify them using the `--os` or `--arch` parameters.

   2. Refer to the step 2 of "Pull the mirror using TiUP", and send this incomplete offline mirror to the control machine in the isolated environment.

   3. Check the path of the current offline mirror on the control machine in the isolated environment. If your TiUP tool is of a recent version, you can get the current mirror address by running the following command:

```
tiup mirror show
```

      If the output of the above command indicates that the `show` command does not exist, you might be using an older version of TiUP. In this case, you can get the current mirror address from `$HOME/.tiup/tiup.toml`. Record this mirror address. In the following steps, `${base_mirror}` is used to refer to this address.

   4. Merge an incomplete offline mirror into an existing offline mirror:

      First, copy the `keys` directory in the current offline mirror to the `$HOME/.tiup` directory:

```
cp -r ${base_mirror}/keys $HOME/.tiup/
```

      Then use the TiUP command to merge the incomplete offline mirror into the mirror in use:

```
tiup mirror merge tiup-custom-mirror-v1.10.0
```

5. When the above steps are completed, check the result by running the `tiup list` command. In this document's example, the outputs of both `tiup list tiup` and `tiup list cluster` show that the corresponding components of `v1.10.0` are available.

Deploy the offline TiUP component

After sending the package to the control machine of the target cluster, install the TiUP component by running the following commands:

```
tar xzvf tidb-community-server-${version}-linux-amd64.tar.gz && \
sh tidb-community-server-${version}-linux-amd64/local_install.sh && \
source /home/tidb/.bash_profile
```

The `local_install.sh` script automatically runs the `tiup mirror set tidb-community-server-${version}-linux-amd64` command to set the current mirror address to `tidb-community-server-${version}-linux-amd64`.

Merge offline packages

If you download the offline packages via download links, you need to merge the server package and the toolkit package into an offline mirror. If you manually package the offline component packages using the `tiup mirror clone` command, you can skip this step.

Run the following commands to merge the offline toolkit package into the server package directory:

```
tar xf tidb-community-toolkit-${version}-linux-amd64.tar.gz
ls -ld tidb-community-server-${version}-linux-amd64 tidb-community-toolkit-${version}-linux-amd64
cd tidb-community-server-${version}-linux-amd64/
cp -rp keys ~/.tiup/
tiup mirror merge ../tidb-community-toolkit-${version}-linux-amd64
```

To switch the mirror to another directory, run the `tiup mirror set <mirror-dir>` command. To switch the mirror to the online environment, run the `tiup mirror set https://tiup-mirrors.pingcap.com` command.

5.4.1.3 Step 3. Initialize cluster topology file

Run the following command to create a cluster topology file:

```
tiup cluster template > topology.yaml
```

In the following two common scenarios, you can generate recommended topology templates by running commands:

- For hybrid deployment: Multiple instances are deployed on a single machine. For details, see Hybrid Deployment Topology.

```
tiup cluster template --full > topology.yaml
```

- For geo-distributed deployment: TiDB clusters are deployed in geographically distributed data centers. For details, see Geo-Distributed Deployment Topology.

```
tiup cluster template --multi-dc > topology.yaml
```

Run `vi topology.yaml` to see the configuration file content:

```
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/tidb-deploy"
  data_dir: "/tidb-data"
server_configs: {}
pd_servers:
  - host: 10.0.1.4
  - host: 10.0.1.5
  - host: 10.0.1.6
tidb_servers:
  - host: 10.0.1.7
  - host: 10.0.1.8
  - host: 10.0.1.9
tikv_servers:
  - host: 10.0.1.1
  - host: 10.0.1.2
  - host: 10.0.1.3
monitoring_servers:
  - host: 10.0.1.4
grafana_servers:
  - host: 10.0.1.4
alertmanager_servers:
  - host: 10.0.1.4
```

The following examples cover seven common scenarios. You need to modify the configuration file (named `topology.yaml`) according to the topology description and templates in the corresponding links. For other scenarios, edit the configuration template accordingly.

Application	Configuration task	Configuration template	Topology description
OLTP	Deploy minimal topology	Simple minimal configuration template	This is the basic cluster topology, including tidb-server, tikv-server, and pd-server.
		Full minimal configuration template	

Application	Configuration task	Configuration template	Topology description
HTAP	Deploy the TiFlash topology	Simple TiFlash configuration template	This is to deploy TiFlash along with the minimal cluster topology. TiFlash is a columnar storage engine, and gradually becomes a standard cluster topology.
		Full TiFlash configuration template	

Application	Configuration task	Configuration template	Topology description
Replicate incremental data using TiCDC	Deploy the TiCDC topology	Simple TiCDC configuration template Full TiCDC configuration template	This is to deploy TiCDC along with the minimal cluster topology. TiCDC supports multiple downstream platforms, such as TiDB, MySQL, and MQ.

Application	Configuration task	Configuration template	Topology description
Replicate incremental data using TiDB Binlog	Deploy the TiDB Binlog topology	Simple TiDB Binlog configuration template (MySQL as downstream) Simple TiDB Binlog configuration template (Files as downstream) Full TiDB Binlog configuration template	This is to deploy TiDB Binlog along with the minimal cluster topology.

Application	Configuration task	Configuration file template	Topology description
Use OLAP on Spark	Deploy the TiSpark topology	Simple TiSpark configuration template Full TiSpark configuration template	This is to deploy TiSpark along with the minimal cluster topology. TiSpark is a component built for running Apache Spark on top of TiDB/TiKV to answer the OLAP queries. Currently, TiUP cluster's support

Application	Configuration task	Configuration template	Topology description
Deploy multiple instances on a single machine	Deploy a hybrid topology	Simple configuration template for hybrid deployment Full configuration template for hybrid deployment	The deployment topologies also apply when you need to add extra configurations for the directory, port, resource ratio, and label.

Application	Configuration task	Configuration template	Topology description
Deploy TiDB clusters across data centers	Deploy a geo-distributed deployment topology	Configuration template for geo-distributed deployment	This topology takes the typical architecture of three data centers in two cities as an example. It introduces the geo-distributed deployment architecture and the key configura-

Application task	Configuration template	Configuration file generation	Topology description

Note:

- For parameters that should be globally effective, configure these parameters of corresponding components in the `server_configs` section of the configuration file.
- For parameters that should be effective on a specific node, configure these parameters in the `config` of this node.
- Use `.` to indicate the subcategory of the configuration, such as `log.slow-threshold`. For more formats, see TiUP configuration template.
- If you need to specify the user group name to be created on the target machine, see this example.

For more configuration description, see the following configuration examples:

- TiDB `config.toml.example`
- TiKV `config.toml.example`
- PD `config.toml.example`
- TiFlash `config.toml.example`

5.4.1.4 Step 4. Run the deployment command

Note:

You can use secret keys or interactive passwords for security authentication when you deploy TiDB using TiUP:

- If you use secret keys, specify the path of the keys through `-i` or `--identity_file`.
- If you use passwords, add the `-p` flag to enter the password interaction window.
- If password-free login to the target machine has been configured, no authentication is required.

> In general, TiUP creates the user and group specified in the `topology.yaml` file on the target machine, with the following exceptions:
>
> - The user name configured in `topology.yaml` already exists on the target machine.
> - You have used the `--skip-create-user` option in the command line to explicitly skip the step of creating the user.

Before you run the `deploy` command, use the `check` and `check --apply` commands to detect and automatically repair potential risks in the cluster:

1. Check for potential risks:

```
tiup cluster check ./topology.yaml --user root [-p] [-i /home/root/.ssh
    ↪ /gcp_rsa]
```

2. Enable automatic repair:

```
tiup cluster check ./topology.yaml --apply --user root [-p] [-i /home/
    ↪ root/.ssh/gcp_rsa]
```

3. Deploy a TiDB cluster:

```
tiup cluster deploy tidb-test v6.1.7 ./topology.yaml --user root [-p]
    ↪ [-i /home/root/.ssh/gcp_rsa]
```

In the `tiup cluster deploy` command above:

- `tidb-test` is the name of the TiDB cluster to be deployed.
- `v6.1.7` is the version of the TiDB cluster to be deployed. You can see the latest supported versions by running `tiup list tidb`.
- `topology.yaml` is the initialization configuration file.
- `--user root` indicates logging into the target machine as the `root` user to complete the cluster deployment. The `root` user is expected to have `ssh` and `sudo` privileges to the target machine. Alternatively, you can use other users with `ssh` and `sudo` privileges to complete the deployment.
- `[-i]` and `[-p]` are optional. If you have configured login to the target machine without password, these parameters are not required. If not, choose one of the two parameters. `[-i]` is the private key of the root user (or other users specified by `--user`) that has access to the target machine. `[-p]` is used to input the user password interactively.

At the end of the output log, you will see `Deployed cluster `tidb-test` successfully`. This indicates that the deployment is successful.

5.4.1.5 Step 5. Check the clusters managed by TiUP

```
tiup cluster list
```

TiUP supports managing multiple TiDB clusters. The preceding command outputs information of all the clusters currently managed by TiUP, including the cluster name, deployment user, version, and secret key information:

5.4.1.6 Step 6. Check the status of the deployed TiDB cluster

For example, run the following command to check the status of the `tidb-test` cluster:

```
tiup cluster display tidb-test
```

Expected output includes the instance ID, role, host, listening port, and status (because the cluster is not started yet, so the status is `Down`/`inactive`), and directory information.

5.4.1.7 Step 7. Start a TiDB cluster

Since TiUP cluster v1.9.0, safe start is introduced as a new start method. Starting a database using this method improves the security of the database. It is recommended that you use this method.

After safe start, TiUP automatically generates a password for the TiDB root user and returns the password in the command-line interface.

Note:

- After safe start of a TiDB cluster, you cannot log in to TiDB using a root user without a password. Therefore, you need to record the password returned in the command output for future logins.
- The password is generated only once. If you do not record it or you forgot it, refer to Forget the `root` password to change the password.

Method 1: Safe start

```
tiup cluster start tidb-test --init
```

If the output is as follows, the start is successful:

```
Started cluster `tidb-test` successfully.
The root password of TiDB database has been changed.
The new password is: 'y_+3Hwp=*AWz8971s6'.
Copy and record it to somewhere safe, it is only displayed once, and will
    ↪ not be stored.
```

```
The generated password can NOT be got again in future.
```

Method 2: Standard start

```
tiup cluster start tidb-test
```

If the output log includes `Started cluster `tidb-test` successfully`, the start is successful. After standard start, you can log in to a database using a root user without a password.

5.4.1.8 Step 8. Verify the running status of the TiDB cluster

```
tiup cluster display tidb-test
```

If the output log shows `Up` status, the cluster is running properly.

5.4.1.9 See also

If you have deployed TiFlash along with the TiDB cluster, see the following documents:

- Use TiFlash
- Maintain a TiFlash Cluster
- TiFlash Alert Rules and Solutions
- Troubleshoot TiFlash

If you have deployed TiCDC along with the TiDB cluster, see the following documents:

- Manage TiCDC Cluster and Replication Tasks
- Troubleshoot TiCDC
- TiCDC FAQs

5.4.2 Deploy a TiDB Cluster on Kubernetes

You can use TiDB Operator to deploy TiDB clusters on Kubernetes. TiDB Operator is an automatic operation system for TiDB clusters on Kubernetes. It provides full life-cycle management for TiDB including deployment, upgrades, scaling, backup, fail-over, and configuration changes. With TiDB Operator, TiDB can run seamlessly on the Kubernetes clusters deployed on a public or private cloud.

Currently, the TiDB on Kubernetes documentation is independent of the TiDB documentation. For detailed steps on how to deploy TiDB clusters on Kubernetes using TiDB Operator, see TiDB on Kubernetes documentation.

5.5 Check Cluster Status

After a TiDB cluster is deployed, you need to check whether the cluster runs normally. This document introduces how to check the cluster status using TiUP commands, TiDB Dashboard and Grafana, and how to log into the TiDB database to perform simple SQL operations.

5.5.1 Check the TiDB cluster status

This section describes how to check the TiDB cluster status using TiUP commands, TiDB Dashboard, and Grafana.

5.5.1.1 Use TiUP

Use the `tiup cluster display <cluster-name>` command to check the cluster status. For example:

```
tiup cluster display tidb-test
```

Expected output: If the `Status` information of each node is `Up`, the cluster runs normally.

5.5.1.2 Use TiDB Dashboard

1. Log in to TiDB Dashboard at `${pd-ip}:${pd-port}/dashboard`. The username and password is the same as that of the TiDB `root` user. If you have modified the `root` password, enter the modified password. The password is empty by default.

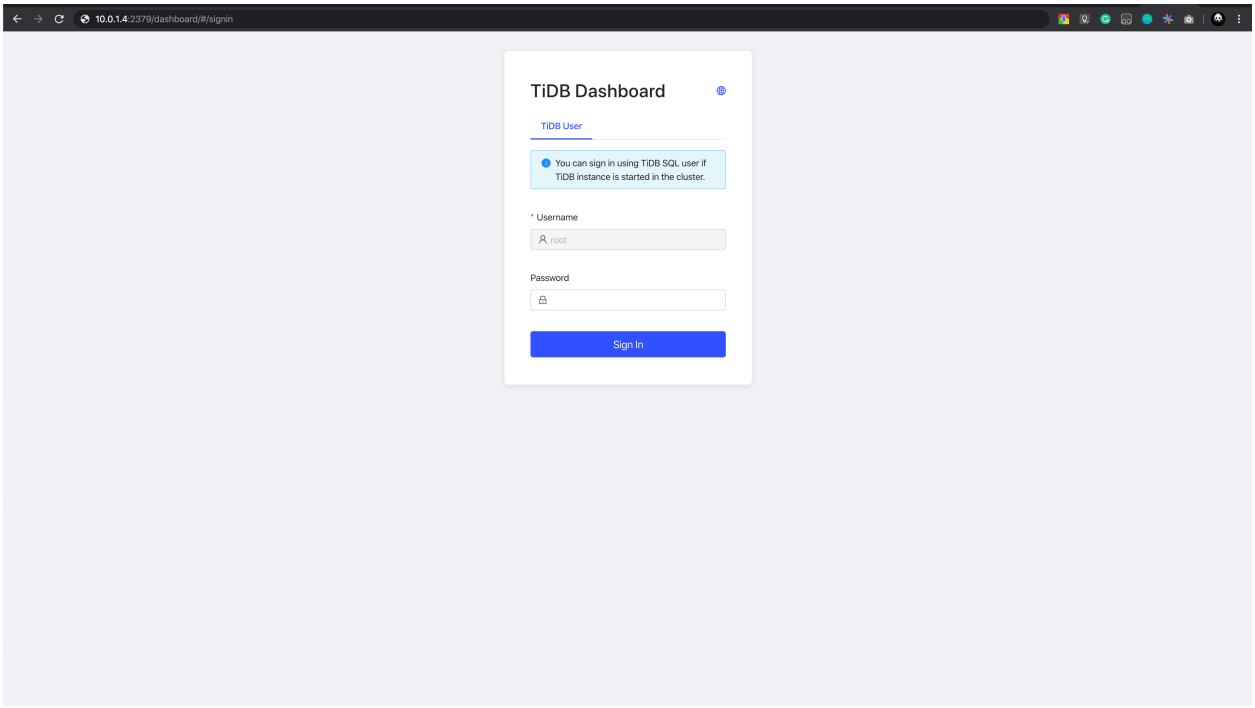


Figure 48: TiDB-Dashboard

2. The home page displays the node information in the TiDB cluster.

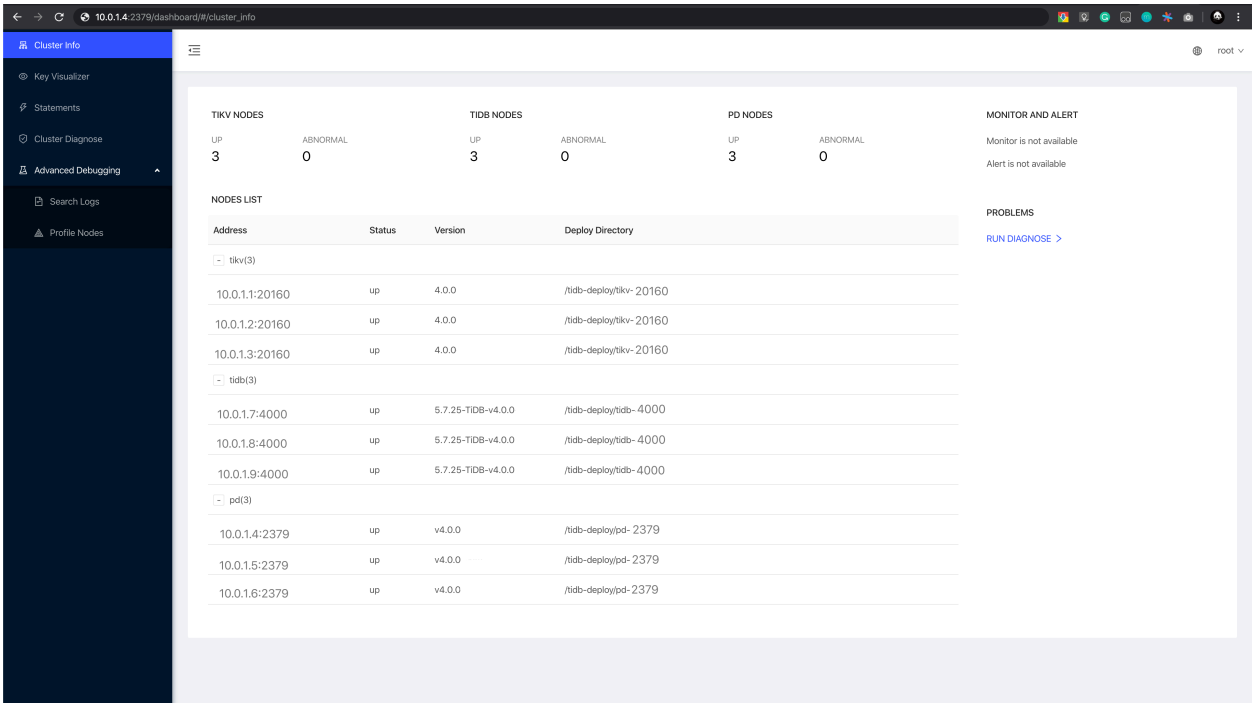


Figure 49: TiDB-Dashboard-status

5.5.1.3 Use Grafana

1. Log in to the Grafana monitoring at `${Grafana-ip}:3000`. The default username and password are both `admin`.

2. To check the TiDB port status and load monitoring information, click **Overview**.

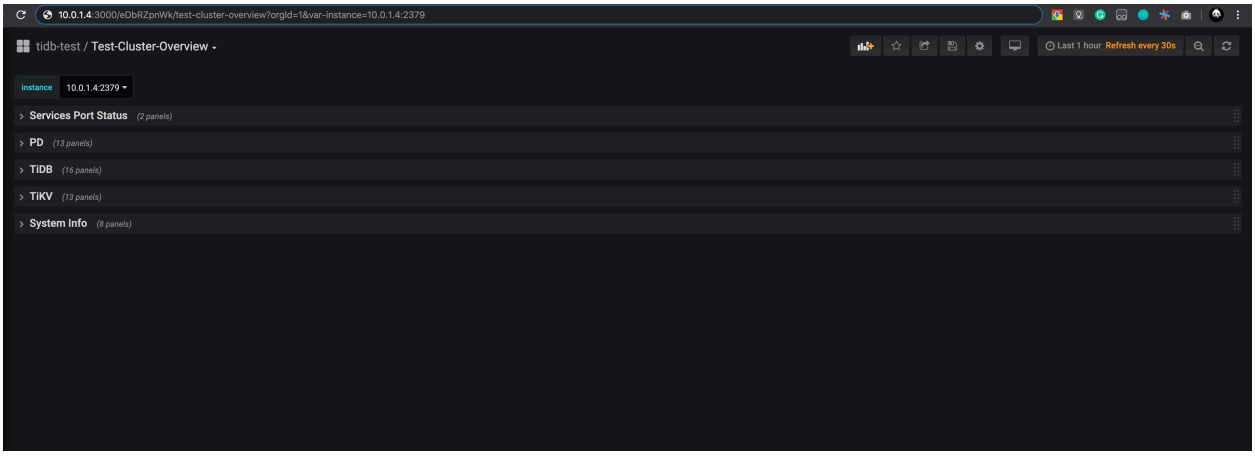


Figure 50: Grafana-overview

5.5.2 Log in to the database and perform simple operations

> **Note:**
>
> Install the MySQL client before you log in to the database.

Log in to the database by running the following command:

```
mysql -u root -h ${tidb_server_host_IP_address} -P 4000
```

`${tidb_server_host_IP_address}` is one of the IP addresses set for `tidb_servers` when you initialize the cluster topology file, such as `10.0.1.7`.

The following information indicates successful login:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.25-TiDB-v5.0.0 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 5.7 compatible
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved
    ↪ .
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
    ↪ statement.
```

5.5.2.1 Database operations

- Check the version of TiDB:

  ```
  select tidb_version()\G
  ```

  Expected output:

  ```
  *************************** 1. row ***************************
  tidb_version(): Release Version: v5.0.0
  Edition: Community
  Git Commit Hash: 689a6b6439ae7835947fcaccf329a3fc303986cb
  Git Branch: HEAD
  UTC Build Time: 2020-05-28 11:09:45
  GoVersion: go1.13.4
  ```

```
Race Enabled: false
TiKV Min Version: v3.0.0-60965b006877ca7234adaced7890d7b029ed1306
Check Table Before Drop: false
1 row in set (0.00 sec)
```

- Create a database named `pingcap`:

```
create database pingcap;
```

Expected output:

```
Query OK, 0 rows affected (0.10 sec)
```

Switch to the `pingcap` database:

```
use pingcap;
```

Expected output:

```
Database changed
```

- Create a table named `tab_tidb`:

```
CREATE TABLE `tab_tidb` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(20) NOT NULL DEFAULT '',
`age` int(11) NOT NULL DEFAULT 0,
`version` varchar(20) NOT NULL DEFAULT '',
PRIMARY KEY (`id`),
KEY `idx_age` (`age`));
```

Expected output:

```
Query OK, 0 rows affected (0.11 sec)
```

- Insert data:

```
insert into `tab_tidb` values (1,'TiDB',5,'TiDB-v5.0.0');
```

Expected output:

```
Query OK, 1 row affected (0.03 sec)
```

- View the entries in `tab_tidb`:

```
select * from tab_tidb;
```

Expected output:

```
+----+------+-----+-------------+
| id | name | age | version |
+----+------+-----+-------------+
|  1 | TiDB |  5 | TiDB-v5.0.0 |
+----+------+-----+-------------+
1 row in set (0.00 sec)
```

- View the store state, `store_id`, capacity, and uptime of TiKV:

```
select STORE_ID,ADDRESS,STORE_STATE,STORE_STATE_NAME,CAPACITY,AVAILABLE
    ↪ ,UPTIME from INFORMATION_SCHEMA.TIKV_STORE_STATUS;
```

Expected output:

```
+--
    ↪ --------+-------------------+-------------+-----------------+---------+-
    ↪
| STORE_ID | ADDRESS         | STORE_STATE | STORE_STATE_NAME | CAPACITY
    ↪  | AVAILABLE | UPTIME    |
+--
    ↪ --------+-------------------+-------------+-----------------+---------+-
    ↪
|        1 | 10.0.1.1:20160 |        0 | Up              | 49.98GiB | 46.3
    ↪ GiB | 5h21m52.474864026s |
|        4 | 10.0.1.2:20160 |        0 | Up              | 49.98GiB |
    ↪ 46.32GiB | 5h21m52.522669177s |
|        5 | 10.0.1.3:20160 |        0 | Up              | 49.98GiB |
    ↪ 45.44GiB | 5h21m52.713660541s |
+--
    ↪ --------+-------------------+-------------+-----------------+---------+-
    ↪
3 rows in set (0.00 sec)
```

- Exit TiDB:

```
exit
```

Expected output:

```
Bye
```

5.6 Test Cluster Performance

5.6.1 How to Test TiDB Using Sysbench

It is recommended to use Sysbench 1.0 or later, which can be downloaded here.

5.6.1.1 Test plan

5.6.1.1.1 TiDB configuration

Higher log level means fewer logs to be printed and thus positively influences TiDB performance. Specifically, you can add the following command in the TiUP configuration file:

```
server_configs:
  tidb:
    log.level: "error"
```

It is also recommended to make sure `tidb_enable_prepared_plan_cache` is enabled and that you allow sysbench to use prepared statements by using `--db-ps-mode=auto`. See the SQL Prepared Execution Plan Cache for documentation about what the SQL plan cache does and how to monitor it.

> **Note:**
>
> In different versions of Sysbench, the default value of `db-ps-mode` might be different. It is recommended to explicitly specify it in the command.

5.6.1.1.2 TiKV configuration

Higher log level also means better performance for TiKV.

There are multiple Column Families on TiKV cluster which are mainly used to store different types of data, including Default CF, Write CF, and Lock CF. For the Sysbench test, you only need to focus on Default CF and Write CF. The Column Family that is used to import data has a constant proportion among TiDB clusters:

Default CF : Write CF = 4 : 1

Configuring the block cache of RocksDB on TiKV should be based on the machine's memory size, in order to make full use of the memory. To deploy a TiKV cluster on a 40GB virtual machine, it is recommended to configure the block cache as follows:

```
server_configs:
  tikv:
    log-level: "error"
    rocksdb.defaultcf.block-cache-size: "24GB"
    rocksdb.writecf.block-cache-size: "6GB"
```

You can also configure TiKV to share block cache:

```
server_configs:
  tikv:
    storage.block-cache.capacity: "30GB"
```

For more detailed information on TiKV performance tuning, see Tune TiKV Performance.

5.6.1.2 Test process

> **Note:**
>
> The test in this document was performed without load balancing tools such as HAproxy. We run the Sysbench test on individual TiDB node and added the results up. The load balancing tools and the parameters of different versions might also impact the performance.

5.6.1.2.1 Sysbench configuration

This is an example of the Sysbench configuration file:

```
mysql-host={TIDB_HOST}
mysql-port=4000
mysql-user=root
mysql-password=password
mysql-db=sbtest
time=600
threads={8, 16, 32, 64, 128, 256}
report-interval=10
db-driver=mysql
```

The above parameters can be adjusted according to actual needs. Among them, `TIDB_HOST` is the IP address of the TiDB server (because we cannot include multiple addresses in the configuration file), `threads` is the number of concurrent connections in the test, which can be adjusted in "8, 16, 32, 64, 128, 256". When importing data, it is recommended to set threads = 8 or 16. After adjusting `threads`, save the file named **config**.

See the following as a sample **config** file:

```
mysql-host=172.16.30.33
mysql-port=4000
mysql-user=root
mysql-password=password
mysql-db=sbtest
```

```
time=600
threads=16
report-interval=10
db-driver=mysql
```

5.6.1.2.2 Data import

> **Note:**
>
> If you enable the optimistic transaction model (TiDB uses the pessimistic transaction mode by default), TiDB rolls back transactions when a concurrency conflict is found. Setting `tidb_disable_txn_auto_retry` to `off` turns on the automatic retry mechanism after meeting a transaction conflict, which can prevent Sysbench from quitting because of the transaction conflict error.

Before importing the data, it is necessary to make some settings to TiDB. Execute the following command in MySQL client:

```
set global tidb_disable_txn_auto_retry = off;
```

Then exit the client.

Restart MySQL client and execute the following SQL statement to create a database `sbtest`:

```
create database sbtest;
```

Adjust the order in which Sysbench scripts create indexes. Sysbench imports data in the order of "Build Table -> Insert Data -> Create Index", which takes more time for TiDB to import data. Users can adjust the order to speed up the import of data. Suppose that you use the Sysbench version 1.0.20. You can adjust the order in either of the following two ways:

- Download the modified oltp_common.lua file for TiDB and overwrite the `/usr/share/sysbench/oltp_common.lua` file with it.
- In `/usr/share/sysbench/oltp_common.lua`, move the lines 235-240 to be right behind the line 198.

> **Note:**
>
> This operation is optional and is only to save the time consumed by data import.

At the command line, enter the following command to start importing data. The config file is the one configured in the previous step:

```
sysbench --config-file=config oltp_point_select --tables=32 --table-size
    ↪ =10000000 prepare
```

5.6.1.2.3 Warming data and collecting statistics

To warm data, we load data from disk into the block cache of memory. The warmed data has significantly improved the overall performance of the system. It is recommended to warm data once after restarting the cluster.

```
sysbench --config-file=config oltp_point_select --tables=32 --table-size
    ↪ =10000000 prewarm
```

5.6.1.2.4 Point select test command

```
sysbench --config-file=config oltp_point_select --tables=32 --table-size
    ↪ =10000000 --db-ps-mode=auto --rand-type=uniform run
```

5.6.1.2.5 Update index test command

```
sysbench --config-file=config oltp_update_index --tables=32 --table-size
    ↪ =10000000 --db-ps-mode=auto --rand-type=uniform run
```

5.6.1.2.6 Read-only test command

```
sysbench --config-file=config oltp_read_only --tables=32 --table-size
    ↪ =10000000 --db-ps-mode=auto --rand-type=uniform run
```

5.6.1.3 Common issues

5.6.1.3.1 TiDB and TiKV are both properly configured under high concurrency, why is the overall performance still low?

This issue often has things to do with the use of a proxy. You can add pressure on single TiDB server, sum each result up and compare the summed result with the result with proxy.

Take HAproxy as an example. The parameter `nbproc` can increase the number of processes it can start at most. Later versions of HAproxy also support `nbthread` and `cpu-map`. All of these can mitigate the negative impact of proxy use on performance.

5.6.1.3.2 Under high concurrency, why is the CPU utilization rate of TiKV still low?

Although the overall CPU utilization rate is low for TiKV, the CPU utilization rate of some modules in the cluster might be high.

The maximum concurrency limits for other modules on TiKV, such as storage readpool, coprocessor, and gRPC, can be adjusted through the TiKV configuration file.

The actual CPU usage can be observed through Grafana's TiKV Thread CPU monitor panel. If there is a bottleneck on the modules, it can be adjusted by increasing the concurrency of the modules.

5.6.1.3.3 Given that TiKV has not yet reached the CPU usage bottleneck under high concurrency, why is TiDB's CPU utilization rate still low?

CPU of NUMA architecture is used on some high-end equipment where cross-CPU access to remote memory will greatly reduce performance. By default, TiDB will use all CPUs of the server, and goroutine scheduling will inevitably lead to cross-CPU memory access.

Therefore, it is recommended to deploy n TiDBs (n is the number of NUMA CPUs) on the server of NUMA architecture, and meanwhile set the TiDB parameter `max-procs` to a value that is the same as the number of NUMA CPU cores.

5.6.2 How to Run TPC-C Test on TiDB

This document describes how to test TiDB using TPC-C.

TPC-C is an online transaction processing (OLTP) benchmark. It tests the OLTP system by using a commodity sales model that involves the following five transactions of different types:

- NewOrder
- Payment
- OrderStatus
- Delivery
- StockLevel

5.6.2.1 Prepare

Before testing, TPC-C Benchmark specifies the initial state of the database, which is the rule for data generation in the database. The `ITEM` table contains a fixed number of 100,000 items, while the number of warehouses can be adjusted. If there are W records in the `WAREHOUSE` table, then:

- The `STOCK` table has W * 100,000 records (Each warehouse corresponds to the stock data of 100,000 items)

- The `DISTRICT` table has W * 10 records (Each warehouse provides services to 10 districts)
- The `CUSTOMER` table has W * 10 * 3,000 records (Each district has 3,000 customers)
- The `HISTORY` table has W * 10 * 3,000 records (Each customer has one transaction history)
- The `ORDER` table has W * 10 * 3,000 records (Each district has 3,000 orders and the last 900 orders generated are added to the `NEW-ORDER` table. Each order randomly generates 5 ~ 15 ORDER-LINE records.

In this document, the testing uses 1,000 warehouses as an example to test TiDB.

TPC-C uses tpmC (transactions per minute) to measure the maximum qualified throughput (MQTh, Max Qualified Throughput). The transactions are the NewOrder transactions and the final unit of measure is the number of new orders processed per minute.

The test in this document is implemented based on go-tpc. You can download the test program using TiUP commands.

```
tiup install bench
```

For detailed usage of the TiUP Bench component, see TiUP Bench.

5.6.2.2 Load data

Loading data is usually the most time-consuming and problematic stage of the entire TPC-C test. This section provides the following command to load data.

Execute the following TiUP command in Shell:

```
tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 1000 prepare
```

Based on different machine configurations, this loading process might take a few hours. If the cluster size is small, you can use a smaller `WAREHOUSE` value for the test.

After the data is loaded, you can execute the `tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 4 check` command to validate the data correctness.

5.6.2.3 Run the test

Execute the following command to run the test:

```
tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 1000 run
```

During the test, test results are continuously printed on the console:

```
[Current] NEW_ORDER - Takes(s): 4.6, Count: 5, TPM: 65.5, Sum(ms): 4604, Avg
    ↪ (ms): 920, 90th(ms): 1500, 99th(ms): 1500, 99.9th(ms): 1500
[Current] ORDER_STATUS - Takes(s): 1.4, Count: 1, TPM: 42.2, Sum(ms): 256,
    ↪ Avg(ms): 256, 90th(ms): 256, 99th(ms): 256, 99.9th(ms): 256
```

```
[Current] PAYMENT - Takes(s): 6.9, Count: 5, TPM: 43.7, Sum(ms): 2208, Avg(
    ↪ ms): 441, 90th(ms): 512, 99th(ms): 512, 99.9th(ms): 512
[Current] STOCK_LEVEL - Takes(s): 4.4, Count: 1, TPM: 13.8, Sum(ms): 224,
    ↪ Avg(ms): 224, 90th(ms): 256, 99th(ms): 256, 99.9th(ms): 256
...
```

After the test is finished, the test summary results are printed:

```
[Summary] DELIVERY - Takes(s): 455.2, Count: 32, TPM: 4.2, Sum(ms): 44376,
    ↪ Avg(ms): 1386, 90th(ms): 2000, 99th(ms): 4000, 99.9th(ms): 4000
[Summary] DELIVERY_ERR - Takes(s): 455.2, Count: 1, TPM: 0.1, Sum(ms): 953,
    ↪ Avg(ms): 953, 90th(ms): 1000, 99th(ms): 1000, 99.9th(ms): 1000
[Summary] NEW_ORDER - Takes(s): 487.8, Count: 314, TPM: 38.6, Sum(ms):
    ↪ 282377, Avg(ms): 899, 90th(ms): 1500, 99th(ms): 1500, 99.9th(ms):
    ↪ 1500
[Summary] ORDER_STATUS - Takes(s): 484.6, Count: 29, TPM: 3.6, Sum(ms):
    ↪ 8423, Avg(ms): 290, 90th(ms): 512, 99th(ms): 1500, 99.9th(ms): 1500
[Summary] PAYMENT - Takes(s): 490.1, Count: 321, TPM: 39.3, Sum(ms): 144708,
    ↪  Avg(ms): 450, 90th(ms): 512, 99th(ms): 1000, 99.9th(ms): 1500
[Summary] STOCK_LEVEL - Takes(s): 487.6, Count: 41, TPM: 5.0, Sum(ms): 9318,
    ↪  Avg(ms): 227, 90th(ms): 512, 99th(ms): 1000, 99.9th(ms): 1000
```

After the test is finished, you can execute the `tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 4 check` command to validate the data correctness.

5.6.2.4 Clean up test data

Execute the following command to clean up the test data:

```
tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 4 cleanup
```

5.6.3 How to Run CH-benCHmark Test on TiDB

This document describes how to test TiDB using CH-benCHmark.

CH-benCHmark is a mixed workload containing both TPC-C and TPC-H tests. It is the most common workload to test HTAP systems. For more information, see The mixed workload CH-benCHmark.

Before running the CH-benCHmark test, you need to deploy TiFlash first, which is a TiDB's HTAP component. After you deploy TiFlash and create TiFlash replicas, TiKV will replicate the latest data of TPC-C online transactions to TiFlash in real time, and the TiDB optimizer will automatically push down OLAP queries from TPC-H workload to the MPP engine of TiFlash for efficient execution.

The CH-benCHmark test in this document is implemented based on go-tpc. You can download the test program using the following TiUP command:

```
tiup install bench
```

For detailed usage of the TiUP Bench component, see TiUP Bench.

5.6.3.1 Load data

5.6.3.1.1 Load TPC-C data

Loading data is usually the most time-consuming and problematic stage of the entire TPC-C test.

Taking 1000 warehouses as an example, you can execute the following TiUP command in shell for data load and test. Note that you need to replace `172.16.5.140` and `4000` in this document with your TiDB host and port values.

```
tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 1000 prepare -T
    ↪ 32
```

Depending on different machine configurations, this loading process might take a few hours. If the cluster size is small, you can use a smaller warehouse value for the test.

After the data is loaded, you can execute the `tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 1000 check` command to validate the data correctness.

5.6.3.1.2 Load additional tables and views required for TPC-H

Run the following TiUP command in the shell:

```
tiup bench ch -H 172.16.5.140 -P 4000 -D tpcc prepare
```

The following is the log output:

```
creating nation
creating region
creating supplier
generating nation table
generate nation table done
generating region table
generate region table done
generating suppliers table
generate suppliers table done
creating view revenue1
```

5.6.3.2 Create TiFlash replicas

After TiFlash is deployed, TiFlash does not automatically replicate TiKV data. You need to execute the following SQL statement to create TiFlash replicas for the `tpcc` database. Once the specified TiFlash replicas are created, TiKV automatically replicates the latest data to TiFlash in real-time. In the following example, two TiFlash nodes are deployed in the cluster and the replica number is set to 2.

```
ALTER DATABASE tpcc SET tiflash replica 2;
```

To check whether the replication of all tables in the `tpcc` database is complete, execute the following statement, in which the `WHERE` clause is used to specify the databases and tables to be checked. If you want to check the replication status of all databases, remove the `WHERE` clause from the statement.

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = 'tpcc
    ↪ ';
```

In the result of the above statement:

- `AVAILABLE` indicates whether the TiFlash replica of a specific table is available or not. `1` means available and `0` means unavailable. Once a replica becomes available, this status does not change anymore. If you use DDL statements to modify the number of replicas, the replication progress will be recalculated.
- `PROGRESS` means the progress of the replication. The value is between `0.0` and `1.0`. `1.0` means at least one replica is replicated.

5.6.3.3 Collect statistics

To ensure that the TiDB optimizer can generate the optimal execution plan, execute the following SQL statements to collect statistics in advance.

```
analyze table customer;
analyze table district;
analyze table history;
analyze table item;
analyze table new_order;
analyze table order_line;
analyze table orders;
analyze table stock;
analyze table warehouse;
analyze table nation;
analyze table region;
analyze table supplier;
```

5.6.3.4 Run the test

Taking 50 TP concurrency and 1 AP concurrency as an example, execute the following command to run the test:

```
go-tpc ch --host 172.16.5.140 -P4000 --warehouses 1000 run -D tpcc -T 50 -t
    ↪ 1 --time 1h
```

During the test, test results are continuously printed on the console. For example:

```
[Current] NEW_ORDER - Takes(s): 10.0, Count: 13524, TPM: 81162.0, Sum(ms):
    ↪ 998317.6, Avg(ms): 73.9, 50th(ms): 71.3, 90th(ms): 100.7, 95th(ms):
    ↪ 113.2, 99th(ms): 159.4, 99.9th(ms): 209.7, Max(ms): 243.3
[Current] ORDER_STATUS - Takes(s): 10.0, Count: 1132, TPM: 6792.7, Sum(ms):
    ↪ 16196.6, Avg(ms): 14.3, 50th(ms): 13.1, 90th(ms): 24.1, 95th(ms):
    ↪ 27.3, 99th(ms): 37.7, 99.9th(ms): 50.3, Max(ms): 52.4
[Current] PAYMENT - Takes(s): 10.0, Count: 12977, TPM: 77861.1, Sum(ms):
    ↪ 773982.0, Avg(ms): 59.7, 50th(ms): 56.6, 90th(ms): 88.1, 95th(ms):
    ↪ 100.7, 99th(ms): 151.0, 99.9th(ms): 201.3, Max(ms): 243.3
[Current] STOCK_LEVEL - Takes(s): 10.0, Count: 1134, TPM: 6806.0, Sum(ms):
    ↪ 31220.9, Avg(ms): 27.5, 50th(ms): 25.2, 90th(ms): 37.7, 95th(ms):
    ↪ 44.0, 99th(ms): 71.3, 99.9th(ms): 117.4, Max(ms): 125.8
[Current] Q11  - Count: 1, Sum(ms): 3682.9, Avg(ms): 3683.6
[Current] DELIVERY - Takes(s): 10.0, Count: 1167, TPM: 7002.6, Sum(ms):
    ↪ 170712.9, Avg(ms): 146.3, 50th(ms): 142.6, 90th(ms): 192.9, 95th(ms):
    ↪  209.7, 99th(ms): 251.7, 99.9th(ms): 335.5, Max(ms): 385.9
[Current] NEW_ORDER - Takes(s): 10.0, Count: 13238, TPM: 79429.5, Sum(ms):
    ↪ 1010795.3, Avg(ms): 76.4, 50th(ms): 75.5, 90th(ms): 104.9, 95th(ms):
    ↪ 117.4, 99th(ms): 159.4, 99.9th(ms): 234.9, Max(ms): 352.3
[Current] ORDER_STATUS - Takes(s): 10.0, Count: 1224, TPM: 7350.6, Sum(ms):
    ↪ 17874.1, Avg(ms): 14.6, 50th(ms): 13.6, 90th(ms): 23.1, 95th(ms):
    ↪ 27.3, 99th(ms): 37.7, 99.9th(ms): 54.5, Max(ms): 60.8
[Current] PAYMENT - Takes(s): 10.0, Count: 12650, TPM: 75901.1, Sum(ms):
    ↪ 761981.3, Avg(ms): 60.3, 50th(ms): 56.6, 90th(ms): 88.1, 95th(ms):
    ↪ 104.9, 99th(ms): 159.4, 99.9th(ms): 218.1, Max(ms): 318.8
[Current] STOCK_LEVEL - Takes(s): 10.0, Count: 1179, TPM: 7084.9, Sum(ms):
    ↪ 32829.8, Avg(ms): 27.9, 50th(ms): 26.2, 90th(ms): 37.7, 95th(ms):
    ↪ 44.0, 99th(ms): 71.3, 99.9th(ms): 100.7, Max(ms): 117.4
[Current] Q12  - Count: 1, Sum(ms): 9945.8, Avg(ms): 9944.7
[Current] Q13  - Count: 1, Sum(ms): 1729.6, Avg(ms): 1729.6
...
```

After the test is finished, the test summary results are printed. For example:

```
Finished: 50 OLTP workers, 1 OLAP workers
[Summary] DELIVERY - Takes(s): 3599.7, Count: 501795, TPM: 8363.9, Sum(ms):
    ↪ 63905178.8, Avg(ms): 127.4, 50th(ms): 125.8, 90th(ms): 167.8, 95th(ms
    ↪ ): 184.5, 99th(ms): 226.5, 99.9th(ms): 318.8, Max(ms): 604.0
```

```
[Summary] DELIVERY_ERR - Takes(s): 3599.7, Count: 14, TPM: 0.2, Sum(ms):
   ↪ 1027.7, Avg(ms): 73.4, 50th(ms): 71.3, 90th(ms): 109.1, 95th(ms):
   ↪ 109.1, 99th(ms): 113.2, 99.9th(ms): 113.2, Max(ms): 113.2
[Summary] NEW_ORDER - Takes(s): 3599.7, Count: 5629221, TPM: 93826.9, Sum(ms
   ↪ ): 363758020.7, Avg(ms): 64.6, 50th(ms): 62.9, 90th(ms): 88.1, 95th(
   ↪ ms): 100.7, 99th(ms): 130.0, 99.9th(ms): 184.5, Max(ms): 570.4
[Summary] NEW_ORDER_ERR - Takes(s): 3599.7, Count: 20, TPM: 0.3, Sum(ms):
   ↪ 404.2, Avg(ms): 20.2, 50th(ms): 18.9, 90th(ms): 37.7, 95th(ms): 50.3,
   ↪  99th(ms): 56.6, 99.9th(ms): 56.6, Max(ms): 56.6
[Summary] ORDER_STATUS - Takes(s): 3599.8, Count: 500318, TPM: 8339.0, Sum(
   ↪ ms): 7135956.6, Avg(ms): 14.3, 50th(ms): 13.1, 90th(ms): 24.1, 95th(
   ↪ ms): 27.3, 99th(ms): 37.7, 99.9th(ms): 50.3, Max(ms): 385.9
[Summary] PAYMENT - Takes(s): 3599.8, Count: 5380815, TPM: 89684.8, Sum(ms):
   ↪  269863092.5, Avg(ms): 50.2, 50th(ms): 48.2, 90th(ms): 75.5, 95th(ms)
   ↪ : 88.1, 99th(ms): 125.8, 99.9th(ms): 184.5, Max(ms): 1073.7
[Summary] PAYMENT_ERR - Takes(s): 3599.8, Count: 11, TPM: 0.2, Sum(ms):
   ↪ 313.0, Avg(ms): 28.5, 50th(ms): 10.0, 90th(ms): 67.1, 95th(ms): 67.1,
   ↪  99th(ms): 88.1, 99.9th(ms): 88.1, Max(ms): 88.1
[Summary] STOCK_LEVEL - Takes(s): 3599.8, Count: 500467, TPM: 8341.5, Sum(ms
   ↪ ): 13208726.4, Avg(ms): 26.4, 50th(ms): 25.2, 90th(ms): 37.7, 95th(ms
   ↪ ): 44.0, 99th(ms): 62.9, 99.9th(ms): 96.5, Max(ms): 570.4
[Summary] STOCK_LEVEL_ERR - Takes(s): 3599.8, Count: 2, TPM: 0.0, Sum(ms):
   ↪ 7.6, Avg(ms): 3.7, 50th(ms): 3.1, 90th(ms): 4.7, 95th(ms): 4.7, 99th(
   ↪ ms): 4.7, 99.9th(ms): 4.7, Max(ms): 4.7
tpmC: 93826.9, efficiency: 729.6%
[Summary] Q1   - Count: 11, Sum(ms): 42738.2, Avg(ms): 3885.3
[Summary] Q10  - Count: 11, Sum(ms): 440370.3, Avg(ms): 40034.3
[Summary] Q11  - Count: 11, Sum(ms): 44208.6, Avg(ms): 4018.7
[Summary] Q12  - Count: 11, Sum(ms): 105320.3, Avg(ms): 9574.6
[Summary] Q13  - Count: 11, Sum(ms): 19199.5, Avg(ms): 1745.4
[Summary] Q14  - Count: 11, Sum(ms): 84582.1, Avg(ms): 7689.5
[Summary] Q15  - Count: 11, Sum(ms): 271944.8, Avg(ms): 24722.8
[Summary] Q16  - Count: 11, Sum(ms): 183894.9, Avg(ms): 16718.1
[Summary] Q17  - Count: 11, Sum(ms): 89018.9, Avg(ms): 8092.7
[Summary] Q18  - Count: 10, Sum(ms): 767814.5, Avg(ms): 76777.6
[Summary] Q19  - Count: 10, Sum(ms): 17099.1, Avg(ms): 1709.8
[Summary] Q2   - Count: 11, Sum(ms): 53513.6, Avg(ms): 4865.2
[Summary] Q20  - Count: 10, Sum(ms): 73717.7, Avg(ms): 7372.1
[Summary] Q21  - Count: 10, Sum(ms): 166001.4, Avg(ms): 16601.1
[Summary] Q22  - Count: 10, Sum(ms): 48268.4, Avg(ms): 4827.7
[Summary] Q3   - Count: 11, Sum(ms): 31110.1, Avg(ms): 2828.5
[Summary] Q4   - Count: 11, Sum(ms): 83814.2, Avg(ms): 7619.3
[Summary] Q5   - Count: 11, Sum(ms): 368301.0, Avg(ms): 33483.5
[Summary] Q6   - Count: 11, Sum(ms): 61702.5, Avg(ms): 5608.9
[Summary] Q7   - Count: 11, Sum(ms): 158928.2, Avg(ms): 14446.3
```

After the test is finished, you can execute the `tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 1000 check` command to validate the data correctness.

6 Migrate

6.1 Data Migration Overview

This document gives an overview of the data migration solutions that you can use with TiDB. The data migration solutions are as follows:

- Full data migration.
    - To import Amazon Aurora snapshots, CSV files, or Mydumper SQL files into TiDB, you can use TiDB Lightning to perform the full migration.
    - To export all TiDB data as CSV files or Mydumper SQL files, you can use Dumpling to perform the full migration, which makes data migration from MySQL or MariaDB easier.
    - To migrate all data from a database with a small data size volume (for example, less than 1 TiB), you can also use TiDB Data Migration (DM).
- Quick initialization of TiDB. TiDB Lightning supports quickly importing data and can quickly initialize a specific table in TiDB. Before you use this feature, pay attention that the quick initialization has a great impact on TiDB and the cluster does not provide services during the initialization period.
- Incremental replication. You can use TiDB DM to replicate binlogs from MySQL, MariaDB, or Aurora to TiDB, which greatly reduces the window downtime during the replication period.
- Data replication between TiDB clusters. TiDB supports backup and restore. This feature can initialize a snapshot in an existing TiDB cluster to a new TiDB cluster.

You might choose different migration solutions according to the database type, deployment location, application data size, and application needs. The following sections introduce some common migration scenarios, and you can refer to these sections to determine the most suitable solution according to your needs.

6.1.1 Migrate data from Aurora MySQL to TiDB

When you migrate data from Aurora to a TiDB cluster deployed on AWS, your data migration takes two operations: full data migration and incremental replication. You can choose the corresponding operation according to your application needs.

- Migrate Data from Amazon Aurora to TiDB.

6.1.2 Migrate data from MySQL to TiDB

If cloud storage (S3) service is not used, the network connectivity is good, and the network latency is low, you can use the following method to migrate data from MySQL to TiDB.

- Migrate MySQL of Small Datasets to TiDB

If you have a high demand on migration speed, or if the data size is large (for example, larger than 1 TiB), and you do not allow other applications to write to TiDB during the migration period, you can use TiDB Lightning to quickly import data. Then, you can use DM to replicate incremental data (binlog) based on your application needs.

- Migrate MySQL of Large Datasets to TiDB

6.1.3 Migrate and merge MySQL shards into TiDB

Suppose that your application uses MySQL shards for data storage, and you need to migrate these shards into TiDB as one table. In this case, you can use DM to perform the shard merge and migration.

- Migrate and Merge MySQL Shards of Small Datasets to TiDB

If the data size of the sharded tables is large (for example, larger than 1 TiB), and you do not allow other applications to write to TiDB during the migration period, you can use TiDB Lightning to quickly merge and import the sharded tables. Then, you can use DM to replicate incremental sharding data (binlog) based on your application needs.

- Migrate and Merge MySQL Shards of Large Datasets to TiDB

6.1.4 Migrate data from files to TiDB

- Migrate data from CSV files to TiDB
- Migrate data from SQL files to TiDB

6.1.5 More advanced migration solutions

The following features can improve the migration process and might meet more needs in your application.

- Continuous Replication from Databases that Use gh-ost or pt-osc
- Migrate Data to a Downstream TiDB Table with More Columns
- Filter Binlog Events
- Filter DML Events Using SQL Expressions

6.2 TiDB Migration Tools Overview

TiDB provides multiple data migration tools for different scenarios such as full data migration, incremental data migration, backup and restore, and data replication.

This document introduces the user scenarios, supported upstreams and downstreams, advantages, and limitations of these tools. You can choose the right tool according to your needs.

6.2.1 TiDB Data Migration (DM)

User scenario	Data migration from MySQL-compatible databases to TiDB
Upstream	MySQL, MariaDB, Aurora
Downstream	TiDB

| **Advantages** |

A convenient and unified data migration task management tool that supports full data migration and incremental replication

Support filtering tables and operations

Support shard merge and migration

| | **Limitation** | Data import speed is roughly the same as that of TiDB Lightning's logical import mode, and a lot lower than that of TiDB Lightning's physical import mode. So it is recommended to use DM to migrate full data with a size of less than 1 TiB. |

6.2.2 TiDB Lightning

User scenario	Full data import into TiDB

| **Upstream (the imported source file)** |

Files exported from Dumpling

Parquet files exported by Amazon Aurora or Apache Hive

CSV files

Data from local disks or Amazon S3

| | **Downstream** | TiDB | | **Advantages** |

Support quickly importing a large amount of data and quickly initializing a specific table in a TiDB cluster

Support checkpoints to store the import progress, so that `tidb-lightning` continues importing from where it lefts off after restarting

Support data filtering

| | **Limitation** |

If physical import mode is used for data import, during the import process, the TiDB cluster cannot provide services.

If you do not want the TiDB services to be impacted, perform the data import according to TiDB Lightning logical import mode.

|

6.2.3 Dumpling

User scenario	Full data export from MySQL or TiDB
Upstream	MySQL, TiDB
Downstream (the output file)	SQL, CSV

| **Advantages** |

Support the table-filter feature that enables you to filter data easier

Support exporting data to Amazon S3

| | **Limitation** |

If you want to restore the exported data to a database other than TiDB, it is recommended to use Dumpling.

If you want to restore the exported data to another TiDB cluster, it is recommended to use Backup & Restore (BR).

|

6.2.4 TiCDC

	This tool is implemented by pulling TiKV change logs. It can restore cluster data to a consistent state with any upstream TSO, and support other systems to subscribe to data changes.
User scenario	
Upstream	TiDB

	This tool is implemented by pulling TiKV change logs. It can restore cluster data to a consistent state with any upstream TSO, and support other systems to subscribe to data changes.
User scenario	
Downstream	TiDB, MySQL, Apache Pulsar, Kafka, Confluent

	This tool is implemented by pulling TiKV change logs. It can restore cluster data to a consistent state with any upstream TSO, and support other systems to subscribe to data changes.
User scenario	
Advantages	Provide TiCDC Open Protocol

	This tool is implemented by pulling TiKV change logs. It can restore cluster data to a consistent state with any upstream TSO, and support other systems to subscribe to data changes.
User scenario	

| **Limitation** | TiCDC only replicates tables that have at least one valid index. The following scenarios are not supported:

The TiKV cluster that uses RawKV alone.

The DDL operation `CREATE SEQUENCE` and the `SEQUENCE` function in TiDB.

|

6.2.5 Backup & Restore (BR)

	Migrate a large amount of TiDB cluster data by backing up and restoring data
User scenario	
Upstream or Downstream (the output file)	TiDB SST, backup.meta files, backup.lock files

| **Advantages** |

Suitable for migrating data to another TiDB cluster

Support backing up data to an external storage for disaster recovery

| | **Limitation** |

When BR restores data to the upstream cluster of TiCDC or Drainer, the restored data cannot be replicated to the downstream by TiCDC or Drainer.

BR supports operations only between clusters that have the same `new_collations_enabled_on_first_` ↪ value.

|

6.2.6 sync-diff-inspector

User scenario	Comparing data stored in the databases with the MySQL protocol
Upstream	TiDB, MySQL
Downstream	TiDB, MySQL
Advantages	Can be used to repair data in the scenario where a small amount of data is inconsistent

Limitation

Online check is not supported for data migration between MySQL and TiDB.

JSON, BIT, BINARY, BLOB and other types of data are not supported.

|

6.2.7 Install tools using TiUP

Since TiDB v4.0, TiUP acts as a package manager that helps you manage different cluster components in the TiDB ecosystem. Now you can manage any cluster component using a single command.

6.2.7.1 Step 1. Install TiUP

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

Redeclare the global environment variable:

```
source ~/.bash_profile
```

6.2.7.2 Step 2. Install components

You can use the following command to see all the available components:

```
tiup list
```

The command output lists all the available components:

```
Available components:
Name           Owner   Description
----           -----   -----------
bench          pingcap Benchmark database with different workloads
br             pingcap TiDB/TiKV cluster backup restore tool
cdc            pingcap CDC is a change data capture tool for TiDB
client         pingcap Client to connect playground
cluster        pingcap Deploy a TiDB cluster for production
ctl            pingcap TiDB controller suite
dm             pingcap Data Migration Platform manager
dmctl          pingcap dmctl component of Data Migration Platform
errdoc         pingcap Document about TiDB errors
pd-recover     pingcap PD Recover is a disaster recovery tool of PD, used to
    ↪ recover the PD cluster which cannot start or provide services
    ↪ normally
playground     pingcap Bootstrap a local TiDB cluster for fun
tidb           pingcap TiDB is an open source distributed HTAP database
    ↪ compatible with the MySQL protocol
tidb-lightning pingcap TiDB Lightning is a tool used for fast full import
    ↪ of large amounts of data into a TiDB cluster
```

```
tiup          pingcap TiUP is a command-line component management tool that
    ↪  can help to download and install TiDB platform components to the
    ↪ local system
```

Choose the components to install:

```
tiup install dumpling tidb-lightning
```

> **Note:**
>
> To install a component of a specific version, use the `tiup install <component>[:version]` command.

6.2.7.3 Step 3. Update TiUP and its components (optional)

It is recommended to see the release log and compatibility notes of the new version.

```
tiup update --self && tiup update dm
```

6.2.8 See also

- Deploy TiUP offline
- Download and install tools in binary

6.3 Migration Scenarios

6.3.1 Migrate Data from Amazon Aurora to TiDB

This document describes how to migrate data from Amazon Aurora to TiDB. The migration process uses DB snapshot, which saves a lot of space and time.

The whole migration has two processes:

- Import full data to TiDB using TiDB Lightning
- Replicate incremental data to TiDB using DM (optional)

6.3.1.1 Prerequisites

- Install Dumpling and TiDB Lightning
- Get the target database privileges required for TiDB Lightning.

6.3.1.2 Import full data to TiDB

6.3.1.2.1 Step 1. Export an Aurora snapshot to Amazon S3

1. In Aurora, query the current binlog position by running the following command:

```
mysql> SHOW MASTER STATUS;
```

The output is similar to the following. Record the binlog name and position for later use.

```
+------------------+----------+--------------+------------------+-------------------
    ↪
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
    ↪ Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------
    ↪
| mysql-bin.000002 | 52806 |              |                  |
    ↪                  |
+------------------+----------+--------------+------------------+-------------------
    ↪
1 row in set (0.012 sec)
```

2. Export the Aurora snapshot. For detailed steps, refer to Exporting DB snapshot data to Amazon S3.

After you obtain the binlog position, export the snapshot within 5 minutes. Otherwise, the recorded binlog position might be outdated and thus cause data conflict during the incremental replication.

After the two steps above, make sure you have the following information ready:

- The Aurora binlog name and position at the time of the snapshot creation.
- The S3 path where the snapshot is stored, and the SecretKey and AccessKey with access to the S3 path.

6.3.1.2.2 Step 2. Export schema

Because the snapshot file from Aurora does not contain the DDL statements, you need to export the schema using Dumpling and create the schema in the target database using TiDB Lightning. If you want to manually create the schema, you can skip this step.

Export the schema using Dumpling by running the following command. The command includes the `--filter` parameter to only export the desired table schema:

```
tiup dumpling --host ${host} --port 3306 --user root --password ${password}
    ↪ --filter 'my_db1.table[12]' --no-data --output 's3://my-bucket/schema
    ↪ -backup?region=us-west-2' --filter "mydb.*"
```

The parameters used in the command above are as follows. For more parameters, refer to Dumpling overview.

Parameter	Description
-u or --user	Aurora MySQL user
-p or --password	MySQL user password
-P or --port	MySQL port
-H or --host	MySQL HP address

Parameter	Description
`-t` or `--thread`	The number of threads for export
`-o` or `--output`	The directory that stores the exported file. Supports local path or external storage URL

Parameter	Description
`--max-row-num`	The maximum number of rows in a single file
`--max-row-size`	The maximum size of a single file, in MiB. Recommended value: 256 MiB.
`--database`	Specifies a database to be exported

Parameter	Description
`-`	Exports
↪ `T`	the
↪	spec-
or	i-
`--`	fied
↪ `tables`	tables
↪	-bles
↪ `list`	
↪	
`-`	Does
↪ `d`	not
↪	ex-
or	port
`--`	data.
↪ `no`	Only
↪	-ex-
↪ `data`	ports
↪	schema.

Parameter	Description
-f	Exports files that match the filter pattern. Do not use -f and -T at the same time. Refer to table-filter for the syntax.

6.3.1.2.3 Step 3. Create the TiDB Lightning configuration file

Create the `tidb-lightning.toml` configuration file as follows:

```
vim tidb-lightning.toml
```

```
[tidb]

### The target TiDB cluster information.
host = ${host}             # e.g.: 172.16.32.1
port = ${port}             # e.g.: 4000
user = "${user_name}       # e.g.: "root"
```

```
password = "${password}"  # e.g.: "rootroot"
status-port = ${status-port} # Obtains the table schema information from
    ↪ TiDB status port, e.g.: 10080
pd-addr = "${ip}:${port}" # The cluster PD address, e.g.: 172.16.31.3:2379.
    ↪  TiDB Lightning obtains some information from PD. When backend = "
    ↪ local", you must specify status-port and pd-addr correctly. Otherwise
    ↪ , the import will be abnormal.

[tikv-importer]
### "local": Default backend. The local backend is recommended to import
    ↪ large volumes of data (1 TiB or more). During the import, the target
    ↪ TiDB cluster cannot provide any service.
### "tidb": The "tidb" backend is recommended to import data less than 1 TiB
    ↪ . During the import, the target TiDB cluster can provide service
    ↪ normally.
backend = "local"

### Set the temporary storage directory for the sorted Key-Value files. The
    ↪ directory must be empty, and the storage space must be greater than
    ↪ the size of the dataset to be imported. For better import performance
    ↪ , it is recommended to use a directory different from `data-source-
    ↪ dir` and use flash storage, which can use I/O exclusively.
sorted-kv-dir = "/mnt/ssd/sorted-kv-dir"

[mydumper]
### The path that stores the snapshot file.
data-source-dir = "${s3_path}" # e.g.: s3://my-bucket/sql-backup?region=us-
    ↪ west-2

[[mydumper.files]]
### The expression that parses the parquet file.
pattern = '(?i)^(?:[^/]*/)*([a-z0-9_]+)\.([a-z0-9_]+)/(?:[^/]*/)*(?:[a-z0
    ↪ -9\-_.]+\.(parquet))$'
schema = '$1'
table = '$2'
type = '$3'
```

If you need to enable TLS in the TiDB cluster, refer to TiDB Lightning Configuration.

6.3.1.2.4 Step 4. Import full data to TiDB

1. Create the tables in the target database using TiDB Lightning:

```
tiup tidb-lightning -config tidb-lightning.toml -d 's3://my-bucket/
    ↪ schema-backup?region=us-west-2'
```

2. Start the import by running `tidb-lightning`. If you launch the program directly in the command line, the process might exit unexpectedly after receiving a SIGHUP signal. In this case, it is recommended to run the program using a `nohup` or `screen` tool. For example:

   Pass the SecretKey and AccessKey that have access to the S3 storage path as environment variables to the Dumpling node. You can also read the credentials from `~/.aws/credentials`.

```
export AWS_ACCESS_KEY_ID=${access_key}
export AWS_SECRET_ACCESS_KEY=${secret_key}
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out 2>&1
    ↪ &
```

3. After the import starts, you can check the progress of the import by either of the following methods:

   - `grep` the keyword `progress` in the log. The progress is updated every 5 minutes by default.
   - Check progress in the monitoring dashboard.
   - Check progress in the TiDB Lightning web interface.

4. After TiDB Lightning completes the import, it exits automatically. Check whether `tidb-lightning.log` contains `the whole procedure completed` in the last lines. If yes, the import is successful. If no, the import encounters an error. Address the error as instructed in the error message.

> **Note:**
>
> Whether the import is successful or not, the last line of the log shows `tidb ↪ lightning exit`. It means that TiDB Lightning exits normally, but does not necessarily mean that the import is successful.

If you encounter any problem during the import, refer to TiDB Lightning FAQ for troubleshooting.

6.3.1.3 Replicate incremental data to TiDB (optional)

6.3.1.3.1 Prerequisites

- Install DM.
- Get the source database and target database privileges required for DM.

6.3.1.3.2 Step 1: Create the data source

1. Create the `source1.yaml` file as follows:

```
# Must be unique.
source-id: "mysql-01"
# Configures whether DM-worker uses the global transaction identifier (
    ↪ GTID) to pull binlogs. To enable this mode, the upstream MySQL
    ↪ must also enable GTID. If the upstream MySQL service is
    ↪ configured to switch master between different nodes automatically
    ↪ , GTID mode is required.
enable-gtid: false

from:
  host: "${host}"      # e.g.: 172.16.10.81
  user: "root"
  password: "${password}" # Supported but not recommended to use
      ↪ plaintext password. It is recommended to use `dmctl encrypt` to
      ↪  encrypt the plaintext password before using it.
  port: 3306
```

2. Load the data source configuration to the DM cluster using `tiup dmctl` by running the following command:

```
tiup dmctl --master-addr ${advertise-addr} operate-source create
    ↪ source1.yaml
```

   The parameters used in the command above are described as follows:

Parameter	Description
--master-addr	The {master-addr} of any DM-master in the cluster where dmctl is to be connected, e.g.: 172.16.10.71:8261
operate-source	Loads the source configuration to the DM cluster.

6.3.1.3.3 Step 2: Create the migration task

Create the `task1.yaml` file as follows:

```
### Task name. Multiple tasks that are running at the same time must each have a unique name.
name: "test"
### Task mode. Options are:
### - full: only performs full data migration.
```

```
### - incremental: only performs binlog real-time replication.
### - all: full data migration + binlog real-time replication.
task-mode: "incremental"
### The configuration of the target TiDB database.
target-database:
  host: "${host}"               # e.g.: 172.16.10.83
  port: 4000
  user: "root"
  password: "${password}"       # Supported but not recommended to use a
     ↪ plaintext password. It is recommended to use `dmctl encrypt` to
     ↪ encrypt the plaintext password before using it.

### Global configuration for block and allow lists. Each instance can
   ↪ reference the configuration by name.
block-allow-list:                # If the DM version is earlier than v2
   ↪ .0.0-beta.2, use black-white-list.
  listA:                         # Name.
    do-tables:                   # Allow list for the upstream tables to be
       ↪  migrated.
    - db-name: "test_db"         # Name of databases to be migrated.
      tbl-name: "test_table"     # Name of tables to be migrated.

### Configures the data source.
mysql-instances:
  - source-id: "mysql-01"          # Data source ID, i.e., source-id in
     ↪ source1.yaml
    block-allow-list: "listA"      # References the block-allow-list
       ↪ configuration above.
###      syncer-config-name: "global" # Name of the syncer configuration.
    meta:                          # When task-mode is "incremental" and the
       ↪  downstream database does not have a checkpoint, DM uses the
       ↪ binlog position as the starting point. If the downstream database
       ↪ has a checkpoint, DM uses the checkpoint as the starting point.
      binlog-name: "mysql-bin.000004" # The binlog position recorded in "
         ↪ Step 1. Export an Aurora snapshot to Amazon S3". When the
         ↪ upstream database has source-replica switching, GTID mode is
         ↪ required.
      binlog-pos: 109227
      # binlog-gtid: "09bec856-ba95-11ea-850a-58f2b4af5188:1-9"

### (Optional) If you need to incrementally replicate data that has already
   ↪ been migrated in the full data migration, you need to enable the safe
   ↪  mode to avoid the incremental data replication error.
   # This scenario is common in the following case: the full migration data
      ↪ does not belong to the data source's consistency snapshot, and
```

```
    ↪ after that, DM starts to replicate incremental data from a position
    ↪  earlier than the full migration.
  # syncers:          # The running configurations of the sync processing
    ↪ unit.
  #   global:          # Configuration name.
  #     safe-mode: true # If this field is set to true, DM changes INSERT of
    ↪  the data source to REPLACE for the target database, and changes
    ↪ UPDATE of the data source to DELETE and REPLACE for the target
    ↪ database. This is to ensure that when the table schema contains a
    ↪ primary key or unique index, DML statements can be imported
    ↪ repeatedly. In the first minute of starting or resuming an
    ↪ incremental replication task, DM automatically enables the safe
    ↪ mode.
```

The YAML file above is the minimum configuration required for the migration task. For more configuration items, refer to DM Advanced Task Configuration File.

6.3.1.3.4 Step 3. Run the migration task

Before you start the migration task, to reduce the probability of errors, it is recommended to confirm that the configuration meets the requirements of DM by running the `check-task` command:

```
tiup dmctl --master-addr ${advertise-addr} check-task task.yaml
```

After that, start the migration task by running `tiup dmctl`:

```
tiup dmctl --master-addr ${advertise-addr} start-task task.yaml
```

The parameters used in the command above are described as follows:

Parameter	Description
`--master-addr`	The {master-advertise-addr} of any DM-master in the cluster where `dmctl` is to be connected, e.g.: 172.16.10.71:8261
`start-task`	Starts the migration task.

If the task fails to start, check the prompt message and fix the configuration. After that, you can re-run the command above to start the task.

If you encounter any problem, refer to DM error handling and DM FAQ.

6.3.1.3.5 Step 4. Check the migration task status

To learn whether the DM cluster has an ongoing migration task and the task status, run the `query-status` command using `tiup dmctl`:

```
tiup dmctl --master-addr ${advertise-addr} query-status ${task-name}
```

For a detailed interpretation of the results, refer to Query Status.

6.3.1.3.6 Step 5. Monitor the task and view logs

To view the history status of the migration task and other internal metrics, take the following steps.

If you have deployed Prometheus, Alertmanager, and Grafana when you deployed DM using TiUP, you can access Grafana using the IP address and port specified during the deployment. You can then select DM dashboard to view DM-related monitoring metrics.

When DM is running, DM-worker, DM-master, and dmctl print the related information in logs. The log directories of these components are as follows:

- DM-master: specified by the DM-master process parameter `--log-file`. If you deploy DM using TiUP, the log directory is `/dm-deploy/dm-master-8261/log/` by default.
- DM-worker: specified by the DM-worker process parameter `--log-file`. If you deploy DM using TiUP, the log directory is `/dm-deploy/dm-worker-8262/log/` by default.

6.3.1.4 What's next

- Pause the migration task.
- Resume the migration task.
- Stop the migration task.
- Export and import the cluster data source and task configuration.
- Handle failed DDL statements.

6.3.2 Migrate MySQL of Small Datasets to TiDB

This document describes how to use TiDB Data Migration (DM) to migrate MySQL of small datasets to TiDB in the full migration mode and incremental replication mode. "Small datasets" in this document mean data size less than 1 TiB.

The migration speed varies from 30 GB/h to 50 GB/h, depending on multiple factors such as the number of indexes in the table schema, hardware, and network environment.

6.3.2.1 Prerequisites

- Deploy a DM Cluster Using TiUP
- Grant the required privileges for the source database and the target database of DM

6.3.2.2 Step 1. Create the data source

First, create the `source1.yaml` file as follows:

```
### The ID must be unique.
source-id: "mysql-01"
```

```
### Configures whether DM-worker uses the global transaction identifier (
    ↪ GTID) to pull binlogs. To enable GTID, the upstream MySQL must have
    ↪ enabled GTID. If the upstream MySQL has automatic source-replica
    ↪ switching, the GTID mode is required.
enable-gtid: true

from:
  host: "${host}"      # For example: 172.16.10.81
  user: "root"
  password: "${password}" # Plaintext password is supported but not
      ↪ recommended. It is recommended to use dmctl encrypt to encrypt the
      ↪ plaintext password before using the password.
  port: 3306
```

Then, load the data source configuration to the DM cluster using `tiup dmctl` by running the following command:

```
tiup dmctl --master-addr ${advertise-addr} operate-source create source1.
    ↪ yaml
```

The parameters used in the command above are described as follows:

Parameter	Description
`--`	`{`
↪ `master`	↪ `advertise`
↪ `-`	↪ `-`
↪ `addr`	↪ `addr`
↪	↪ `}`
	↪ of any DM-master node in the cluster where `dmctl` ↪ is to connect. For example, 172.16.10.71:8261.
`operate`	Load
↪ `-`	the
↪ `source`	data
↪	source
↪ `create`	to
↪	the DM cluster.

6.3.2.3 Step 2. Create the migration task

Create the `task1.yaml` file as follows:

```
### Task name. Each of the multiple tasks running at the same time must have
    ↪  a unique name.
name: "test"
```

```
### Task mode. Options are:
### full: only performs full data migration.
### incremental: only performs binlog real-time replication.
### all: full data migration + binlog real-time replication.
task-mode: "all"
### The configuration of the target TiDB database.
target-database:
  host: "${host}"               # For example: 172.16.10.83
  port: 4000
  user: "root"
  password: "${password}"       # Plaintext password is supported but not
     ↪ recommended. It is recommended to use dmctl encrypt to encrypt the
     ↪ plaintext password before using the password.

### The configuration of all MySQL instances of source database required for
   ↪  the current migration task.
mysql-instances:
-
  # The ID of an upstream instance or a replication group
  source-id: "mysql-01"
  # The names of the block list and allow list configuration of the schema
     ↪ name or table name that is to be migrated. These names are used to
     ↪ reference the global configuration of the block and allowlist. For
     ↪ the global configuration, refer to the `block-allow-list`
     ↪ configuration below.
  block-allow-list: "listA"

### The global configuration of blocklist and allowlist. Each instance is
   ↪ referenced by a configuration item name.
block-allow-list:
  listA:                         # name
    do-tables:                   # The allowlist of upstream tables that
       ↪ need to be migrated.
    - db-name: "test_db"         # The schema name of the table to be
       ↪ migrated.
      tbl-name: "test_table"     # The name of the table to be migrated.
```

The above is the minimum task configuration to perform the migration. For more configuration items regarding the task, refer to DM task complete configuration file introduction.

6.3.2.4 Step 3. Start the migration task

To avoid errors, before starting the migration task, it is recommended to use the `check ↪ -task` command to check whether the configuration meets the requirements of DM configuration.

```
tiup dmctl --master-addr ${advertise-addr} check-task task.yaml
```

Start the migration task by running the following command with `tiup dmctl`.

```
tiup dmctl --master-addr ${advertise-addr} start-task task.yaml
```

The parameters used in the command above are described as follows:

Parameter	Description
`--`	`{`
↪ `master`	`advertise`
↪ `-`	`-`
↪ `addr`	`addr`
↪	↪ `}`
	↪ of any DM-master node in the cluster where `dmctl` ↪ is to connect. For example: 172.16.10.71:8261.
`start`	Start
↪ `-`	the
↪ `task`	migration task

If the task fails to start, after changing the configuration according to the returned result,

you can run the `start-task task.yaml` command to restart the task. If you encounter problems, refer to Handle Errors and FAQ.

6.3.2.5 Step 4: Check the migration task status

To learn whether the DM cluster has an ongoing migration task, the task status and some other information, run the `query-status` command using `tiup dmctl`:

```
tiup dmctl --master-addr ${advertise-addr} query-status ${task-name}
```

For a detailed interpretation of the results, refer to Query Status.

6.3.2.6 Step 5. Monitor the task and view logs (optional)

To view the historical status of the migration task and other internal metrics, take the following steps.

If you have deployed Prometheus, Alertmanager, and Grafana when deploying DM using TiUP, you can access Grafana using the IP address and port specified during the deployment. You can then select the DM dashboard to view DM-related monitoring metrics.

- The log directory of DM-master: specified by the DM-master process parameter `--log-file`. If you have deployd DM using TiUP, the log directory is `/dm-deploy/dm-master-8261/log/` by default.
- The log directory of DM-worker: specified by the DM-worker process parameter `--log-file`. If you have deployd DM using TiUP, the log directory is `/dm-deploy/dm-worker-8262/log/` by default.

6.3.2.7 What's next

- Pause the migration task
- Resume the migration task
- Stop the migration task
- Export and import the cluster data source and task configuration
- Handle failed DDL statements

6.3.3 Migrate MySQL of Large Datasets to TiDB

When the data volume to be migrated is small, you can easily use DM to migrate data, both for full migration and incremental replication. However, because DM imports data at a slow speed (30~50 GiB/h), when the data volume is large, the migration might take a long time. "Large datasets" in this document usually mean data around one TiB or more.

This document describes how to migrate large datasets from MySQL to TiDB. The whole migration has two processes:

1. *Full migration*. Use Dumpling and TiDB Lightning to perform the full migration. TiDB Lightning's **local backend** mode can import data at a speed of up to 500 GiB/h.
2. *Incremental replication*. After the full migration is completed, you can replicate the incremental data using DM.

6.3.3.1 Prerequisites

- Install DM.
- Install Dumpling and TiDB Lightning.
- Grant the source database and target database privileges required for DM.
- Grant the target database privileges required for TiDB Lightning.
- Grant the source database privileges required for Dumpling.

6.3.3.2 Resource requirements

Operating system: The example in this document uses fresh CentOS 7 instances. You can deploy a virtual machine either on your local host or in the cloud. Because TiDB Lightning consumes as much CPU resources as needed by default, it is recommended that you deploy it on a dedicated server. If this is not possible, you can deploy it on a single server together with other TiDB components (for example, `tikv-server`) and then configure `region-concurrency` to limit the CPU usage from TiDB Lightning. Usually, you can configure the size to 75% of the logical CPU.

Memory and CPU: Because TiDB Lightning consumes high resources, it is recommended to allocate more than 64 GiB of memory and more than 32 CPU cores. To get the best performance, make sure that the CPU core to memory (GiB) ratio is greater than 1:2.

Disk space:

- Dumpling requires a disk space that can store the whole data source (or to store all upstream tables to be exported). SSD is recommended. To calculate the required space, see Downstream storage space requirements.
- During the import, TiDB Lightning needs temporary space to store the sorted key-value pairs. The disk space should be enough to hold the largest single table from the data source.
- If the full data volume is large, you can increase the binlog storage time in the upstream. This is to ensure that the binlogs are not lost during the incremental replication.

Note: It is difficult to calculate the exact data volume exported by Dumpling from MySQL, but you can estimate the data volume by using the following SQL statement to summarize the `data-length` field in the `information_schema.tables` table:

```
/* Calculate the size of all schemas, in MiB. Replace ${schema_name} with
   ↪ your schema name. */
```

```
SELECT table_schema,SUM(data_length)/1024/1024 AS data_length,SUM(
    ↪ index_length)/1024/1024 AS index_length,SUM(data_length+index_length)
    ↪ /1024/1024 AS SUM FROM information_schema.tables WHERE table_schema =
    ↪  "${schema_name}" GROUP BY table_schema;

/* Calculate the size of the largest table, in MiB. Replace ${schema_name}
    ↪  with your schema name. */
SELECT table_name,table_schema,SUM(data_length)/1024/1024 AS data_length,
    ↪ SUM(index_length)/1024/1024 AS index_length,SUM(data_length+
    ↪ index_length)/1024/1024 AS SUM from information_schema.tables WHERE
    ↪ table_schema = "${schema_name}" GROUP BY table_name,table_schema
    ↪ ORDER BY SUM DESC LIMIT 5;
```

6.3.3.2.1 Disk space for the target TiKV cluster

The target TiKV cluster must have enough disk space to store the imported data. In addition to the standard hardware requirements, the storage space of the target TiKV cluster must be larger than **the size of the data source x the number of replicas x 2**. For example, if the cluster uses 3 replicas by default, the target TiKV cluster must have a storage space larger than 6 times the size of the data source. The formula has `x 2` because:

- Index might take extra space.
- RocksDB has a space amplification effect.

6.3.3.3 Step 1. Export all data from MySQL

1. Export all data from MySQL by running the following command:

```
tiup dumpling -h ${ip} -P 3306 -u root -t 16 -r 200000 -F 256MiB -B
    ↪ my_db1 -f 'my_db1.table[12]' -o 's3://my-bucket/sql-backup?region
    ↪ =us-west-2'
```

   Dumpling exports data in SQL files by default. You can specify a different file format by adding the `--filetype` option.

   The parameters used above are as follows. For more Dumpling parameters, refer to Dumpling Overview.

parameters	Description
`-` ↪ `u` or `--` ↪ `user` ↪	MySQL user
`-` ↪ `p` or `--` ↪ `password` ↪	MySQL user password
`-` ↪ `P` or `--` ↪ `port` ↪	MySQL port
`-` ↪ `h` or `--` ↪ `host` ↪	MySQL HP address
`-` ↪ `t` or `--` ↪ `thread` ↪	The number of threads used for export

parameter	Description
-o, --output-dir	The directory that stores the exported file. Supports a local path or an external storage URL
-r, --row-number	The maximum number of rows in a single file

parameter	Description
-F	The maximum size of a single file, in MiB. Recommended value: 256 MiB.
-B or --database	Specifies a database to be exported
-f	Exports tables that match the filter pattern. Refer to table-filter for the syntax.

Make sure `${data-path}` has the space to store all exported upstream tables. To calculate the required space, see Downstream storage space requirements. To prevent the export from being interrupted by a large table consuming all the spaces, it is strongly recommended to use the `-F` option to limit the size of a single file.

2. View the `metadata` file in the `${data-path}` directory. This is a Dumpling-generated metadata file. Record the binlog position information, which is required for the incremental replication in Step 3.

```
SHOW MASTER STATUS:
Log: mysql-bin.000004
Pos: 109227
GTID:
```

6.3.3.4 Step 2. Import full data to TiDB

1. Create the `tidb-lightning.toml` configuration file:

```
[lightning]
# log.
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
# "local": Default backend. The local backend is recommended to import
    ↪ large volumes of data (1 TiB or more). During the import, the
    ↪ target TiDB cluster cannot provide any service.
# "tidb": The "tidb" backend is recommended to import data less than 1
    ↪ TiB. During the import, the target TiDB cluster can provide
    ↪ service normally. For more information on the backends, refer to
    ↪ https://docs.pingcap.com/tidb/stable/tidb-lightning-backends.
backend = "local"
# Sets the temporary storage directory for the sorted Key-Value files.
    ↪ The directory must be empty, and the storage space must be
    ↪ greater than the size of the dataset to be imported. For better
    ↪ import performance, it is recommended to use a directory
    ↪ different from `data-source-dir` and use flash storage, which can
    ↪  use I/O exclusively.
sorted-kv-dir = "${sorted-kv-dir}"

[mydumper]
# The data source directory. The same directory where Dumpling exports
    ↪ data in "Step 1. Export all data from MySQL".
data-source-dir = "${data-path}" # A local path or S3 path. For example
    ↪ , 's3://my-bucket/sql-backup?region=us-west-2'.
```

```
[tidb]
# The target TiDB cluster information.
host = ${host}             # e.g.: 172.16.32.1
port = ${port}             # e.g.: 4000
user = "${user_name}"      # e.g.: "root"
password = "${password}"   # e.g.: "rootroot"
status-port = ${status-port} # During the import, TiDB Lightning needs
    ↪ to obtain the table schema information from the TiDB status port.
    ↪  e.g.: 10080
pd-addr = "${ip}:${port}" # The address of the PD cluster, e.g.:
    ↪ 172.16.31.3:2379. TiDB Lightning obtains some information from PD
    ↪ . When backend = "local", you must specify status-port and pd-
    ↪ addr correctly. Otherwise, the import will be abnormal.
```

For more information on TiDB Lightning configuration, refer to TiDB Lightning Configuration.

2. Start the import by running `tidb-lightning`. If you launch the program directly in the command line, the process might exit unexpectedly after receiving a SIGHUP signal. In this case, it is recommended to run the program using a `nohup` or `screen` tool. For example:

   If you import data from S3, pass the SecretKey and AccessKey that have access to the S3 storage path as environment variables to the TiDB Lightning node. You can also read the credentials from `~/.aws/credentials`.

```
export AWS_ACCESS_KEY_ID=${access_key}
export AWS_SECRET_ACCESS_KEY=${secret_key}
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out 2>&1
    ↪ &
```

3. After the import starts, you can check the progress of the import by one of the following methods:

   - `grep` the keyword `progress` in the log. The progress is updated every 5 minutes by default.
   - Check progress in the monitoring dashboard.
   - Check progress in the TiDB Lightning web interface.

4. After TiDB Lightning completes the import, it exits automatically. Check whether `tidb-lightning.log` contains `the whole procedure completed` in the last lines. If yes, the import is successful. If no, the import encounters an error. Address the error as instructed in the error message.

Note:

Whether the import is successful or not, the last line of the log shows `tidb lightning exit`. It means that TiDB Lightning exits normally, but does not necessarily mean that the import is successful.

If the import fails, refer to TiDB Lightning FAQ for troubleshooting.

6.3.3.5 Step 3. Replicate incremental data to TiDB

6.3.3.5.1 Add the data source

1. Create a `source1.yaml` file as follows:

```
# Must be unique.
source-id: "mysql-01"

# Configures whether DM-worker uses the global transaction identifier (
    ↪ GTID) to pull binlogs. To enable this mode, the upstream MySQL
    ↪ must also enable GTID. If the upstream MySQL service is
    ↪ configured to switch master between different nodes automatically
    ↪ , GTID mode is required.
enable-gtid: true

from:
  host: "${host}"        # e.g.: 172.16.10.81
  user: "root"
  password: "${password}" # Supported but not recommended to use a
      ↪ plaintext password. It is recommended to use `dmctl encrypt` to
      ↪  encrypt the plaintext password before using it.
  port: 3306
```

2. Load the data source configuration to the DM cluster using `tiup dmctl` by running the following command:

```
tiup dmctl --master-addr ${advertise-addr} operate-source create
    ↪ source1.yaml
```

The parameters used in the command above are described as follows:

Parameter	Description
--master-addr	The {master-addr} of any DM-master in the cluster where dmctl is to be connected, e.g.: 172.16.10.71:8261
operate-source	Loads the source configuration to the DM cluster.

6.3.3.5.2 Add a replication task

1. Edit the `task.yaml` file. Configure the incremental replication mode and the starting point of each data source:

```
name: task-test                     # Task name. Must be globally unique.
task-mode: incremental              # Task mode. The "incremental" mode
    ↪ only performs incremental data replication.
```

```
# Configures the target TiDB database.
target-database:                # The target database instance.
  host: "${host}"               # e.g.: 127.0.0.1
  port: 4000
  user: "root"
  password: "${password}"       # It is recommended to use `dmctl
     ↪ encrypt` to encrypt the plaintext password before using it.

# Use block and allow lists to specify the tables to be replicated.
block-allow-list:               # The collection of filtering rules
   ↪ that matches the tables in the source database instance. If the
   ↪ DM version is earlier than v2.0.0-beta.2, use black-white-list.
  bw-rule-1:                    # The block-allow-list configuration
     ↪ item ID.
    do-dbs: ["${db-name}"]      # Name of databases to be replicated.

# Configures the data source.
mysql-instances:
  - source-id: "mysql-01"       # Data source ID, i.e., source-id in
     ↪ source1.yaml
    block-allow-list: "bw-rule-1" # You can use the block-allow-list
       ↪ configuration above.
    # syncer-config-name: "global" # You can use the syncers
       ↪ incremental data configuration below.
    meta:                       # When task-mode is "incremental" and
       ↪ the target database does not have a checkpoint, DM uses the
       ↪ binlog position as the starting point. If the target database
       ↪  has a checkpoint, DM uses the checkpoint as the starting
       ↪ point.
      # binlog-name: "mysql-bin.000004" # The binlog position recorded
         ↪ in "Step 1. Export all data from MySQL". If the upstream
         ↪ database service is configured to switch master between
         ↪ different nodes automatically, GTID mode is required.
      # binlog-pos: 109227
      binlog-gtid: "09bec856-ba95-11ea-850a-58f2b4af5188:1-9"

# (Optional) If you need to incrementally replicate data that has
   ↪ already been migrated in the full data migration, you need to
   ↪ enable the safe mode to avoid the incremental data replication
   ↪ error.
# This scenario is common in the following case: the full migration
   ↪ data does not belong to the data source's consistency snapshot,
   ↪ and after that, DM starts to replicate incremental data from a
   ↪ position earlier than the full migration.
```

```
# syncers:          # The running configurations of the sync processing
    ↪ unit.
#   global:         # Configuration name.
#     safe-mode: true # If this field is set to true, DM changes INSERT
    ↪ of the data source to REPLACE for the target database, and
    ↪ changes UPDATE of the data source to DELETE and REPLACE for the
    ↪ target database. This is to ensure that when the table schema
    ↪ contains a primary key or unique index, DML statements can be
    ↪ imported repeatedly. In the first minute of starting or resuming
    ↪ an incremental replication task, DM automatically enables the
    ↪ safe mode.
```

The YAML above is the minimum configuration required for the migration task. For more configuration items, refer to DM Advanced Task Configuration File.

Before you start the migration task, to reduce the probability of errors, it is recommended to confirm that the configuration meets the requirements of DM by running the `check-task` command:

```
tiup dmctl --master-addr ${advertise-addr} check-task task.yaml
```

2. Start the migration task by running the following command:

```
tiup dmctl --master-addr ${advertise-addr} start-task task.yaml
```

The parameters used in the command above are described as follows:

Parameter	Description
`--master-addr {advertise-addr}`	The `advertise-addr` of any DM-master in the cluster where `dmctl` is to be connected, e.g.: 172.16.10.71:8261
`start-task`	Starts the migration task.

If the task fails to start, check the prompt message and fix the configuration. After that, you can re-run the command above to start the task.

If you encounter any problem, refer to DM error handling and DM FAQ.

6.3.3.5.3 Check the migration task status

To learn whether the DM cluster has an ongoing migration task and view the task status, run the `query-status` command using `tiup dmctl`:

```
tiup dmctl --master-addr ${advertise-addr} query-status ${task-name}
```

For a detailed interpretation of the results, refer to Query Status.

6.3.3.5.4 Monitor the task and view logs

To view the history status of the migration task and other internal metrics, take the following steps.

If you have deployed Prometheus, Alertmanager, and Grafana when you deployed DM using TiUP, you can access Grafana using the IP address and port specified during the deployment. You can then select DM dashboard to view DM-related monitoring metrics.

When DM is running, DM-worker, DM-master, and dmctl print the related information in logs. The log directories of these components are as follows:

- DM-master: specified by the DM-master process parameter `--log-file`. If you deploy DM using TiUP, the log directory is `/dm-deploy/dm-master-8261/log/` by default.
- DM-worker: specified by the DM-worker process parameter `--log-file`. If you deploy DM using TiUP, the log directory is `/dm-deploy/dm-worker-8262/log/` by default.

6.3.3.6 What's next

- Pause a Data Migration Task
- Resume a Data Migration Task
- Stop a Data Migration Task
- Export and Import Data Sources and Task Configuration of Clusters
- Handle Failed DDL Statements

6.3.4 Migrate and Merge MySQL Shards of Small Datasets to TiDB

If you want to migrate and merge multiple MySQL database instances upstream to one TiDB database downstream, and the amount of data is not too large, you can use DM to migrate MySQL shards. "Small datasets" in this document usually mean data around or less than one TiB. Through examples in this document, you can learn the operation steps, precautions, and troubleshooting of the migration.

This document applies to migrating MySQL shards less than 1 TiB in total. If you want to migrate MySQL shards with a total of more than 1 TiB of data, it will take a long time to migrate only using DM. In this case, it is recommended that you follow the operation introduced in Migrate and Merge MySQL Shards of Large Datasets to TiDB to perform migration.

This document takes a simple example to illustrate the migration procedure. The MySQL shards of the two data source MySQL instances in the example are migrated to the downstream TiDB cluster.

In this example, both MySQL Instance 1 and MySQL Instance 2 contain the following schemas and tables. In this example, you migrate and merge tables from `store_01` and `store_02` schemas with a `sale` prefix in both instances, into the downstream `sale` table in the `store` schema.

Schema	Table
store_01	sale_01, sale_02
store_02	sale_01, sale_02

Target schemas and tables:

Schema	Table
store	sale

6.3.4.1 Prerequisites

Before starting the migration, make sure you have completed the following tasks:

- Deploy a DM Cluster Using TiUP
- Privileges required by DM-worker

6.3.4.1.1 Check conflicts for the sharded tables

If the migration involves merging data from different sharded tables, primary key or unique index conflicts may occur during the merge. Therefore, before migration, you need to take a deep look at the current sharding scheme from the business point of view, and find a way to avoid the conflicts. For more details, see Handle conflicts between primary keys or unique indexes across multiple sharded tables. The following is a brief description.

In this example, `sale_01` and `sale_02` have the same table structure as follows

```
CREATE TABLE `sale_01` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `sid` bigint(20) NOT NULL,
  `pid` bigint(20) NOT NULL,
  `comment` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `sid` (`sid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

The `id` column is the primary key, and the `sid` column is the sharding key. The `id` column is auto-incremental, and duplicated multiple sharded table ranges will cause data conflicts. The `sid` can ensure that the index is globally unique, so you can follow the steps in Remove the primary key attribute of the auto-increment primary key to bypasses the `id` column.

```
CREATE TABLE `sale` (
  `id` bigint(20) NOT NULL,
  `sid` bigint(20) NOT NULL,
  `pid` bigint(20) NOT NULL,
```

```
  `comment` varchar(255) DEFAULT NULL,
  INDEX (`id`),
  UNIQUE KEY `sid` (`sid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

6.3.4.2 Step 1. Load data sources

Create a new data source file called `source1.yaml`, which configures an upstream data source into DM, and add the following content:

```
### Configuration.
source-id: "mysql-01" # Must be unique.
### Specifies whether DM-worker pulls binlogs with GTID (Global Transaction
    ↪ Identifier).
### The prerequisite is that you have already enabled GTID in the upstream
    ↪ MySQL.
### If you have configured the upstream database service to switch master
    ↪ between different nodes automatically, you must enable GTID.
enable-gtid: true
from:
  host: "${host}"        # For example: 172.16.10.81
  user: "root"
  password: "${password}" # Plaintext passwords are supported but not
      ↪ recommended. It is recommended that you use dmctl encrypt to encrypt
      ↪  plaintext passwords.
  port: ${port}          # For example: 3306
```

Run the following command in a terminal. Use `tiup dmctl` to load the data source configuration into the DM cluster:

```
tiup dmctl --master-addr ${advertise-addr} operate-source create source1.
    ↪ yaml
```

The parameters are described as follows.

Parameter	Description
`--`	`{`
↪ `master`	`advertise`
↪ `-`	`-`
↪ `addr`	`addr`
↪	`}`
	↪ of any DM-master node in the cluster that dm-ctl connects to. For example: 172.16.10.71:8261
`operate`	Load
↪ `-`	data
↪ `source`	sources
↪ `create`	to
↪	the DM clusters.

Repeat the above steps until all data sources are added to the DM cluster.

6.3.4.3 Step 2. Configure the migration task

Create a task configuration file named `task1.yaml` and writes the following content to it:

```
name: "shard_merge"          # The name of the task. Should be globally
    ↪ unique.
```

```
### Task mode. You can set it to the following:
### - full: Performs only full data migration (incremental replication is
    ↪ skipped)
### - incremental: Only performs real-time incremental replication using
    ↪ binlog. (full data migration is skipped)
### - all: Performs both full data migration and incremental replication.
    ↪ For migrating small to medium amount of data here, use this option.
task-mode: all
### Required for the MySQL shards. By default, the "pessimistic" mode is
    ↪ used.
### If you have a deep understanding of the principles and usage limitations
    ↪  of the optimistic mode, you can also use the "optimistic" mode.
### For more information, see [Merge and Migrate Data from Sharded Tables](
    ↪ https://docs.pingcap.com/tidb/dev/feature-shard-merge/)
shard-mode: "pessimistic"
meta-schema: "dm_meta"                  # A schema will be created in the
    ↪ downstream database to store the metadata
ignore-checking-items: ["auto_increment_ID"] # In this example, there are
    ↪ auto-incremental primary keys upstream, so you do not need to check
    ↪ this item.

target-database:
  host: "${host}"                       # For example: 192.168.0.1
  port: 4000
  user: "root"
  password: "${password}"               # Plaintext passwords are supported
      ↪  but not recommended. It is recommended that you use dmctl encrypt
      ↪ to encrypt plaintext passwords.

mysql-instances:
  -
    source-id: "mysql-01"                               # ID of the data
        ↪ source, which is source-id in source1.yaml
    route-rules: ["sale-route-rule"]                    # Table route rules
        ↪ applied to the data source
    filter-rules: ["store-filter-rule", "sale-filter-rule"] # Binlog event
        ↪ filter rules applied to the data source
    block-allow-list: "log-bak-ignored"                 # Block & Allow Lists
        ↪ rules applied to the data source
  -
    source-id: "mysql-02"
    route-rules: ["sale-route-rule"]
    filter-rules: ["store-filter-rule", "sale-filter-rule"]
    block-allow-list: "log-bak-ignored"
```

```
### Configurations for merging MySQL shards
routes:                     # Table renaming rules ('routes') from upstream
   ↪ to downstream tables, in order to support merging different sharded
   ↪ tables into a single target table.
  sale-route-rule:          # Rule name. Migrate and merge tables from
     ↪ upstream to the downstream.
    schema-pattern: "store_*" # Rule for matching upstream schema names. It
       ↪ supports the wildcards "*" and "?".
    table-pattern: "sale_*" # Rule for matching upstream table names. It
       ↪ supports the wildcards "*" and "?".
    target-schema: "store" # Name of the target schema.
    target-table: "sale"   # Name of the target table.

### Filters out some DDL events.
filters:
  sale-filter-rule:         # Filter name.
    schema-pattern: "store_*" # The binlog events or DDL SQL statements of
       ↪ upstream MySQL instance schemas that match schema-pattern are
       ↪ filtered by the rules below.
    table-pattern: "sale_*" # The binlog events or DDL SQL statements of
       ↪ upstream MySQL instance tables that match table-pattern are
       ↪ filtered by the rules below.
    events: ["truncate table", "drop table", "delete"] # The binlog event
       ↪ array.
    action: Ignore                                  # The string (`Do`/`Ignore
       ↪ `). `Do` is the allow list. `Ignore` is the block list.
  store-filter-rule:
    schema-pattern: "store_*"
    events: ["drop database"]
    action: Ignore

### Block and allow list
block-allow-list:         # filter or only migrate all operations of some
   ↪ databases or some tables.
  log-bak-ignored:        # Rule name.
    do-dbs: ["store_*"]  # The allow list of the schemas to be migrated,
       ↪ similar to replicate-do-db in MySQL.
```

The above example is the minimum configuration to perform the migration task. For more information, see DM Advanced Task Configuration File.

For more information on `routes`, `filters` and other configurations in the task file, see the following documents:

- Table routing

- Block & Allow Table Lists
- Binlog event filter
- Filter Certain Row Changes Using SQL Expressions

6.3.4.4 Step 3. Start the task

Before starting a migration task, run the `check-task` subcommand in `tiup dmctl` to check whether the configuration meets the requirements of DM so as to avoid possible errors.

```
tiup dmctl --master-addr ${advertise-addr} check-task task.yaml
```

Run the following command in `tiup dmctl` to start a migration task:

```
tiup dmctl --master-addr ${advertise-addr} start-task task.yaml
```

Parameter	Description
`--`	`{`
↪ `master-addr`	`advertise`
↪ `-`	`-`
↪ `addr`	`addr`
↪	`}`
	↪ of any DM-master node in the cluster that dmctl connects to. For example: 172.16.10.71:8261

Parameter	Description
`start-task`	Starts the task migration task.

If the migration task fails to start, modify the configuration information according to the error information, and then run `start-task task.yaml` again to start the migration task. If you encounter problems, see Handle Errors and FAQ.

6.3.4.5 Step 4. Check the task

After starting the migration task, you can use `dmtcl tiup` to run `query-status` to view the status of the task.

```
tiup dmctl --master-addr ${advertise-addr} query-status ${task-name}
```

If you encounter errors, use `query-status ${task-name}` to view more detailed information. For details about the query results, task status and sub task status of the `query-status` command, see TiDB Data Migration Query Status.

6.3.4.6 Step 5. Monitor tasks and check logs (optional)

You can view the history of a migration task and internal operational metrics through Grafana or logs.

- Via Grafana

  If Prometheus, Alertmanager, and Grafana are correctly deployed when you deploy the DM cluster using TiUP, you can view DM monitoring metrics in Grafana. Specifically, enter the IP address and port specified during deployment in Grafana and select the DM dashboard.

- Via logs

  When DM is running, DM-master, DM-worker, and dmctl output logs, which includes information about migration tasks. The log directory of each component is as follows.

  - DM-master log directory: It is specified by the DM-master process parameter `--log-file`. If DM is deployed using TiUP, the log directory is `/dm-deploy/dm-master-8261/log/`.
  - DM-worker log directory: It is specified by the DM-worker process parameter `--log-file`. If DM is deployed using TiUP, the log directory is `/dm-deploy/dm-worker-8262/log/`.

6.3.4.7 See also

- Migrate and Merge MySQL Shards of Large Datasets to TiDB.
- Merge and Migrate Data from Sharded Tables
- Best Practices of Data Migration in the Shard Merge Scenario
- Handle Errors
- Handle Performance Issues
- FAQ

6.3.5 Migrate and Merge MySQL Shards of Large Datasets to TiDB

If you want to migrate a large MySQL dataset (for example, more than 1 TiB) from different partitions into TiDB, and you are able to suspend all the TiDB cluster write operations from your business during the migration, you can use TiDB Lightning to do the migration quickly. After migration, you can also use TiDB DM to perform incremental replication according to your business needs. "Large datasets" in this document usually mean data around one TiB or more.

This document uses an example to walk through the whole procedure of such kind of migration.

If the data size of the MySQL shards is less than 1 TiB, you can follow the procedure described in Migrate and Merge MySQL Shards of Small Datasets to TiDB, which supports both full and incremental migration and the steps are easier.

The example in this document assumes that you have two databases, `my_db1` and `my_db2`. You use Dumpling to export two tables `table1` and `table2` from `my_db1`, and two tables `table3` and `table4` from `my_db2`, respectively. After that, you use TiDB Lightning to import and merge the four exported tables into the same `table5` from `mydb` in the target TiDB.

In this document, you can migrate data following this procedure:

1. Use Dumpling to export full data. In this example, you export 2 tables respectively from 2 upstream databases:
    - Export `table1` and `table2` from `my_db1`
    - Export `table3` and `table4` from `my_db2`
2. Start TiDB Lightning to migrate data to `mydb.table5` in TiDB.
3. (Optional) Use TiDB DM to perform incremental replication.

6.3.5.1 Prerequisites

Before getting started, see the following documents to prepare for the migration task.

- Deploy a DM Cluster Using TiUP

- Use TiUP to Deploy Dumpling and Lightning
- Downstream privilege requirements for Dumpling
- Downstream privilege requirements for TiDB Lightning
- Downstream storage space for TiDB Lightning
- Privileges required by DM-worker

6.3.5.1.1 Check conflicts for Sharded Tables

If the migration involves merging data from different sharded tables, primary key or unique index conflicts may occur during the merge. Therefore, before migration, you need to take a deep look at the current sharding scheme from the business point of view, and find a way to avoid conflicts. For more details, see Handle conflicts between primary keys or unique indexes across multiple sharded tables. The following is a brief description.

Assume that tables 1~4 have the same table structure as follows.

```
CREATE TABLE `table1` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `sid` bigint(20) NOT NULL,
  `pid` bigint(20) NOT NULL,
  `comment` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `sid` (`sid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

For those four tables, the `id` column is the primary key. It is auto-incremental, which will cause different sharded tables to generate duplicated `id` ranges and cause the primary key conflict on the target table during the migration. On the other hand, the `sid` column is the sharding key, which ensures that the index is unique globally. So you can remove the unique constraint of the `id` column in the target `table5` to avoid the data merge conflicts.

```
CREATE TABLE `table5` (
  `id` bigint(20) NOT NULL,
  `sid` bigint(20) NOT NULL,
  `pid` bigint(20) NOT NULL,
  `comment` varchar(255) DEFAULT NULL,
  INDEX (`id`),
  UNIQUE KEY `sid` (`sid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

6.3.5.2 Step1. Use Dumpling to export full data

If those multiple sharded tables to be exported are in the same upstream MySQL instance, you can directly use the `-f` parameter of Dumpling to export them in a single operation.

If the sharded tables are stored in different MySQL instances, you can use Dumping to export them respectively and place the exported results in the same parent directory.

In the following example, both methods are used, and then the exported data is stored in the same parent directory.

First, run the following command to use Dumping to export `table1` and `table2` from `my_db1`:

```
tiup dumping -h ${ip} -P 3306 -u root -t 16 -r 200000 -F 256MB -B my_db1 -f
    ↪  'my_db1.table[12]' -o ${data-path}/my_db1
```

The following table describes parameters in the command above. For more information about Dumping parameters, see Dumping Overview.

Parameter	Description
-u or --user	Specifies the user name to user used.
-p or --password	Specifies the password used.
-P or --port	Specifies the port to be used.
-h or --host	Specifies the IP address of the data source.

Parameter	Description
`-` ↪ `t` ↪ or `--` ↪ `thread` ↪	Specifies the number of threads for the export. Increasing the number of threads improves the concurrency of Dumpling and the export speed, and increases the database's memory consumption. Therefore, it is not recom-

Parameter	Description
`-s` or `--storage`	Specifies the export storage output directory of the storage, which supports a local file path or a URL of an external storage.

Parameter	Description
-r or --row	Specifies the maximum row number of rows in a single file. If you use this parameter, Dumpling enables the in-table concurrency to speed up the export and reduce the memory usage.

Parameter	Description
-	Specifies the maximum size of a single file. The unit is `MiB`. It is recommended to keep the value to 256 MiB.
-	Specifies databases to be excluded
or --	

Parameter	Description
-f or --filter	Exports tables that match the filter pattern. For the filter syntax, see table-filter.

Ensure that there is enough free space in `${data-path}`. It is strongly recommended to use the `-F` option to avoid interruptions in the backup process due to oversized single tables.

Then, run the following command to use Dumpling to export `table3` and `table4` from `my_db2`. Note that the path is `${data-path}/my_db2` instead of `${data-path}/my_db1`.

```
tiup dumpling -h ${ip} -P 3306 -u root -t 16 -r 200000 -F 256MB -B my_db2 -f
    ↪  'my_db2.table[34]' -o ${data-path}/my_db2
```

After the preceding procedures, all source data tables are now exported to the `${data-path}` directory. Putting all the exported data on the same directory makes subsequent import by TiDB Lightning convenient.

The starting position information needed for incremental replication is in the `metadata` files in `my_db1` and `my_db2` sub-directories of `${data-path}` directory respectively. They are meta-information files automatically generated by Dumpling. To perform incremental replication, you need to record the binlog locations information in these files.

6.3.5.3 Step 2. Start TiDB Lightning to import full exported data

Before starting TiDB Lightning for migration, it is recommended that you understand how to handle checkpoints, and then choose the appropriate way to proceed according to your needs.

6.3.5.3.1 Checkpoints

Migrating a large volume of data usually takes hours or even days. There is a certain chance that the long-running process is interrupted unexpectedly. It can be very frustrating to redo everything from scratch, even if some part of data has already been imported.

Fortunately, TiDB Lightning provides a feature called `checkpoints`, which makes TiDB Lightning save the import progress as `checkpoints` from time to time, so that an interrupted import task can be resumed from the latest checkpoint upon restart.

If the TiDB Lightning task crashes due to unrecoverable errors (for example, data corruption), it will not pick up from the checkpoint, but will report an error and quit the task. To ensure the safety of the imported data, you must resolve these errors by using the `tidb-lightning-ctl` command before proceeding with other steps. The options include:

- –checkpoint-error-destroy: This option allows you to restart importing data into failed target tables from scratch by destroying all the existing data in those tables first.
- –checkpoint-error-ignore: If migration has failed, this option clears the error status as if no errors ever happened.
- –checkpoint-remove: This option simply clears all checkpoints, regardless of errors.

For more information, see TiDB Lightning Checkpoints.

6.3.5.3.2 Create a target schema

Create `mydb.table5` at downstream.

```
CREATE TABLE `table5` (
  `id` bigint(20) NOT NULL,
  `sid` bigint(20) NOT NULL,
  `pid` bigint(20) NOT NULL,
  `comment` varchar(255) DEFAULT NULL,
  INDEX (`id`),
  UNIQUE KEY `sid` (`sid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

6.3.5.3.3 Start the migration task

Follow these steps to start `tidb-lightning`:

1. Edit the toml file. `tidb-lightning.toml` is used in the following example:

```
[lightning]
# Logs
level = "info"
file = "tidb-lightning.log"
```

```
[mydumper]
data-source-dir = ${data-path}

[tikv-importer]
# Choose a local backend.
# "local": The default mode. It is used for large data volumes greater
    ↪ than 1 TiB. During migration, downstream TiDB cannot provide
    ↪ services.
# "tidb": Used for data volumes less than 1 TiB. During migration,
    ↪ downstream TiDB can provide services normally.
# For more information, see [TiDB Lightning Backends](https://docs.
    ↪ pingcap.com/tidb/stable/tidb-lightning-backends)
backend = "local"
# Set the temporary directory for the sorted key value pairs. It must
    ↪ be empty.
# The free space must be greater than the size of the dataset to be
    ↪ imported.
# It is recommended that you use a directory different from `data-
    ↪ source-dir` to get better migration performance by consuming I/O
    ↪ resources exclusively.
sorted-kv-dir = "${sorted-kv-dir}"

# Set the renaming rules ('routes') from source to target tables, in
    ↪ order to support merging different table shards into a single
    ↪ target table. Here you migrate `table1` and `table2` in `my_db1`,
    ↪  and `table3` and `table4` in `my_db2`, to the target `table5` in
    ↪  downstream `my_db`.
[[mydumper.files]]
pattern = '(^|/)my_db1\.table[1-2]\..*\.sql$'
schema = "my_db"
table = "table5"
type = "sql"

[[mydumper.files]]
pattern = '(^|/)my_db2\.table[3-4]\..*\.sql$'
schema = "my_db"
table = "table5"
type = "sql"

# Information of the target TiDB cluster. For example purposes only.
    ↪ Replace the IP address with your IP address.
[tidb]
# Information of the target TiDB cluster.
# Values here are only for illustration purpose. Replace them with your
    ↪  own values.
```

```
host = ${host}        # For example: "172.16.31.1"
port = ${port}        # For example: 4000
user = "${user_name}" # For example: "root"
password = "${password}" # For example: "rootroot"
status-port = ${status-port} # The table information is read from the
    ↪ status port. For example: 10080
# the IP address of the PD cluster. TiDB Lightning gets some
    ↪ information through the PD cluster.
# For example: "172.16.31.3:2379".
# When backend = "local", make sure that the values of status-port and
    ↪ pd-addr are correct. Otherwise an error will occur.
pd-addr = "${ip}:${port}"
```

2. Run `tidb-lightning`. If you run the program by directly invoking the program name in a shell, the process may quit unexpectedly after receiving a SIGHUP signal. It is recommended that you run the program using tools such as `nohup` or `screen` or `tiup`, and put the process to the shell background. If you migrate from S3, the SecretKey and AccessKey of the account that has access to the Amazon S3 backend store needs to be passed into the Lightning node as environment variables. Reading credential files from `~/.aws/credentials` is also supported. For example:

```
export AWS_ACCESS_KEY_ID=${access_key}
export AWS_SECRET_ACCESS_KEY=${secret_key}
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out 2>&1
    ↪ &
```

3. After starting the migration task, you can check the progress by using either of the following methods:

- Use `grep` tool to search the keyword `progress` in the log. By default, a message reporting the progress is flushed into the log file every 5 minutes.
- View progress via the monitoring dashboard. For more information, see TiDB Lightning Monitoring.
- View the progress via the Web page. See Web Interface.

After TiDB Lightning completes the import, it exits automatically. Check whether `tidb-lightning.log` contains `the whole procedure completed` in the last lines. If yes, the import is successful. If no, the import encounters an error. Address the error as instructed in the error message.

Note:

> Whether the migration is successful or not, the last line in the log will always be `tidb lightning exit`. It just means that TiDB Lightning quits normally, and does not guarantee that the importing task is completed successfully.

If you encounter any problems during migration, see TiDB Lightning FAQs.

6.3.5.4 Step 3. (Optional) Use DM to perform incremental replication

To replicate the data changes based on binlog from a specified position in the source database to TiDB, you can use TiDB DM to perform incremental replication.

6.3.5.4.1 Add the data source

Create a new data source file called `source1.yaml`, which configures an upstream data source into DM, and add the following content:

```
### Configuration.
source-id: "mysql-01" # Must be unique.

### Specifies whether DM-worker pulls binlogs with GTID (Global Transaction
    ↪ Identifier).
### The prerequisite is that you have already enabled GTID in the upstream
    ↪ MySQL.
### If you have configured the upstream database service to switch master
    ↪ between different nodes automatically, you must enable GTID.
enable-gtid: true

from:
  host: "${host}"        # For example: 172.16.10.81
  user: "root"
  password: "${password}" # Plaintext passwords are supported but not
      ↪ recommended. It is recommended that you use dmctl encrypt to encrypt
      ↪  plaintext passwords.
  port: ${port}          # For example: 3306
```

Run the following command in a terminal. Use `tiup dmctl` to load the data source configuration into the DM cluster:

```
tiup dmctl --master-addr ${advertise-addr} operate-source create source1.
    ↪ yaml
```

The parameters are described as follows.

Parameter	Description
--master-addr	{advertise-master-addr} of DM-master node in the cluster that dmctl connects to. For example: 172.16.10.71:8261
operate-source	Load data sources to DM clusters.

Repeat the above steps until all MySQL upstream instances are added to the DM as data sources.

6.3.5.4.2 Create a replication task

Edit a task configuration file called `task.yaml`, to configure the incremental replication mode and replication starting point for each data source.

```
name: task-test           # The name of the task. Should be globally unique
  ↪ .
task-mode: incremental    # The mode of the task. "incremental" means full
  ↪ data migration is skipped and only incremental replication is
  ↪ performed.
```

```
### Required for incremental replication from sharded tables. By default,
   ↪ the "pessimistic" mode is used.
### If you have a deep understanding of the principles and usage limitations
   ↪  of the optimistic mode, you can also use the "optimistic" mode.
### For more information, see [Merge and Migrate Data from Sharded Tables](
   ↪ https://docs.pingcap.com/zh/tidb/dev/feature-shard-merge/).

shard-mode: "pessimistic"

### Configure the access information of the target TiDB database instance:
target-database:          # The target database instance
  host: "${host}"         # For example: 127.0.0.1
  port: 4000
  user: "root"
  password: "${password}" # It is recommended to use a dmctl encrypted
     ↪ password.

### Use block-allow-list to configure tables that require sync:
block-allow-list:         # The set of filter rules on matching tables in
   ↪ the data sources, to decide which tables need to migrate and which
   ↪ not. Use the black-white-list if the DM version is earlier than or
   ↪ equal to v2.0.0-beta.2.
  bw-rule-1:              # The ID of the block and allow list rule.
    do-dbs: ["my_db1"]    # The databases to be migrated. Here, my_db1 of
       ↪ instance 1 and my_db2 of instance 2 are configured as two separate
       ↪  rules to demonstrate how to prevent my_db2 of instance 1 from
       ↪ being replicated.
  bw-rule-2:
    do-dbs: ["my_db2"]
routes:                           # Table renaming rules ('routes') from
   ↪ upstream to downstream tables, in order to support merging different
   ↪ sharded table into a single target table.
  route-rule-1:                   # Rule name. Migrate and merge table1 and
     ↪ table2 from my_db1 to the downstream my_db.table5.
    schema-pattern: "my_db1"      # Rule for matching upstream schema names.
       ↪  It supports the wildcards "*" and "?".
    table-pattern: "table[1-2]"   # Rule for matching upstream table names.
       ↪ It supports the wildcards "*" and "?".
    target-schema: "my_db"        # Name of the target schema.
    target-table: "table5"        # Name of the target table.
  route-rule-2:                   # Rule name. Migrate and merge table3 and
     ↪ table4 from my_db2 to the downstream my_db.table5.
    schema-pattern: "my_db2"
    table-pattern: "table[3-4]"
    target-schema: "my_db"
```

```
    target-table: "table5"

### Configure data sources. The following uses two data sources as an
    ↪ example.
mysql-instances:
  - source-id: "mysql-01"         # Data source ID. It is the source-id in
      ↪ source1.yaml.
    block-allow-list: "bw-rule-1" # Use the block and allow list
       ↪ configuration above. Replicate `my_db1` in instance 1.
    route-rules: ["route-rule-1"] # Use the configured routing rule above to
       ↪  merge upstream tables.
###      syncer-config-name: "global" # Use the syncers configuration below.
    meta:                         # The migration starting point of binlog
       ↪ when task-mode is incremental and there is no checkpoint in the
       ↪ downstream database. If there is a checkpoint, the checkpoint will
       ↪  be used.
      binlog-name: "${binlog-name}" # The log location recorded in ${data-
         ↪ path}/my_db1/metadata in Step 1. You can either specify binlog-
         ↪ name + binlog-pos or binlog-gtid. When the upstream database
         ↪ service is configured to switch master between different nodes
         ↪ automatically, use binlog GTID here.
      binlog-pos: ${binlog-position}
      # binlog-gtid:              " For example: 09bec856-ba95-11ea-850a-58
         ↪ f2b4af5188:1-9"
  - source-id: "mysql-02"         # Data source ID. It is the source-id in
      ↪ source1.yaml.
    block-allow-list: "bw-rule-2" # Use the block and allow list
       ↪ configuration above. Replicate `my_db2` in instance2.
    route-rules: ["route-rule-2"] # Use the routing rule configured above.

###      syncer-config-name: "global" # Use the syncers configuration below.
    meta:                         # The migration starting point of binlog
       ↪ when task-mode is incremental and there is no checkpoint in the
       ↪ downstream database. If there is a checkpoint, the checkpoint will
       ↪  be used.
      # binlog-name: "${binlog-name}" # The log location recorded in ${data-
         ↪ path}/my_db2/metadata in Step 1. You can either specify binlog-
         ↪ name + binlog-pos or binlog-gtid. When the upstream database
         ↪ service is configured to switch master between different nodes
         ↪ automatically, use binlog GTID here.
      # binlog-pos: ${binlog-position}
      binlog-gtid: "09bec856-ba95-11ea-850a-58f2b4af5188:1-9"
### (Optional) If you need to incrementally replicate some data changes that
    ↪  have been covered in the full migration, you need to enable the safe
    ↪  mode to avoid data migration errors during incremental replication.
```

```
### This scenario is common when the fully migrated data is not part of a
    ↪ consistent snapshot of the data source, and the incremental data is
    ↪ replicated from a location earlier than the fully migrated data.
### syncers:         # The running parameters of the sync processing unit.
### global:          # Configuration name.
### If set to true, DM changes INSERT to REPLACE, and changes UPDATE to a
    ↪ pair of DELETE and REPLACE for data source replication operations.
### Thus, it can apply DML repeatedly during replication when primary keys
    ↪ or unique indexes exist in the table structure.
### TiDB DM automatically starts safe mode within 1 minute before starting
    ↪ or resuming an incremental replication task.
###   safe-mode: true
```

For more configurations, see DM Advanced Task Configuration File.

Before you start the data migration task, it is recommended to use the `check-task` ↪ subcommand in `tiup dmctl` to check if the configuration meets the DM configuration requirements.

```
tiup dmctl --master-addr ${advertise-addr} check-task task.yaml
```

Use `tiup dmctl` to run the following command to start the data migration task.

```
tiup dmctl --master-addr ${advertise-addr} start-task task.yaml
```

The parameters in this command are described as follows.

Parameter	Description
–	{advertise-addr}
master-addr	of any DM-master node in the cluster that dmctl connects to. For example: 172.16.10.71:8261
start-task	Starts the data migration task.

If the task fails to start, first make configuration changes according to the prompt messages from the returned result, and then run the `start-task task.yaml` subcommand in `tiup dmctl` to restart the task. If you encounter problems, see Handle Errors and TiDB Data Migration FAQ.

6.3.5.4.3 Check the migration status

You can check if there are running migration tasks in the DM cluster and their status by running the `query-status` command in `tiup dmctl`.

```
tiup dmctl --master-addr ${advertise-addr} query-status ${task-name}
```

For more information, see Query Status.

6.3.5.4.4 Monitor tasks and view logs

You can view the history of a migration task and internal operational metrics through Grafana or logs.

- Via Grafana

  If Prometheus, Alertmanager, and Grafana are correctly deployed when you deploy the DM cluster using TiUP, you can view DM monitoring metrics in Grafana. Specifically, enter the IP address and port specified during deployment in Grafana and select the DM dashboard.

- Via logs

  When DM is running, DM-master, DM-worker, and dmctl output logs, which includes information about migration tasks. The log directory of each component is as follows.

  - DM-master log directory: It is specified by the DM-master command line parameter `--log-file`. If DM is deployed using TiUP, the log directory is `/dm-deploy/dm-master-8261/log/`.
  - DM-worker log directory: It is specified by the DM-worker command line parameter `--log-file`. If DM is deployed using TiUP, the log directory is `/dm-deploy/dm-worker-8262/log/`.

6.3.5.5 See also

- Dumpling
- TiDB Lightning
- Pessimistic mode and optimistic mode
- Pause a Data Migration Task
- Resume a Data Migration Task
- Stop a Data Migration Task
- Export and Import Data Sources and Task Configuration of Clusters
- Handle Failed DDL Statements
- Handle Errors

6.3.6 Migrate Data from CSV Files to TiDB

This document describes how to migrate data from CSV files to TiDB.

TiDB Lightning can read data from CSV files and other delimiter formats, such as tab-separated values (TSV). For other flat file data sources, you can also refer to this document and migrate data to TiDB.

6.3.6.1 Prerequisites

- Install TiDB Lightning.
- Get the target database privileges required for TiDB Lightning.

6.3.6.2 Step 1. Prepare the CSV files

Put all the CSV files in the same directory. If you need TiDB Lightning to recognize all CSV files, the file names should meet the following requirements:

- If a CSV file contains the data for an entire table, name the file `${db_name}.${table_name}.csv`.
- If the data of one table is separated into multiple CSV files, append a numeric suffix to these CSV files. For example, `${db_name}.${table_name}.003.csv`. The numeric suffixes can be inconsecutive but must be in ascending order. You also need to add extra zeros before the number to ensure all the suffixes are in the same length.

TiDB Lightning recursively searches for all `.csv` files in this directory and its subdirectories.

6.3.6.3 Step 2. Create the target table schema

Because CSV files do not contain schema information, before importing data from CSV files into TiDB, you need to create the target table schema. You can create the target table schema by either of the following two methods:

- **Method 1**: create the target table schema using TiDB Lightning.

  Create SQL files that contain the required DDL statements:

  - Add `CREATE DATABASE` statements in the `${db_name}-schema-create.sql` files.
  - Add `CREATE TABLE` statements in the `${db_name}.${table_name}-schema.sql` files.

- **Method 2**: create the target table schema manually.

6.3.6.4 Step 3. Create the configuration file

Create a `tidb-lightning.toml` file with the following content:

```
[lightning]
### Log
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
### "local": Default backend. The local backend is recommended to import
   ↪ large volumes of data (1 TiB or more). During the import, the target
   ↪ TiDB cluster cannot provide any service.
### "tidb": The "tidb" backend is recommended to import data less than 1 TiB
   ↪ . During the import, the target TiDB cluster can provide service
   ↪ normally.
```

```
backend = "local"
### Set the temporary storage directory for the sorted Key-Value files. The
    ↪ directory must be empty, and the storage space must be greater than
    ↪ the size of the dataset to be imported. For better import performance
    ↪ , it is recommended to use a directory different from `data-source-
    ↪ dir` and use flash storage, which can use I/O exclusively.
sorted-kv-dir = "/mnt/ssd/sorted-kv-dir"

[mydumper]
### Directory of the data source.
data-source-dir = "${data-path}" # A local path or S3 path. For example, 's3
    ↪ ://my-bucket/sql-backup?region=us-west-2'.

### Defines CSV format.
[mydumper.csv]
### Field separator of the CSV file. Must not be empty. If the source file
    ↪ contains fields that are not string or numeric, such as binary, blob,
    ↪  or bit, it is recommended not to usesimple delimiters such as ",",
    ↪ and use an uncommon character combination like "|+|" instead.
separator = ','
### Delimiter. Can be zero or multiple characters.
delimiter = '"'
### Configures whether the CSV file has a table header.
### If this item is set to true, TiDB Lightning uses the first line of the
    ↪ CSV file to parse the corresponding relationship of fields.
header = true
### Configures whether the CSV file contains NULL.
### If this item is set to true, any column of the CSV file cannot be parsed
    ↪  as NULL.
not-null = false
### If `not-null` is set to false (CSV contains NULL),
### The following value is parsed as NULL.
null = '\N'
### Whether to treat the backslash ('\') in the string as an escape
    ↪ character.
backslash-escape = true
### Whether to trim the last separator at the end of each line.
trim-last-separator = false

[tidb]
### The target cluster.
host = ${host}         # e.g.: 172.16.32.1
port = ${port}         # e.g.: 4000
user = "${user_name}"  # e.g.: "root"
password = "${password}" # e.g.: "rootroot"
```

```
status-port = ${status-port} # During the import, TiDB Lightning needs to
    ↪ obtain the table schema information from the TiDB status port. e.g.:
    ↪ 10080
pd-addr = "${ip}:${port}" # The address of the PD cluster, e.g.:
    ↪ 172.16.31.3:2379. TiDB Lightning obtains some information from PD.
    ↪ When backend = "local", you must specify status-port and pd-addr
    ↪ correctly. Otherwise, the import will be abnormal.
```

For more information on the configuration file, refer to TiDB Lightning Configuration.

6.3.6.5 Step 4. Tune the import performance (optional)

When you import data from CSV files with a uniform size of about 256 MiB, TiDB Lightning works in the best performance. However, if you import data from a single large CSV file, TiDB Lightning can only use one thread to process the import by default, which might slow down the import speed.

To speed up the import, you can split a large CSV file into smaller ones. For a CSV file in a common format, before TiDB Lightning reads the entire file, it is hard to quickly locate the beginning and ending positions of each line. Therefore, TiDB Lightning does not automatically split CSV files by default. But if your CSV files to be imported meet certain format requirements, you can enable the `strict-format` mode. In this mode, TiDB Lightning automatically splits a single large CSV file into multiple files, each in about 256 MiB, and processes them in parallel.

> **Note:**
>
> If a CSV file is not in a strict format but the `strict-format` mode is set to `true` by mistake, a field that spans multiple lines will be split into two fields. This causes the parsing to fail, and TiDB Lightning might import the corrupted data without reporting any error.

In a strict-format CSV file, each field only takes up one line. It must meet the following requirements:

- The delimiter is empty.
- Each field does not contain CR (`\r`) or LF (`\n`).

If your CSV file meets the above requirements, you can speed up the import by enabling the `strict-format` mode as follows:

```
[mydumper]
strict-format = true
```

6.3.6.6 Step 5. Import the data

To start the import, run `tidb-lightning`. If you launch the program in the command line, the process might exit unexpectedly after receiving a SIGHUP signal. In this case, it is recommended to run the program using a `nohup` or `screen` tool. For example:

```
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out 2>&1 &
```

After the import starts, you can check the progress of the import by either of the following methods:

- `grep` the keyword `progress` in the log. The progress is updated every 5 minutes by default.
- Check progress in the monitoring dashboard.
- Check progress in the TiDB Lightning web interface.

After TiDB Lightning completes the import, it exits automatically. Check whether `tidb-lightning.log` contains `the whole procedure completed` in the last lines. If yes, the import is successful. If no, the import encounters an error. Address the error as instructed in the error message.

> **Note:**
>
> Whether the import is successful or not, the last line of the log shows `tidb-lightning exit`. It means that TiDB Lightning exits normally, but does not necessarily mean that the import is successful.

If the import fails, refer to TiDB Lightning FAQ for troubleshooting.

6.3.6.7 Other file formats

If your data source is in other formats, to migrate data from your data source, you must end the file name with `.csv` and make corresponding changes in the `[mydumper.csv]` section of the `tidb-lightning.toml` configuration file. Here are example changes for common formats:

TSV:

```
### Format example
### ID   Region   Count
### 1    East     32
### 2    South    NULL
### 3    West     10
### 4    North    39
```

```
### Format configuration
[mydumper.csv]
separator = "\t"
delimiter = ''
header = true
not-null = false
null = 'NULL'
backslash-escape = false
trim-last-separator = false
```

TPC-H DBGEN:

```
### Format example
### 1|East|32|
### 2|South|0|
### 3|West|10|
### 4|North|39|

### Format configuration
[mydumper.csv]
separator = '|'
delimiter = ''
header = false
not-null = true
backslash-escape = false
trim-last-separator = true
```

6.3.6.8 What's next

- CSV Support and Restrictions.

6.3.7 Migrate Data from SQL Files to TiDB

This document describes how to migrate data from MySQL SQL files to TiDB using TiDB Lightning. For how to generate MySQL SQL files, refer to Export to SQL files using Dumpling.

6.3.7.1 Prerequisites

- Install TiDB Lightning using TiUP
- Grant the required privileges to the target database for TiDB Lightning

6.3.7.2 Step 1. Prepare SQL files

Put all the SQL files in the same directory, like `/data/my_datasource/` or `s3://my-bucket/sql-backup?region=us-west-2`. TiDB Lightning recursively searches for all `.sql` files in this directory and its subdirectories.

6.3.7.3 Step 2. Define the target table schema

To import data to TiDB, you need to create the table schema for the target database.

If you use Dumpling to export data, the table schema file is automatically exported. For the data exported in other ways, you can create the table schema in one of the following methods:

- **Method 1**: Create the target table schema using TiDB Lightning.

  Create SQL files that contain the required DDL statements:

  - Add `CREATE DATABASE` statements in the `${db_name}-schema-create.sql` files.
  - Add `CREATE TABLE` statements in the `${db_name}.${table_name}-schema.sql` files.

- **Method 2**: Create the target table schema manually.

6.3.7.4 Step 3. Create the configuration file

Create a `tidb-lightning.toml` file with the following content:

```
[lightning]
### Log
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
### "local": Default. The local backend is used to import large volumes of
    ↪ data (around or more than 1 TiB). During the import, the target TiDB
    ↪ cluster cannot provide any service.
### "tidb": The "tidb" backend can also be used to import small volumes of
    ↪ data (less than 1 TiB). During the import, the target TiDB cluster
    ↪ can provide service normally. For the information about backend mode,
    ↪  refer to https://docs.pingcap.com/tidb/stable/tidb-lightning-
    ↪ backends.

backend = "local"
### Sets the temporary storage directory for the sorted key-value files. The
    ↪  directory must be empty, and the storage space must be greater than
    ↪ the size of the dataset to be imported. For better import performance
    ↪ , it is recommended to use a directory different from `data-source-
    ↪ dir` and use flash storage and exclusive I/O for the directory.
```

```
sorted-kv-dir = "${sorted-kv-dir}"

[mydumper]
### Directory of the data source
data-source-dir = "${data-path}" # Local or S3 path, such as 's3://my-bucket
    ↪ /sql-backup?region=us-west-2'

[tidb]
### The information of target cluster
host = ${host}             # For example, 172.16.32.1
port = ${port}             # For example, 4000
user = "${user_name}"      # For example, "root"
password = "${password}"   # For example, "rootroot"
status-port = ${status-port} # During the import process, TiDB Lightning
    ↪ needs to obtain table schema information from the "Status Port" of
    ↪ TiDB, such as 10080.
pd-addr = "${ip}:${port}" # The address of the cluster's PD. TiDB Lightning
    ↪  obtains some information through PD, such as 172.16.31.3:2379. When
    ↪ backend = "local", you must correctly specify status-port and pd-addr
    ↪ . Otherwise, the import will encounter errors.
```

For more information about the configuration file, refer to TiDB Lightning Configuration.

6.3.7.5 Step 4. Import the data

To start the import, run `tidb-lightning`. If you launch the program in the command line, the program might exit because of the `SIGHUP` signal. In this case, it is recommended to run the program with a `nohup` or `screen` tool.

If you import the data from S3, you need to pass in `SecretKey` and `AccessKey` of the account as environment variables. The account has the permission to access the S3 backend storage.

```
export AWS_ACCESS_KEY_ID=${access_key}
export AWS_SECRET_ACCESS_KEY=${secret_key}
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out 2>&1 &
```

TiDB Lightning also supports reading credential files from `~/.aws/credentials`.

After the import is started, you can check the progress in one of the following ways:

- Search the `progress` keyword in the `grep` log, which is updated every 5 minutes by default.
- Use the Grafana dashboard. For details, see TiDB Lightning Monitoring.
- Use web interface. For details, see TiDB Lightning Web Interface.

After the import is completed, TiDB Lightning automatically exits. Check whether `tidb -lightning.log` contains `the whole procedure completed` in the last lines. If yes, the import is successful. If no, the import encounters an error. Address the error as instructed in the error message.

> **Note:**
>
> No matter whether the import is successful or not, the last line displays `tidb lightning exit`. It only means that TiDB Lightning has exited normally, not the completion of the task. If you encounter problems during the import process, refer to TiDB Lightning FAQ for troubleshooting.

6.3.8 Migrate from One TiDB Cluster to Another TiDB Cluster

This document describes how to migrate data from one TiDB cluster to another TiDB cluster. This function applies to the following scenarios:

- Split databases: You can split databases when a TiDB cluster is excessively large, or you want to avoid impact between services of a cluster.
- Relocate databases: Physically relocate databases, such as changing the data center.
- Migrate data to a TiDB cluster of a newer version: Migrate data to a TiDB cluster of a newer version to satisfy data security and accuracy requirements.

This document exemplifies the whole migration process and contains the following steps:

1. Set up the environment.
2. Migrate full data.
3. Migrate incremental data.
4. Migrate services to the new TiDB cluster.

6.3.8.1 Step 1. Set up the environment

1. Deploy TiDB clusters.

   Deploy two TiDB clusters, one upstream and the other downstream by using TiUP Playground. For more information, refer to Deploy and Maintain an Online TiDB Cluster Using TiUP.

```
# Create an upstream cluster
tiup --tag upstream playground --host 0.0.0.0 --db 1 --pd 1 --kv 1 --
    ↪ tiflash 0 --ticdc 1
# Create a downstream cluster
tiup --tag downstream playground --host 0.0.0.0 --db 1 --pd 1 --kv 1 --
    ↪ tiflash 0 --ticdc 1
# View cluster status
tiup status
```

2. Initialize data.

   By default, test databases are created in the newly deployed clusters. Therefore, you can use sysbench to generate test data and simulate data in real scenarios.

```
sysbench oltp_write_only --config-file=./tidb-config --tables=10 --
    ↪ table-size=10000 prepare
```

   In this document, we use sysbench to run the `oltp_write_only` script. This script generates 10 tables in the test database, each with 10,000 rows. The tidb-config is as follows:

```
mysql-host=172.16.6.122 # Replace the value with the IP address of your
    ↪  upstream cluster
mysql-port=4000
mysql-user=root
mysql-password=
db-driver=mysql      # Set database driver to MySQL
mysql-db=test        # Set the database as a test database
report-interval=10   # Set data collection period to 10s
threads=10           # Set the number of worker threads to 10
time=0               # Set the time required for executing the script.
    ↪ 0 indicates time unlimited
rate=100             # Set average TPS to 100
```

3. Simulate service workload.

   In real scenarios, service data is continuously written to the upstream cluster. In this document, we use sysbench to simulate this workload. Specifically, run the following command to enable 10 workers to continuously write data to three tables, sbtest1, sbtest2, and sbtest3, with a total TPS not exceeding 100.

```
sysbench oltp_write_only --config-file=./tidb-config --tables=3 run
```

4. Prepare external storage.

   In full data backup, both the upstream and downstream clusters need to access backup files. It is recommended that you use External storage to store backup files. In this document, Minio is used to simulate an S3-compatible storage service.

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
# Configure access-key access-screct-id to access minio
export HOST_IP='172.16.6.122' # Replace the value with the IP address
    ↪ of your upstream cluster
export MINIO_ROOT_USER='minio'
export MINIO_ROOT_PASSWORD='miniostorage'
# Create the database directory. backup is the bucket name.
mkdir -p data/backup
# Start minio at port 6060
./minio server ./data --address :6060 &
```

The preceding command starts a minio server on one node to simulate S3 services. Parameters in the command are configured as follows:

- Endpoint: `http://${HOST_IP}:6060/`
- Access-key: `minio`
- Secret-access-key: `miniostorage`
- Bucket: `backup`

The access link is as follows:

```
s3://backup?access-key=minio&secret-access-key=miniostorage&endpoint=
    ↪ http://${HOST_IP}:6060&force-path-style=true
```

6.3.8.2 Step 2. Migrate full data

After setting up the environment, you can use the backup and restore functions of BR to migrate full data. BR can be started in three ways. In this document, we use the SQL statements, `BACKUP` and `RESTORE`.

Note:

- `BACKUP` and `RESTORE` SQL statements are experimental. It is not recommended that you use them in the production environment. They might be changed or removed without prior notice. If you find a bug, you can report an issue on GitHub.
- In production clusters, performing a backup with GC disabled might affect cluster performance. It is recommended that you back up data in off-peak hours, and set `RATE_LIMIT` to a proper value to avoid performance degradation.

- If the versions of the upstream and downstream clusters are different, you should check BR compatibility. In this document, we assume that the upstream and downstream clusters are the same version.

1. Disable GC.

   To ensure that newly written data is not deleted during incremental migration, you should disable GC for the upstream cluster before backup. In this way, history data is not deleted.

   Run the following command to disable GC:

```
MySQL [test]> SET GLOBAL tidb_gc_enable=FALSE;
```

```
Query OK, 0 rows affected (0.01 sec)
```

   To verify that the change takes effect, query the value of `tidb_gc_enable`:

```
MySQL [test]> SELECT @@global.tidb_gc_enable;
```

```
+-------------------------+:
| @@global.tidb_gc_enable |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set (0.00 sec)
```

2. Back up data.

   Run the `BACKUP` statement in the upstream cluster to back up data:

```
MySQL [(none)]> BACKUP DATABASE * TO 's3://backup?access-key=minio&
    ↪ secret-access-key=miniostorage&endpoint=http://${HOST_IP}:6060&
    ↪ force-path-style=true' RATE_LIMIT = 120 MB/SECOND;
```

```
+--------------+----------+--------------------+---------------------+------------
    ↪
| Destination | Size     | BackupTS           | Queue Time          |
    ↪ Execution Time    |
+--------------+----------+--------------------+---------------------+------------
    ↪
| s3://backup | 10315858 | 431434047157698561 | 2022-02-25 19:57:59 |
    ↪ 2022-02-25 19:57:59 |
+--------------+----------+--------------------+---------------------+------------
    ↪
1 row in set (2.11 sec)
```

After the `BACKUP` command is executed, TiDB returns metadata about the backup data. Pay attention to `BackupTS`, because data generated before it is backed up. In this document, we use `BackupTS` as **the end of data check** and **the start of incremental migration scanning by TiCDC**.

3. Restore data.

   Run the `RESTORE` command in the downstream cluster to restore data:

```
mysql> RESTORE DATABASE * FROM 's3://backup?access-key=minio&secret-
    ↪ access-key=miniostorage&endpoint=http://${HOST_IP}:6060&force-
    ↪ path-style=true';
```

```
+--------------+-----------+--------------------+---------------------+------------
    ↪
| Destination | Size    | BackupTS         | Queue Time        |
    ↪ Execution Time    |
+--------------+-----------+--------------------+---------------------+------------
    ↪
| s3://backup | 10315858 | 431434141450371074 | 2022-02-25 20:03:59 |
    ↪ 2022-02-25 20:03:59 |
+--------------+-----------+--------------------+---------------------+------------
    ↪
1 row in set (41.85 sec)
```

4. (Optional) Validate data.

   You can use sync-diff-inspector to check data consistency between upstream and downstream at a certain time. The preceding `BACKUP` output shows that the upstream cluster finishes backup at 431434047157698561. The preceding `RESTORE` output shows that the downstream finishes restoration at 431434141450371074.

```
sync_diff_inspector -C ./config.yaml
```

   For details about how to configure the sync-diff-inspector, see Configuration file description. In this document, the configuration is as follows:

```
# Diff Configuration.
######################### Datasource config ##########################
[data-sources]
[data-sources.upstream]
    host = "172.16.6.122" # Replace the value with the IP address of
        ↪ your upstream cluster
    port = 4000
    user = "root"
    password = ""
    snapshot = "431434047157698561" # Set snapshot to the actual backup
        ↪  time (BackupTS in the "Back up data" section in [Step 2.
        ↪ Migrate full data](#step-2-migrate-full-data))
```

```
[data-sources.downstream]
    host = "172.16.6.125" # Replace the value with the IP address of
        ↪ your downstream cluster
    port = 4000
    user = "root"
    password = ""

######################### Task config ########################
[task]
    output-dir = "./output"
    source-instances = ["upstream"]
    target-instance = "downstream"
    target-check-tables = ["*.*"]
```

6.3.8.3 Step 3. Migrate incremental data

1. Deploy TiCDC.

   After finishing full data migration, deploy and configure a TiCDC to replicate incremental data. In production environments, deploy TiCDC as instructed in Deploy TiCDC. In this document, a TiCDC node has been started upon the creation of the test clusters. Therefore, you can skip the step of deploying TiCDC and proceed with changefeed configuration.

2. Create a changefeed.

   In the upstream cluster, run the following command to create a changefeed from the upstream to the downstream clusters:

   ```
   tiup cdc cli changefeed create --pd=http://172.16.6.122:2379 --sink-uri
       ↪ ="mysql://root:@172.16.6.125:4000" --changefeed-id="upstream-to-
       ↪ downstream" --start-ts="431434047157698561"
   ```

   In this command, the parameters are as follows:

   - `--pd`: PD address of the upstream cluster
   - `--sink-uri`: URI of the downstream cluster
   - `--changefeed-id`: changefeed ID, must be in the format of a regular expression, [4]+(-[a-zA-Z0-9]+)*$
   - `--start-ts`: start timestamp of the changefeed, must be the backup time (or BackupTS in the "Back up data" section in Step 2. Migrate full data)

   For more information about the changefeed configurations, see Task configuration file.

[4] a-zA-Z0-9

3. Enable GC.

   In incremental migration using TiCDC, GC only removes history data that is replicated. Therefore, after creating a changefeed, you need to run the following command to enable GC. For details, see What is the complete behavior of TiCDC garbage collection (GC) safepoint?.

   To enable GC, run the following command:

```
MySQL [test]> SET GLOBAL tidb_gc_enable=TRUE;
```

```
Query OK, 0 rows affected (0.01 sec)
```

   To verify that the change takes effect, query the value of `tidb_gc_enable`:

```
MySQL [test]> SELECT @@global.tidb_gc_enable;
```

```
+-------------------------+
| @@global.tidb_gc_enable |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)
```

6.3.8.4 Step 4. Migrate services to the new TiDB cluster

After creating a changefeed, data written to the upstream cluster is replicated to the downstream cluster with low latency. You can migrate read traffic to the downstream cluster gradually. Observe for a period. If the downstream cluster is stable, you can migrate write traffic to the downstream cluster by performing the following steps:

1. Stop write services in the upstream cluster. Make sure that all upstream data are replicated to downstream before stopping the changefeed.

```
# Stop the changefeed from the upstream cluster to the downstream
   ↪ cluster
tiup cdc cli changefeed pause -c "upstream-to-downstream" --pd=http
   ↪ ://172.16.6.122:2379

# View the changefeed status
tiup cdc cli changefeed list
```

```
[
  {
    "id": "upstream-to-downstream",
    "summary": {
    "state": "stopped", # Ensure that the status is stopped
```

```
    "tso": 431747241184329729,
    "checkpoint": "2022-03-11 15:50:20.387", # This time must be later
       ↪ than the time of stopping writing
    "error": null
    }
  }
]
```

2. Create a changefeed from downstream to upstream. You can leave `start-ts` unspecified so as to use the default setting, because the upstream and downstream data are consistent and there is no new data written to the cluster.

```
tiup cdc cli changefeed create --pd=http://172.16.6.125:2379 --sink-uri
   ↪ ="mysql://root:@172.16.6.122:4000" --changefeed-id="downstream -
   ↪ to-upstream"
```

3. After migrating writing services to the downstream cluster, observe for a period. If the downstream cluster is stable, you can discard the upstream cluster.

6.3.9 Migrate Data from TiDB to MySQL-compatible Databases

This document describes how to migrate data from TiDB clusters to MySQL-compatible databases, such as Aurora, MySQL, and MariaDB. The whole process contains four steps:

1. Set up the environment.
2. Migrate full data.
3. Migrate incremental data.
4. Migrate services to the MySQL-compatible cluster.

6.3.9.1 Step 1. Set up the environment

1. Deploy a TiDB cluster upstream.

   Deploy a TiDB cluster by using TiUP Playground. For more information, refer to Deploy and Maintain an Online TiDB Cluster Using TiUP.

```
# Create a TiDB cluster
tiup playground --db 1 --pd 1 --kv 1 --tiflash 0 --ticdc 1
# View cluster status
tiup status
```

2. Deploy a MySQL instance downstream.

   - In a lab environment, you can use Docker to quickly deploy a MySQL instance by running the following command:

```
docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -p
    ↪ 3306:3306 -d mysql
```

- In a production environment, you can deploy a MySQL instance by following instructions in Installing MySQL.

3. Simulate service workload.

   In the lab environment, you can use `go-tpc` to write data to the TiDB cluster upstream. This is to generate event changes in the TiDB cluster. Run the following command to create a database named `tpcc` in the TiDB cluster, and then use TiUP bench to write data to this database.

```
tiup bench tpcc -H 127.0.0.1 -P 4000 -D tpcc --warehouses 4 prepare
tiup bench tpcc -H 127.0.0.1 -P 4000 -D tpcc --warehouses 4 run --time
    ↪ 300s
```

   For more details about `go-tpc`, refer to How to Run TPC-C Test on TiDB.

6.3.9.2 Step 2. Migrate full data

After setting up the environment, you can use Dumpling to export the full data from the upstream TiDB cluster.

> **Note:**
>
> In production clusters, performing a backup with GC disabled might affect cluster performance. It is recommended that you complete this step in off-peak hours.

1. Disable Garbage Collection (GC).

   To ensure that newly written data is not deleted during incremental migration, you should disable GC for the upstream cluster before exporting full data. In this way, history data is not deleted. For TiDB v4.0.0 and later versions, Dumpling might automatically adjust the GC safe point to block GC. Nevertheless, manually disabling GC is still necessary because the GC process might begin after Dumpling exits, leading to the failure of incremental changes migration.

   Run the following command to disable GC:

```
MySQL [test]> SET GLOBAL tidb_gc_enable=FALSE;
```

```
Query OK, 0 rows affected (0.01 sec)
```

To verify that the change takes effect, query the value of `tidb_gc_enable`:

```
MySQL [test]> SELECT @@global.tidb_gc_enable;
```

```
+-------------------------+:
| @@global.tidb_gc_enable |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set (0.00 sec)
```

2. Back up data.

    1. Export data in SQL format using Dumpling:

    ```
    tiup dumpling -u root -P 4000 -h 127.0.0.1 --filetype sql -t 8 -o
        ↪ ./dumpling_output -r 200000 -F256MiB
    ```

    2. After finishing exporting data, run the following command to check the metadata. `Pos` in the metadata is the TSO of the export snapshot and can be recorded as the BackupTS.

    ```
    cat dumpling_output/metadata
    ```

    ```
    Started dump at: 2022-06-28 17:49:54
    SHOW MASTER STATUS:
            Log: tidb-binlog
            Pos: 434217889191428107
            GTID:
    Finished dump at: 2022-06-28 17:49:57
    ```

3. Restore data.

    Use MyLoader (an open-source tool) to import data to the downstream MySQL instance. For details about how to install and use MyLoader, see MyDumpler/MyLoader. Run the following command to import full data exported by Dumpling to MySQL:

    ```
    myloader -h 127.0.0.1 -P 3306 -d ./dumpling_output/
    ```

4. (Optional) Validate data.

    You can use sync-diff-inspector to check data consistency between upstream and downstream at a certain time.

    ```
    sync_diff_inspector -C ./config.yaml
    ```

    For details about how to configure the sync-diff-inspector, see Configuration file description. In this document, the configuration is as follows:

```
# Diff Configuration.
######################### Datasource config #########################
[data-sources]
[data-sources.upstream]
       host = "127.0.0.1" # Replace the value with the IP address of
          ↪ your upstream cluster
       port = 4000
       user = "root"
       password = ""
       snapshot = "434217889191428107" # Set snapshot to the actual
          ↪ backup time (BackupTS in the "Back up data" section in [
          ↪ Step 2. Migrate full data](#step-2-migrate-full-data))
[data-sources.downstream]
       host = "127.0.0.1" # Replace the value with the IP address of
          ↪ your downstream cluster
       port = 3306
       user = "root"
       password = ""
######################### Task config #########################
[task]
       output-dir = "./output"
       source-instances = ["upstream"]
       target-instance = "downstream"
       target-check-tables = ["*.*"]
```

6.3.9.3 Step 3. Migrate incremental data

1. Deploy TiCDC.

   After finishing full data migration, deploy and configure a TiCDC cluster to replicate incremental data. In production environments, deploy TiCDC as instructed in Deploy TiCDC. In this document, a TiCDC node has been started upon the creation of the test cluster. Therefore, you can skip the step of deploying TiCDC and proceed with the next step to create a changefeed.

2. Create a changefeed.

   In the upstream cluster, run the following command to create a changefeed from the upstream to the downstream clusters:

```
tiup ctl:<cluster-version> cdc changefeed create --pd=http
   ↪ ://127.0.0.1:2379 --sink-uri="mysql://root:@127.0.0.1:3306" --
   ↪ changefeed-id="upstream-to-downstream" --start-ts
   ↪ ="434217889191428107"
```

In this command, the parameters are as follows:

- `--pd`: PD address of the upstream cluster
- `--sink-uri`: URI of the downstream cluster
- `--changefeed-id`: changefeed ID, must be in the format of a regular expression, `^[a-zA-Z0-9]+(\-[a-zA-Z0-9]+)*$`
- `--start-ts`: start timestamp of the changefeed, must be the backup time (or BackupTS in the "Back up data" section in Step 2. Migrate full data)

For more information about the changefeed configurations, see Task configuration file.

3. Enable GC.

   In incremental migration using TiCDC, GC only removes history data that is replicated. Therefore, after creating a changefeed, you need to run the following command to enable GC. For details, see What is the complete behavior of TiCDC garbage collection (GC) safepoint.

To enable GC, run the following command:

````
```sql
MySQL [test]> SET GLOBAL tidb_gc_enable=TRUE;
```


```
Query OK, 0 rows affected (0.01 sec)
```

To verify that the change takes effect, query the value of `tidb_gc_enable`:

```sql
MySQL [test]> SELECT @@global.tidb_gc_enable;
```


```
+-------------------------+
| @@global.tidb_gc_enable |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)
```
````

6.3.9.4 Step 4. Migrate services

After creating a changefeed, data written to the upstream cluster is replicated to the downstream cluster with low latency. You can migrate read traffic to the downstream cluster gradually. Observe the read traffic for a period. If the downstream cluster is stable, you can migrate write traffic to the downstream cluster as well in the following steps:

1. Stop write services in the upstream cluster. Make sure that all upstream data are replicated to downstream before stopping the changefeed.

```
# Stop the changefeed from the upstream cluster to the downstream
    ↪ cluster
tiup cdc cli changefeed pause -c "upstream-to-downstream" --pd=http
    ↪ ://172.16.6.122:2379
# View the changefeed status
tiup cdc cli changefeed list
```

```
[
  {
    "id": "upstream-to-downstream",
    "summary": {
    "state": "stopped", # Ensure that the status is stopped
    "tso": 434218657561968641,
    "checkpoint": "2022-06-28 18:38:45.685", # This time should be
       ↪ later than the time of stopping writing
    "error": null
    }
  }
]
```

2. After migrating writing services to the downstream cluster, observe for a period. If the downstream cluster is stable, you can discard the upstream cluster.

6.4 Advanced Migration

6.4.1 Continuous Replication from Databases that Use gh-ost or pt-osc

In production scenarios, table locking during DDL execution can block the reads from or writes to the database to a certain extent. Therefore, online DDL tools are often used to execute DDLs to minimize the impact on reads and writes. Common DDL tools are gh-ost and pt-osc.

When using DM to migrate data from MySQL to TiDB, you can enbale `online-ddl` to allow collaboration of DM and gh-ost or pt-osc.

For the detailed replication instructions, refer to the following documents by scenarios:

- Migrate MySQL of Small Datasets to TiDB

- Migrate MySQL of Large Datasets to TiDB
- Migrate and Merge MySQL Shards of Small Datasets to TiDB
- Migrate and Merge MySQL Shards of Large Datasets to TiDB

6.4.1.1 Enable online-ddl on DM

In the task configuration file of DM, set the global parameter `online-ddl` to `true`, as shown below:

```
### ----------- Global configuration -----------
#### ********* Basic configuration *********
name: test                    # The name of the task. Should be globally
    ↪ unique.
task-mode: all                # The task mode. Can be set to `full`, `
    ↪ incremental`, or `all`.
shard-mode: "pessimistic"     # The shard merge mode. Optional modes are `
    ↪ pessimistic` and `optimistic`. The `pessimistic` mode is used by
    ↪ default. After understanding the principles and restrictions of the "
    ↪ optimistic" mode, you can set it to the "optimistic" mode.
meta-schema: "dm_meta"        # The downstream database that stores the `meta`
    ↪  information.
online-ddl: true              # Enable online-ddl support on DM to support
    ↪ automatic processing of "gh-ost" and "pt-osc" for the upstream
    ↪ database.
```

6.4.1.2 Workflow after enabling online-ddl

After online-ddl is enabled on DM, the DDL statements generated by DM replicating gh-ost or pt-osc will change.

The workflow of gh-ost or pt-osc:

- Create a ghost table according to the table schema of the DDL real table.
- Apply DDLs on the ghost table.
- Replicate the data of the DDL real table to the ghost table.
- After the data are consistent between the two tables, use the rename statement to replace the real table with the ghost table.

The workflow of DM:

- Skip creating the ghost table downstream.
- Record DDLs applied to the ghost table.

- Replicate data only from the ghost table.
- Apply DDLs recorded downstream.

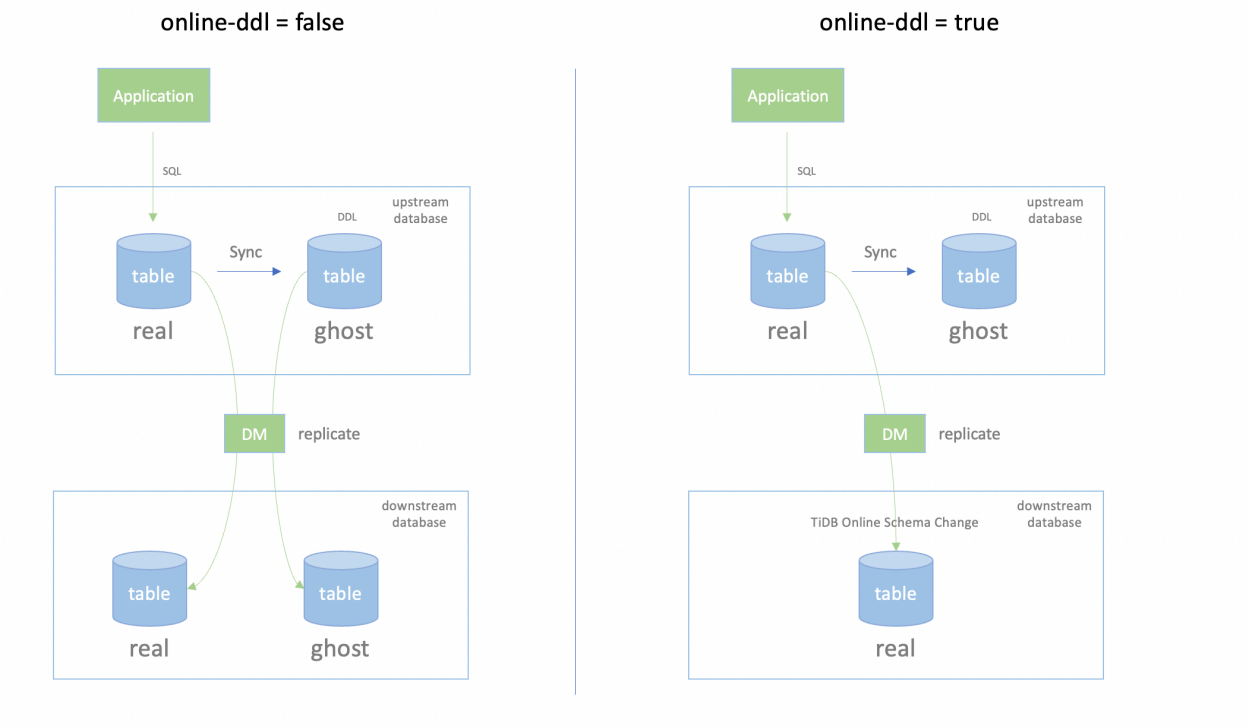


Figure 51: dm-online-ddl

The change in the workflow brings the following advantages:

- The downstream TiDB does not need to create and replicate the ghost table, saving the storage space and network transmission overhead.
- When you migrate and merge data from sharded tables, the RENAME operation is ignored for each sharded ghost table to ensure the correctness of the replication.

6.4.1.3 See also

Working details for DM with online DDL tools

6.4.2 Migrate Data to a Downstream TiDB Table with More Columns

This document provides the additional steps to be taken when you migrate data to a downstream TiDB table with more columns than the corresponding upstream table. For regular migration steps, see the following migration scenarios:

- Migrate MySQL of Small Datasets to TiDB
- Migrate MySQL of Large Datasets to TiDB
- Migrate and Merge MySQL Shards of Small Datasets to TiDB
- Migrate and Merge MySQL Shards of Large Datasets to TiDB

6.4.2.1 Use DM to migrate data to a downstream TiDB table with more columns

When replicating the upstream binlog, DM tries to use the current table schema of the downstream to parse the binlog and generate the corresponding DML statements. If the column number of the table in the upstream binlog does not match the column number in the downstream table schema, the following error occurs:

```
"errors": [
    {
        "ErrCode": 36027,
        "ErrClass": "sync-unit",
        "ErrScope": "internal",
        "ErrLevel": "high",
        "Message": "startLocation: [position: (mysql-bin.000001, 2022), gtid-
            ↪ set:09bec856-ba95-11ea-850a-58f2b4af5188:1-9 ], endLocation: [
            ↪  position: (mysql-bin.000001, 2022), gtid-set: 09bec856-ba95
            ↪ -11ea-850a-58f2b4af5188:1-9]: gen insert sqls failed, schema:
            ↪ log, table: messages: Column count doesn't match value count:
            ↪ 3 (columns) vs 2 (values)",
        "RawCause": "",
        "Workaround": ""
    }
]
```

The following is an example upstream table schema:

```
### Upstream table schema
CREATE TABLE `messages` (
  `id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
)
```

The following is an example downstream table schema:

```
### Downstream table schema
CREATE TABLE `messages` (
  `id` int(11) NOT NULL,
  `message` varchar(255) DEFAULT NULL, # This is the additional column that
     ↪  only exists in the downstream table.
  PRIMARY KEY (`id`)
)
```

When DM tries to use the downstream table schema to parse the binlog event generated by the upstream, DM reports the above `Column count doesn't match` error.

In such cases, you can use the `binlog-schema` command to set a table schema for the table to be migrated from the data source. The specified table schema needs to correspond to the binlog event data to be replicated by DM. If you are migrating sharded tables, for each sharded table, you need to set a table schema in DM to parse binlog event data. The steps are as follows:

1. Create a SQL file in DM and add the `CREATE TABLE` statement that corresponds to the upstream table schema to the file. For example, save the following table schema to `log.messages.sql`. For DM v6.0 or later versions, you can update the table schema by adding the `--from-source` or `--from-target` flag without creating a SQL file. For details, see Manage Table Schemas of Tables to be Migrated.

```
# Upstream table schema
CREATE TABLE `messages` (
`id` int(11) NOT NULL,
PRIMARY KEY (`id`)
)
```

2. Use the `binlog-schema` command to set the table schema for the table to be migrated from the data source. At this time, the data migration task should be in the Paused state due to the above `Column count doesn't match` error.

```
tiup dmctl --master-addr ${advertise-addr} binlog-schema update -s ${
    ↪ source-id} ${task-name} ${database-name} ${table-name} ${schema-
    ↪ file}
```

   The descriptions of parameters in this command are as follows:

Parameter	Description
- master-addr	Specifies ${ advertise-addr } of any DM-master node in the cluster where dm-ctl is to be connected.
- advertise-addr	${ advertise-addr } indicates the address that DM-master advertises to the outside world.

Parameter	Description
`binlog-schema-set`	Manually set the schema information.
`-s`	Specifies the source. `${source-id}` indicates the source ID of MySQL data.

Parameter	Description
`${task-name}`	Specifies the name of the migration task defined in the `task.yaml` configuration file of the data migration task.

Parameter	Description
${database-name}	Specifies the database.
${table-name}	Specifies the name of the upstream table.
${schema-file}	Specifies the schema file to be set.

For example:

```
tiup dmctl --master-addr 172.16.10.71:8261 binlog-schema update -s
   ↪ mysql-01 task-test -d log -t message log.message.sql
```

3. Use the `resume-task` command to resume the migration task in the Paused state.

```
tiup dmctl --master-addr ${advertise-addr} resume-task ${task-name}
```

4. Use the `query-status` command to confirm that the data migration task is running correctly.

```
tiup dmctl --master-addr ${advertise-addr} query-status ${task-name}
```

6.4.3 Filter Binlog Events

This document describes how to filter binlog events when you use DM to perform continuous incremental data replication. For the detailed replication instructions, refer to the following documents by scenarios:

- Migrate MySQL of Small Datasets to TiDB
- Migrate MySQL of Large Datasets to TiDB
- Migrate and Merge MySQL Shards of Small Datasets to TiDB
- Migrate and Merge MySQL Shards of Large Datasets to TiDB

6.4.3.1 Configuration

To use binlog event filter, add a `filter` to the task configuration file of DM, as shown below:

```
filters:
  rule-1:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    events: ["truncate table", "drop table"]
    sql-pattern: ["^DROP\\s+PROCEDURE", "^CREATE\\s+PROCEDURE"]
    action: Ignore
```

- `schema-pattern/table-pattern`: Filters matching schemas or tables
- `events`: Filters binlog events. Supported events are listed in the table below:

Event	Category	Description
all		Includes all events
all dml		Includes all DML events
all ddl		Includes all DDL events
none		Includes no event
none ddl		Excludes all DDL events
none dml		Excludes all DML events
insert	DML	Insert DML event
update	DML	Update DML event
delete	DML	Delete DML event
create database	DDL	Create database event

Event	Category	Description
drop database	DDL	Drop database event
create table	DDL	Create table event
create index	DDL	Create index event
drop table	DDL	Drop table event
truncate table	DDL	Truncate table event
rename table	DDL	Rename table event
drop index	DDL	Drop index event
alter table	DDL	Alter table event

- `sql-pattern`: Filters specified DDL SQL statements. The matching rule supports using a regular expression.
- `action`: `Do` or `Ignore`
    - `Do`: the allow list. A binlog event is replicated if meeting either of the following two conditions:
        - The event matches the rule setting.
        - sql-pattern has been specified and the SQL statement of the event matches any of the sql-pattern options.
    - `Ignore`: the block list. A binlog event is filtered out if meeting either of the following two conditions:
        - The event matches the rule setting.
        - sql-pattern has been specified and the SQL statement of the event matches any of the sql-pattern options.

    If both `Do` and `Ignore` are configured, `Ignore` has higher priority over `Do`. That is, an event satisfying both `Ignore` and `Do` conditions will be filtered out.

6.4.3.2 Application scenarios

This section describes the application scenarios of binlog event filter.

6.4.3.2.1 Filter out all sharding deletion operations

To filter out all deletion operations, configure a `filter-table-rule` and a `filter-schema-rule`, as shown below:

```
filters:
  filter-table-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
```

```
  filter-schema-rule:
    schema-pattern: "test_*"
    events: ["drop database"]
    action: Ignore
```

6.4.3.2.2 Migrate only DML operations of sharded schemas and tables

To replicate only DML statements, configure two `Binlog event filter rule`, as shown below:

```
filters:
  do-table-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    events: ["create table", "all dml"]
    action: Do
  do-schema-rule:
    schema-pattern: "test_*"
    events: ["create database"]
    action: Do
```

6.4.3.2.3 Filter out SQL statements not supported by TiDB

To filter out SQL statements not supported by TiDB, configure a `filter-procedure-rule`, as shown below:

```
filters:
  filter-procedure-rule:
    schema-pattern: "*"
    sql-pattern: [".*\\s+DROP\\s+PROCEDURE", ".*\\s+CREATE\\s+PROCEDURE", "
      ↪ ALTER\\s+TABLE[\\s\\S]*ADD\\s+PARTITION", "ALTER\\s+TABLE[\\s\\S]*
      ↪ DROP\\s+PARTITION"]
    action: Ignore
```

> **Warning:**
>
> To avoid filtering out data that needs to be migrated, configure the global filtering rule as strictly as possible.

6.4.3.3 See also

Filter Binlog Events Using SQL Expressions

6.4.4 Filter DML Events Using SQL Expressions

This document introduces how to filter binlog events using SQL expressions when you use DM to perform continuous incremental data replication. For the detailed replication instruction, refer to the following documents:

- Migrate MySQL of Small Datasets to TiDB
- Migrate MySQL of Large Datasets to TiDB
- Migrate and Merge MySQL Shards of Small Datasets to TiDB
- Migrate and Merge MySQL Shards of Large Datasets to TiDB

When performing incremental data replication, you can use the Binlog Event Filter to filter certain types of binlog events. For example, you can choose not to replicate `DELETE` events to the downstream for the purposes like archiving and auditing. However, the Binlog Event Filter cannot determine whether to filter the `DELETE` event of a row that requires finer granularity.

To address the issue, since v2.0.5, DM supports using `binlog value filter` in incremental data replication to filter data. Among the DM-supported and `ROW`-formatted binlog, the binlog events carry values of all columns, and you can configure SQL expressions based on these values. If the expression calculates a row change as `TRUE`, DM does not replicate this row change to the downstream.

Similar to Binlog Event Filter, you need to configure `binlog value filter` in the task configuration file. For details, see the following configuration example. For the advanced task configuration and the description, refer to DM advanced task configuration file.

```
name: test
task-mode: all

mysql-instances:
  - source-id: "mysql-replica-01"
    expression-filters: ["even_c"]

expression-filter:
  even_c:
    schema: "expr_filter"
    table: "tbl"
    insert-value-expr: "c % 2 = 0"
```

In the above configuration example, the `even_c` rule is configured and referenced by the data source `mysql-replica-01`. According to this rule, for the `tb1` table in the `expr_filter` ↪ schema, when an even number is inserted into the `c` column (`c % 2 = 0`), this `insert` statement is not replicated to the downstream. The following example shows the effect of this rule.

Incrementally insert the following data in the upstream data source:

```
INSERT INTO tbl(id, c) VALUES (1, 1), (2, 2), (3, 3), (4, 4);
```

Then query the `tb1` table on downstream. You can see that only the rows with odd numbers on `c` are replicated.

```
MySQL [test]> select * from tbl;
+------+------+
| id   | c    |
+------+------+
|    1 |    1 |
|    3 |    3 |
+------+------+
2 rows in set (0.001 sec)
```

6.4.4.1 Configuration parameters and description

- `schema`: The name of the upstream schema to match. Wildcard matching or regular matching is not supported.
- `table`: The name of the upstream table to match. Wildcard matching or regular matching is not supported.
- `insert-value-expr`: Configures an expression that takes effect on values carried by the `INSERT` type binlog events (WRITE_ROWS_EVENT). You cannot use this expression together with `update-old-value-expr`, `update-new-value-expr` or `delete-value-expr` in the same configuration item.
- `update-old-value-expr`: Configures an expression that takes effect on the old values carried by the `UPDATE` type binlog events (UPDATE_ROWS_EVENT). You cannot use this expression together with `insert-value-expr` or `delete-value-expr` in the same configuration item.
- `update-new-value-expr`: Configures an expression that takes effect on the new values carried by the `UPDATE` type binlog events (UPDATE_ROWS_EVENT). You cannot use this expression together with `insert-value-expr` or `delete-value-expr` in the same configuration item.
- `delete-value-expr`: Configures an expression that takes effect on values carried by the `DELETE` type binlog events (DELETE_ROWS_EVENT). You cannot use this expression together with `insert-value-expr`, `update-old-value-expr` or `update-new-value-expr`.

Note:

- You can configure `update-old-value-expr` and `update-new-value-expr` together.

- When `update-old-value-expr` and `update-new-value-expr` are configured together, the rows whose "update + old values" meet `update-old-value-expr` **and** whose "update + new values" meet `update-new-value-expr` are filtered.
- When one of `update-old-value-expr` and `update-new-value-expr` is configured, the configured expression determines whether to filter the **entire row change**, which means that the deletion of old values and the insertion of new values are filtered as a whole.

You can use the SQL expression on one column or on multiple columns. You can also use the SQL functions supported by TiDB, such as `c % 2 = 0`, `a*a + b*b = c*c`, and `ts > NOW()`.

The `TIMESTAMP` default time zone is the time zone specified in the task configuration file. The default value is the time zone of the downstream. You can explicitly specify the time zone in a way like `c_timestamp = '2021-01-01 12:34:56.5678+08:00'`.

You can configure multiple filtering rules under the `expression-filter` configuration item. The upstream data source references the required rule in `expression-filters` to make it effective. When multiple rules are used, if **any** one of the rules are matched, the entire row change is filtered.

Note:

Configuring too many expression filtering rules increases the calculation overhead of DM and slows down the data replication.

7 Integrate

7.1 Data Integration Overview

Data integration means the flow, transfer, and consolidation of data among various data sources. As data grows exponentially in volume and data value is more profoundly explored, data integration has become increasingly popular and urgent. To avoid the situation that TiDB becomes data silos and to integrate data with different platforms, TiCDC offers the capability to replicate TiDB incremental data change logs to other data platforms. This document describes the data integration applications using TiCDC. You can choose an integration solution that suits your business scenarios.

7.1.1 Integrate with Confluent Cloud and Snowflake

You can use TiCDC to replicate incremental data from TiDB to Confluent Cloud, and replicate the data to Snowflake, ksqlDB, and SQL Server via Confluent Cloud. For details, see Integrate with Confluent Cloud and Snowflake.

7.1.2 Integrate with Apache Kafka and Apache Flink

You can use TiCDC to replicate incremental data from TiDB to Apache Kafka, and consume the data using Apache Flink. For details, see Integrate with Apache Kafka and Apache Flink.

7.2 Integration Scenarios

7.2.1 Integrate Data with Confluent Cloud and Snowflake

Confluent is an Apache Kafka-compatible streaming data platform that provides strong data integration capabilities. On this platform, you can access, store, and manage non-stop real-time streaming data.

Starting from TiDB v6.1.0, TiCDC supports replicating incremental data to Confluent in Avro format. This document introduces how to replicate TiDB incremental data to Confluent using TiCDC, and further replicate data to Snowflake, ksqlDB, and SQL Server via Confluent Cloud. The organization of this document is as follows:

1. Quickly deploy a TiDB cluster with TiCDC included.
2. Create a changefeed that replicates data from TiDB to Confluent Cloud.
3. Create Connectors that replicate data from Confluent Cloud to Snowflake, ksqlDB, and SQL Server.
4. Write data to TiDB using go-tpc, and observe data changes in Snowflake, ksqlDB, and SQL Server.

The preceding steps are performed in a lab environment. You can also deploy a cluster in a production environment by referring to these steps.

7.2.1.1 Replicate incremental data to Confluent Cloud

7.2.1.1.1 Step 1. Set up the environment

1. Deploy a TiDB cluster with TiCDC included.

   In a lab or testing environment, you can deploy a TiDB cluster with TiCDC included quickly by using TiUP Playground.

```
tiup playground --host 0.0.0.0 --db 1 --pd 1 --kv 1 --tiflash 0 --ticdc
    ↪ 1
# View cluster status
tiup status
```

If TiUP is not installed yet, refer to Install TiUP. In a production environment, you can deploy a TiCDC as instructed in Deploy TiCDC.

2. Register Confluent Cloud and create a Confluent cluster.

   Create a Basic cluster and make it accessible via Internet. For details, see Quick Start for Confluent Cloud.

7.2.1.1.2 Step 2. Create an access key pair

1. Create a cluster API key.

   Sign in to Confluent Cloud. Choose **Data integration** > **API keys** > **Create key**. On the **Select scope for API key** page that is displayed, select **Global access**.

   After creation, a key pair file is generated, as shown below.

```
=== Confluent Cloud API key: xxx-xxxxx ===

API key:
L5WWA4GK4NAT2EQV

API secret:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Bootstrap server:
xxx-xxxxx.ap-east-1.aws.confluent.cloud:9092
```

2. Record the Schema Registry Endpoints.

   In the Confluent Cloud Console, choose **Schema Registry** > **API endpoint**. Record the Schema Registry Endpoints. The following is an example:

```
https://yyy-yyyyy.us-east-2.aws.confluent.cloud
```

3. Create a Schema Registry API key.

   In the Confluent Cloud Console, choose **Schema Registry** > **API credentials**. Click **Edit** and then **Create key**.

   After creation, a key pair file is generated, as shown below:

```
=== Confluent Cloud API key: yyy-yyyyy === API key: 7NBH2CAFM2LMGTH7
↪ API secret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

You can also perform this step by using Confluent CLI. For details, see Connect Confluent CLI to Confluent Cloud Cluster.

7.2.1.1.3 Step 3. Create a Kafka changefeed

1. Create a changefeed configuration file.

   As required by Avro and Confluent Connector, incremental data of each table must be sent to an independent topic, and a partition must be dispatched for each event based on the primary key value. Therefore, you need to create a changefeed configuration file `changefeed.conf` with the following contents:

   ```
   [sink]
   dispatchers = [
   {matcher = ['*.*'], topic = "tidb_{schema}_{table}", partition="index-
       ↪ value"},
   ]
   ```

   For detailed description of `dispatchers` in the configuration file, see Customize the rules for Topic and Partition dispatchers of Kafka Sink.

2. Create a changefeed to replicate incremental data to Confluent Cloud:

   ```
   tiup ctl:<cluster-version> cdc changefeed create --pd="http
       ↪ ://127.0.0.1:2379" --sink-uri="kafka://<broker_endpoint>/ticdc-
       ↪ meta?protocol=avro&replication-factor=3&enable-tls=true&auto-
       ↪ create-topic=true&sasl-mechanism=plain&sasl-user=<broker_api_key
       ↪ >&sasl-password=<broker_api_secret>" --schema-registry="https://<
       ↪ schema_registry_api_key>:<schema_registry_api_secret>@<
       ↪ schema_registry_endpoint>" --changefeed-id="confluent-changefeed"
       ↪  --config changefeed.conf
   ```

   You need to replace the values of the following fields with those created or recorded in Step 2. Create an access key pair:

   - `<broker_endpoint>`
   - `<broker_api_key>`
   - `<broker_api_secret>`
   - `<schema_registry_api_key>`
   - `<schema_registry_api_secret>`
   - `<schema_registry_endpoint>`

   Note that you should encode `<schema_registry_api_secret>` based on HTML URL Encoding Reference before replacing its value. After you replace all the preceding fields, the configuration file is as follows:

```
tiup ctl:<cluster-version> cdc changefeed create --pd="http
    ↪ ://127.0.0.1:2379" --sink-uri="kafka://xxx-xxxxx.ap-east-1.aws.
    ↪ confluent.cloud:9092/ticdc-meta?protocol=avro&replication-factor
    ↪ =3&enable-tls=true&auto-create-topic=true&sasl-mechanism=plain&
    ↪ sasl-user=L5WWA4GK4NAT2EQV&sasl-password=
    ↪ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" --schema-registry="https://7
    ↪ NBH2CAFM2LMGTH7:xxxxxxxxxxxxxxxxx@yyy-yyyyy.us-east-2.aws.
    ↪ confluent.cloud" --changefeed-id="confluent-changefeed" --config
    ↪ changefeed.conf
```

- Run the command to create a changefeed.
  - If the changefeed is successfully created, changefeed information, such as changefeed ID, is displayed, as shown below:

    ```
    Create changefeed successfully!
    ID: confluent-changefeed
    Info: {... changfeed info json struct ...}
    ```

  - If no result is returned after you run the command, check the network connectivity between the server where you run the command and Confluent Cloud. For details, see Test connectivity to Confluent Cloud.

3. After creating the changefeed, run the following command to check the changefeed status:

```
tiup ctl:<cluster-version> cdc changefeed list --pd="http
    ↪ ://127.0.0.1:2379"
```

You can refer to Manage TiCDC Cluster and Replication Tasks to manage the changefeed.

7.2.1.1.4 Step 4. Write data to generate change logs

After the preceding steps are done, TiCDC sends change logs of incremental data in the TiDB cluster to Confluent Cloud. This section describes how to write data into TiDB to generate change logs.

1. Simulate service workload.

   To generate change logs in a lab environment, you can use go-tpc to write data to the TiDB cluster. Specifically, run the following command to create a database `tpcc` in the TiDB cluster. Then, use TiUP bench to write data to this new database.

```
tiup bench tpcc -H 127.0.0.1 -P 4000 -D tpcc --warehouses 4 prepare
tiup bench tpcc -H 127.0.0.1 -P 4000 -D tpcc --warehouses 4 run --time
    ↪ 300s
```

For more details about go-tpc, refer to How to Run TPC-C Test on TiDB.

2. Observe data in Confluent Cloud.

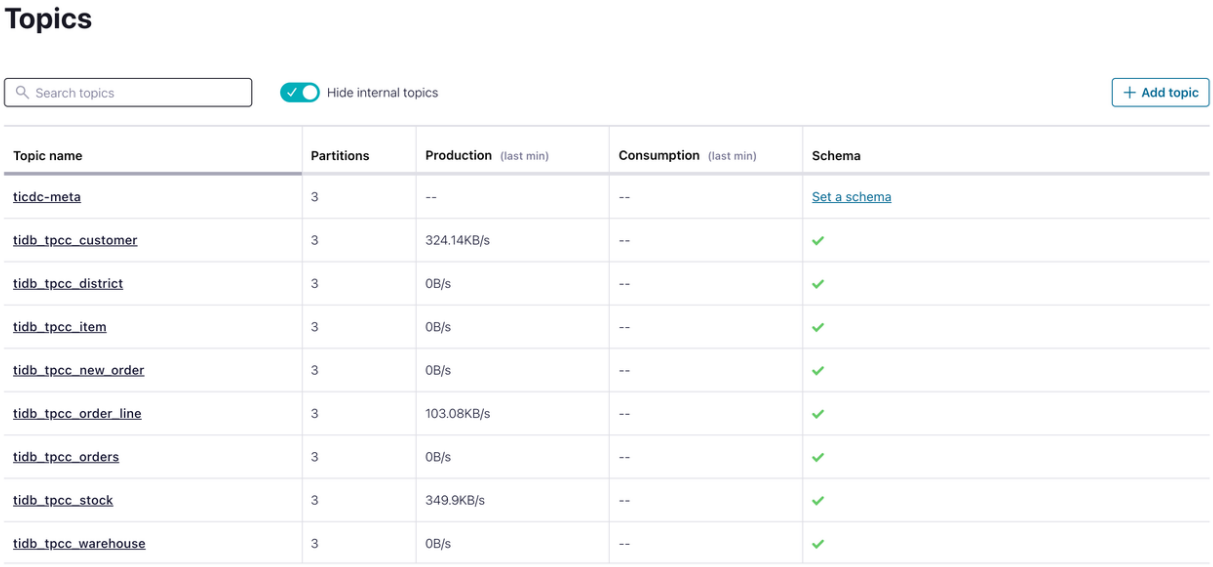


Figure 52: Confluent topics

In the Confluent Cloud Console, click **Topics**. You can see that the target topics have been created and are receiving data. At this time, incremental data of the TiDB database is successfully replicated to Confluent Cloud.

7.2.1.2 Integrate data with Snowflake

Snowflake is a cloud native data warehouse. With Confluent, you can replicate TiDB incremental data to Snowflake by creating Snowflake Sink Connectors.

7.2.1.2.1 Prerequisites

- You have registered and created a Snowflake cluster. See Getting Started with Snowflake.
- Before connecting to the Snowflake cluster, you have generated a private key for it. See Key Pair Authentication & Key Pair Rotation.

7.2.1.2.2 Integration procedure

1. Create a database and a schema in Snowflake.

   In the Snowflake control console, choose **Data** > **Database**. Create a database named `TPCC` and a schema named `TiCDC`.

2. In the Confluent Cloud Console, choose **Data integration** > **Connectors** > **Snowflake Sink**. The page shown below is displayed.

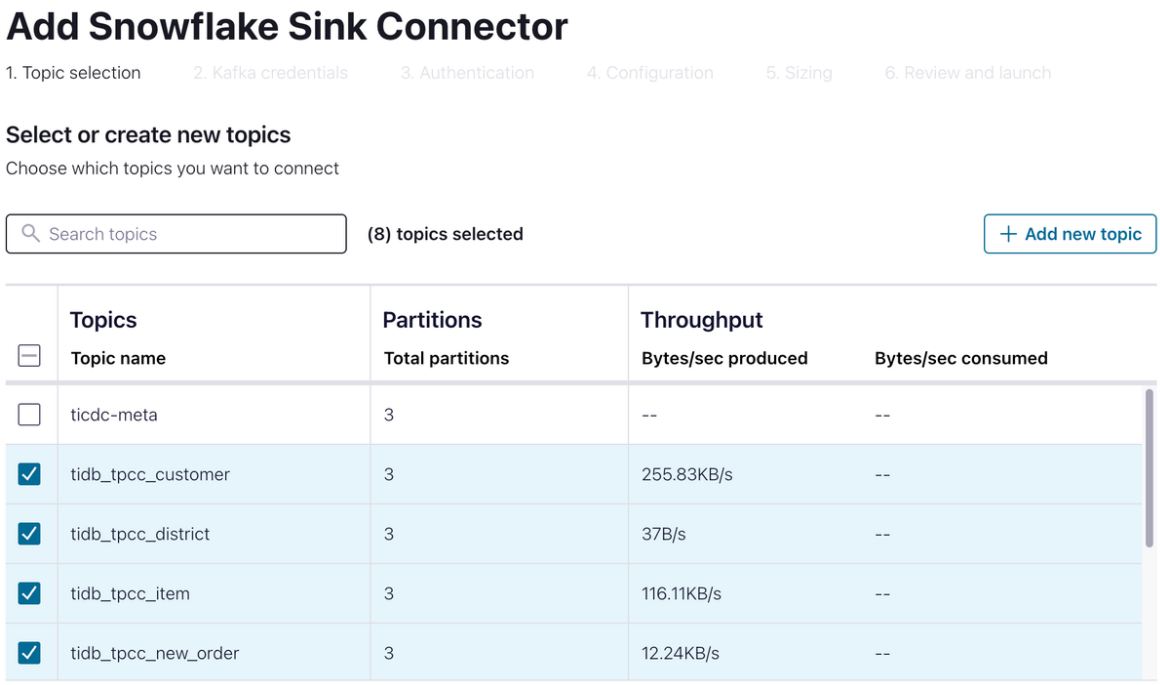


Figure 53: Add snowflake sink connector

3. Select the topic you want to replicate to Snowflake. Then go to the next page.

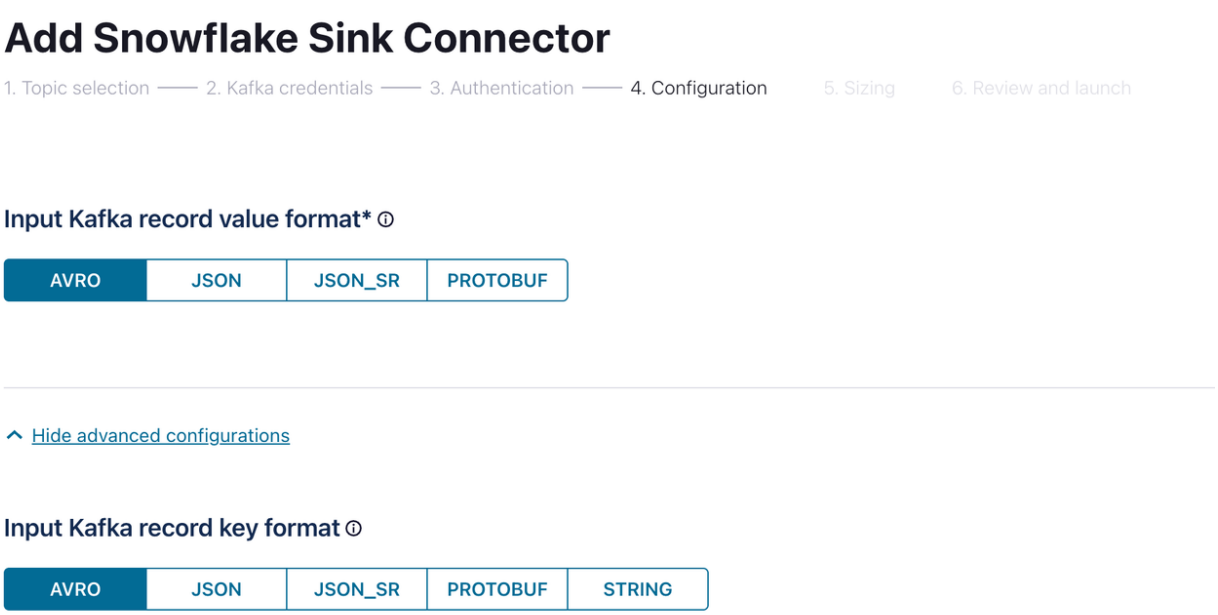


Figure 54: Configuration

4. Specify the authentication information for connecting Snowflake. Fill in **Database name** and **Schema name** with the values you created in the previous step. Then go to the next page.

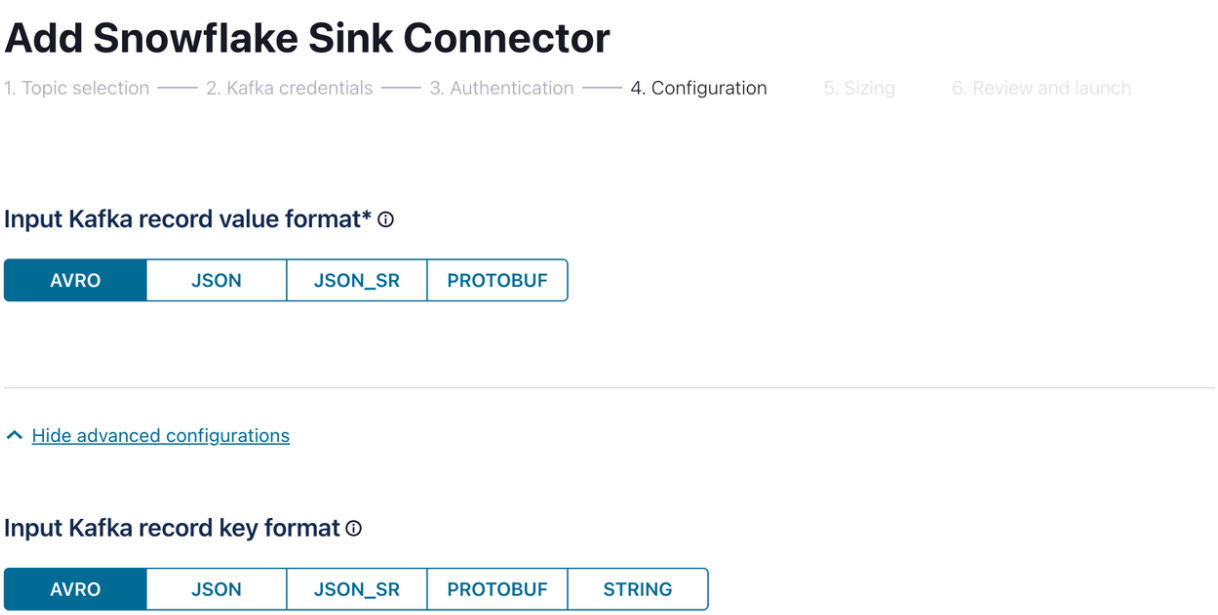


Figure 55: Configuration

5. On the **Configuration** page, select `AVRO` for both **Input Kafka record value format** and **Input Kafka record key format**. Then click **Continue**. Wait until the connector is created and the status becomes **Running**, which might take several minutes.

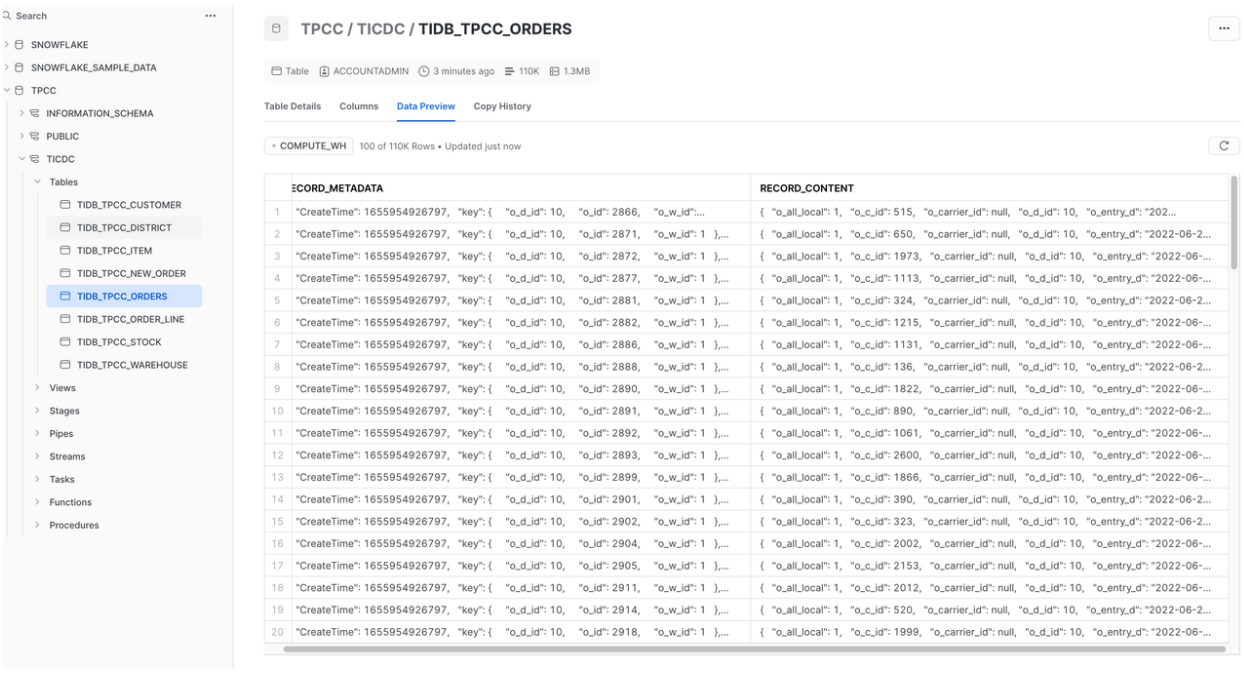


Figure 56: Data preview

6. In the Snowflake console, choose **Data** > **Database** > **TPCC** > **TiCDC**. You can see that TiDB incremental data has been replicated to Snowflake. Data integration with Snowflake is done (see the preceding figure). However, the table structure in Snowflake is different from that in TiDB, and data is inserted into Snowflake incrementally. In most scenarios, you expect the data in Snowflake to be a replica of the data in TiDB, rather than storing TiDB change logs. This problem will be addressed in the next section.

7.2.1.2.3 Create data replicas of TiDB tables in Snowflake

In the previous section, the change logs of TiDB incremental data have been replicated to Snowflake. This section describes how to process these change logs using the TASK and STREAM features of Snowflake according to the event type of `INSERT`, `UPDATE`, and `DELETE` ↪ , and then write them to a table with the same structure as that in upstream, thereby creating a data replica of the TiDB table in Snowflake. The following takes the `ITEM` table as an example.

The structure of the `ITEM` table is as follows:

```
CREATE TABLE `item` (
  `i_id` int(11) NOT NULL,
  `i_im_id` int(11) DEFAULT NULL,
  `i_name` varchar(24) DEFAULT NULL,
  `i_price` decimal(5,2) DEFAULT NULL,
  `i_data` varchar(50) DEFAULT NULL,
```

```
  PRIMARY KEY (`i_id`)
);
```

In Snowflake, there is a table named `TIDB_TEST_ITEM`, which is automatically created by the Confluent Snowflake Sink Connector. The table structure is as follows:

```
create or replace TABLE TIDB_TEST_ITEM (
       RECORD_METADATA VARIANT,
       RECORD_CONTENT VARIANT
);
```

1. In Snowflake, create a table with the same structure as that in TiDB:

    ```
    create or replace table TEST_ITEM (
        i_id INTEGER primary key,
        i_im_id INTEGER,
        i_name VARCHAR,
        i_price DECIMAL(36,2),
        i_data VARCHAR
    );
    ```

2. Create a stream for `TIDB_TEST_ITEM` and set `append_only` to `true` as follows.

    ```
    create or replace stream TEST_ITEM_STREAM on table TIDB_TEST_ITEM
        ↪ append_only=true;
    ```

    In this way, the created stream captures only `INSERT` events in real time. Specifically, when a new change log is generated for `ITEM` in TiDB, the change log will be inserted into `TIDB_TEST_ITEM` and be captured by the stream.

3. Process the data in the stream. According to the event type, insert, update, or delete the stream data in the `TEST_ITEM` table.

    ```
    --Merge data into the TEST_ITEM table
    merge into TEST_ITEM n
      using
          -- Query TEST_ITEM_STREAM
          (SELECT RECORD_METADATA:key as k, RECORD_CONTENT:val as v from
              ↪ TEST_ITEM_STREAM) stm
          -- Match the stream with table on the condition that i_id is equal
          on k:i_id = n.i_id
      -- If the TEST_ITEM table contains a record that matches i_id and v
          ↪ is empty, delete this record
      when matched and IS_NULL_VALUE(v) = true then
          delete
    ```

```
  -- If the TEST_ITEM table contains a record that matches i_id and v
     ↪ is not empty, update this record
  when matched and IS_NULL_VALUE(v) = false then
     update set n.i_data = v:i_data, n.i_im_id = v:i_im_id, n.i_name =
        ↪ v:i_name, n.i_price = v:i_price

  -- If the TEST_ITEM table does not contain a record that matches i_id
     ↪ , insert this record
  when not matched then
     insert
        (i_data, i_id, i_im_id, i_name, i_price)
     values
        (v:i_data, v:i_id, v:i_im_id, v:i_name, v:i_price)
;
```

In the preceding example, the `MERGE INTO` statement of Snowflake is used to match the stream and the table on a specific condition, and then execute corresponding operations, such as deleting, updating, or inserting a record. In this example, three `WHERE` clauses are used for the following three scenarios:

- Delete the record in the table when the stream and the table match and the data in the stream is empty.
- Update the record in the table when the stream and the table match and the data in the stream is not empty.
- Insert the record in the table when the stream and the table do not match.

4. Periodically execute the statement in Step 3 to ensure that data is always up-to-date. You can also use the `SCHEDULED TASK` feature of Snowflake:

```
-- Create a TASK to periodically execute the MERGE INTO statement
create or replace task STREAM_TO_ITEM
    warehouse = test
    -- Execute the TASK every minute
    schedule = '1 minute'
when
    -- Skip the TASK when there is no data in TEST_ITEM_STREAM
    system$stream_has_data('TEST_ITEM_STREAM')
as
-- Merge data into the TEST_ITEM table. The statement is the same as
    ↪ that in the preceding example
merge into TEST_ITEM n
  using
     (select RECORD_METADATA:key as k, RECORD_CONTENT:val as v from
        ↪ TEST_ITEM_STREAM) stm
     on k:i_id = n.i_id
  when matched and IS_NULL_VALUE(v) = true then
```

```
    delete
  when matched and IS_NULL_VALUE(v) = false then
    update set n.i_data = v:i_data, n.i_im_id = v:i_im_id, n.i_name =
      ↪ v:i_name, n.i_price = v:i_price
  when not matched then
    insert
      (i_data, i_id, i_im_id, i_name, i_price)
    values
      (v:i_data, v:i_id, v:i_im_id, v:i_name, v:i_price)
;
```

At this time, you have established a data channel with certain ETL capabilities. Through this data channel, you can replicate TiDB's incremental data change logs to Snowflake, maintain a data replica of TiDB, and use the data in Snowflake.

The last step is to regularly clean up the useless data in the `TIDB_TEST_ITEM` table:

```
-- Clean up the TIDB_TEST_ITEM table every two hours
create or replace task TRUNCATE_TIDB_TEST_ITEM
    warehouse = test
    schedule = '120 minute'
when
    system$stream_has_data('TIDB_TEST_ITEM')
as
    TRUNCATE table TIDB_TEST_ITEM;
```

7.2.1.3 Integrate data with ksqlDB

ksqlDB is a database purpose-built for stream processing applications. You can create ksqlDB clusters on Confluent Cloud and access incremental data replicated by TiCDC.

1. Select **ksqlDB** in the Confluent Cloud Console and create a ksqlDB cluster as instructed.

   Wait until the ksqlDB cluster status is **Running**. This process takes several minutes.

2. In the ksqlDB Editor, run the following command to create a stream to access the `tidb_tpcc_orders` topic:

   ```
   CREATE STREAM orders (o_id INTEGER, o_d_id INTEGER, o_w_id INTEGER,
      ↪ o_c_id INTEGER, o_entry_d STRING, o_carrier_id INTEGER, o_ol_cnt
      ↪  INTEGER, o_all_local INTEGER) WITH (kafka_topic='
      ↪ tidb_tpcc_orders', partitions=3, value_format='AVRO');
   ```

3. Run the following command to check the orders STREAM data:

```
SELECT * FROM ORDERS EMIT CHANGES;
```

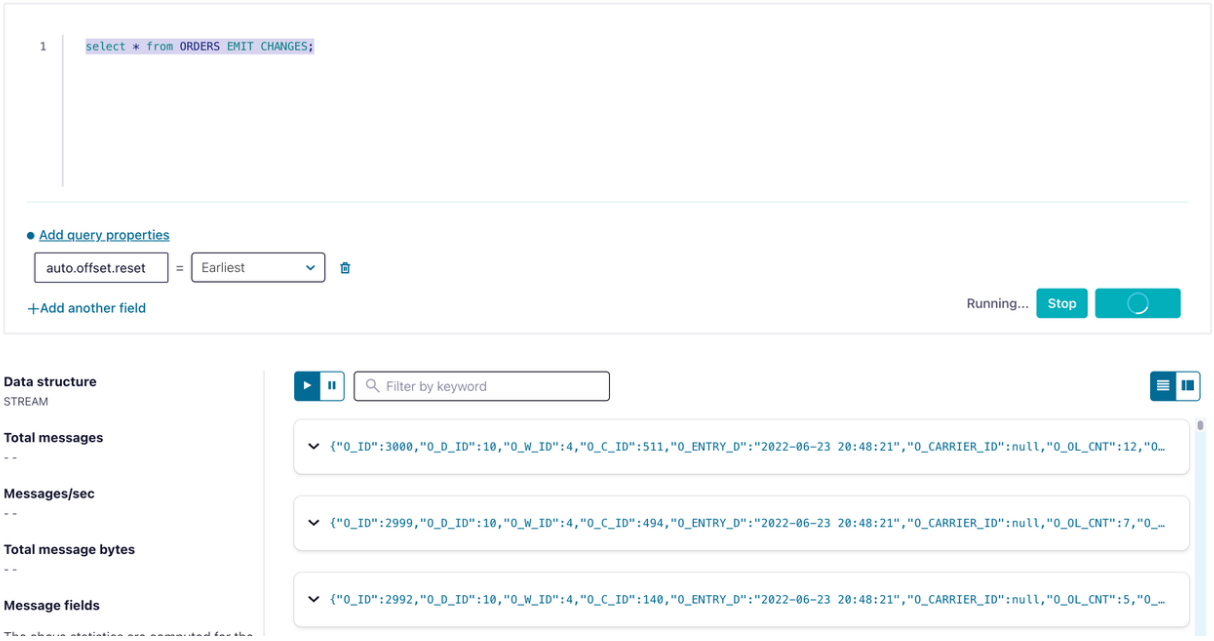


Figure 57: Select from orders

You can see that the incremental data has been replicated to ksqlDB, as shown in the preceding figure. Data integration with ksqlDB is done.

7.2.1.4 Integrate data with SQL Server

Microsoft SQL Server is a relational database management system (RDBMS) developed by Microsoft. With Confluent, you can replicate TiDB incremental data to SQL Server by creating SQL Server Sink Connectors.

1. Connect to SQL Server and create a database named `tpcc`.

```
[ec2-user@ip-172-1-1-1 bin]$ sqlcmd -S 10.61.43.14,1433 -U admin
Password:
1> create database tpcc
2> go
1> select name from master.dbo.sysdatabases
2> go
name
----------------------------------------------------------------------
master
tempdb
```

```
model
msdb
rdsadmin
tpcc
(6 rows affected)
```

2. In the Confluent Cloud Console, choose **Data integration** > **Connectors** > **Microsoft SQL Server Sink**. The page shown below is displayed.

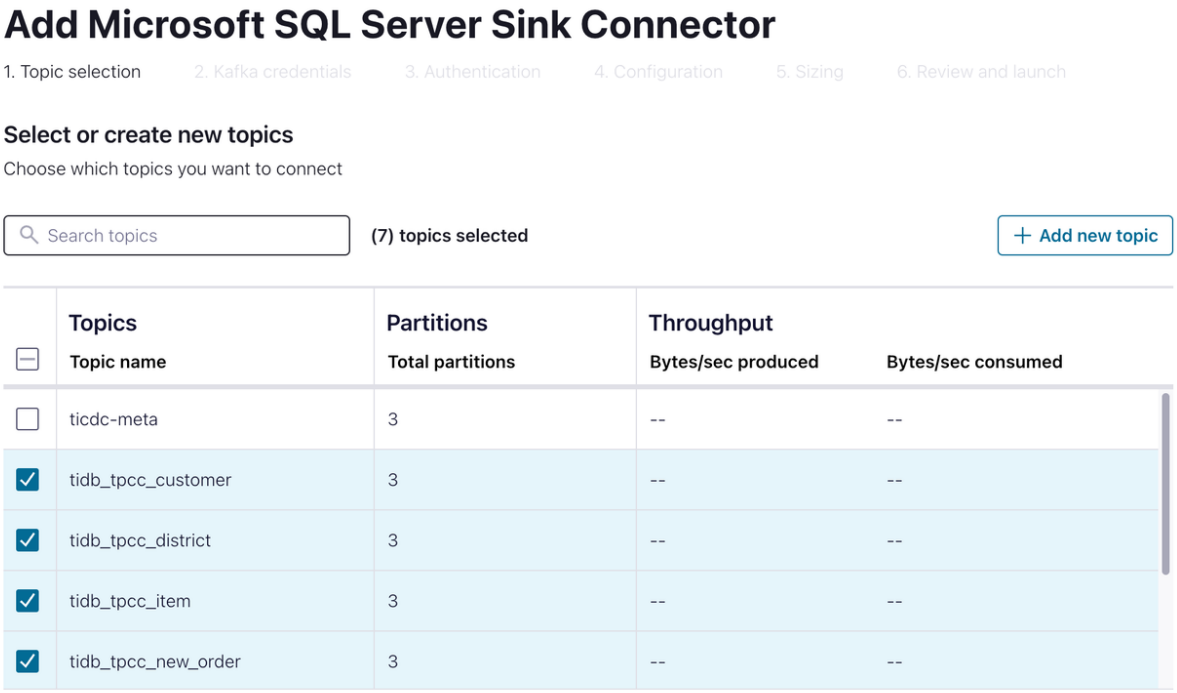


Figure 58: Topic selection

3. Select the topic you want to replicate to SQL Server. Then go to the next page.

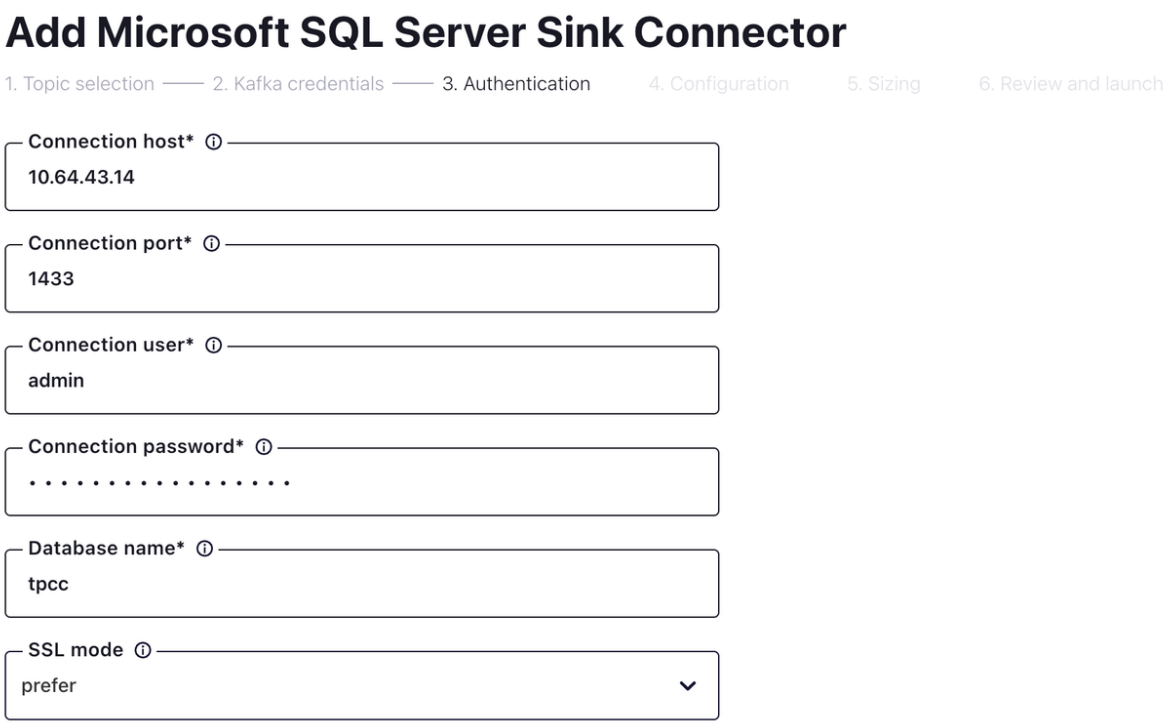


Figure 59: Authentication

4. Fill in the connection and authentication information. Then go to the next page.

5. On the **Configuration** page, configure the following fields and click **Continue**.

Field	Value
Input Kafka record value format	AVRO
Insert mode	UPSERT
Auto create table	true
Auto add columns	true
PK mode	record_key
Input Kafka record key format	AVRO
Delete on null	true

6. After configuration, click **Continue**. Wait until the connector status becomes **Running**, which might take several minutes.

Run Cancel | Disconnect Change Connection | tpcc

```
SELECT TOP (1000) [o_id]
      ,[o_d_id]
      ,[o_w_id]
      ,[o_c_id]
      ,[o_entry_d]
      ,[o_carrier_id]
      ,[o_ol_cnt]
      ,[o_all_local]
  FROM [tpcc].[dbo].[tidb_tpcc_orders]
```

Results Messages

	o_id	o_d_id	o_w_id	o_c_id	o_entry_d	o_carrier_id	o_ol_cnt	o_all_local
1	1	1	1	1860	2022-06-22 22:08:06	3	6	1
2	1	3	1	884	2022-06-22 22:08:06	10	5	1
3	2	1	1	1247	2022-06-22 22:08:06	7	10	1
4	2	3	1	752	2022-06-22 22:08:06	2	12	1
5	3	1	1	902	2022-06-22 22:08:06	5	11	1
6	3	2	1	1250	2022-06-22 22:08:06	8	7	1
7	4	1	1	1468	2022-06-22 22:08:06	7	6	1
8	4	2	1	2246	2022-06-22 22:08:06	3	9	1
9	4	3	1	358	2022-06-22 22:08:06	4	13	1
10	5	1	1	639	2022-06-22 22:08:06	9	10	1

Figure 60: Results

7. Connect SQL Server and observe the data. You can see that the incremental data has been replicated to SQL Server, as shown in the preceding figure. Data integration with SQL Server is done.

7.2.2 Integrate Data with Apache Kafka and Apache Flink

This document describes how to replicate TiDB data to Apache Kafka and Apache Flink using TiCDC. The organization of this document is as follows:

1. Quickly deploy a TiDB cluster with TiCDC included, and create a Kafka cluster and a Flink cluster.
2. Create a changefeed that replicates data from TiDB to Kafka.
3. Write data to TiDB using go-tpc.
4. Observe data on Kafka console consumer and check that the data is replicated to a specified Kafka topic.
5. (Optional) Configure the Flink cluster to consume Kafka data.

The preceding steps are performed in a lab environment. You can also deploy a cluster in a production environment by referring to these steps.

7.2.2.1 Step 1. Set up the environment

1. Deploy a TiDB cluster with TiCDC included.

   In a lab or testing environment, you can deploy a TiDB cluster with TiCDC included quickly by using TiUP Playground.

   ```
   tiup playground --host 0.0.0.0 --db 1 --pd 1 --kv 1 --tiflash 0 --ticdc
       ↪ 1
   # View cluster status
   tiup status
   ```

   If TiUP is not installed yet, refer to Install TiUP. In a production environment, you can deploy a TiCDC as instructed in Deploy TiCDC.

2. Create a Kafka cluster.

   - Lab environment: refer to Apache Kakfa Quickstart to start a Kafka cluster.
   - Production environment: refer to Running Kafka in Production to deploy a Kafka production cluster.

3. (Optional) Create a Flink cluster.

   - Lab environment: refer to Apache Flink First steps to start a Flink cluster.
   - Production environment: refer to Apache Kafka Deployment to deploy a Flink production cluster.

7.2.2.2 Step 2. Create a Kafka changefeed

1. Create a changefeed configuration file.

   As required by Flink, incremental data of each table must be sent to an independent topic, and a partition must be dispatched for each event based on the primary key value. Therefore, you need to create a changefeed configuration file `changefeed.conf` with the following contents:

   ```
   [sink]
   dispatchers = [
   {matcher = ['*.*'], topic = "tidb_{schema}_{table}", partition="index-
       ↪ value"},
   ]
   ```

   For detailed description of `dispatchers` in the configuration file, see Customize the rules for Topic and Partition dispatchers of Kafka Sink.

2. Create a changefeed to replicate incremental data to Kafka:

```
tiup ctl:<cluster-version> cdc changefeed create --pd="http
    ↪ ://127.0.0.1:2379" --sink-uri="kafka://127.0.0.1:9092/kafka-topic
    ↪ -name?protocol=canal-json" --changefeed-id="kafka-changefeed" --
    ↪ config="changefeed.conf"
```

- If the changefeed is successfully created, changefeed information, such as changefeed ID, is displayed, as shown below:

```
Create changefeed successfully!
ID: kafka-changefeed
Info: {... changfeed info json struct ...}
```

- If no result is returned after you run the command, check the network connectivity between the server where you run the command and the Kafka machine specified in the sink URI.

In a production environment, a Kafka cluster has multiple broker nodes. Therefore, you can add the addresses of multiple brokers to the sink UIR. This ensures stable access to the Kafka cluster. When the Kafka cluster is down, the changefeed still works. Suppose that a Kafka cluster has three broker nodes, with IP addresses being 127.0.0.1:9092, 127.0.0.2:9092, and 127.0.0.3:9092, respectively. You can create a changefeed with the following sink URI.

```
tiup ctl:<cluster-version> cdc changefeed create --pd="http
    ↪ ://127.0.0.1:2379" --sink-uri="kafka
    ↪ ://127.0.0.1:9092,127.0.0.2:9092,127.0.0.3:9092/kafka-topic-name?
    ↪ protocol=canal-json&partition-num=3&replication-factor=1&max-
    ↪ message-bytes=1048576" --config="changefeed.conf"
```

3. After creating the changefeed, run the following command to check the changefeed status:

```
tiup ctl:<cluster-version> cdc changefeed list --pd="http
    ↪ ://127.0.0.1:2379"
```

You can refer to Manage TiCDC Cluster and Replication Tasks to manage the changefeed.

7.2.2.3 Step 3. Write data to generate change logs

After the preceding steps are done, TiCDC sends change logs of incremental data in the TiDB cluster to Kafka. This section describes how to write data into TiDB to generate change logs.

1. Simulate service workload.

   To generate change logs in a lab environment, you can use go-tpc to write data to the TiDB cluster. Specifically, run the following command to use TiUP bench to create a `tpcc` database and write data to this new database.

   ```
   tiup bench tpcc -H 127.0.0.1 -P 4000 -D tpcc --warehouses 4 prepare
   tiup bench tpcc -H 127.0.0.1 -P 4000 -D tpcc --warehouses 4 run --time
       ↪ 300s
   ```

   For more details about go-tpc, refer to How to Run TPC-C Test on TiDB.

2. Consume data in the Kafka topic.

   When a changefeed works normally, it writes data to the Kafka topic. Run `kafka-console-consumer.sh`. You can see that data is successfully written to the Kafka topic.

   ```
   ./bin/kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --
       ↪ from-beginning --topic `${topic-name}`
   ```

At this time, incremental data of the TiDB database is successfully replicated to Kafka. Next, you can use Flink to consume Kafka data. Alternatively, you can develop a Kafka consumer client yourself for specific service scenarios.

7.2.2.4 (Optional) Step 4. Configure Flink to consume Kafka data

1. Install a Flink Kafka connector.

   In the Flink ecosystem, a Flink Kafka connector is used to consume Kafka data and output data to Flink. However, Flink Kafka connectors are not automatically installed. To use it, add a Flink Kafka connector and its dependencies to the Flink installation directory after installing Flink. Specifically, download the following jar files to the `lib` directory of the Flink installation directory. If you have already run the Flink cluster, restart it to load the new plugin.

   - flink-connector-kafka-1.15.0.jar
   - flink-sql-connector-kafka-1.15.0.jar
   - kafka-clients-3.2.0.jar

2. Create a table.

   In the directory where Flink is installed, run the following command to start the Flink SQL client:

   ```
   [root@flink flink-1.15.0]# ./bin/sql-client.sh
   ```

   Then, run the following command to create a table named `tpcc_orders`.

```
CREATE TABLE tpcc_orders (
    o_id INTEGER,
    o_d_id INTEGER,
    o_w_id INTEGER,
    o_c_id INTEGER,
    o_entry_d STRING,
    o_carrier_id INTEGER,
    o_ol_cnt INTEGER,
    o_all_local INTEGER
) WITH (
'connector' = 'kafka',
'topic' = 'tidb_tpcc_orders',
'properties.bootstrap.servers' = '127.0.0.1:9092',
'properties.group.id' = 'testGroup',
'format' = 'canal-json',
'scan.startup.mode' = 'earliest-offset',
'properties.auto.offset.reset' = 'earliest'
)
```

Replace `topic` and `properties.bootstrap.servers` with the actual values in the environment.

3. Query data of the table.

   Run the following command to query data of the `tpcc_orders` table:

```
SELECT * FROM tpcc_orders;
```

   After this command is executed, you can see that there is new data in the table, as shown in the following figure.

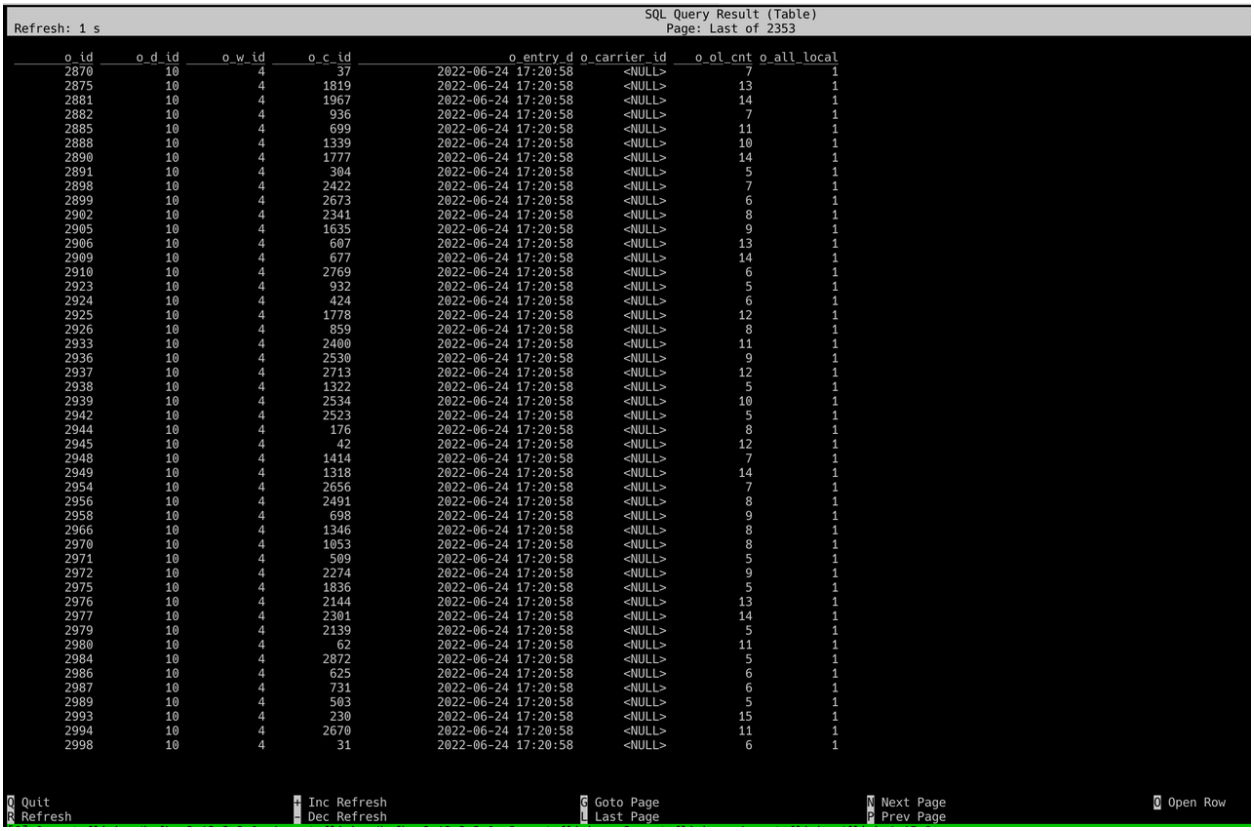


Figure 61: SQL query result

Data integration with Kafka is done.

8 Maintain

8.1 Security

8.1.1 Best Practices for TiDB Security Configuration

The security of TiDB is crucial for protecting data integrity and confidentiality. This document provides guidelines for configuring TiDB clusters securely during deployment. By following these best practices, you can effectively reduce potential security risks, prevent data breaches, and ensure the continuous, stable, and reliable operation of your TiDB database system.

Note:

This document offers general recommendations on TiDB security configurations. PingCAP does not guarantee the completeness or accuracy of the information, and it assumes no responsibility for any issues arising from the use of this guide. Users should assess these recommendations based on their specific needs and consult professionals for tailored advice.

8.1.1.1 Set the initial password for the root user

By default, the root user in a newly created TiDB cluster has no password, which poses a potential security risk. If a password is not set, anyone can attempt to log in to the TiDB database as the root user, potentially gaining access to and modifying data.

To avoid this risk, it is recommended to set a root password during deployment:

- For deployments using TiUP, refer to Deploy TiDB Cluster Using TiUP to generate a random password for the root user.
- For deployments using TiDB Operator, refer to Set initial account and password to set the root password.

8.1.1.2 Change the default Grafana password

TiDB installation includes the Grafana component by default, and the default username and password are typically `admin/admin`. If the password is not changed promptly, attackers could exploit this to gain control of the system.

It is recommended to immediately change the Grafana password to a strong one during the TiDB deployment, and regularly update the password to ensure system security. Here are the steps to change the Grafana password:

- Upon first login to Grafana, follow the prompts to change the password.

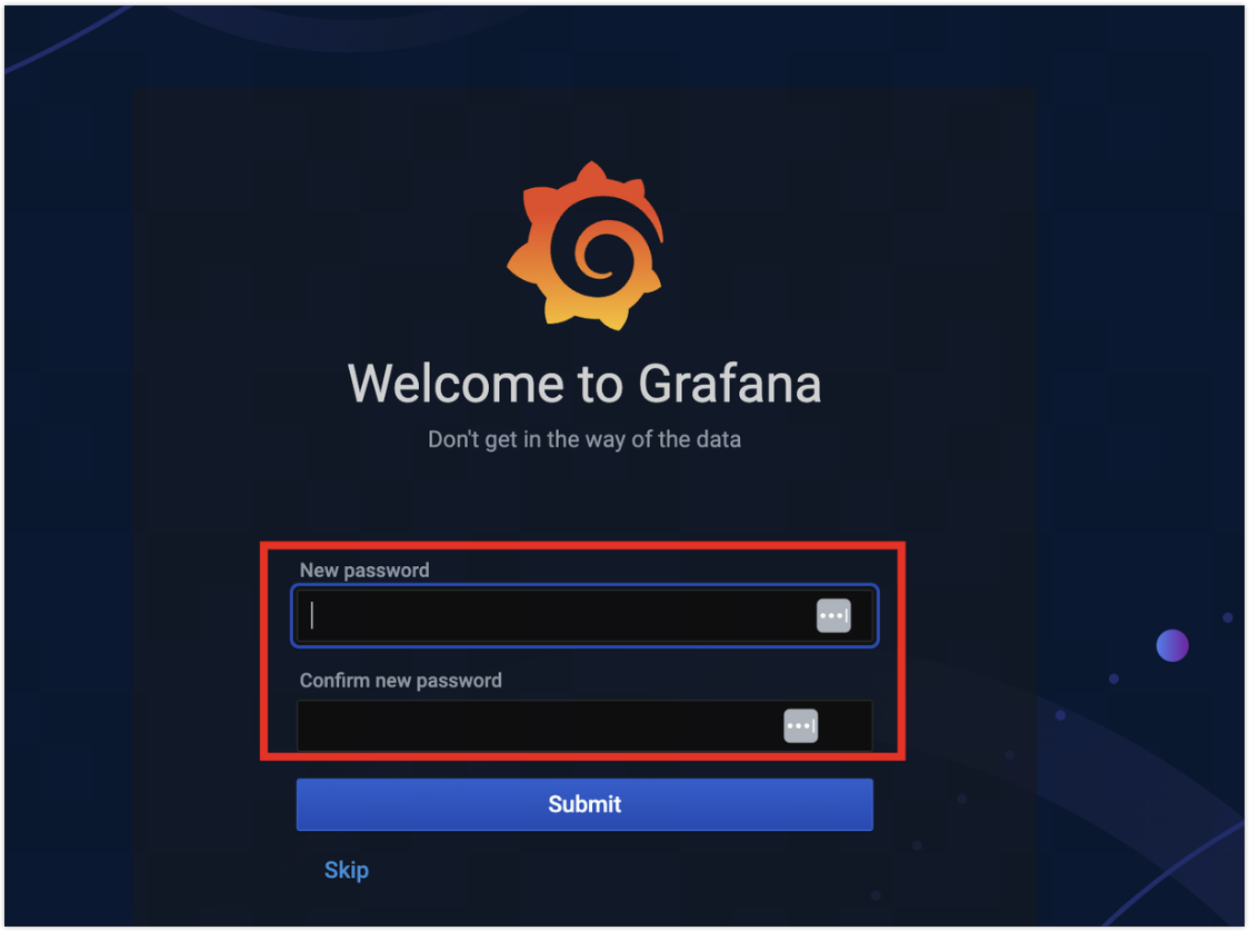


Figure 62: Grafana Password Reset Guide

- Access the Grafana personal configuration center to change the password.

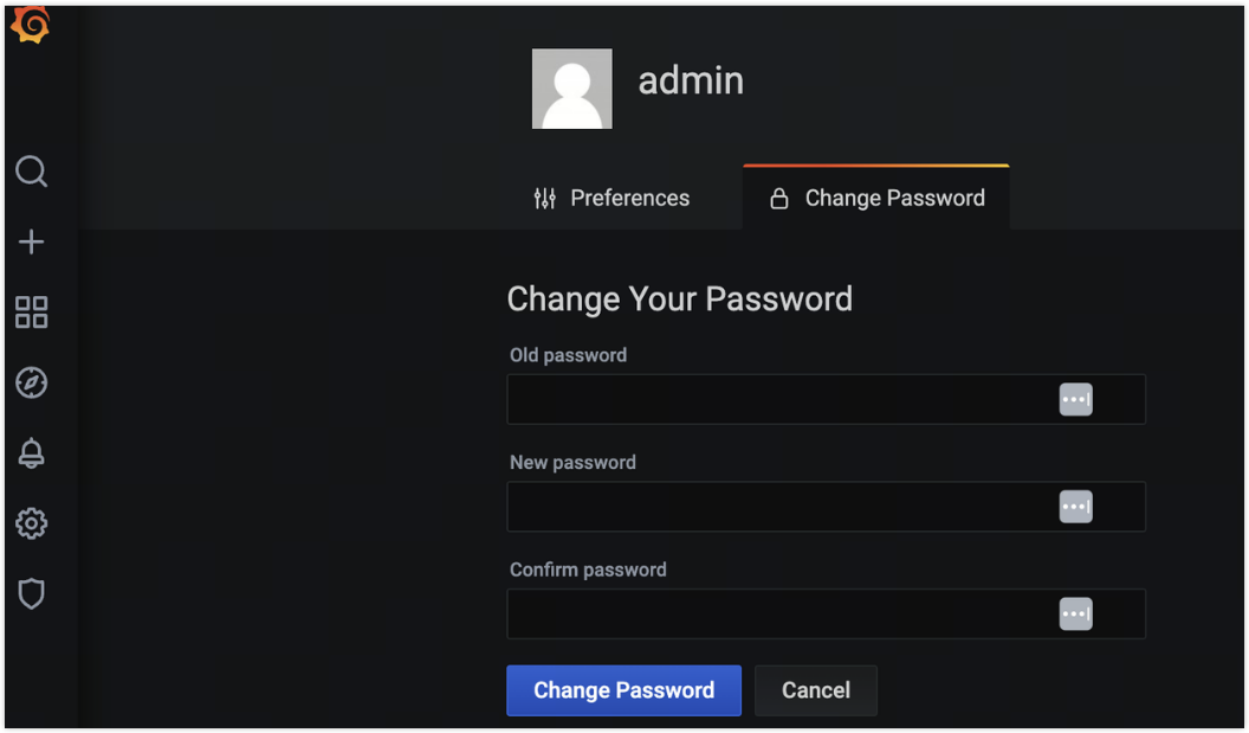


Figure 63: Grafana Password Reset Guide

8.1.1.3 Enhance TiDB Dashboard security

8.1.1.3.1 Use a least privilege user

TiDB Dashboard shares the account system with TiDB SQL users, and TiDB Dashboard authorization is based on TiDB SQL user permissions. TiDB Dashboard requires minimal permissions and can even operate with read-only access.

To enhance security, it is recommended to create a least-privilege SQL user for accessing the TiDB Dashboard and to avoid using high-privilege users.

8.1.1.3.2 Restrict access control

By default, TiDB Dashboard is designed for trusted users. The default port includes additional API interfaces besides TiDB Dashboard. If you want to allow access to TiDB Dashboard from external networks or untrusted users, take the following measures to avoid security vulnerabilities:

- Use a firewall or other mechanisms to restrict the default 2379 port to trusted domains, preventing access by external users.

Note:

TiDB, TiKV, and other components need to communicate with the PD component via the PD client port. Do not block internal network access between components, which will make the cluster unavailable.

- Configure a reverse proxy to securely provide TiDB Dashboard services to external users on a different port.

8.1.1.4 Protect internal ports

By default, TiDB installation includes several privileged interfaces for inter-component communication. These ports typically do not need to be accessible to users, because they are primarily for internal communication. Exposing these ports on public networks increases the attack surface, violates the principle of least privilege, and raises the risk of security vulnerabilities. The following table lists the default listening ports in a TiDB cluster:

Component	Default port	Protocol
TiDB	4000	MySQL
TiDB	10080	HTTP
TiKV	20160	Protocol
TiKV	20180	HTTP
PD	2379	HTTP/Protocol
PD	2380	Protocol
TiFlash	3930	Protocol
TiFlash	20170	Protocol
TiFlash	20292	HTTP
TiFlash	8234	HTTP
TiFlow	8261	HTTP
TiFlow	8291	HTTP
TiFlow	8262	HTTP
TiFlow	8300	HTTP
TiDB Lightning	8289	HTTP
TiDB Operator	6060	HTTP
TiDB Dashboard	2379	HTTP
TiDB Binlog	8250	HTTP
TiDB Binlog	8249	HTTP
TMS	8082	HTTP
TEM	8080	HTTP
TEM	8000	HTTP
TEM	4110	HTTP
TEM	4111	HTTP
TEM	4112	HTTP
TEM	4113	HTTP

Component	Default port	Protocol
TEM	4124	HTTP
Prometheus	9090	HTTP
Grafana	3000	HTTP
AlertManager	9093	HTTP
AlertManager	9094	Protocol
Node Exporter	9100	HTTP
Blackbox Exporter	9115	HTTP
NG Monitoring	12020	HTTP

It is recommended to only expose the `4000` port for the database and the `9000` port for the Grafana dashboard to ordinary users, while restricting access to other ports using network security policies or firewalls. The following is an example of using `iptables` to restrict port access:

```
### Allow internal port communication from the whitelist of component IP
    ↪ addresses
sudo iptables -A INPUT -s internal IP address range -j ACCEPT

### Only open ports 4000 and 9000 to external users
sudo iptables -A INPUT -p tcp --dport 4000 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 9000 -j ACCEPT

### Deny all other traffic by default
sudo iptables -P INPUT DROP
```

If you need to access TiDB Dashboard, it is recommended to configure a reverse proxy to securely provide services to external networks on a separate port.

8.1.1.5 Resolving false positives from third-party MySQL vulnerability scanners

Most vulnerability scanners detect MySQL vulnerabilities based on version information. Because TiDB is MySQL protocol-compatible but not MySQL itself, version-based vulnerability scans might lead to false positives. It is recommended to focus vulnerability scans on principle-based assessments. If compliance scanning tools require a specific MySQL version, you can modify the server version number to meet the requirement.

By changing the server version number, you can avoid false positives from vulnerability scanners. The `server-version` value is used by TiDB nodes to verify the current TiDB version. Before upgrading the TiDB cluster, ensure that the `server-version` value is either empty or the actual version of TiDB to avoid unexpected behavior.

8.1.2 Enable TLS between TiDB Clients and Servers

Non-encrypted connection between TiDB's server and clients is allowed by default, which enables third parties that monitor channel traffic to know the data sent and received between the server and the client, including query content and query results. If a channel is untrustworthy (such as if the client is connected to the TiDB server via a public network), then a non-encrypted connection is prone to information leakage. In this case, for security reasons, it is recommended to require an encrypted connection.

The TiDB server supports the encrypted connection based on the TLS (Transport Layer Security). The protocol is consistent with MySQL encrypted connections and is directly supported by existing MySQL clients such as MySQL Client, MySQL Shell and MySQL drivers. TLS is sometimes referred to as SSL (Secure Sockets Layer). Because the SSL protocol has known security vulnerabilities, TiDB does not support SSL. TiDB supports the following protocols: TLSv1.0, TLSv1.1, TLSv1.2 and TLSv1.3.

When an encrypted connection is used, the connection has the following security properties:

- Confidentiality: the traffic plaintext is encrypted to avoid eavesdropping
- Integrity: the traffic plaintext cannot be tampered
- Authentication: (optional) the client can verify the identity of the server and the server can verify the identity of the client to avoid man-in-the-middle attacks

To use connections secured with TLS, you first need to configure the TiDB server to enable TLS. Then you need to configure the client application to use TLS. Most client libraries enable TLS automatically when the server has TLS support configured correctly.

Similar to MySQL, TiDB allows TLS and non-TLS connections on the same TCP port. For a TiDB server with TLS enabled, you can choose to securely connect to the TiDB server through an encrypted connection, or to use an unencrypted connection. You can use the following ways to require the use of secure connections:

- Configure the system variable `require_secure_transport` to require secure connections to the TiDB server for all users.
- Specify `REQUIRE SSL` when you create a user (`create user`), or modify an existing user (`alter user`), which is to specify that specified users must use the encrypted connection to access TiDB. The following is an example of creating a user:

```
CREATE USER 'u1'@'%' IDENTIFIED BY 'my_random_password' REQUIRE SSL;
```

> **Note:**
>
> If the login user has configured using the TiDB Certificate-Based Authentication for Login, the user is implicitly required to enable the encrypted connection to TiDB.

8.1.2.1 Configure TiDB server to use secure connections

See the following descriptions about the related parameters to enable secure connections:

- `auto-tls`: enables automatic certificate generation (since v5.2.0)
- `ssl-cert`: specifies the file path of the SSL certificate
- `ssl-key`: specifies the private key that matches the certificate
- `ssl-ca`: (optional) specifies the file path of the trusted CA certificate
- `tls-version`: (optional) specifies the minimum TLS version, e.g. "TLSv1.2"

`auto-tls` allows secure connections but does not provide client certificate validation. For certificate validation, and to control how certificates are generated, see the advice on configuring the `ssl-cert`, `ssl-key` and `ssl-ca` variables below.

To enable secure connections with your own certificates in the TiDB server, you must specify both of the `ssl-cert` and `ssl-key` parameters in the configuration file when you start the TiDB server. You can also specify the `ssl-ca` parameter for client authentication (see Enable authentication).

All the files specified by the parameters are in PEM (Privacy Enhanced Mail) format. Currently, TiDB does not support the import of a password-protected private key, so it is required to provide a private key file without a password. If the certificate or private key is invalid, the TiDB server starts as usual, but the client cannot connect to the TiDB server through an encrypted connection.

If the certificate parameters are correct, TiDB outputs `secure connection is enabled` when started; otherwise, it outputs `secure connection is NOT ENABLED`.

For TiDB versions earlier than v5.2.0, you can use `mysql_ssl_rsa_setup --datadir =./certs` to generate certficates. The `mysql_ssl_rsa_setup` tool is a part of MySQL Server.

8.1.2.2 Configure the MySQL client to use encrypted connections

The client of MySQL 5.7 or later versions attempts to establish an encrypted connection by default. If the server does not support encrypted connections, it automatically returns to unencrypted connections. The client of MySQL earlier than version 5.7 uses the unencrypted connection by default.

You can change the connection behavior of the client using the following `--ssl-mode` parameters:

- `--ssl-mode=REQUIRED`: The client requires an encrypted connection. The connection cannot be established if the server side does not support encrypted connections.
- In the absence of the `--ssl-mode` parameter: The client attempts to use an encrypted connection, but the encrypted connection cannot be established if the server side does not support encrypted connections. Then the client uses an unencrypted connection.
- `--ssl-mode=DISABLED`: The client uses an unencrypted connection.

MySQL 8.0 clients have two SSL modes in addition to this parameter:

- `--ssl-mode=VERIFY_CA`: Validates the certificate from the server against the CA that requires `--ssl-ca`.
- `--ssl-mode=VERIFY_IDENTITY`: The same as `VERIFY_CA`, but also validating whether the hostname you are connecting to matches the certificate.

For more information, see Client-Side Configuration for Encrypted Connections in MySQL.

8.1.2.3 Enable authentication

If the `ssl-ca` parameter is not specified in the TiDB server or MySQL client, the client or the server does not perform authentication by default and cannot prevent man-in-the-middle attack. For example, the client might "securely" connect to a disguised client. You can configure the `ssl-ca` parameter for authentication in the server and client. Generally, you only need to authenticate the server, but you can also authenticate the client to further enhance the security.

- To authenticate the TiDB server from the MySQL client:

1. Specify the `ssl-cert` and `ssl-key` parameters in the TiDB server.
2. Specify the `--ssl-ca` parameter in the MySQL client.
3. Specify the `--ssl-mode` to `VERIFY_CA` at least in the MySQL client.
4. Make sure that the certificate (`ssl-cert`) configured in the TiDB server is signed by the CA specified by the client `--ssl-ca` parameter; otherwise, the authentication fails.

- To authenticate the MySQL client from the TiDB server:

1. Specify the `ssl-cert`, `ssl-key`, and `ssl-ca` parameters in the TiDB server.
2. Specify the `--ssl-cert` and `--ssl-key` parameters in the client.
3. Make sure the server-configured certificate and the client-configured certificate are both signed by the `ssl-ca` specified by the server.

- To perform mutual authentication, meet both of the above requirements.

By default, the server-to-client authentication is optional. Even if the client does not present its certificate of identification during the TLS handshake, the TLS connection can be still established. You can also require the client to be authenticated by specifying `require x509` when creating a user (`create user`), granting permissions (`grant`), or modifying an existing user (`alter user`). The following is an example of creating a user:

```
create user 'u1'@'%' require x509;
```

> **Note:**
>
> If the login user has configured using the TiDB Certificate-Based Authentication for Login, the user is implicitly required to enable the encrypted connection to TiDB.

8.1.2.4 Check whether the current connection uses encryption

Use the `SHOW STATUS LIKE "%Ssl%";` statement to get the details of the current connection, including whether encryption is used, the encryption protocol used by encrypted connections and the TLS version number.

See the following example of the result in an encrypted connection. The results change according to different TLS versions or encryption protocols supported by the client.

```
mysql> SHOW STATUS LIKE "%Ssl%";
......
| Ssl_verify_mode | 5                          |
| Ssl_version     | TLSv1.2                    |
| Ssl_cipher      | ECDHE-RSA-AES128-GCM-SHA256 |
......
```

For the official MySQL client, you can also use the `STATUS` or `\s` statement to view the connection status:

```
mysql> \s
...
SSL: Cipher in use is ECDHE-RSA-AES128-GCM-SHA256
...
```

8.1.2.5 Supported TLS versions, key exchange protocols, and encryption algorithms

The TLS versions, key exchange protocols and encryption algorithms supported by TiDB are determined by the official Golang libraries.

The crypto policy for your operating system and the client library you are using might also impact the list of supported protocols and cipher suites.

8.1.2.5.1 Supported TLS versions

- TLSv1.0 (disabled by default)
- TLSv1.1

- TLSv1.2
- TLSv1.3

The `tls-version` configuration option can be used to limit the TLS versions that can be used.

The actual TLS versions that can be used depend on the OS crypto policy, MySQL client version and the SSL/TLS library that is used by the client.

8.1.2.5.2 Supported key exchange protocols and encryption algorithms

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

8.1.2.6 Reload certificate, key, and CA

To replace the certificate, the key or CA, first replace the corresponding files, then execute the `ALTER INSTANCE RELOAD TLS` statement on the running TiDB instance to reload the certificate (`ssl-cert`), the key (`ssl-key`), and the CA (`ssl-ca`) from the original configuration path. In this way, you do not need to restart the TiDB instance.

The newly loaded certificate, key, and CA take effect on the connection that is established after the statement is successfully executed. The connection established before the statement execution is not affected.

8.1.2.7 Monitoring

Since TiDB v5.2.0, you can use the `Ssl_server_not_after` and `Ssl_server_not_before` ↪ status variables to monitor the start and end dates of the validity of the certificate.

```
SHOW GLOBAL STATUS LIKE 'Ssl\_server\_not\_%';
```

```
+----------------------+--------------------------+
| Variable_name        | Value                    |
+----------------------+--------------------------+
| Ssl_server_not_after | Nov 28 06:42:32 2021 UTC |
| Ssl_server_not_before | Aug 30 06:42:32 2021 UTC |
+----------------------+--------------------------+
2 rows in set (0.0076 sec)
```

8.1.2.8 See also

- Enable TLS Between TiDB Components.

8.1.3 Enable TLS Between TiDB Components

This document describes how to enable encrypted data transmission between components within a TiDB cluster. Once enabled, encrypted transmission is used between the following components:

- TiDB and TiKV; TiDB and PD
- TiKV and PD
- TiDB Control and TiDB; TiKV Control and TiKV; PD Control and PD
- Internal communication within each TiKV, PD, TiDB cluster

Currently, it is not supported to only enable encrypted transmission of some specific components.

8.1.3.1 Configure and enable encrypted data transmission

1. Prepare certificates.

   It is recommended to prepare a server certificate for TiDB, TiKV, and PD separately. Make sure that these components can authenticate each other. The Control tools of TiDB, TiKV, and PD can choose to share one client certificate.

   You can use tools like `openssl`, `easy-rsa` and `cfssl` to generate self-signed certificates.

   If you choose `openssl`, you can refer to generating self-signed certificates.

2. Configure certificates.

   To enable mutual authentication among TiDB components, configure the certificates of TiDB, TiKV, and PD as follows.

- TiDB

  Configure in the configuration file or command-line arguments:

```
[security]
# Path of the file that contains list of trusted SSL CAs for
   ↪ connection with cluster components.
cluster-ssl-ca = "/path/to/ca.pem"
# Path of the file that contains X509 certificate in PEM format for
   ↪  connection with cluster components.
cluster-ssl-cert = "/path/to/tidb-server.pem"
# Path of the file that contains X509 key in PEM format for
   ↪ connection with cluster components.
cluster-ssl-key = "/path/to/tidb-server-key.pem"
```

- TiKV

  Configure in the configuration file or command-line arguments, and set the corresponding URL to `https`:

```
[security]
## The path for certificates. An empty string means that secure
   ↪ connections are disabled.
# Path of the file that contains a list of trusted SSL CAs. If it
   ↪ is set, the following settings `cert_path` and `key_path`
   ↪ are also needed.
ca-path = "/path/to/ca.pem"
# Path of the file that contains X509 certificate in PEM format.
cert-path = "/path/to/tikv-server.pem"
# Path of the file that contains X509 key in PEM format.
key-path = "/path/to/tikv-server-key.pem"
```

- PD

  Configure in the configuration file or command-line arguments, and set the corresponding URL to `https`:

```
[security]
## The path for certificates. An empty string means that secure
   ↪ connections are disabled.
# Path of the file that contains a list of trusted SSL CAs. If it
   ↪ is set, the following settings `cert_path` and `key_path`
   ↪ are also needed.
cacert-path = "/path/to/ca.pem"
# Path of the file that contains X509 certificate in PEM format.
cert-path = "/path/to/pd-server.pem"
# Path of the file that contains X509 key in PEM format.
key-path = "/path/to/pd-server-key.pem"
```

- TiFlash (New in v4.0.5)

  Configure in the `tiflash.toml` file, and change the `http_port` item to `https_port`:

  ```
  toml [security] ## The path for certificates. An empty string means
  ↪  that secure connections are disabled. # Path of the file that
  ↪ contains a list of trusted SSL CAs. If it is set, the following
  ↪  settings `cert_path` and `key_path` are also needed. ca_path
  ↪  = "/path/to/ca.pem" # Path of the file that contains X509
  ↪ certificate in PEM format. cert_path = "/path/to/tiflash-server
  ↪ .pem" # Path of the file that contains X509 key in PEM format.
  ↪ key_path = "/path/to/tiflash-server-key.pem"
  ```

  Configure in the `tiflash-learner.toml` file:

  ```
  [security]
  # Path of the file that contains a list of trusted SSL CAs. If it
     ↪ is set, the following settings `cert_path` and `key_path`
     ↪ are also needed.
  ca-path = "/path/to/ca.pem"
  # Path of the file that contains X509 certificate in PEM format.
  cert-path = "/path/to/tiflash-server.pem"
  # Path of the file that contains X509 key in PEM format.
  key-path = "/path/to/tiflash-server-key.pem"
  ```

- TiCDC

  Configure in the command-line arguments and set the corresponding URL to `https`:

  ```
  cdc server --pd=https://127.0.0.1:2379 --log-file=ticdc.log --addr
     ↪ =0.0.0.0:8301 --advertise-addr=127.0.0.1:8301 --ca=/path/to/
     ↪ ca.pem --cert=/path/to/ticdc-cert.pem --key=/path/to/ticdc-
     ↪ key.pem
  ```

  Now, encrypted transmission among TiDB components is enabled.

> **Note:**
>
> After enabling encrypted transmission in a TiDB cluster, if you need to connect to the cluster using tidb-ctl, tikv-ctl, or pd-ctl, specify the client certificate. For example:

```
./tidb-ctl -u https://127.0.0.1:10080 --ca /path/to/ca.pem --ssl-cert /
   ↪ path/to/client.pem --ssl-key /path/to/client-key.pem
```

```
tiup ctl:<cluster-version> pd -u https://127.0.0.1:2379 --cacert /path/
    ↪ to/ca.pem --cert /path/to/client.pem --key /path/to/client-key.
    ↪ pem
```

```
./tikv-ctl --host="127.0.0.1:20160" --ca-path="/path/to/ca.pem" --cert-
    ↪ path="/path/to/client.pem" --key-path="/path/to/clinet-key.pem"
```

8.1.3.1.1 Verify component caller's identity

The Common Name is used for caller verification. In general, the callee needs to verify the caller's identity, in addition to verifying the key, the certificates, and the CA provided by the caller. For example, TiKV can only be accessed by TiDB, and other visitors are blocked even though they have legitimate certificates.

To verify component caller's identity, you need to mark the certificate user identity using `Common Name` when generating the certificate, and to check the caller's identity by configuring the `Common Name` list for the callee.

- TiDB

  Configure in the configuration file or command-line arguments:

```
[security]
cluster-verify-cn = [
    "TiDB-Server",
    "TiKV-Control",
]
```

- TiKV

  Configure in the configuration file or command-line arguments:

```
[security]
cert-allowed-cn = [
    "TiDB-Server", "PD-Server", "TiKV-Control", "RawKvClient1",
]
```

- PD

  Configure in the configuration file or command-line arguments:

```
[security]
cert-allowed-cn = ["TiKV-Server", "TiDB-Server", "PD-Control"]
```

- TiCDC

  Configure in the command-line arguments:

```
cdc server --pd=https://127.0.0.1:2379 --log-file=ticdc.log --addr
    ↪ =0.0.0.0:8301 --advertise-addr=127.0.0.1:8301 --ca=/path/to/ca.
    ↪ pem --cert=/path/to/ticdc-cert.pem --key=/path/to/ticdc-key.pem
    ↪ --cert-allowed-cn="client1,client2"
```

- TiFlash (New in v4.0.5)

  Configure in the `tiflash.toml` file or command-line arguments:

```
[security]
cert_allowed_cn = ["TiKV-Server", "TiDB-Server"]
```

  Configure in the `tiflash-learner.toml` file:

```
[security]
cert-allowed-cn = ["PD-Server", "TiKV-Server", "TiFlash-Server"]
```

8.1.3.1.2 Reload certificates

To reload the certificates and the keys, TiDB, PD, TiKV, and all kinds of clients reread the current certificates and the key files each time a new connection is created. Currently, you cannot reload the CA certificate.

8.1.3.2 See also

- Enable TLS Between TiDB Clients and Servers

8.1.4 Generate Self-Signed Certificates

> **Note:**
>
> To enable TLS between clients and servers, you only need to set `auto-tls`.

This document provides an example of using `openssl` to generate a self-signed certificate. You can also generate certificates and keys that meet requirements according to your demands.

Assume that the topology of the instance cluster is as follows:

Name	Host IP	Services
node1	172.16.10.11	PD1, TiDB1

Name	Host IP	Services
node2	172.16.10.12	PD2
node3	172.16.10.13	PD3
node4	172.16.10.14	TiKV1
node5	172.16.10.15	TiKV2
node6	172.16.10.16	TiKV3

8.1.4.1 Install OpenSSL

- For Debian or Ubuntu OS:

```
apt install openssl
```

- For RedHat or CentOS OS:

```
yum install openssl
```

You can also refer to OpenSSL's official download document for installation.

8.1.4.2 Generate the CA certificate

A certificate authority (CA) is a trusted entity that issues digital certificates. In practice, contact your administrator to issue the certificate or use a trusted CA. CA manages multiple certificate pairs. Here you only need to generate an original pair of certificates as follows.

1. Generate the root key:

```
openssl genrsa -out root.key 4096
```

2. Generate root certificates:

```
openssl req -new -x509 -days 1000 -key root.key -out root.crt
```

3. Validate root certificates:

```
openssl x509 -text -in root.crt -noout
```

8.1.4.3 Issue certificates for individual components

This section describes how to issue certificates for individual components.

8.1.4.3.1 Certificates that might be used in the cluster

- tidb-server certificate: used by TiDB to authenticate TiDB for other components and clients
- tikv-server certificate: used by TiKV to authenticate TiKV for other components and clients
- pd-server certificate: used by PD to authenticate PD for other components and clients
- client certificate: used to authenticate the clients from PD, TiKV and TiDB, such as `pd-ctl`, `tikv-ctl`

8.1.4.3.2 Issue certificates to TiKV instances

To issue a certificate to a TiKV instance, perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out tikv.key 2048
```

2. Make a copy of the OpenSSL configuration template file (Refer to the actual location of your template file because it might have more than one location):

```
cp /usr/lib/ssl/openssl.cnf .
```

   If you do not know the actual location, look for it in the root directory:

```
find / -name openssl.cnf
```

3. Edit `openssl.cnf`, add `req_extensions = v3_req` under the `[ req ]` field, and add `subjectAltName = @alt_names` under the `[ v3_req ]` field. Finally, create a new field and edit the information of SAN.

```
[ alt_names ]
IP.1 = 127.0.0.1
IP.2 = 172.16.10.14
IP.3 = 172.16.10.15
IP.4 = 172.16.10.16
```

4. Save the `openssl.cnf` file, and generate the certificate request file (in this step, you can also assign a Common Name to the certificate, which is used to allow the server to validate the identity of the client. Each component does not enable the validation by default, and you can enable it in the configuration file):

```
openssl req -new -key tikv.key -out tikv.csr -config openssl.cnf
```

5. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA root.crt -CAkey root.key -
    ↪ CAcreateserial -in tikv.csr -out tikv.crt -extensions v3_req -
    ↪ extfile openssl.cnf
```

6. Verify that the certificate includes the SAN field (optional):

```
openssl x509 -text -in tikv.crt -noout
```

7. Confirm that the following files exist in your current directory:

```
root.crt
tikv.crt
tikv.key
```

The process of issuing certificates for other TiDB components is similar and will not be repeated in this document.

8.1.4.3.3 Issue certificates for clients

To issue a certificate to a client, perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out client.key 2048
```

2. Generate the certificate request file (in this step, you can also assign a Common Name to the certificate, which is used to allow the server to validate the identity of the client. Each component does not enable the validation by default, and you can enable it in the configuration file):

```
openssl req -new -key client.key -out client.csr
```

3. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA root.crt -CAkey root.key -
    ↪ CAcreateserial -in client.csr -out client.crt
```

8.1.5 Encryption at Rest

Note:

If your cluster is deployed on AWS and uses the EBS storage, it is recommended to use the EBS encryption. See AWS documentation - EBS Encryption. You are using the non-EBS storage on AWS such as the local NVMe storage, it is recommended to use encryption at rest introduced in this document.

Encryption at rest means that data is encrypted when it is stored. For databases, this feature is also referred to as TDE (transparent data encryption). This is opposed to encryption in flight (TLS) or encryption in use (rarely used). Different things could be doing encryption at rest (such as SSD drive, file system, and cloud vendor), but by having TiKV do the encryption before storage this helps ensure that attackers must authenticate with the database to gain access to data. For example, when an attacker gains access to the physical machine, data cannot be accessed by copying files on disk.

8.1.5.1 Encryption support in different TiDB components

In a TiDB cluster, different components use different encryption methods. This section introduces the encryption supports in different TiDB components such as TiKV, TiFlash, PD, and Backup & Restore (BR).

When a TiDB cluster is deployed, the majority of user data is stored on TiKV and TiFlash nodes. Some metadata is stored on PD nodes (for example, secondary index keys used as TiKV Region boundaries). To get the full benefits of encryption at rest, you need to enable encryption for all components. Backups, log files, and data transmitted over the network should also be considered when you implement encryption.

8.1.5.1.1 TiKV

TiKV supports encryption at rest. This feature allows TiKV to transparently encrypt data files using AES in CTR mode. To enable encryption at rest, an encryption key must be provided by the user and this key is called master key. TiKV automatically rotates data keys that it used to encrypt actual data files. Manually rotating the master key can be done occasionally. Note that encryption at rest only encrypts data at rest (namely, on disk) and not while data is transferred over network. It is advised to use TLS together with encryption at rest.

Optionally, you can use AWS KMS for both cloud and self-hosted deployments. You can also supply the plaintext master key in a file.

TiKV currently does not exclude encryption keys and user data from core dumps. It is advised to disable core dumps for the TiKV process when using encryption at rest. This is not currently handled by TiKV itself.

TiKV tracks encrypted data files using the absolute path of the files. As a result, once encryption is turned on for a TiKV node, the user should not change data file paths configuration such as `storage.data-dir`, `raftstore.raftdb-path`, `rocksdb.wal-dir` and `raftdb.wal-dir`.

8.1.5.1.2 TiFlash

TiFlash supports encryption at rest. Data keys are generated by TiFlash. All files (including data files, schema files, and temporary files) written into TiFlash (including TiFlash Proxy) are encrypted using the current data key. The encryption algorithms, the encryption configuration (in the `tiflash-learner.toml` file) supported by TiFlash, and the meanings of monitoring metrics are consistent with those of TiKV.

If you have deployed TiFlash with Grafana, you can check the **TiFlash-Proxy-Details -> Encryption** panel.

8.1.5.1.3 PD

Encryption-at-rest for PD is an experimental feature, which is configured in the same way as in TiKV.

8.1.5.1.4 Backups with BR

BR supports S3 server-side encryption (SSE) when backing up data to S3. A customer-owned AWS KMS key can also be used together with S3 server-side encryption. See BR S3 server-side encryption for details.

8.1.5.1.5 Logging

TiKV, TiDB, and PD info logs might contain user data for debugging purposes. The info log and this data in it are not encrypted. It is recommended to enable log redaction.

8.1.5.2 TiKV encryption at rest

8.1.5.2.1 Overview

TiKV currently supports encrypting data using AES128, AES192 or AES256, in CTR mode. TiKV uses envelope encryption. As a result, two types of keys are used in TiKV when encryption is enabled.

- Master key. The master key is provided by user and is used to encrypt the data keys TiKV generates. Management of master key is external to TiKV.
- Data key. The data key is generated by TiKV and is the key actually used to encrypt data.

The same master key can be shared by multiple instances of TiKV. The recommended way to provide a master key in production is via AWS KMS. Create a customer master key (CMK) through AWS KMS, and then provide the CMK key ID to TiKV in the configuration file. The TiKV process needs access to the KMS CMK while it is running, which can be done by using an IAM role. If TiKV fails to get access to the KMS CMK, it will fail to start or restart. Refer to AWS documentation for KMS and IAM usage.

Alternatively, if using custom key is desired, supplying the master key via file is also supported. The file must contain a 256 bits (or 32 bytes) key encoded as hex string, end with a newline (namely, `\n`), and contain nothing else. Persisting the key on disk, however, leaks the key, so the key file is only suitable to be stored on the `tempfs` in RAM.

Data keys are passed to the underlying storage engine (namely, RocksDB). All files written by RocksDB, including SST files, WAL files, and the MANIFEST file, are encrypted by the current data key. Other temporary files used by TiKV that may include user data are also encrypted using the same data key. Data keys are automatically rotated by TiKV every week by default, but the period is configurable. On key rotation, TiKV does not rewrite all existing files to replace the key, but RocksDB compaction are expected to rewrite old data into new data files, with the most recent data key, if the cluster gets constant write workload. TiKV keeps track of the key and encryption method used to encrypt each of the files and use the information to decrypt the content on reads.

Regardless of data encryption method, data keys are encrypted using AES256 in GCM mode for additional authentication. This required the master key to be 256 bits (32 bytes), when passing from file instead of KMS.

8.1.5.2.2 Key creation

To create a key on AWS, follow these steps:

1. Go to the AWS KMS on the AWS console.
2. Make sure that you have selected the correct region on the top right corner of your console.
3. Click **Create key** and select **Symmetric** as the key type.
4. Set an alias for the key.

You can also perform the operations using the AWS CLI:

```
aws --region us-west-2 kms create-key
aws --region us-west-2 kms create-alias --alias-name "alias/tidb-tde" --
    ↪ target-key-id 0987dcba-09fe-87dc-65ba-ab0987654321
```

The `--target-key-id` to enter in the second command is in the output of the first command.

8.1.5.2.3 Configure encryption

To enable encryption, you can add the encryption section in the configuration files of TiKV and PD:

```
[security.encryption]
data-encryption-method = "aes128-ctr"
data-key-rotation-period = "168h" # 7 days
```

Possible values for `data-encryption-method` are "aes128-ctr", "aes192-ctr", "aes256-ctr" and "plaintext". The default value is "plaintext", which means encryption is not turned on. `data-key-rotation-period` defines how often TiKV rotates the data key. Encryption can be turned on for a fresh TiKV cluster, or an existing TiKV cluster, though only data written after encryption is enabled is guaranteed to be encrypted. To disable encryption, remove `data-encryption-method` in the configuration file, or reset it to "plaintext", and restart TiKV. To change encryption method, update `data-encryption-method` in the configuration file and restart TiKV.

The master key has to be specified if encryption is enabled (that is,`data-encryption-method` is not "plaintext"). To specify a AWS KMS CMK as master key, add the `encryption.master-key` section after the `encryption` section:

```
[security.encryption.master-key]
type = "kms"
key-id = "0987dcba-09fe-87dc-65ba-ab0987654321"
region = "us-west-2"
endpoint = "https://kms.us-west-2.amazonaws.com"
```

The `key-id` specifies the key ID for the KMS CMK. The `region` is the AWS region name for the KMS CMK. The `endpoint` is optional and you do not need to specify it normally unless you are using an AWS KMS-compatible service from a non-AWS vendor or need to use a VPC endpoint for KMS.

You can also use multi-Region keys in AWS. For this, you need to set up a primary key in a specific region and add replica keys in the regions you require.

To specify a master key that's stored in a file, the master key configuration would look like the following:

```
[security.encryption.master-key]
type = "file"
path = "/path/to/key/file"
```

Here `path` is the path to the key file. The file must contain a 256 bits (or 32 bytes) key encoded as hex string, end with a newline (`\n`) and contain nothing else. Example of the file content:

```
3b5896b5be691006e0f71c3040a29495ddcad20b14aff61806940ebd780d3c62
```

8.1.5.2.4 Rotate the master key

To rotate master key, you have to specify both of the new master key and old master key in the configuration, and restart TiKV. Use `security.encryption.master-key` to specify the new master key, and use `security.encryption.previous-master-key` to specify the old master key. The configuration format for `security.encryption.previous-master-key` ↪ is the same as `encryption.master-key`. On restart TiKV must access both of the old and new master key, but once TiKV is up and running, TiKV will only need access to the new key. It is okay to leave the `encryption.previous-master-key` configuration in the configuration file from that on. Even on restart, TiKV only tries to use the old key if it fails to decrypt existing data using the new master key.

Currently online master key rotation is not supported, so you need to restart TiKV. It is advised to do a rolling restart to a running TiKV cluster serving online query.

Here is an example configuration for rotating the KMS CMK:

```
[security.encryption.master-key]
type = "kms"
key-id = "50a0c603-1c6f-11e6-bb9e-3fadde80ce75"
region = "us-west-2"

[security.encryption.previous-master-key]
type = "kms"
key-id = "0987dcba-09fe-87dc-65ba-ab0987654321"
region = "us-west-2"
```

8.1.5.2.5 Monitoring and debugging

To monitor encryption at rest, if you deploy TiKV with Grafana, you can look at the **Encryption** panel in the **TiKV-Details** dashboard. There are a few metrics to look for:

- Encryption initialized: 1 if encryption is initialized during TiKV startup, 0 otherwise. In case of master key rotation, after encryption is initialized, TiKV do not need access to the previous master key.
- Encryption data keys: number of existing data keys. The number is bumped by 1 after each time data key rotation happened. Use this metrics to monitor if data key rotation works as expected.
- Encrypted files: number of encrypted data files currently exists. Compare this number to existing data files in the data directory to estimate portion of data being encrypted, when turning on encryption for a previously unencrypted cluster.
- Encryption meta file size: size of the encryption meta data files.
- Read/Write encryption meta duration: the extra overhead to operate on metadata for encryption.

For debugging, the `tikv-ctl` command can be used to dump encryption metadata such as encryption method and data key id used to encryption the file, as well as list of data keys.

Since the operation can expose sensitive data, it is not recommended to use in production. Please refer to TiKV Control document.

8.1.5.2.6 Compatibility between TiKV versions

To reduce the overhead caused by I/O and mutex contention when TiKV manages the encryption metadata, an optimization is introduced in TiKV v4.0.9 and controlled by `security.encryption.enable-file-dictionary-log` in the TiKV configuration file. This configuration parameter takes effect only on TiKV v4.0.9 or later versions.

When it is enabled (by default), the data format of encryption metadata is unrecognizable by TiKV v4.0.8 or earlier versions. For example, assume that you use TiKV v4.0.9 or later with encryption at rest and the default `enable-file-dictionary-log` configuration. If you downgrade your cluster to TiKV v4.0.8 or earlier, TiKV will fail to start, with an error in the info log similar to the following one:

```
[2020/12/07 07:26:31.106 +08:00] [ERROR] [mod.rs:110] ["encryption: failed
   ↪ to load file dictionary."]
[2020/12/07 07:26:33.598 +08:00] [FATAL] [lib.rs:483] ["called `Result::
   ↪ unwrap()` on an `Err` value: Other(\"[components/encryption/src/
   ↪ encrypted_file/header.rs:18]: unknown version 2\")"]
```

To avoid the error above, you can first set `security.encryption.enable-file-` ↪ `dictionary-log` to `false` and start TiKV with v4.0.9 or later. Once TiKV starts successfully, the data format of encryption metadata is downgraded to the version recognizable to earlier TiKV versions. At this point, you can then downgrade your TiKV cluster to an earlier version.

8.1.5.3 BR S3 server-side encryption

To enable S3 server-side encryption when backup to S3 using BR, pass `--s3.sse` argument and set value to "aws:kms". S3 will use its own KMS key for encryption. Example:

```
./br backup full --pd <pd-address> --storage "s3://<bucket>/<prefix>" --s3.
   ↪ sse aws:kms
```

To use a custom AWS KMS CMK that you created and owned, pass `--s3.sse-kms-` ↪ `key-id` in addition. In this case, both the BR process and all the TiKV nodes in the cluster would need access to the KMS CMK (for example, via AWS IAM), and the KMS CMK needs to be in the same AWS region as the S3 bucket used to store the backup. It is advised to grant access to the KMS CMK to BR process and TiKV nodes via AWS IAM. Refer to AWS documentation for usage of IAM. For example:

```
./br backup full --pd <pd-address> --storage "s3://<bucket>/<prefix>" --s3.
   ↪ region <region> --s3.sse aws:kms --s3.sse-kms-key-id 0987dcba-09fe-87
   ↪ dc-65ba-ab0987654321
```

When restoring the backup, both `--s3.sse` and `--s3.sse-kms-key-id` should NOT be used. S3 will figure out encryption settings by itself. The BR process and TiKV nodes in the cluster to restore the backup to would also need access to the KMS CMK, or the restore will fail. Example:

```
./br restore full --pd <pd-address> --storage "s3://<bucket>/<prefix> --s3.
    ↪ region <region>"
```

8.1.6 Enable Encryption for Disk Spill

When the `oom-use-tmp-storage` configuration item is set to `true`, if the memory usage of a single SQL statement exceeds the limit of the system variable `tidb_mem_quota_query`, some operators can save the intermediate results during execution as a temporary file to the disk and delete the file after the query is completed.

You can enable encryption for disk spill to prevent attackers from accessing data by reading these temporary files.

8.1.6.1 Configure

To enable encryption for the disk spill files, you can configure the item `spilled-file-encryption-method` in the `[security]` section of the TiDB configuration file.

```
[security]
spilled-file-encryption-method = "aes128-ctr"
```

Value options for `spilled-file-encryption-method` are `aes128-ctr` and `plaintext`. The default value is `plaintext`, which means that encryption is disabled.

8.1.7 Log Redaction

When TiDB provides detailed log information, it might print sensitive data (for example, user data) in the log, which causes data security risks. To avoid such risks, each component (TiDB, TiKV, and PD) provides a configuration item that enables log redaction to shield user data values.

8.1.7.1 Log redaction in TiDB side

To enable log redaction in the TiDB side, set the value of `global.tidb_redact_log` to `1`. This configuration value defaults to `0`, which means that log redaction is disabled.

You can use the `set` syntax to set the global variable `tidb_redact_log`:

```
set @@global.tidb_redact_log=1;
```

After the setting, all logs generated in new sessions are redacted:

```
create table t (a int, unique key (a));
Query OK, 0 rows affected (0.00 sec)

insert into t values (1),(1);
ERROR 1062 (23000): Duplicate entry '1' for key 'a'
```

The error log for the `INSERT` statement above is printed as follows:

```
[2020/10/20 11:45:49.539 +08:00] [INFO] [conn.go:800] ["command dispatched
    ↪ failed"] [conn=5] [connInfo="id:5, addr:127.0.0.1:57222 status:10,
    ↪ collation:utf8_general_ci, user:root"] [command=Query] [status="inTxn
    ↪ :0, autocommit:1"] [sql="insert into t values ( ? ) , ( ? )"] [
    ↪ txn_mode=OPTIMISTIC] [err="[kv:1062]Duplicate entry '?' for key 'a'"]
```

From the error log above, you can see that all sensitive information is shielded using `?` after `tidb_redact_log` is enabled. In this way, data security risks are avoided.

8.1.7.2 Log redaction in TiKV side

To enable log redaction in the TiKV side, set the value of `security.redact-info-log` ↪ to `true`. This configuration value defaults to `false`, which means that log redaction is disabled.

8.1.7.3 Log redaction in PD side

To enable log redaction in the PD side, set the value of `security.redact-info-log` to `true`. This configuration value defaults to `false`, which means that log redaction is disabled.

8.1.7.4 Log redaction in TiFlash side

To enable log redaction in the TiFlash side, set both the `security.redact_info_log` value in tiflash-server and the `security.redact-info-log` value in tiflash-learner to `true`. Both configuration values default to `false`, which means that log redaction is disabled.

8.2 Upgrade

8.2.1 Upgrade TiDB Using TiUP

This document is targeted for the following upgrade paths:

- Upgrade from TiDB 4.0 versions to TiDB 6.1 versions.
- Upgrade from TiDB 5.0-5.4 versions to TiDB 6.1 versions.
- Upgrade from TiDB 6.0 to TiDB 6.1 versions.

Warning:

- You cannot upgrade TiFlash online from versions earlier than 5.3 to 5.3 or later. Instead, you must first stop all the TiFlash instances of the early version, and then upgrade the cluster offline. If other components (such as TiDB and TiKV) do not support an online upgrade, follow the instructions in warnings in Online upgrade.
- **DO NOT** upgrade a TiDB cluster when a DDL statement is being executed in the cluster (usually for the time-consuming DDL statements such as `ADD INDEX` and the column type changes).
- Before the upgrade, it is recommended to use the `ADMIN SHOW DDL` command to check whether the TiDB cluster has an ongoing DDL job. If the cluster has a DDL job, to upgrade the cluster, wait until the DDL execution is finished or use the `ADMIN CANCEL DDL` command to cancel the DDL job before you upgrade the cluster.
- In addition, during the cluster upgrade, **DO NOT** execute any DDL statement. Otherwise, the issue of undefined behavior might occur.

Note:

If your cluster to be upgraded is v3.1 or an earlier version (v3.0 or v2.1), the direct upgrade to v6.1.0 or its patch versions is not supported. You need to upgrade your cluster first to v4.0 and then to v6.1.0.

8.2.1.1 Upgrade caveat

- TiDB currently does not support version downgrade or rolling back to an earlier version after the upgrade.
- For the v4.0 cluster managed using TiDB Ansible, you need to import the cluster to TiUP (`tiup cluster`) for new management according to Upgrade TiDB Using TiUP (v4.0). Then you can upgrade the cluster to v6.1.0 or its patch versions according to this document.
- To update versions earlier than v3.0 to v6.1.0:
    1. Update this version to 3.0 using TiDB Ansible.
    2. Use TiUP (`tiup cluster`) to import the TiDB Ansible configuration.
    3. Update the 3.0 version to 4.0 according to Upgrade TiDB Using TiUP (v4.0).
    4. Upgrade the cluster to v6.1.0 according to this document.

- Support upgrading the versions of TiDB Binlog, TiCDC, TiFlash, and other components.
- For detailed compatibility changes of different versions, see the Release Notes of each version. Modify your cluster configuration according to the "Compatibility Changes" section of the corresponding release notes.
- For clusters that upgrade from versions earlier than v5.3 to v5.3 or later versions, the default deployed Prometheus will upgrade from v2.8.1 to v2.27.1. Prometheus v2.27.1 provides more features and fixes a security issue. Compared with v2.8.1, alert time representation in v2.27.1 is changed. For more details, see Prometheus commit for more details.

8.2.1.2 Preparations

This section introduces the preparation works needed before upgrading your TiDB cluster, including upgrading TiUP and the TiUP Cluster component.

8.2.1.2.1 Step 1: Review compatibility changes

Review the compatibility changes in TiDB v6.1.0 release notes. If any changes affect your upgrade, take actions accordingly.

8.2.1.2.2 Step 2: Upgrade TiUP or TiUP offline mirror

Before upgrading your TiDB cluster, you first need to upgrade TiUP or TiUP mirror.

Upgrade TiUP and TiUP Cluster

> **Note:**
>
> If the control machine of the cluster to upgrade cannot access `https://tiup-mirrors.pingcap.com`, skip this section and see Upgrade TiUP offline mirror.

1. Upgrade the TiUP version. It is recommended that the TiUP version is `1.10.0` or later.

```
tiup update --self
tiup --version
```

2. Upgrade the TiUP Cluster version. It is recommended that the TiUP Cluster version is `1.10.0` or later.

```
tiup update cluster
tiup cluster --version
```

Upgrade TiUP offline mirror

> **Note:**
>
> If the cluster to upgrade was deployed not using the offline method, skip this step.

Refer to Deploy a TiDB Cluster Using TiUP - Deploy TiUP offline to download the TiUP mirror of the new version and upload it to the control machine. After executing `local_install.sh`, TiUP will complete the overwrite upgrade.

```
tar xzvf tidb-community-server-${version}-linux-amd64.tar.gz
sh tidb-community-server-${version}-linux-amd64/local_install.sh
source /home/tidb/.bash_profile
```

After the overwrite upgrade, run the following command to merge the server and toolkit offline mirrors to the server directory:

```
tar xf tidb-community-toolkit-${version}-linux-amd64.tar.gz
ls -ld tidb-community-server-${version}-linux-amd64 tidb-community-toolkit-$
    ↪ {version}-linux-amd64
cd tidb-community-server-${version}-linux-amd64/
cp -rp keys ~/.tiup/
tiup mirror merge ../tidb-community-toolkit-${version}-linux-amd64
```

After merging the mirrors, run the following command to upgrade the TiUP Cluster component:

```
tiup update cluster
```

Now, the offline mirror has been upgraded successfully. If an error occurs during TiUP operation after the overwriting, it might be that the `manifest` is not updated. You can try `rm -rf ~/.tiup/manifests/*` before running TiUP again.

8.2.1.2.3 Step 3: Edit TiUP topology configuration file

> **Note:**
>
> Skip this step if one of the following situations applies:
>
> - You have not modified the configuration parameters of the original cluster. Or you have modified the configuration parameters using `tiup ↪ cluster` but no more modification is needed.

- After the upgrade, you want to use v6.1.0's default parameter values for the unmodified configuration items.

1. Enter the `vi` editing mode to edit the topology file:

```
tiup cluster edit-config <cluster-name>
```

2. Refer to the format of topology configuration template and fill the parameters you want to modify in the `server_configs` section of the topology file.

3. After the modification, enter : + w + q to save the change and exit the editing mode. Enter Y to confirm the change.

Note:

Before you upgrade the cluster to v6.1.0, make sure that the parameters you have modified in v4.0 are compatible in v6.1.0. For details, see TiKV Configuration File.

8.2.1.2.4 Step 4: Check the health status of the current cluster

To avoid the undefined behaviors or other issues during the upgrade, it is recommended to check the health status of Regions of the current cluster before the upgrade. To do that, you can use the `check` sub-command.

```
tiup cluster check <cluster-name> --cluster
```

After the command is executed, the "Region status" check result will be output.

- If the result is "All Regions are healthy", all Regions in the current cluster are healthy and you can continue the upgrade.
- If the result is "Regions are not fully healthy: m miss-peer, n pending-peer" with the "Please fix unhealthy regions before other operations." prompt, some Regions in the current cluster are abnormal. You need to troubleshoot the anomalies until the check result becomes "All Regions are healthy". Then you can continue the upgrade.

8.2.1.3 Upgrade the TiDB cluster

This section describes how to upgrade the TiDB cluster and verify the version after the upgrade.

8.2.1.3.1 Upgrade the TiDB cluster to a specified version

You can upgrade your cluster in one of the two ways: online upgrade and offline upgrade.

By default, TiUP Cluster upgrades the TiDB cluster using the online method, which means that the TiDB cluster can still provide services during the upgrade process. With the online method, the leaders are migrated one by one on each node before the upgrade and restart. Therefore, for a large-scale cluster, it takes a long time to complete the entire upgrade operation.

If your application has a maintenance window for the database to be stopped for maintenance, you can use the offline upgrade method to quickly perform the upgrade operation.

Online upgrade

```
tiup cluster upgrade <cluster-name> <version>
```

For example, if you want to upgrade the cluster to v6.1.7:

```
tiup cluster upgrade <cluster-name> v6.1.7
```

Note:

- An online upgrade upgrades all components one by one. During the upgrade of TiKV, all leaders in a TiKV instance are evicted before stopping the instance. The default timeout time is 5 minutes (300 seconds). The instance is directly stopped after this timeout time.
- You can use the `--force` parameter to upgrade the cluster immediately without evicting the leader. However, the errors that occur during the upgrade will be ignored, which means that you are not notified of any upgrade failure. Therefore, use the `--force` parameter with caution.
- To keep a stable performance, make sure that all leaders in a TiKV instance are evicted before stopping the instance. You can set `--transfer-timeout` to a larger value, for example, `--transfer-timeout 3600` (unit: second).
- To upgrade TiFlash from versions earlier than 5.3 to 5.3 or later, you should stop TiFlash and then upgrade it. The following steps help you upgrade TiFlash without interrupting other components:

1. Stop the TiFlash instance: `tiup cluster stop <cluster-name> -R tiflash`
2. Upgrade the TiDB cluster without restarting it (only updating the files): `tiup cluster upgrade <cluster-name> <version> --offline`

3. Reload the TiDB cluster: `tiup cluster reload <cluster-name>`. After the reload, the TiFlash instance is started and you do not need to manually start it.

- Try to avoid creating a new clustered index table when you apply rolling updates to the clusters using TiDB Binlog.

Offline upgrade

1. Before the offline upgrade, you first need to stop the entire cluster.

```
tiup cluster stop <cluster-name>
```

2. Use the `upgrade` command with the `--offline` option to perform the offline upgrade.

```
tiup cluster upgrade <cluster-name> <version> --offline
```

3. After the upgrade, the cluster will not be automatically restarted. You need to use the `start` command to restart it.

```
tiup cluster start <cluster-name>
```

8.2.1.3.2 Verify the cluster version

Execute the `display` command to view the latest cluster version `TiDB Version`:

```
tiup cluster display <cluster-name>
```

```
Cluster type:     tidb
Cluster name:     <cluster-name>
Cluster version:  v6.1.7
```

8.2.1.4 FAQ

This section describes common problems encountered when updating the TiDB cluster using TiUP.

8.2.1.4.1 If an error occurs and the upgrade is interrupted, how to resume the upgrade after fixing this error?

Re-execute the `tiup cluster upgrade` command to resume the upgrade. The upgrade operation restarts the nodes that have been previously upgraded. If you do not want the upgraded nodes to be restarted, use the `replay` sub-command to retry the operation:

1. Execute `tiup cluster audit` to see the operation records:

```
tiup cluster audit
```

   Find the failed upgrade operation record and keep the ID of this operation record. The ID is the `<audit-id>` value in the next step.

2. Execute `tiup cluster replay <audit-id>` to retry the corresponding operation:

```
tiup cluster replay <audit-id>
```

8.2.1.4.2 The evict leader has waited too long during the upgrade. How to skip this step for a quick upgrade?

You can specify `--force`. Then the processes of transferring PD leader and evicting TiKV leader are skipped during the upgrade. The cluster is directly restarted to update the version, which has a great impact on the cluster that runs online. Here is the command:

```
tiup cluster upgrade <cluster-name> <version> --force
```

8.2.1.4.3 How to update the version of tools such as pd-ctl after upgrading the TiDB cluster?

You can upgrade the tool version by using TiUP to install the `ctl` component of the corresponding version:

```
tiup install ctl:v6.1.7
```

8.2.2 Use TiDB Operator

8.3 Scale

8.3.1 Scale a TiDB Cluster Using TiUP

The capacity of a TiDB cluster can be increased or decreased without interrupting the online services.

This document describes how to scale the TiDB, TiKV, PD, TiCDC, or TiFlash cluster using TiUP. If you have not installed TiUP, refer to the steps in Step 2. Deploy TiUP on the control machine.

To view the current cluster name list, run `tiup cluster list`.

For example, if the original topology of the cluster is as follows:

Host IP	Service
10.0.1.3	TiDB + TiFlash

Host IP	Service
10.0.1.4	TiDB + PD
10.0.1.5	TiKV + Monitor
10.0.1.1	TiKV
10.0.1.2	TiKV

8.3.1.1 Scale out a TiDB/PD/TiKV cluster

This section exemplifies how to add a TiDB node to the `10.0.1.5` host.

> **Note:**
>
> You can take similar steps to add a PD node. Before you add a TiKV node, it is recommended that you adjust the PD scheduling parameters in advance according to the cluster load.

1. Configure the scale-out topology:

> **Note:**
>
> - The port and directory information is not required by default.
> - If multiple instances are deployed on a single machine, you need to allocate different ports and directories for them. If the ports or directories have conflicts, you will receive a notification during deployment or scaling.
> - Since TiUP v1.0.0, the scale-out configuration inherits the global configuration of the original cluster.

Add the scale-out topology configuration in the `scale-out.yml` file:

```
vi scale-out.yml
```

```
tidb_servers:
- host: 10.0.1.5
  ssh_port: 22
  port: 4000
  status_port: 10080
  deploy_dir: /tidb-deploy/tidb-4000
  log_dir: /tidb-deploy/tidb-4000/log
```

Here is a TiKV configuration file template:

```
tikv_servers:
- host: 10.0.1.5
  ssh_port: 22
  port: 20160
  status_port: 20180
  deploy_dir: /tidb-deploy/tikv-20160
  data_dir: /tidb-data/tikv-20160
  log_dir: /tidb-deploy/tikv-20160/log
```

Here is a PD configuration file template:

```
pd_servers:
- host: 10.0.1.5
  ssh_port: 22
  name: pd-1
  client_port: 2379
  peer_port: 2380
  deploy_dir: /tidb-deploy/pd-2379
  data_dir: /tidb-data/pd-2379
  log_dir: /tidb-deploy/pd-2379/log
```

To view the configuration of the current cluster, run `tiup cluster edit-config <cluster-name>`. Because the parameter configuration of `global` and `server_configs` is inherited by `scale-out.yml` and thus also takes effect in `scale-out.yml`.

2. Run the scale-out command:

   Before you run the `scale-out` command, use the `check` and `check --apply` commands to detect and automatically repair potential risks in the cluster:

   1. Check for potential risks:

      ```
      tiup cluster check <cluster-name> scale-out.yml --cluster --user
          ↪ root [-p] [-i /home/root/.ssh/gcp_rsa]
      ```

   2. Enable automatic repair:

      ```
      tiup cluster check <cluster-name> scale-out.yml --cluster --apply
          ↪ --user root [-p] [-i /home/root/.ssh/gcp_rsa]
      ```

   3. Run the `scale-out` command:

      ```
      tiup cluster scale-out <cluster-name> scale-out.yml [-p] [-i /home/
          ↪ root/.ssh/gcp_rsa]
      ```

   In the preceding commands:

- `scale-out.yml` is the scale-out configuration file.
- `--user root` indicates logging in to the target machine as the `root` user to complete the cluster scale out. The `root` user is expected to have `ssh` and `sudo` privileges to the target machine. Alternatively, you can use other users with `ssh` and `sudo` privileges to complete the deployment.
- `[-i]` and `[-p]` are optional. If you have configured login to the target machine without password, these parameters are not required. If not, choose one of the two parameters. `[-i]` is the private key of the root user (or other users specified by `--user`) that has access to the target machine. `[-p]` is used to input the user password interactively.

If you see `Scaled cluster <cluster-name> out successfully`, the scale-out operation succeeds.

3. Refresh the cluster configuration.

Note:

- Refreshing cluster configuration is only required after you add PD nodes. If you only add TiDB or TiKV nodes, skip this step.
- If you are using TiUP v1.15.0 or a later version, skip this step because TiUP does it. If you are using a TiUP version earlier than v1.15.0, perform the following sub-steps.

1. Refresh the cluster configuration:

```
tiup cluster reload <cluster-name> --skip-restart
```

2. Refresh the Prometheus configuration and restart Prometheus:

```
tiup cluster reload <cluster-name> -R prometheus
```

4. Check the cluster status:

```
tiup cluster display <cluster-name>
```

Access the monitoring platform at http://10.0.1.5:3000 using your browser to monitor the status of the cluster and the new node.

After the scale-out, the cluster topology is as follows:

Host IP	Service
10.0.1.3	TiDB + TiFlash
10.0.1.4	TiDB + PD
10.0.1.5	**TiDB** + TiKV + Monitor
10.0.1.1	TiKV
10.0.1.2	TiKV

8.3.1.2 Scale out a TiFlash cluster

This section exemplifies how to add a TiFlash node to the `10.0.1.4` host.

> **Note:**
>
> When adding a TiFlash node to an existing TiDB cluster, note the following:
>
> - Confirm that the current TiDB version supports using TiFlash. Otherwise, upgrade your TiDB cluster to v5.0 or later versions.
> - Run the `tiup ctl:<cluster-version> pd -u http://<pd_ip>:<pd_port> config set enable-placement-rules true` command to enable the Placement Rules feature. Or run the corresponding command in pd-ctl.

1. Add the node information to the `scale-out.yml` file:

   Create the `scale-out.yml` file to add the TiFlash node information.

   ```
   tiflash_servers:
   - host: 10.0.1.4
   ```

   Currently, you can only add IP addresses but not domain names.

2. Run the scale-out command:

   ```
   tiup cluster scale-out <cluster-name> scale-out.yml
   ```

> **Note:**
>
> The preceding command is based on the assumption that the mutual trust has been configured for the user to run the command and the new machine. If the mutual trust cannot be configured, use the `-p` option to enter the password of the new machine, or use the `-i` option to specify the private key file.

3. View the cluster status:

   ```
   tiup cluster display <cluster-name>
   ```

   Access the monitoring platform at http://10.0.1.5:3000 using your browser, and view the status of the cluster and the new node.

After the scale-out, the cluster topology is as follows:

Host IP	Service
10.0.1.3	TiDB + TiFlash
10.0.1.4	TiDB + PD + **TiFlash**
10.0.1.5	TiDB+ TiKV + Monitor
10.0.1.1	TiKV
10.0.1.2	TiKV

8.3.1.3 Scale out a TiCDC cluster

This section exemplifies how to add two TiCDC nodes to the `10.0.1.3` and `10.0.1.4` hosts.

1. Add the node information to the `scale-out.yml` file:

   Create the `scale-out.yml` file to add the TiCDC node information.

```
cdc_servers:
  - host: 10.0.1.3
    gc-ttl: 86400
    data_dir: /tidb-data/cdc-8300
  - host: 10.0.1.4
    gc-ttl: 86400
    data_dir: /tidb-data/cdc-8300
```

2. Run the scale-out command:

```
tiup cluster scale-out <cluster-name> scale-out.yml
```

> **Note:**
>
> The preceding command is based on the assumption that the mutual trust has been configured for the user to run the command and the new machine. If the mutual trust cannot be configured, use the `-p` option to enter the password of the new machine, or use the `-i` option to specify the private key file.

3. View the cluster status:

```
tiup cluster display <cluster-name>
```

   Access the monitoring platform at http://10.0.1.5:3000 using your browser, and view the status of the cluster and the new nodes.

After the scale-out, the cluster topology is as follows:

Host IP	Service
10.0.1.3	TiDB + TiFlash + **TiCDC**
10.0.1.4	TiDB + PD + TiFlash + **TiCDC**
10.0.1.5	TiDB+ TiKV + Monitor
10.0.1.1	TiKV
10.0.1.2	TiKV

8.3.1.4 Scale in a TiDB/PD/TiKV cluster

This section exemplifies how to remove a TiKV node from the `10.0.1.5` host.

> **Note:**
>
> - You can take similar steps to remove a TiDB or PD node.
> - Because the TiKV, TiFlash, and TiDB Binlog components are taken offline asynchronously and the stopping process takes a long time, TiUP takes them offline in different methods. For details, see Particular handling of components' offline process.
> - The PD Client in TiKV caches the list of PD nodes. The current version of TiKV has a mechanism to automatically and regularly update PD nodes, which can help mitigate the issue of an expired list of PD nodes cached by TiKV. However, after scaling out PD, you should try to avoid directly removing all PD nodes at once that exist before the scaling. If necessary, before making all the previously existing PD nodes offline, make sure to switch the PD leader to a newly added PD node.

1. View the node ID information:

```
tiup cluster display <cluster-name>
```

```
Starting /root/.tiup/components/cluster/v1.10.0/cluster display <
    ↪ cluster-name>
TiDB Cluster: <cluster-name>
TiDB Version: v6.1.7
ID            Role        Host       Ports                            Status
    ↪  Data Dir             Deploy Dir
--            ----        ----       -----                            ------
    ↪  --------             ----------
10.0.1.3:8300 cdc         10.0.1.3   8300                             Up
    ↪      data/cdc-8300        deploy/cdc-8300
10.0.1.4:8300 cdc         10.0.1.4   8300                             Up
    ↪      data/cdc-8300        deploy/cdc-8300
```

```
10.0.1.4:2379 pd          10.0.1.4   2379/2380
    ↪ Healthy data/pd-2379        deploy/pd-2379
10.0.1.1:20160 tikv       10.0.1.1   20160/20180                  Up
    ↪     data/tikv-20160      deploy/tikv-20160
10.0.1.2:20160 tikv       10.0.1.2   20160/20180                  Up
    ↪     data/tikv-20160      deploy/tikv-20160
10.0.1.5:20160 tikv       10.0.1.5   20160/20180                  Up
    ↪     data/tikv-20160      deploy/tikv-20160
10.0.1.3:4000 tidb        10.0.1.3   4000/10080                   Up
    ↪     -                    deploy/tidb-4000
10.0.1.4:4000 tidb        10.0.1.4   4000/10080                   Up
    ↪     -                    deploy/tidb-4000
10.0.1.5:4000 tidb        10.0.1.5   4000/10080                   Up
    ↪     -                    deploy/tidb-4000
10.0.1.3:9000 tiflash     10.0.1.3   9000/8123/3930/20170/20292/8234 Up
    ↪  data/tiflash-9000    deploy/tiflash-9000
10.0.1.4:9000 tiflash     10.0.1.4   9000/8123/3930/20170/20292/8234 Up
    ↪  data/tiflash-9000    deploy/tiflash-9000
10.0.1.5:9090 prometheus  10.0.1.5   9090                         Up
    ↪     data/prometheus-9090 deploy/prometheus-9090
10.0.1.5:3000 grafana     10.0.1.5   3000                         Up
    ↪     -                    deploy/grafana-3000
10.0.1.5:9093 alertmanager 10.0.1.5 9093/9294                     Up
    ↪     data/alertmanager-9093 deploy/alertmanager-9093
```

2. Run the scale-in command:

```
tiup cluster scale-in <cluster-name> --node 10.0.1.5:20160
```

The `--node` parameter is the ID of the node to be taken offline.

If you see `Scaled cluster <cluster-name> in successfully`, the scale-in operation succeeds.

3. Refresh the cluster configuration.

Note:

- Refreshing cluster configuration is only required after you remove PD nodes. If you only remove TiDB or TiKV nodes, skip this step.
- If you are using TiUP v1.15.0 or a later version, skip this step because TiUP does it. If you are using a TiUP version earlier than v1.15.0, perform the following sub-steps.

1. Refresh the cluster configuration:

```
tiup cluster reload <cluster-name> --skip-restart
```

2. Refresh the Prometheus configuration and restart Prometheus:

```
tiup cluster reload <cluster-name> -R prometheus
```

4. Check the cluster status:

   The scale-in process takes some time. You can run the following command to check the scale-in status:

```
tiup cluster display <cluster-name>
```

   If the node to be scaled in becomes `Tombstone`, the scale-in operation succeeds.

   Access the monitoring platform at http://10.0.1.5:3000 using your browser, and view the status of the cluster.

The current topology is as follows:

Host IP	Service
10.0.1.3	TiDB + TiFlash + TiCDC
10.0.1.4	TiDB + PD + TiFlash + TiCDC
10.0.1.5	TiDB + Monitor **(TiKV is deleted)**
10.0.1.1	TiKV
10.0.1.2	TiKV

8.3.1.5 Scale in a TiFlash cluster

This section exemplifies how to remove a TiFlash node from the `10.0.1.4` host.

8.3.1.5.1 1. Adjust the number of replicas of the tables according to the number of remaining TiFlash nodes

1. Query whether any table has TiFlash replicas more than the number of TiFlash nodes after scale-in. `tobe_left_nodes` means the number of TiFlash nodes after scale-in. If the query result is empty, you can start scaling in TiFlash. If the query result is not empty, you need to modify the number of TiFlash replicas of the related table(s).

```
SELECT * FROM information_schema.tiflash_replica WHERE REPLICA_COUNT >
    ↪  'tobe_left_nodes';
```

2. Execute the following statement for all tables with TiFlash replicas more than the number of TiFlash nodes after scale-in. `new_replica_num` must be less than or equal to `tobe_left_nodes`:

```
ALTER TABLE <db-name>.<table-name> SET tiflash replica '
    ↪ new_replica_num';
```

3. Perform step 1 again and make sure that there is no table with TiFlash replicas more than the number of TiFlash nodes after scale-in.

8.3.1.5.2 2. Perform the scale-in operation

Perform the scale-in operation with one of the following solutions.

Solution 1. Use TiUP to remove a TiFlash node

1. Confirm the name of the node to be taken down:

```
tiup cluster display <cluster-name>
```

2. Remove the TiFlash node (assume that the node name is `10.0.1.4:9000` from Step 1):

```
tiup cluster scale-in <cluster-name> --node 10.0.1.4:9000
```

Solution 2. Manually remove a TiFlash node

In special cases (such as when a node needs to be forcibly taken down), or if the TiUP scale-in operation fails, you can manually remove a TiFlash node with the following steps.

1. Use the store command of pd-ctl to view the store ID corresponding to this TiFlash node.

    - Enter the store command in pd-ctl (the binary file is under `resources/bin` in the tidb-ansible directory).
    - If you use TiUP deployment, replace `pd-ctl` with `tiup ctl:<cluster-version > pd`:

```
tiup ctl:<cluster-version> pd -u http://<pd_ip>:<pd_port> store
```

> **Note:**
> If multiple PD instances exist in the cluster, you only need to specify the IP address:port of an active PD instance in the above command.

2. Remove the TiFlash node in pd-ctl:

- Enter `store delete <store_id>` in pd-ctl (`<store_id>` is the store ID of the TiFlash node found in the previous step.
- If you use TiUP deployment, replace `pd-ctl` with `tiup ctl:<cluster-version > pd`:

```
tiup ctl:<cluster-version> pd -u http://<pd_ip>:<pd_port> store
    ↪ delete <store_id>
```

> **Note:**
>
> If multiple PD instances exist in the cluster, you only need to specify the IP address:port of an active PD instance in the above command.

3. Wait for the store of the TiFlash node to disappear or for the `state_name` to become `Tombstone` before you stop the TiFlash process.
4. Manually delete TiFlash data files (the location can be found in the `data_dir` directory under the TiFlash configuration of the cluster topology file).
5. Delete information about the TiFlash node that goes down from the cluster topology using the following command:

```
tiup cluster scale-in <cluster-name> --node <pd_ip>:<pd_port> --force
```

> **Note:**
>
> Before all TiFlash nodes in the cluster stop running, if not all tables replicated to TiFlash are canceled, you need to manually clean up the replication rules in PD, or the TiFlash node cannot be taken down successfully.

The steps to manually clean up the replication rules in PD are below:

1. View all data replication rules related to TiFlash in the current PD instance:

```
curl http://<pd_ip>:<pd_port>/pd/api/v1/config/rules/group/tiflash
```

```
[
  {
    "group_id": "tiflash",
    "id": "table-45-r",
    "override": true,
```

```
    "start_key": "7480000000000000FF2D5F720000000000FA",
    "end_key": "7480000000000000FF2E00000000000000F8",
    "role": "learner",
    "count": 1,
    "label_constraints": [
      {
        "key": "engine",
        "op": "in",
        "values": [
          "tiflash"
        ]
      }
    ]
  }
]
```

2. Remove all data replication rules related to TiFlash. Take the rule whose `id` is `table-45-r` as an example. Delete it by the following command:

```
curl -v -X DELETE http://<pd_ip>:<pd_port>/pd/api/v1/config/rule/tiflash/table-45-r
```

3. View the cluster status:

```
tiup cluster display <cluster-name>
```

   Access the monitoring platform at http://10.0.1.5:3000 using your browser, and view the status of the cluster and the new nodes.

After the scale-out, the cluster topology is as follows:

Host IP	Service
10.0.1.3	TiDB + TiFlash + TiCDC
10.0.1.4	TiDB + PD + TiCDC **(TiFlash is deleted)**
10.0.1.5	TiDB+ Monitor
10.0.1.1	TiKV
10.0.1.2	TiKV

8.3.1.6 Scale in a TiCDC cluster

This section exemplifies how to remove the TiCDC node from the `10.0.1.4` host.

1. Take the node offline:

```
tiup cluster scale-in <cluster-name> --node 10.0.1.4:8300
```

2. View the cluster status:

```
tiup cluster display <cluster-name>
```

Access the monitoring platform at http://10.0.1.5:3000 using your browser, and view the status of the cluster.

The current topology is as follows:

Host IP	Service
10.0.1.3	TiDB + TiFlash + TiCDC
10.0.1.4	TiDB + PD + **(TiCDC is deleted)**
10.0.1.5	TiDB + Monitor
10.0.1.1	TiKV
10.0.1.2	TiKV

8.3.2 Use TiDB Operator

8.4 Backup and Restore

8.4.1 Use BR to Back Up Cluster Data

This document describes how to back up TiDB cluster data in the following scenarios:

- Back up TiDB cluster snapshots
- Back up a database
- Back up a table
- Back up multiple tables with table filter
- Back up data to external storage
- Back up incremental data
- Encrypt backup data

If you are not familiar with the backup and restore tools, it is recommended that you read the following documents to fully understand usage principles and methods of these tools:

- BR Overview
- Use BR Command-line for Backup and Restoration

If you need to back up a small amount of data (for example, less than 50 GB) and do not require high backup speed, you can use Dumpling to export data to implement backup. For detailed backup operations, see Use Dumpling to back up full data.

8.4.1.1 Back up TiDB cluster snapshots

A snapshot of a TiDB cluster contains only the latest and transactionally consistent data at a specific time. You can back up the latest or specified snapshot data of a TiDB cluster by running the `br backup full` command. To get help on this command, run the `br backup full --help` command.

Example: Back up the snapshot generated at `2022-01-30 07:42:23` to the `2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup full \
    --pd "${PDIP}:2379" \
    --backupts '2022-01-30 07:42:23' \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file backupfull.log
```

In the preceding command:

- `--backupts`: The physical time of the snapshot. If data of this snapshot is processed by Garbage Collection (GC), the `br backup` command will exit with an error. If you leave this parameter unspecified, BR picks the snapshot corresponding to the backup start time.
- `--ratelimit`: The maximum speed **per TiKV** performing backup tasks (in MiB/s).
- `--log-file`: The target file for BR logging.

During backup, a progress bar is displayed in the terminal, as shown below. When the progress bar advances to 100%, the backup is complete.

```
br backup full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file backupfull.log
Full Backup <---------/.....................................................>
    ↪ 17.12%.
```

After the backup is completed, BR compares the checksum of the backup data with the admin checksum table of the cluster to ensure data correctness and security.

8.4.1.2 Back up a database or a table

BR supports backing up partial data of a specified database or table from a cluster snapshot or incremental data backup. This feature allows you to filter out unwanted data from snapshot backup and incremental data backup, and back up only business-critical data.

8.4.1.2.1 Back up a database

To back up a database in a cluster, run the `br backup db` command. To get help on this command, run the `br backup db --help` command.

Example: Back up the `test` database to the `db-test/2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup db \
    --pd "${PDIP}:2379" \
    --db test \
    --storage "s3://backup-data/db-test/2022-01-30/" \
    --ratelimit 128 \
    --log-file backuptable.log
```

In the preceding command, `--db` specifies the database name, and other parameters are the same as those in Back up TiDB cluster snapshots.

8.4.1.2.2 Back up a table

To back up a table in a cluster, run the `br backup table` command. To get help on this command, run the `br backup table --help` command.

Example: Back up `test.usertable` to the `table-db-usertable/2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup table \
    --pd "${PDIP}:2379" \
    --db test \
    --table usertable \
    --storage "s3://backup-data/table-db-usertable/2022-01-30/" \
    --ratelimit 128 \
    --log-file backuptable.log
```

In the preceding command, `--db` and `--table` specify the database name and table name respectively, and other parameters are the same as those in Back up TiDB cluster snapshots.

8.4.1.2.3 Back up multiple tables with table filter

To back up multiple tables with more criteria, run the `br backup full` command and specify the table filters with `--filter` or `-f`.

Example: Back up `db*.tbl*` data of a table to the `table-filter/2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup full \
    --pd "${PDIP}:2379" \
    --filter 'db*.tbl*' \
    --storage "s3://backup-data/table-filter/2022-01-30/" \
```

```
    --ratelimit 128 \
    --log-file backupfull.log
```

8.4.1.3 Back up data to external storage

BR supports backing up data to Amazon S3, Google Cloud Storage (GCS), Azure Blob Storage, NFS, or other S3-compatible file storage services. For details, see the following documents:

- Back up data on Amazon S3 using BR
- Back up data on Google Cloud Storage using BR
- Back up data on Azure Blob Storage using BR

8.4.1.4 Back up incremental data

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

Incremental data of a TiDB cluster is differentiated data between the snapshot of a starting point and that of an end point. Compared with snapshot data, incremental data is smaller and therefore it is a supplementary to snapshot backup, which reduces the volume of backup data.

To back up incremental data, run the `br backup` command with **the last backup timestamp** `--lastbackupts` specified. To get `--lastbackupts`, run the `validate` command. The following is an example:

```
LAST_BACKUP_TS=`br validate decode --field="end-version" -s s3://backup-data
    ↪ /2022-01-30/ | tail -n1`
```

> **Note:**
>
> - You need to save the incremental backup data under a different path from the previous snapshot backup.
> - GC safepoint must be prior to `lastbackupts`. The defalt GC lifetime is 10 minutes in TiDB, which means that TiDB only backs up incremental data generated in the last 10 minutes. To back up earlier incremental data, you need to adjust TiDB GC Lifetime setting.

```
br backup full\
    --pd ${PDIP}:2379 \
    --ratelimit 128 \
    --storage "s3://backup-data/2022-01-30/incr" \
    --lastbackupts ${LAST_BACKUP_TS}
```

The preceding command backs up the incremental data between `(LAST_BACKUP_TS, current PD timestamp]` and the DDLs generated during this time period. When restoring incremental data, BR restores all DDLs first, and then restores data.

8.4.1.5 Encrypt backup data

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

BR supports encrypting backup data at the backup end and at the storage end when backing up to Amazon S3. You can choose either encryption method as required.

8.4.1.5.1 Encrypt backup data at the backup end

Since TiDB v5.3.0, you can encrypt backup data by configuring the following parameters:

- `--crypter.method`: Encryption algorithm, which can be `aes128-ctr`, `aes192-ctr`, or `aes256-ctr`. The default value is `plaintext`, indicating that data is not encrypted.
- `--crypter.key`: Encryption key in hexadecimal string format. It is a 128-bit (16 bytes) key for the algorithm `aes128-ctr`, 24-byte key for the algorithm `aes192-ctr`, and 32-byte key for the algorithm `aes256-ctr`.
- `--crypter.key-file`: The key file. You can directly pass in the file path where the key is stored as a parameter without passing in "crypter.key".

Example: Encrypt backup data at the backup end.

```
br backup full\
    --pd ${PDIP}:2379 \
    --storage "s3://backup-data/2022-01-30/" \
    --crypter.method aes128-ctr \
    --crypter.key 0123456789abcdef0123456789abcdef
```

Note:

- If the key is lost, the backup data cannot be restored to the cluster.
- The encryption feature needs to be used on BR tools and TiDB clusters v5.3.0 or later versions. The encrypted backup data cannot be restored on clusters earlier than v5.3.0.

8.4.1.5.2 Encrypt backup data when backing up to Amazon S3

BR supports server-side encryption (SSE) when backing up data to S3. In this scenario, you can use AWS KMS keys you have created to encrypt data. For details, see BR S3 server-side encryption.

8.4.1.6 Validate backup data

After you back up data using BR, you can validate the backup data, including checking its integrity and viewing the metadata (such as TSO) by decoding the `backupmeta` file.

8.4.1.6.1 Check the integrity of backup data

To check the integrity of backup data, you can run the `tiup br debug checksum` command to calculate the checksum of the backup data.

Example: Calculate the checksum of the backup data in the `${prefix}` directory in the `backup-data` bucket on Amazon S3.

```
br debug checksum \
    --storage 's3://backup-data/${prefix}' \
    --s3.endpoint '${S3-endpoint-URL}' \
    --log-file checksum.log
```

8.4.1.6.2 Decode `backupmeta` to a readable JSON file

After a backup is complete, you can run the `tiup br debug decode` command to decode the `backupmeta` file into a readable JSON file, through which you can view the metadata (such as TSO) of the snapshot.

Example: Decode the `backupmeta` file in the `${prefix}` directory in the `backup-data` bucket on Amazon S3 into a JSON file `backupmeta.json`. The decoded file is stored in `s3://backup-data/${prefix}/backupmeta.json`.

```
br debug decode \
    --storage 's3://backup-data/${prefix}' \
    --s3.endpoint '${S3-endpoint-URL}' \
```

```
    --log-file decode-backupmeta.log
```

Open the `backupmeta.json` file and search for `end_version` to view the TSO of the snapshot.

If necessary, you can also encode the JSON format `backupmeta` file back to the original state. Specifically, run the `tiup br debug encode` command to generate the file named `backupmeta_from_json`.

Example: Encode the `backupmeta.json` file in the `${prefix}` directory in the `backup-data` bucket on Amazon S3 into a `backupmeta` file. The encoded file is stored in `s3://backup-data/${prefix}/backupmeta_from_json`.

```
br debug encode \
    --storage 's3://backup-data/${prefix}' \
    --s3.endpoint '${S3-endpoint-URL}' \
    --log-file encode-backupmeta.log
```

8.4.1.7 Backup performance and impact

The backup feature has some impact on cluster performance (transaction latency and QPS). However, you can mitigate the impact by adjusting the number of backup threads `backup.num-threads` or by adding more clusters.

To illustrate the impact of backup, this document lists the test conclusions of several snapshot backup tests:

- (5.3.0 and earlier) When the backup threads of BR on a TiKV node takes up 75% of the total CPU of the node, the QPS is reduced by 30% of the original QPS.
- (5.4.0 and later) When there are no more than `8` threads of BR on a TiKV node and the cluster's total CPU utilization does not exceed 80%, the impact of BR tasks on the cluster (write and read) is 20% at most.
- (5.4.0 and later) When there are no more than `8` threads of BR on a TiKV node and the cluster's total CPU utilization does not exceed 75%, the impact of BR tasks on the cluster (write and read) is 10% at most.
- (5.4.0 and later) When there are no more than `8` threads of BR on a TiKV node and the cluster's total CPU utilization does not exceed 60%, BR tasks has little impact on the cluster (write and read).

You can mitigate impact on cluster performance by reducing the number of backup threads. However, this might cause backup performance to deteriorate. Based on the preceding test results: (On a single TiKV node) the backup speed is proportional to the number of backup threads. When the number of threads is small, the backup speed is about 20 MB/thread. For example, a single node with 5 backup threads can deliver a backup speed of 100 MB/s.

Note:

The impact and speed of backup depends much on cluster configuration, deployment, and running services. The preceding test conclusions, based on simulation tests in many scenarios and verified in some customer sites, are worthy of reference. However, the exact impact and performance cap may vary depending on the scenarios. Therefore, you should always run the test and verify the test results.

Since v5.3.0, BR introduces the auto tunning feature (enabled by default) to adjust the number of backup threads. It can maintain the CPU utilization of the cluster below 80% during backup tasks. For details, see BR Auto-Tune.

8.4.2 Use BR to Restore Cluster Data

This document describes how to restore TiDB cluster data in the following scenarios:

- Restore TiDB cluster snapshots
- Restore a database
- Restore a table
- Restore multiple tables with table filter
- Restore backup data from external storage
- Restore incremental data
- Restore encrypted backup data
- Restore tables in the `mysql` schema

If you are not familiar with backup and restore tools, it is recommended that you read the following documents to fully understand usage principles and methods of these tools:

- BR Overview
- Use BR Command-line for Backup and Restoration

If you need to restore data exported by Dumpling, CSV files, or Apache Parquet files generated by Amazon Aurora, you can use TiDB Lightning to import data to implement restore. For details, see Use TiDB Lightning to restore full data.

8.4.2.1 Restore TiDB cluster snapshots

BR supports restoring snapshot backup on an empty cluster to restore the target cluster to the latest state when the snapshot is backed up.

Example: Restore the snapshot generated at `2022-01-30 07:42:23` from the `2022-01-30/` directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file restorefull.log
```

In the preceding command,

- `--ratelimit`: The maximum speed for **each TiKV** to perform a restoration task (unit: MiB/s)
- `--log-file` The target file for BR logging

During restoration, a progress bar is displayed in the terminal, as shown below. When the progress bar advances to 100%, the restoration is complete. To ensure data security, BR performs a check on the restored data.

```
br restore full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file restorefull.log
Full Restore <---------/.....................................................>
    ↪ 17.12%.
```

8.4.2.2 Restore a database or a table

BR supports restoring partial data of a specified database or table from backup data. This feature allows you to filter out unwanted data and back up only a specific database or table.

8.4.2.2.1 Restore a database

To restore a database to the cluster, run the `br restore db` command. To get help on this command, run the `br restore db --help` command.

Example: Restore the `test` database from the `db-test/2022-01-30/` directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore db \
    --pd "${PDIP}:2379" \
    --db "test" \
    --ratelimit 128 \
```

```
    --storage "s3://backup-data/db-test/2022-01-30/" \
    --log-file restore_db.log
```

In the preceding command, `--db` specifies the name of the database to be restored, and other parameters are the same as those in Restore TiDB cluster snapshots.

> **Note:**
>
> When you restore the backup data, the database name specified by `--db` must be the same as the one specified by `-- db` in the backup command. Otherwise, the restoration fails. This is because the metafile of the backup data (`backupmeta` file) records the database name, and you can only restore data to the database with the same name. The recommended method is to restore the backup data to the database with the same name in another cluster.

8.4.2.2.2 Restore a table

To restore a single table to the cluster, run the `br restore table` command. To get help on this command, run the `br restore table --help` command.

Example: Restore `test.usertable` from the `table-db-usertable/2022-01-30/` ↪ directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore table \
    --pd "${PDIP}:2379" \
    --db "test" \
    --table "usertable" \
    --ratelimit 128 \
    --storage "s3://backup-data/table-db-usertable/2022-01-30/" \
    --log-file restore_table.log
```

In the preceding command, `--table` specifies the name of the table to be restored, and other parameters are the same as those in Restore TiDB cluster snapshots.

8.4.2.2.3 Restore multiple tables with table filter

To restore multiple tables with more criteria, run the `br restore full` command and specify the table filters with `--filter` or `-f`.

Example: Restore data matching the `db*.tbl*` table from the `table-filter` ↪ `/2022-01-30/` directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore full \
    --pd "${PDIP}:2379" \
    --filter 'db*.tbl*' \
    --storage "s3://backup-data/table-filter/2022-01-30/" \
    --log-file restorefull.log
```

8.4.2.3 Restore backup data from external storage

BR supports restoring data to Amazon S3, Google Cloud Storage (GCS), Azure Blob Storage, NFS, or other S3-compatible file storage services. For details, see the following documents:

- Restore data on Amazon S3 using BR
- Restore data on Google Cloud Storage using BR
- Restore data on Azure Blob Storage using BR

8.4.2.4 Restore incremental data

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

Restoring incremental data is similar to restoring full data using BR. When restoring incremental data, make sure that all the data backed up before `last backup ts` has been restored to the target cluster. Also, because incremental restoration updates ts data, you need to ensure that there are no other writes during the restoration. Otherwise, conflicts might occur.

```
br restore full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/incr" \
    --ratelimit 128 \
    --log-file restorefull.log
```

8.4.2.5 Restore encrypted backup data

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

After encrypting the backup data, you need to pass in the corresponding decryption parameters to restore the data. Ensure that the decryption algorithm and key are correct. If the decryption algorithm or key is incorrect, the data cannot be restored.

```
br restore full\
    --pd ${PDIP}:2379 \
    --storage "s3://backup-data/2022-01-30/" \
    --crypter.method aes128-ctr \
    --crypter.key 0123456789abcdef0123456789abcdef
```

8.4.2.6 Restore tables in the `mysql` schema

Starting from BR v5.1.0, when you perform a full backup, BR backs up the **system tables**. Before BR v6.2.0, under default configuration, BR only restores user data, but does not restore data in the system tables. Starting from BR v6.2.0, if the backup data contains system tables, and if you configure `--with-sys-table`, BR restores **data in some system tables**.

BR can restore data in **the following system tables**:

```
+---------------------------------+
| mysql.columns_priv              |
| mysql.db                        |
| mysql.default_roles             |
| mysql.global_grants             |
| mysql.global_priv               |
| mysql.role_edges                |
| mysql.tables_priv               |
| mysql.user                      |
+---------------------------------+
```

BR does not restore the following system tables:

- Statistics tables (`mysql.stat_*`)
- System variable tables (`mysql.tidb`, `mysql.global_variables`)
- Other system tables

When you restore data related to system privileges, note that before BR restores data, it checks whether the system tables in the target cluster are compatible with those in the backup data. "Compatible" means that all the following conditions are met:

- The target cluster has the same system tables as the backup data.
- The **number of columns** in the system privilege table of the target cluster is consistent with that of the backup data. The order of the columns can be different.
- The columns in the system privilege table of the target cluster are compatible with those in the backup data. If the data type of the column is a type with length (for example, int or char), the length in the target cluster must be >= the length in the backup data. If the data type of the column is an enum type, the enum values in the target cluster must be a superset of the enum values in the backup data.

If the target cluster is not empty or the target cluster is not compatible with the backup data, BR returns the following information. You can remove `--with-sys-table` to skip restoring system tables.

```
   ##########################################################################
↪  # the target cluster is not compatible with the backup data, ### br
↪ cannot restore system tables. ### you can remove 'with-sys-table' flag
↪ to skip restoring system tables ######################################################
↪
```

To restore a table created by the user in the `mysql` schema (not system tables), you can explicitly include the table using table filters. The following example shows how to restore the `mysql.usertable` table when BR performs a normal restoration.

```
br restore full -f '*.*' -f '!mysql.*' -f 'mysql.usertable' -s
   ↪ $external_storage_url --with-sys-table
```

In the preceding command,

- `-f '*.*'` is used to override the default rules
- `-f '!mysql.*'` instructs BR not to restore tables in `mysql` unless otherwise stated.
- `-f 'mysql.usertable'` indicates that `mysql.usertable` should be restored.

If you only need to restore `mysql.usertable`, run the following command:

```
br restore full -f 'mysql.usertable' -s $external_storage_url --with-sys-
   ↪ table
```

8.4.2.7 Restoration performance and impact

- TiDB fully uses TiKV CPU, disk IO, network bandwidth, and other resources when restoring data. Therefore, it is recommended that you restore backup data on an empty cluster to avoid affecting running services.
- The restoration speed depends much on cluser configuration, deployment, and running services. Generally, the restoration speed can reach 100 MB/s (per TiKV node).

Note:

The preceding test conclusions, based on simulation tests in many scenarios and verified in some customer sites, are worthy of reference. However, the restoration speed may vary depending on the scenarios. Therefore, you should always run the test and verify the test results.

8.4.3 BR Use Cases

Backup & Restore (BR) is a tool for distributed backup and restoration of the TiDB cluster data.

This document describes common backup and restoration scenarios:

- Back up a single table to a network disk (recommended for production environments)
- Restore data from a network disk (recommended for production environments)
- Back up a single table to a local disk
- Restore data from a local disk

This document aims to help you achieve the following goals:

- Back up and restore data using a network disk or local disk correctly.
- Get the status of a backup or restoration operation through monitoring metrics.
- Learn how to tune performance during the backup or restoration operation.
- Troubleshoot the possible anomalies during the backup operation.

8.4.3.1 Audience

You are expected to have a basic understanding of TiDB and TiKV.

Before reading on, make sure you have read BR Overview, especially Usage Restrictions and Some tips.

8.4.3.2 Prerequisites

This section introduces the recommended method of deploying TiDB, cluster versions, the hardware information of the TiKV cluster, and the cluster configuration for the use case demonstrations.

You can estimate the performance of your backup or restoration operation based on your own hardware and configuration. It is recommended that you use a network disk to back up and restore data. This spares you from collecting backup files and greatly improves the backup efficiency especially when the TiKV cluster is in a large scale.

8.4.3.2.1 Deployment method

It is recommended that you deploy the TiDB cluster using TiUP and install BR using TiUP.

8.4.3.2.2 Cluster versions

- TiDB: v6.2.0
- TiKV: v6.2.0
- PD: v6.2.0
- BR: v6.2.0

> **Note:**
>
> It is recommended that you use the latest version of TiDB/TiKV/PD/BR and make sure that the BR version is **consistent with** the TiDB version.

8.4.3.2.3 TiKV hardware information

- Operating system: CentOS Linux release 7.6.1810 (Core)
- CPU: 16-Core Common KVM processor
- RAM: 32 GB
- Disk: 500 GB SSD * 2
- NIC: 10 Gigabit network card

8.4.3.2.4 Cluster configuration

BR directly sends commands to the TiKV cluster and are not dependent on the TiDB server, so you do not need to configure the TiDB server when using BR.

- TiKV: default configuration
- PD: default configuration

8.4.3.2.5 Others

In addition to the preceding prerequisites, you should also perform the following checks before performing the backup and restoration.

Check before backup

Before running the `br backup` command, make sure the following conditions are met:

- No DDL statements are running on the TiDB cluster.

- The target storage device has required space (no less than 1/3 of the disk space of the backup cluster).

Check before restoration

Before running the `br restore` command, check the target cluster to ensure that the table in this cluster does not have a duplicate name.

8.4.3.3 Back up a single table to a network disk (recommended for production environments)

Run the `br backup` command to back up the single table data `--db batchmark --table order_line` to the specified path `local:///br_data` in the network disk.

8.4.3.3.1 Backup prerequisites

- Check before backup
- Configure a high-performance SSD hard disk host as the NFS server to store data, and all BR nodes, TiKV nodes, and TiFlash nodes as NFS clients. Mount the same path (for example, `/br_data`) to the NFS server for NFS clients to access the server.
- The total transfer rate between the NFS server and all NFS clients must reach at least `the number of TiKV instances * 150MB/s`. Otherwise, the network I/O might become the performance bottleneck.

Note:

- During data backup, because only the data of leader replicas are backed up, even if there is a TiFlash replica in the cluster, BR can complete the backup without mounting TiFlash nodes.
- When restoring data, BR will restore the data of all replicas. Also, TiFlash nodes need access to the backup data for BR to complete the restore. Therefore, before the restore, you must mount TiFlash nodes to the NFS server.

8.4.3.3.2 Topology

The following diagram shows the typology of BR:

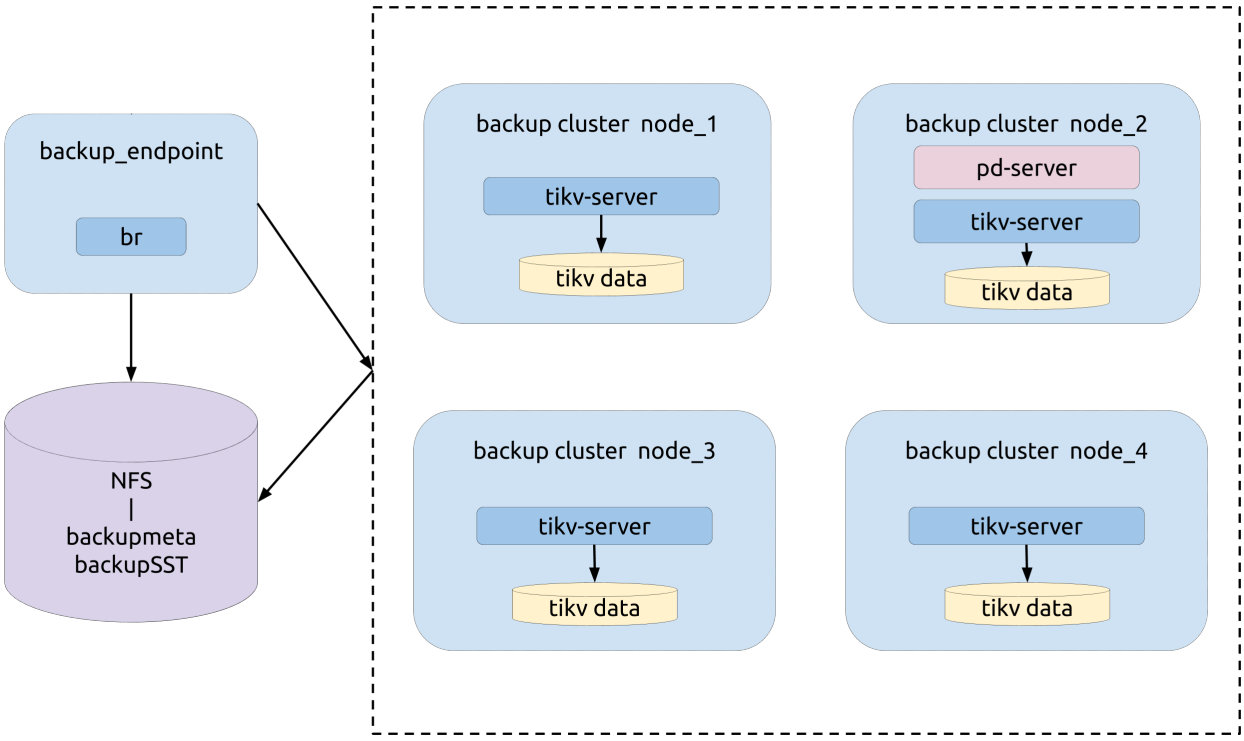


Figure 64: img

8.4.3.3.3 Backup operation

Run the `br backup` command:

```
bin/br backup table \
    --db batchmark \
    --table order_line \
    -s local:///br_data \
    --pd ${PD_ADDR}:2379 \
    --log-file backup-nfs.log
```

8.4.3.3.4 Monitoring metrics for the backup

During the backup process, pay attention to the following metrics on the monitoring panels to get the status of the backup process.

Backup CPU Utilization: the CPU usage rate of each working TiKV node in the backup operation (for example, backup-worker and backup-endpoint).

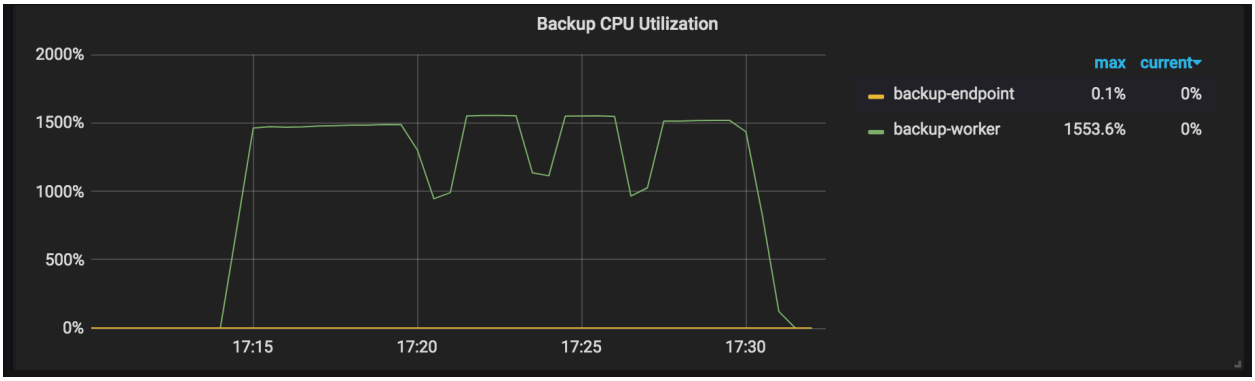


Figure 65: img

IO Utilization: the I/O usage rate of each working TiKV node in the backup operation.

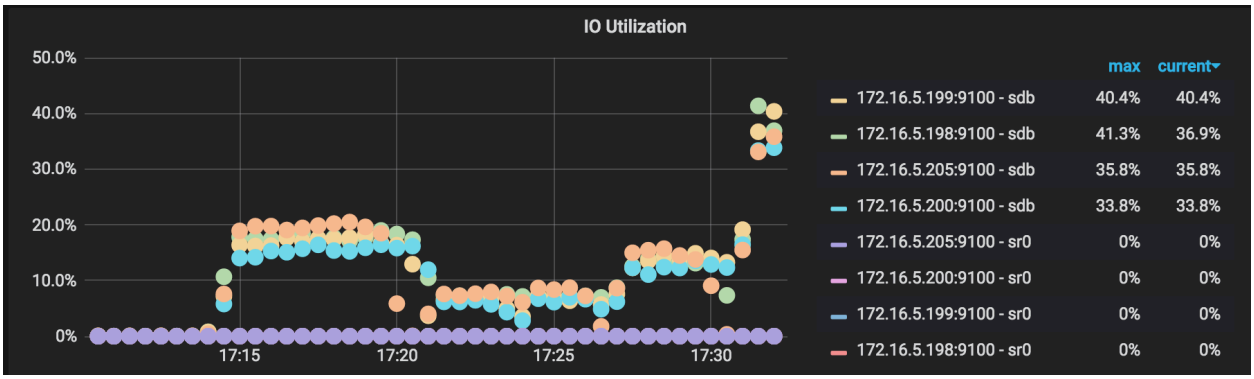


Figure 66: img

BackupSST Generation Throughput: the backupSST generation throughput of each working TiKV node in the backup operation, which is normally around 150 MB/s.

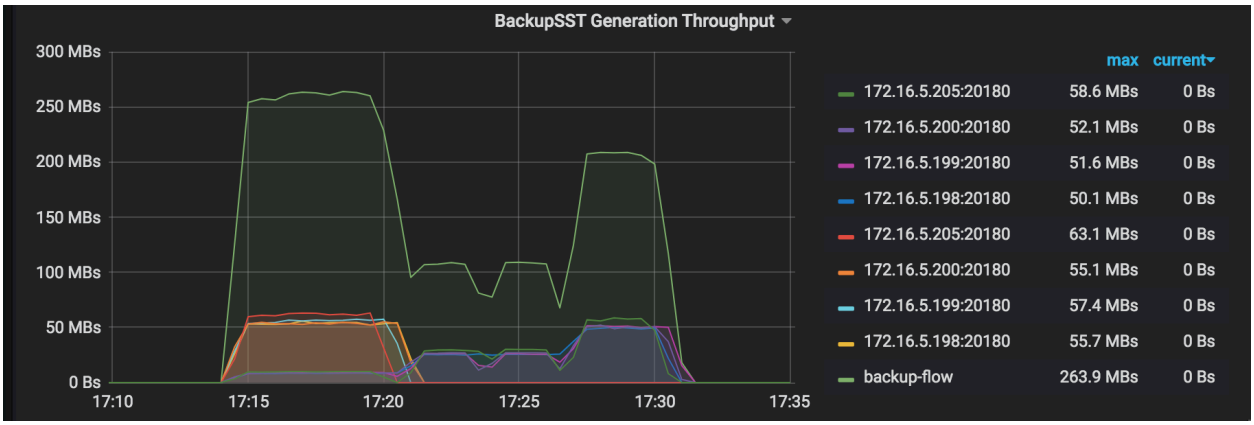


Figure 67: img

One Backup Range Duration: the duration of backing up a range, which is the total time cost of scanning KVs and storing the range as the backupSST file.

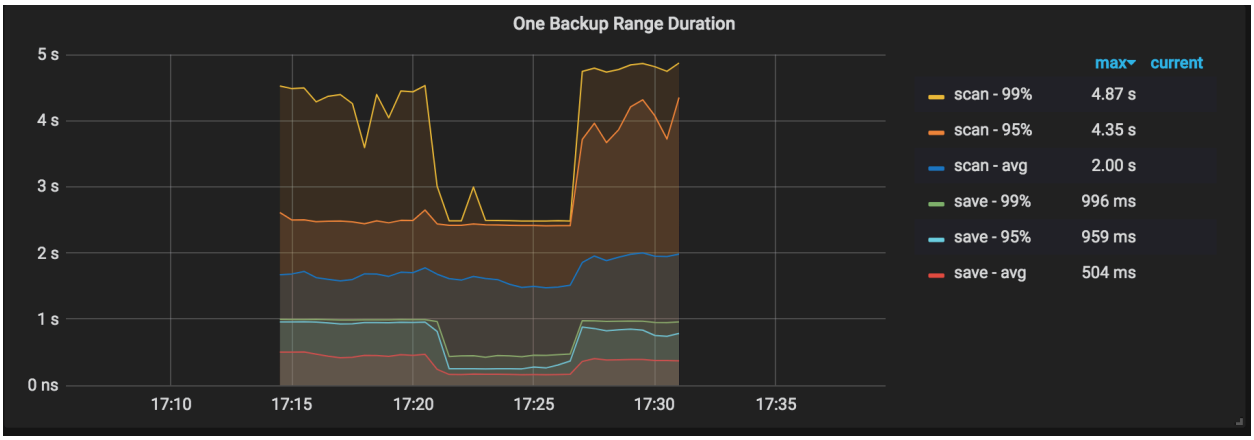


Figure 68: img

One Backup Subtask Duration: the duration of each sub-task into which a backup task is divided.

Note:

- In this task, the single table to be backed up has three indexes and the task is normally divided into four sub-tasks.
- The panel in the following image has 20 points on it, 10 blue and 10 yellow, indicating that there are 10 sub-tasks. Region scheduling might occur during the backup process, so a few retries is normal.

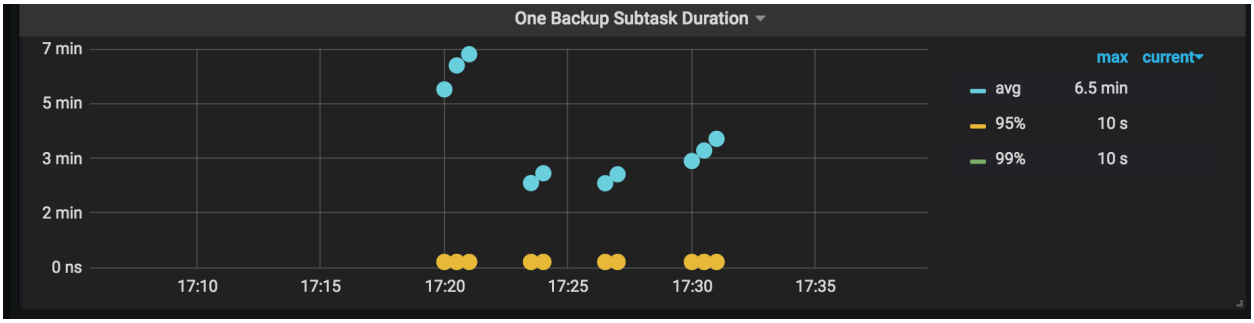


Figure 69: img

Backup Errors: the errors occurred during the backup process. No error occurs in normal situations. Even if a few errors occur, the backup operation has the retry mechanism which might increase the backup time but does not affect the operation correctness.

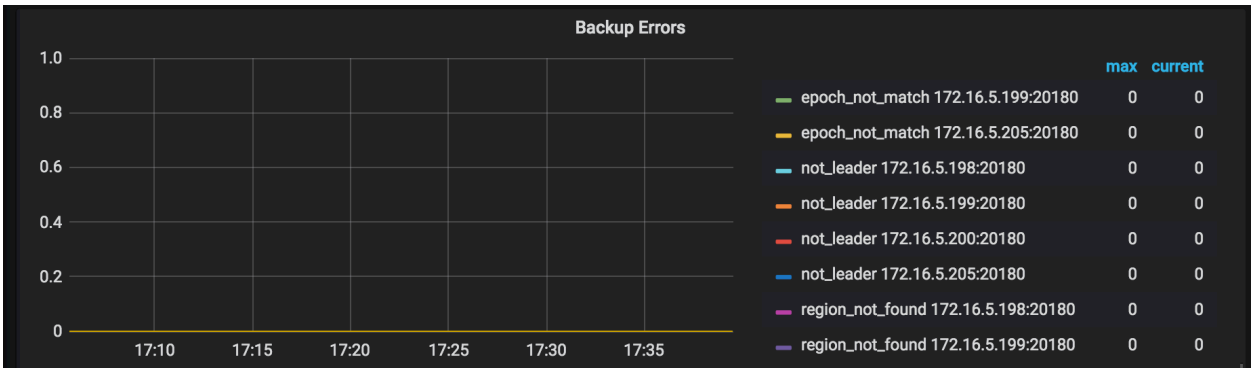


Figure 70: img

Checksum Request Duration: the duration of the admin checksum request in the backup cluster.

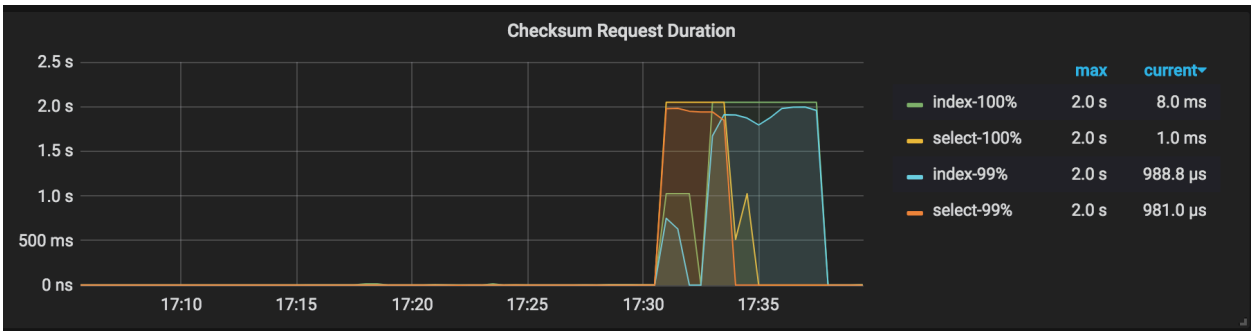


Figure 71: img

8.4.3.3.5 Backup results explanation

When finishing the backup, BR outputs the backup summary to the console.

In the log specified before running the backup command, you can get the statistical information of the backup operation from this log. Search "summary" in this log, you can see the following information:

```
["Full backup Success summary:
    total backup ranges: 2,
    total success: 2,
    total failed: 0,
    total take(Full backup time): 31.802912166s,
    total take(real time): 49.799662427s,
    total size(MB): 5997.49,
    avg speed(MB/s): 188.58,
    total kv: 120000000"]
    ["backup checksum"=17.907153678s]
    ["backup fast checksum"=349.333µs]
    ["backup total regions"=43]
    [BackupTS=422618409346269185]
    [Size=826765915]
```

The preceding log includes the following information:

- `total take(Full backup time)`: Backup duration
- `total take(real time)`: Total runtime of the application
- `total size(MB)`: The size of the backup data
- `avg speed(MB/s)`: Backup throughput
- `total kv`: The number of backed-up KV pairs
- `backup checksum`: Backup checksum duration
- `backup fast checksum`: The total duration of calculating the checksum, KV pairs, and bytes of each table
- `backup total regions`: The total number of backup Regions
- `BackupTS`: The snapshot timestamp of the backup data
- `Size`: The actual size of the backup data in the disk after compression

From the preceding information, the throughput of a single TiKV instance can be calculated: `avg speed(MB/s)/tikv_count` = `62.86`.

8.4.3.3.6 Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the backup process (for example, in the Monitoring metrics for the backup, the highest CPU usage rate of backup-worker is around `1500%` and the overall I/O usage rate is below `30%`), you can try to increase the value of `--concurrency` (`4` by default) to tune the performance. But this

performance tuning method is not suitable for the use cases of many small tables. See the following example:

```
bin/br backup table \
    --db batchmark \
    --table order_line \
    -s local:///br_data/ \
    --pd ${PD_ADDR}:2379 \
    --log-file backup-nfs.log \
    --concurrency 16
```

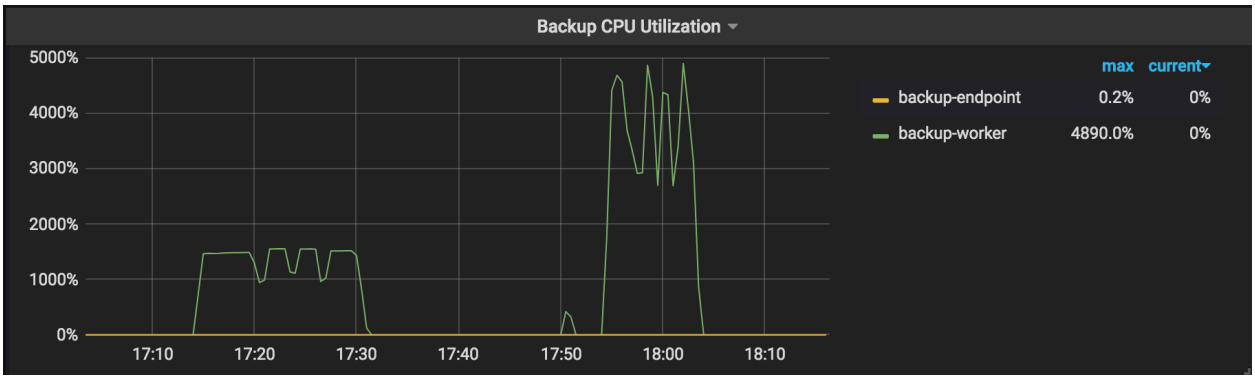


Figure 72: img

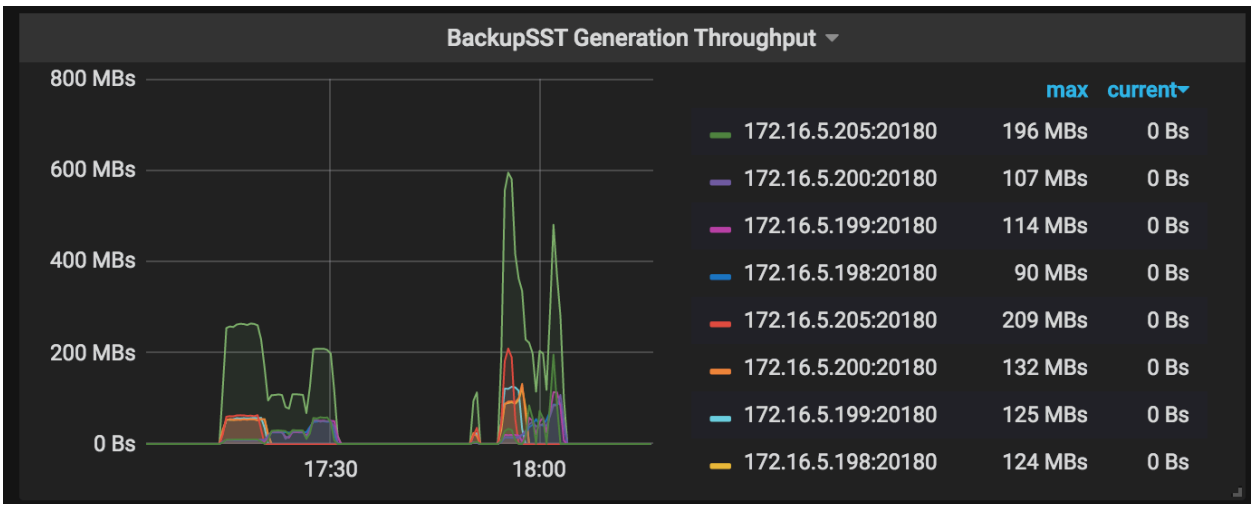


Figure 73: img

The tuned performance results are as follows (with the same data size):

- Backup duration (`total take(s)`): reduced from `986.43` to `535.53`

- Backup throughput (`avg speed(MB/s)`): increased from `358.09` to `659.59`
- Throughput of a single TiKV instance (`avg speed(MB/s)/tikv_count`): increased from `89` to `164.89`

8.4.3.4 Restore data from a network disk (recommended for production environments)

Use the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

8.4.3.4.1 Restoration prerequisites

- Check before restore

8.4.3.4.2 Topology

The following diagram shows the typology of BR:

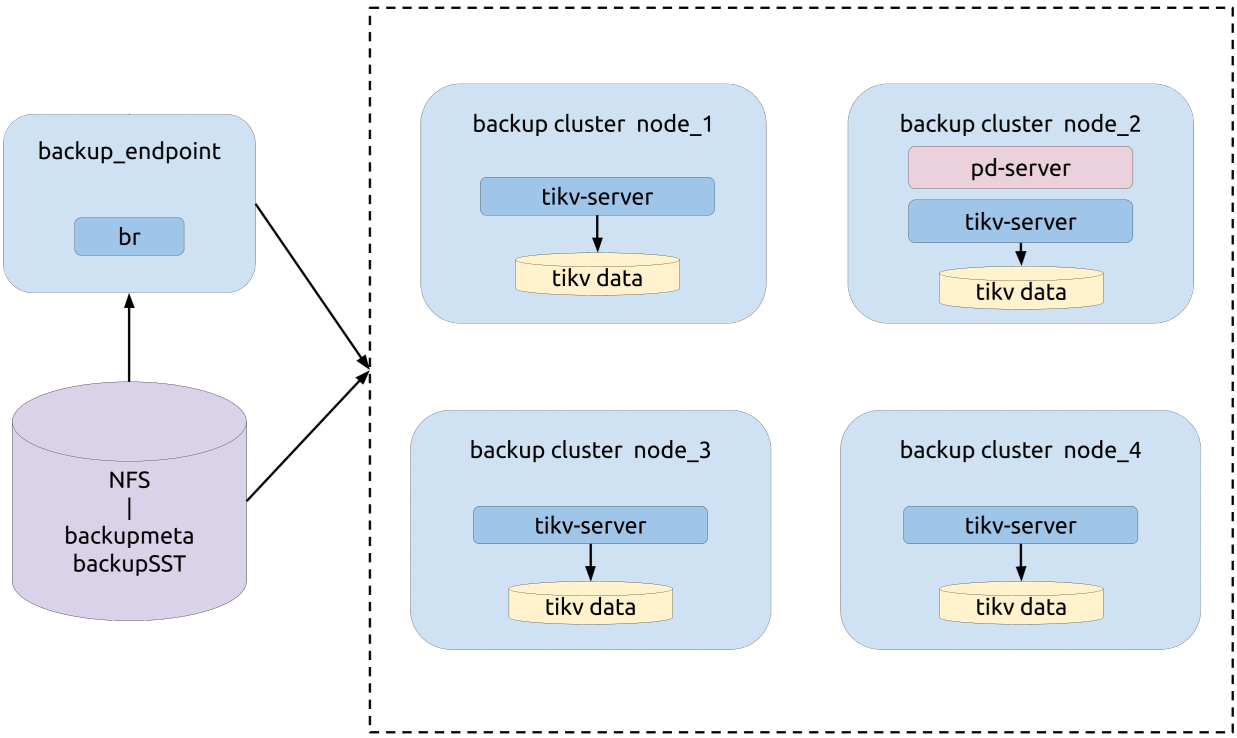


Figure 74: img

8.4.3.4.3 Restoration operation

Run the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data
    ↪ --pd 172.16.5.198:2379 --log-file restore-nfs.log
```

8.4.3.4.4 Monitoring metrics for the restoration

During the restoration process, pay attention to the following metrics on the monitoring panels to get the status of the restoration process.

CPU: the CPU usage rate of each working TiKV node in the restoration operation.

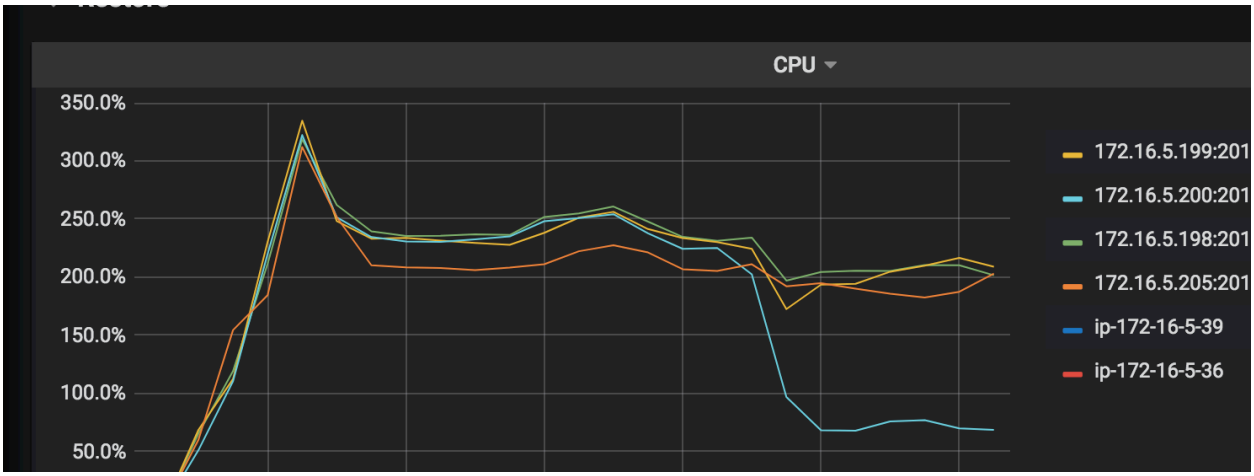


Figure 75: img

IO Utilization: the I/O usage rate of each working TiKV node in the restoration operation.

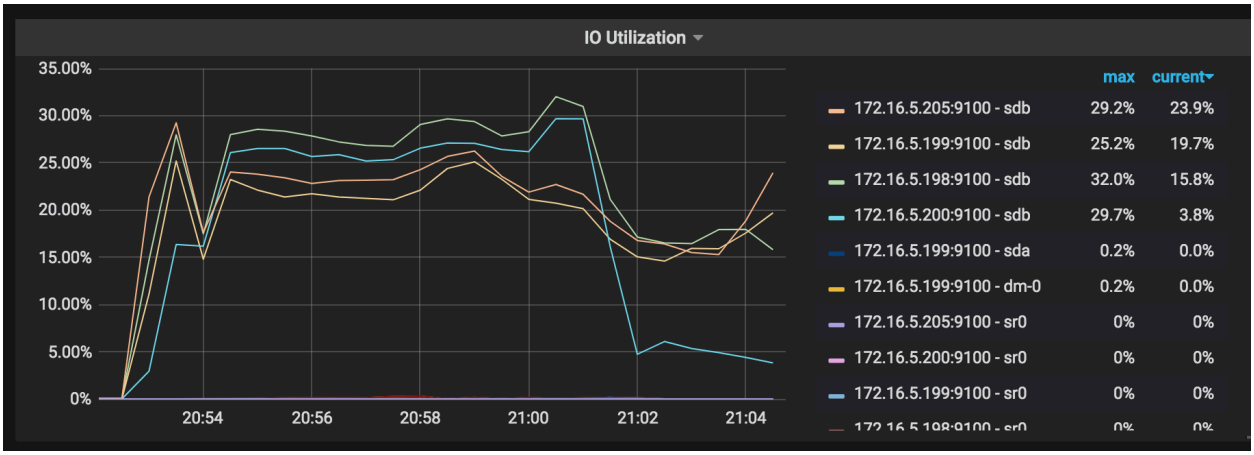


Figure 76: img

Region: the Region distribution. The more even Regions are distributed, the better the restoration resources are used.

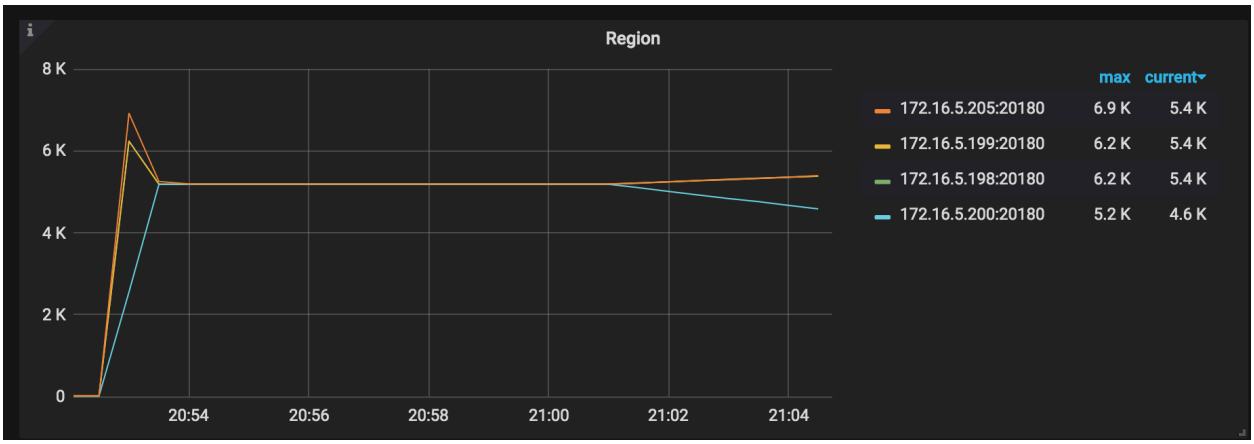


Figure 77: img

Process SST Duration: the delay of processing the SST files. When restoring a table, if `tableID` is changed, you need to rewrite `tableID`. Otherwise, `tableID` is renamed. Generally, the delay of rewriting is longer than that of renaming.

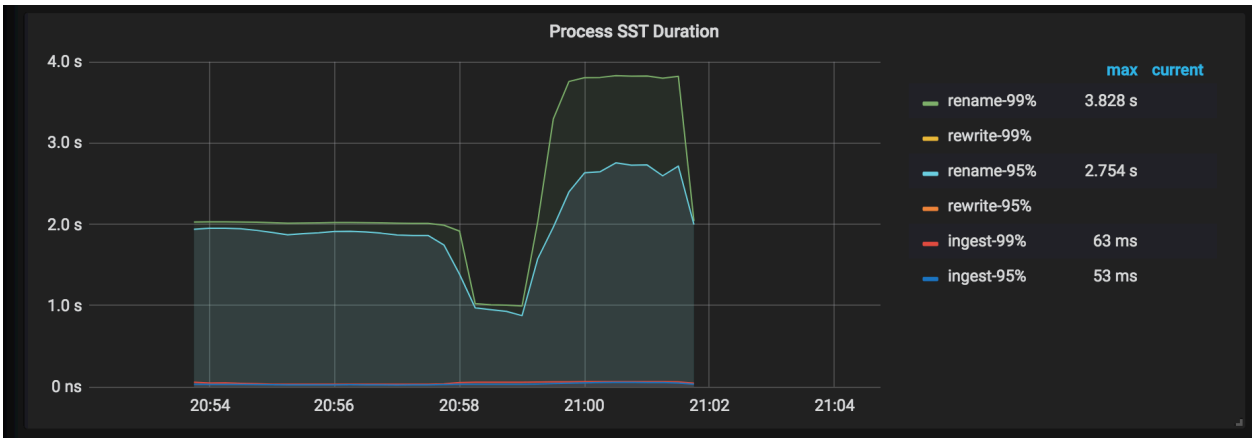


Figure 78: img

DownLoad SST Throughput: the throughput of downloading SST files from External Storage.

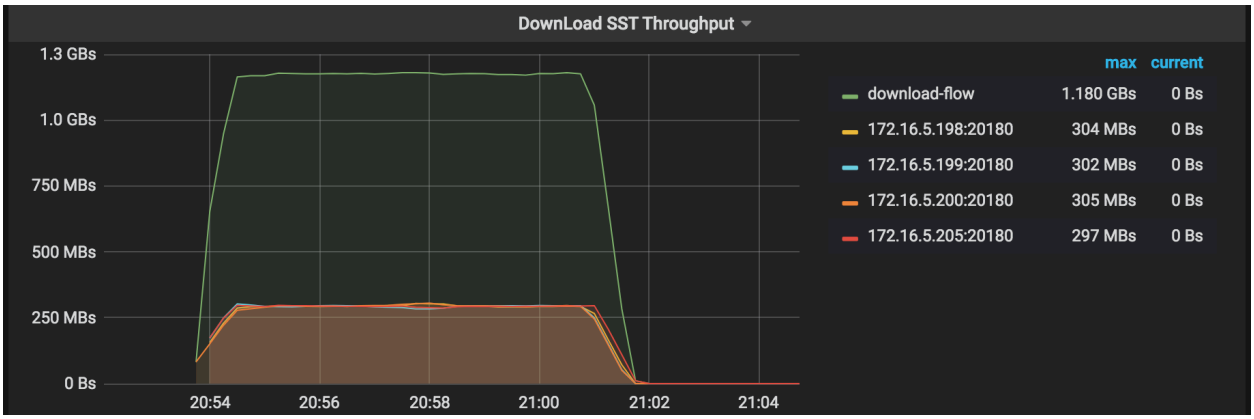


Figure 79: img

Restore Errors: the errors occurred during the restoration process.

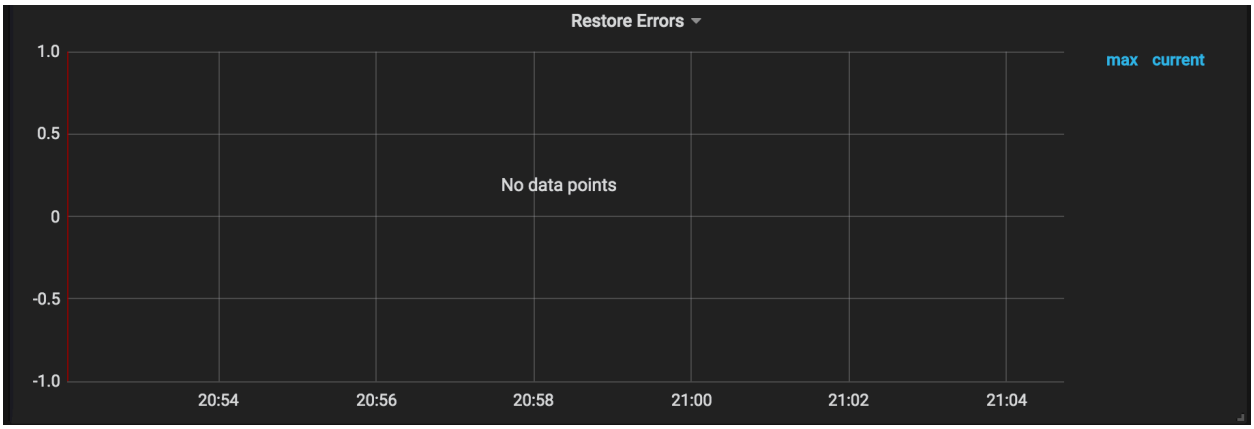


Figure 80: img

Checksum Request Duration: the duration of the admin checksum request. This duration for the restoration is longer than that for the backup.

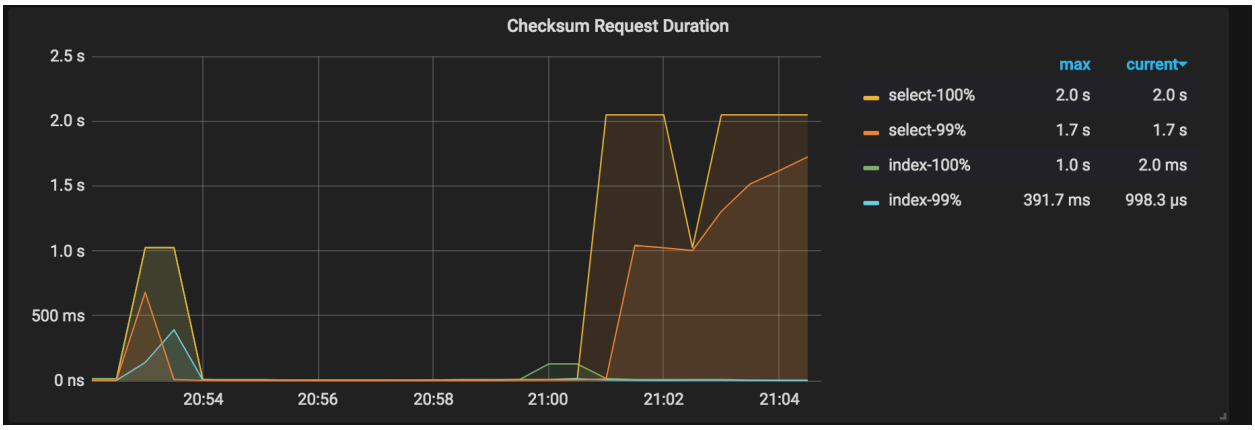


Figure 81: img

8.4.3.4.5 Restoration results explanation

In the log specified before running the restoration command, you can get the statistical information of the restoration operation from this log. Search "summary" in this log, you can see the following information:

```
["Table Restore summary:
    total restore tables: 1,
    total success: 1,
    total failed: 0,
    total take(Full restore time): 17m1.001611365s,
    total take(real time): 16m1.371611365s,
    total kv: 5659888624,
    total size(MB): 353227.18,
    avg speed(MB/s): 367.42"]
    ["restore files"=9263]
    ["restore ranges"=6888]
    ["split region"=49.049182743s]
    ["restore checksum"=6m34.879439498s]
    [Size=48693068713]
```

The preceding log includes the following information:

- `total take(Full restore time)`: The restoration duration
- `total take(real time)`: The total runtime of the application
- `total size(MB)`: The size of the data to be restored
- `total kv`: The number of restored KV pairs
- `avg speed(MB/s)`: The restoration throughput
- `split region`: The Region split duration
- `restore checksum`: The restoration checksum duration

- `Size`: The actual size of the restored data in the disk

From the preceding information, the following items can be calculated:

- The throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` = `91.8`
- The average restore speed of a single TiKV instance: `total size(MB)/(split time + restore time)/tikv_count` = `87.4`

Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the restore process, you can increase the value of `--concurrency` (defaults to `128`). See the following example:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data/
    ↪ --pd 172.16.5.198:2379 --log-file restore-concurrency.log --
    ↪ concurrency 1024
```

The tuned performance results are as follows (with the same data size):

- Restoration duration (`total take(s)`): reduced from `961.37` to `443.49`
- Restoration throughput (`avg speed(MB/s)`): increased from `367.42` to `796.47`
- Throughput of a single TiKV instance (`avg speed(MB/s)/tikv_count`): increased from `91.8` to `199.1`
- Average restore speed of a single TiKV instance (`total size(MB)/(split time + restore time)/tikv_count`): increased from `87.4` to `162.3`

8.4.3.5 Back up a single table to a local disk (recommended for testing environments)

Run the `br backup` command to back up a single table `--db batchmark --table` ↪ `order_line` to the specified path `local:///home/tidb/backup_local` in the local disk.

8.4.3.5.1 Backup prerequisites

- Check before backup
- Each TiKV node has a separate disk to store backupSST files.
- The `backup_endpoint` node has a separate disk to store `backupmeta` files.
- TiKV and the `backup_endpoint` node share the same directory (for example, `/home/` ↪ `tidb/backup_local`) for backup.

8.4.3.5.2 Topology

The following diagram shows the typology of BR:

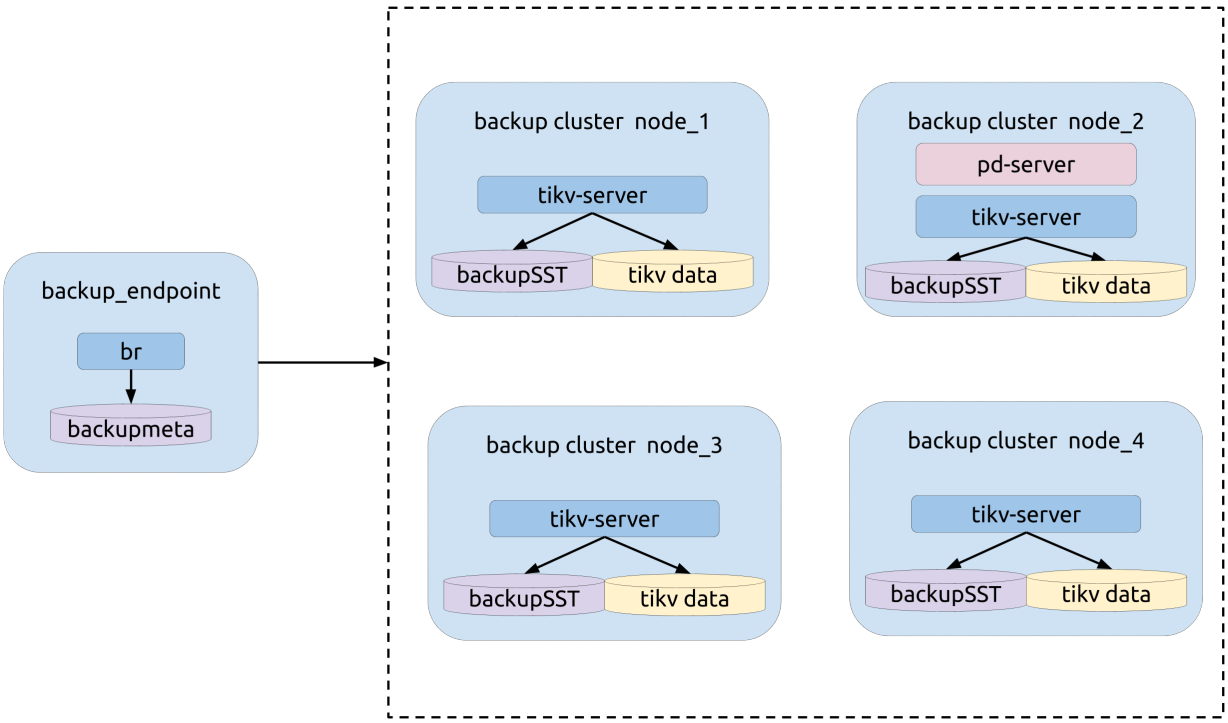


Figure 82: img

8.4.3.5.3 Backup operation

Run the `br backup` command:

```
bin/br backup table \
    --db batchmark \
    --table order_line \
    -s local:///home/tidb/backup_local/ \
    --pd ${PD_ADDR}:2379 \
    --log-file backup_local.log
```

During the backup process, pay attention to the metrics on the monitoring panels to get the status of the backup process. See Monitoring metrics for the backup for details.

Backup results explanation

In the log specified before running the backup command, you can get the statistical information of the restoration operation from this log. Search “summary” in this log, you can see the following information:

```
["Table backup summary:
```

```
total backup ranges: 4,
total success: 4,
total failed: 0,
total take(s): 551.31,
total kv: 5659888624,
total size(MB): 353227.18,
avg speed(MB/s): 640.71"]
["backup total regions"=6795]
["backup checksum"=6m33.962719217s]
["backup fast checksum"=22.995552ms]
```

The preceding log includes the following information:

- `total take(s)`: The backup duration
- `total size(MB)`: The data size
- `avg speed(MB/s)`: The backup throughput
- `backup checksum`: The backup checksum duration

From the preceding information, the throughput of a single TiKV instance can be calculated: `avg speed(MB/s)/tikv_count` = 160.

8.4.3.6 Restore data from a local disk (recommended for testing environments)

Run the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

8.4.3.6.1 Restoration prerequisites

- Check before restore
- The TiKV cluster and the backup data do not have a duplicate database or table. Currently, BR does not support table route.
- Each TiKV node has a separate disk to store backupSST files.
- The `restore_endpoint` node has a separate disk to store `backupmeta` files.
- TiKV and the `restore_endpoint` node share the same directory (for example, `/home/tidb/backup_local/`) for restoration.

Before the restoration, follow these steps:

1. Collect all backupSST files into the same directory.
2. Copy the collected backupSST files to all TiKV nodes of the cluster.
3. Copy the `backupmeta` files to the `restore endpoint` node.

8.4.3.6.2 Topology

The following diagram shows the typology of BR:

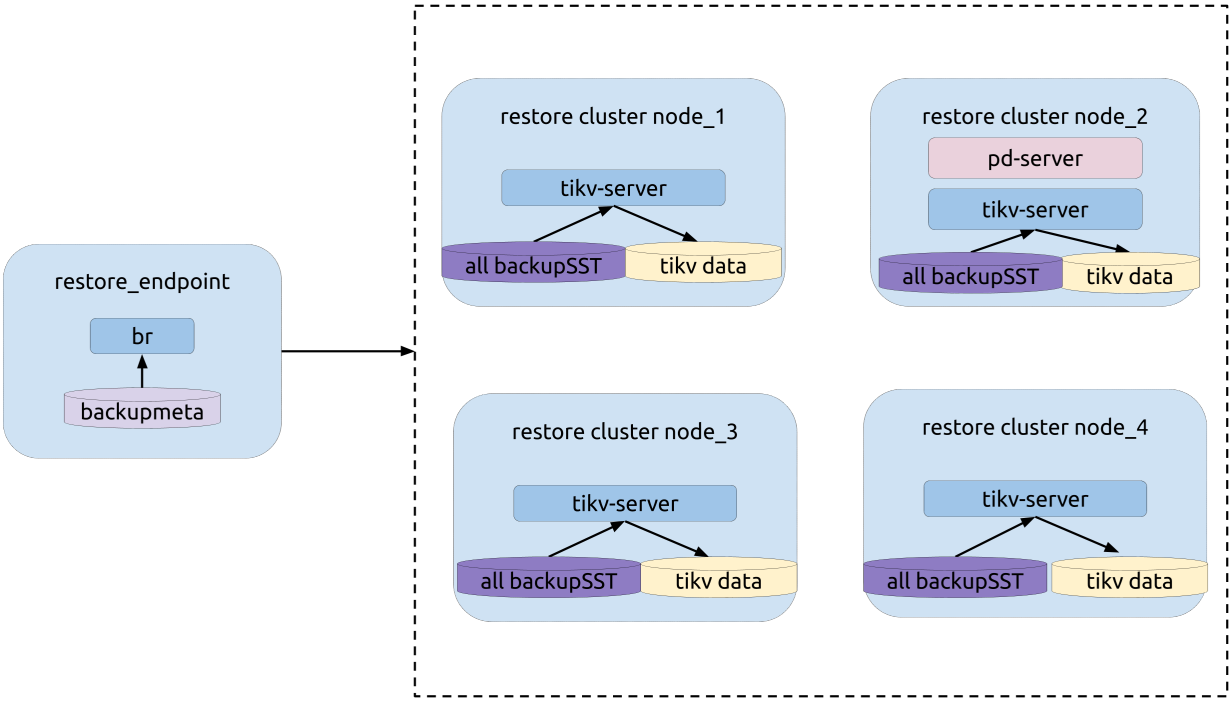


Figure 83: img

8.4.3.6.3 Restoration operation

Run the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///home/tidb
    ↪ /backup_local/ --pd 172.16.5.198:2379 --log-file restore_local.log
```

During the restoration process, pay attention to the metrics on the monitoring panels to get the status of the restoration process. See Monitoring metrics for the restoration for details.

8.4.3.6.4 Restoration results explanation

In the log specified before running the restoration command, you can get the statistical information of the restoration operation from this log. Search "summary" in this log, you can see the following information:

```
["Table Restore summary:
    total restore tables: 1,
    total success: 1,
    total failed: 0,
```

```
total take(s): 908.42,
total kv: 5659888624,
total size(MB): 353227.18,
avg speed(MB/s): 388.84"]
["restore files"=9263]
["restore ranges"=6888]
["split region"=58.7885518s]
["restore checksum"=6m19.349067937s]
```

The preceding log includes the following information:

- `total take(s)`: The restoration duration
- `total size(MB)`: The data size
- `avg speed(MB/s)`: The restoration throughput
- `split region`: The region split duration
- `restore checksum`: The restoration checksum duration

From the preceding information, the following items can be calculated:

- The throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` = 97.2
- The average restoration speed of a single TiKV instance: `total size(MB)/(split time + restore time)/tikv_count` = 92.4

8.4.3.7 Error handling during backup

This section introduces the common errors that might occur during the backup process.

8.4.3.7.1 `key locked Error` in the backup log

Error message in the log: `log - ["backup occur kv error"][error="{\"KvError \":{\"locked\":`

If a key is locked during the backup process, BR tries to resolve the lock. A small number of this error do not affect the correctness of the backup.

8.4.3.7.2 Backup failure

Error message in the log: `log - Error: msg:"Io(Custom { kind: AlreadyExists, error: \"[5_5359_42_123_default.sst] is already exists in /dir/backup_local /\" })"`

If the backup operation fails and the preceding message occurs, perform one of the following operations and then start the backup operation again:

- Change the directory for the backup. For example, change `/dir/backup-2020-01-01/` to `/dir/backup_local/`.
- Delete the backup directory of all TiKV nodes and BR nodes.

8.5 Time Zone Support

The time zone in TiDB is decided by the global `time_zone` system variable and the session `time_zone` system variable. The default value of `time_zone` is `SYSTEM`. The actual time zone corresponding to `System` is configured when the TiDB cluster bootstrap is initialized. The detailed logic is as follows:

- Prioritize the use of the `TZ` environment variable.
- If the `TZ` environment variable fails, extract the time zone from the actual soft link address of `/etc/localtime`.
- If both of the above methods fail, use `UTC` as the system time zone.

You can use the following statement to set the global server `time_zone` value at runtime:

```
SET GLOBAL time_zone = timezone;
```

Each client has its own time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

```
SET time_zone = timezone;
```

You can use the following statement to view the current values of the global, client-specific and system time zones:

```
SELECT @@global.time_zone, @@session.time_zone, @@global.system_time_zone;
```

To set the format of the value of the `time_zone`:

- The value 'SYSTEM' indicates that the time zone should be the same as the system time zone.
- The value can be given as a string indicating an offset from UTC, such as '+10:00' or '-6:00'.
- The value can be given as a named time zone, such as 'Europe/Helsinki', 'US/Eastern', or 'MET'.

The current session time zone setting affects the display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`.

> **Note:**
>
> Only the values of the Timestamp data type is affected by time zone. This is because the Timestamp data type uses the literal value + time zone information. Other data types, such as Datetime/Date/Time, do not have time zone information, thus their values are not affected by the changes of time zone.

```
create table t (ts timestamp, dt datetime);
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
set @@time_zone = 'UTC';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
insert into t values ('2017-09-30 11:11:11', '2017-09-30 11:11:11');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
set @@time_zone = '+8:00';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select * from t;
```

```
+---------------------|---------------------+
| ts                  | dt                  |
+---------------------|---------------------+
| 2017-09-30 19:11:11 | 2017-09-30 11:11:11 |
+---------------------|---------------------+
1 row in set (0.00 sec)
```

In this example, no matter how you adjust the value of the time zone, the value of the Datetime data type is not affected. But the displayed value of the Timestamp data type changes if the time zone information changes. In fact, the value that is stored in the storage does not change, it's just displayed differently according to different time zone setting.

Note:

- Time zone is involved during the conversion of the value of Timestamp and Datetime, which is handled based on the current `time_zone` of the session.
- For data migration, you need to pay special attention to the time zone setting of the primary database and the secondary database.

8.6 Daily Check

As a distributed database, TiDB is more complicated than the stand-alone database in terms of the mechanism, and monitoring items. To help operate and maintain TiDB in a more convenient way, this document introduces some key performance indicators.

8.6.1 Key indicators of TiDB Dashboard

Starting from v4.0, TiDB provides a new operation and maintenance management tool, TiDB Dashboard. This tool is integrated into the PD component. You can access TiDB Dashboard at the default address `http://${pd-ip}:${pd_port}/dashboard`.

TiDB Dashboard simplifies the operation and maintenance of the TiDB database. You can view the running status of the entire TiDB cluster through one interface. The following are descriptions of some performance indicators.

8.6.1.1 Instance panel

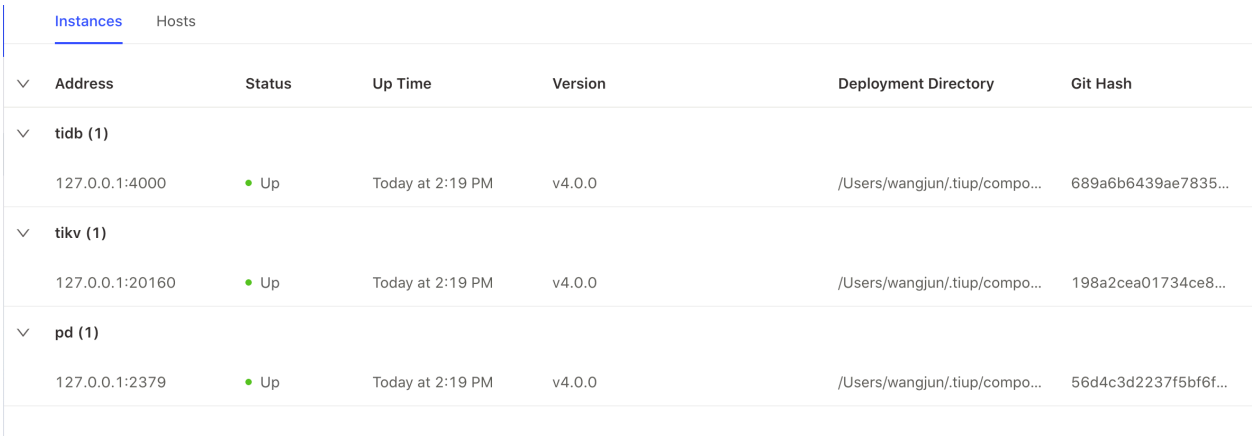


Figure 84: Instance panel

- **Status**: This indicator is used to check whether the status is normal. For an online node, this can be ignored.
- **Up Time**: The key indicator. If you find that the `Up Time` is changed, you need to locate the reason why the component is restarted.
- **Version**, **Deployment Directory**, **Git Hash**: These indicators need to be checked to avoid inconsistent or even incorrect version/deployment directory.

8.6.1.2 Host panel

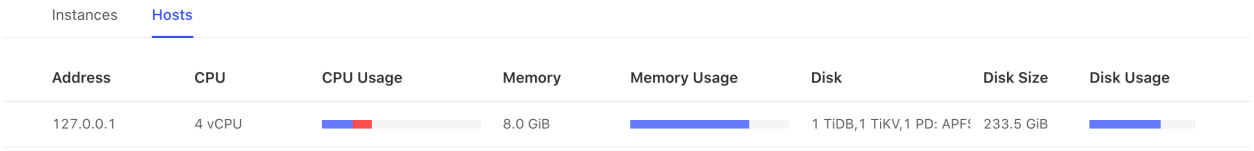


Figure 85: Host panel

You can view the usage of CPU, memory, and disk. When the usage of any resource exceeds 80%, it is recommended to scale out the capacity accordingly.

8.6.1.3 SQL analysis panel

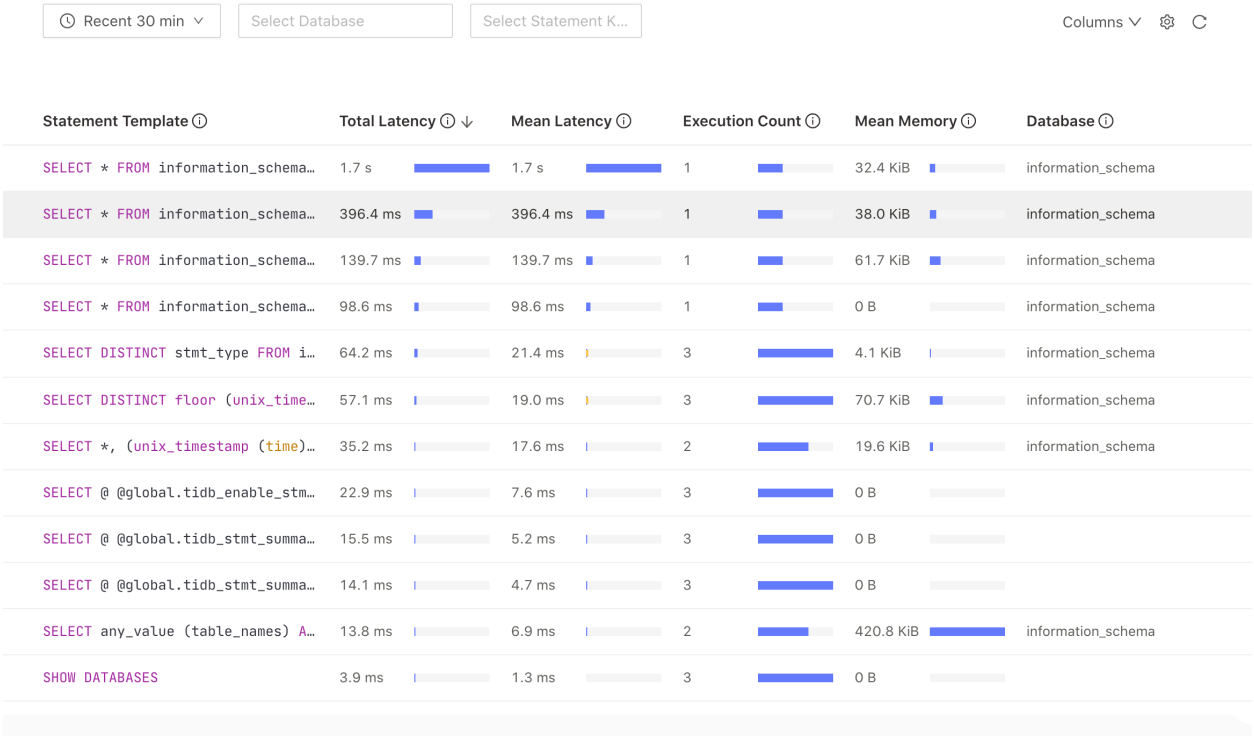


Figure 86: SQL analysis panel

You can locate the slow SQL statement executed in the cluster. Then you can optimize the specific SQL statement.

8.6.1.4 Region panel

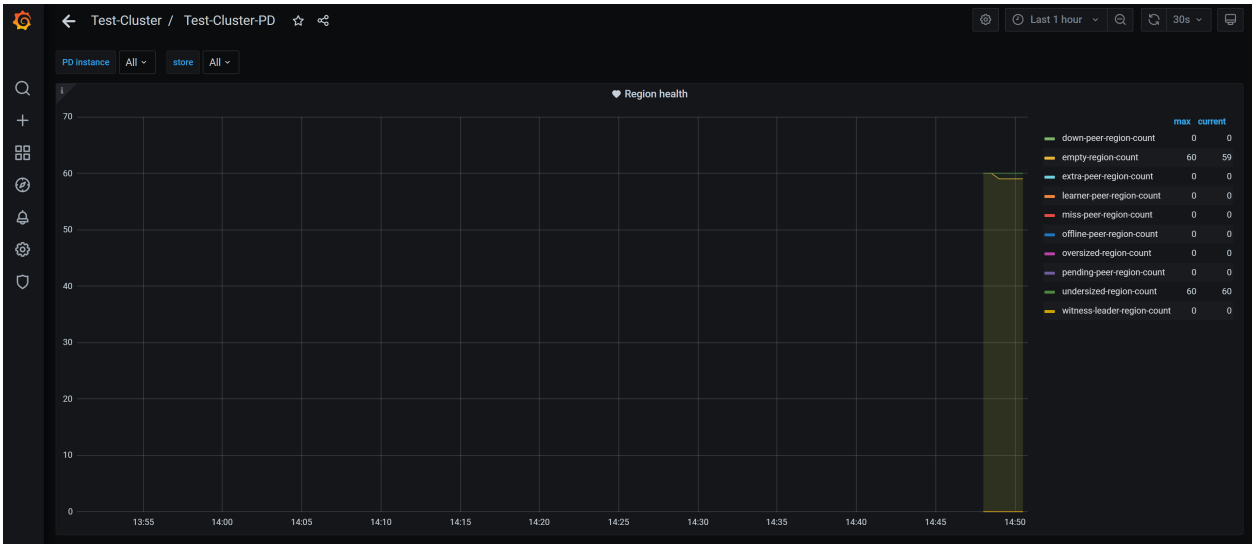


Figure 87: Region panel

- `down-peer-region-count`: The number of Regions with an unresponsive peer reported by the Raft leader.
- `empty-region-count`: The number of empty Regions, with a size of smaller than 1 MiB. These Regions are generated by executing the `TRUNCATE TABLE/DROP TABLE` statement. If this number is large, you can consider enabling `Region Merge` to merge Regions across tables.
- `extra-peer-region-count`: The number of Regions with extra replicas. These Regions are generated during the scheduling process.
- `learner-peer-region-count`: The number of Regions with the learner peer. The sources of learner peers can be various, for example, the learner peers in TiFlash, and the learner peers included in the configured Placement Rules.
- `miss-peer-region-count`: The number of Regions without enough replicas. This value is not always greater than `0`.
- `offline-peer-region-count`: The number of Regions during the peer offline process.
- `oversized-region-count`: The number of Regions with a size larger than `region-max-size` or `region-max-keys`.
- `pending-peer-region-count`: The number of Regions with outdated Raft logs. It is normal that a few pending peers are generated in the scheduling process. However, it is not normal if this value is large for a period of time (longer than 30 minutes).
- `undersized-region-count`: The number of Regions with a size smaller than `max-merge-region-size` or `max-merge-region-keys`.

Generally, it is normal that these values are not `0`. However, it is not normal that they are not `0` for quite a long time.

8.6.1.5 KV Request Duration

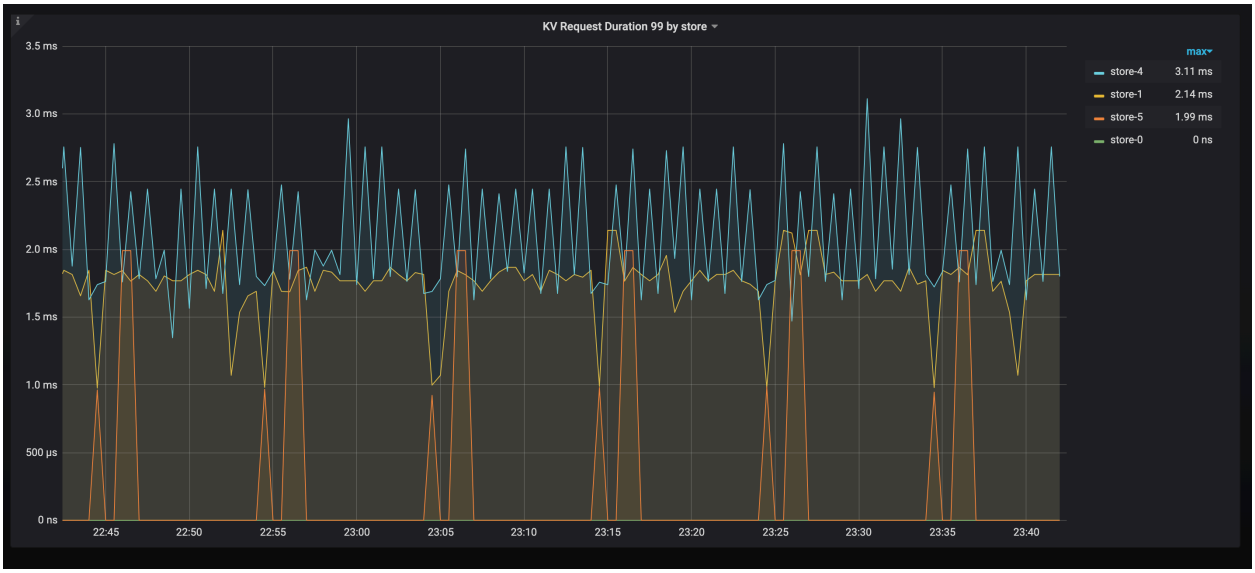


Figure 88: TiKV request duration

The KV request duration 99 in TiKV. If you find nodes with a long duration, check whether there are hot spots, or whether there are nodes with poor performance.

8.6.1.6 PD TSO Wait Duration

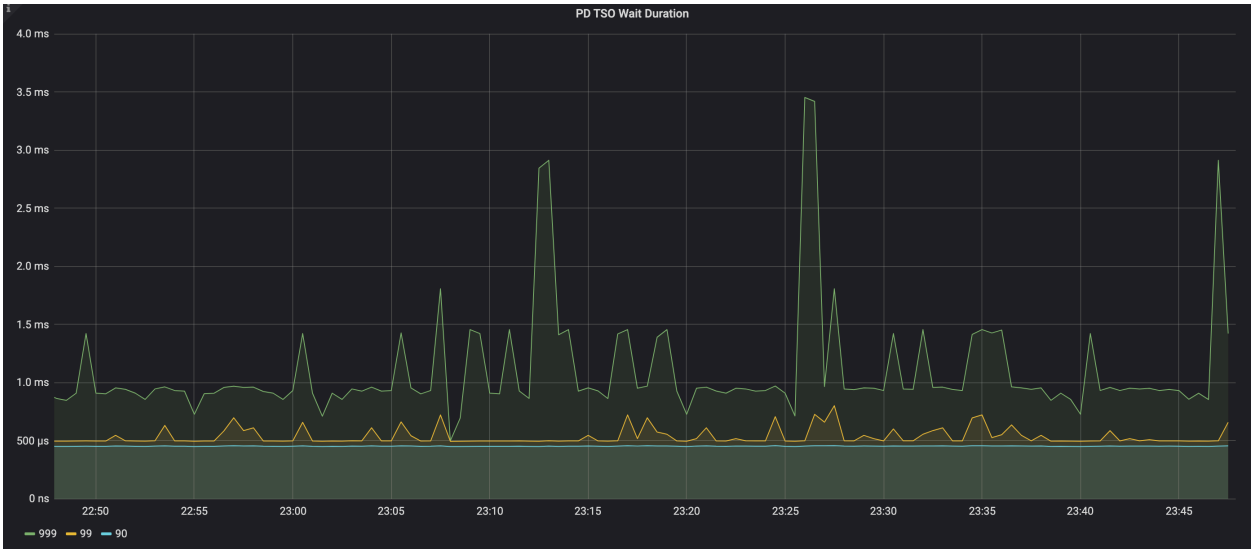


Figure 89: TiDB TSO Wait Duration

The time it takes for TiDB to obtain TSO from PD. The following are reasons for the long wait duration:

- High network latency from TiDB to PD. You can manually execute the ping command to test the network latency.
- High load for the TiDB server.
- High load for the PD server.

8.6.1.7 Overview panel

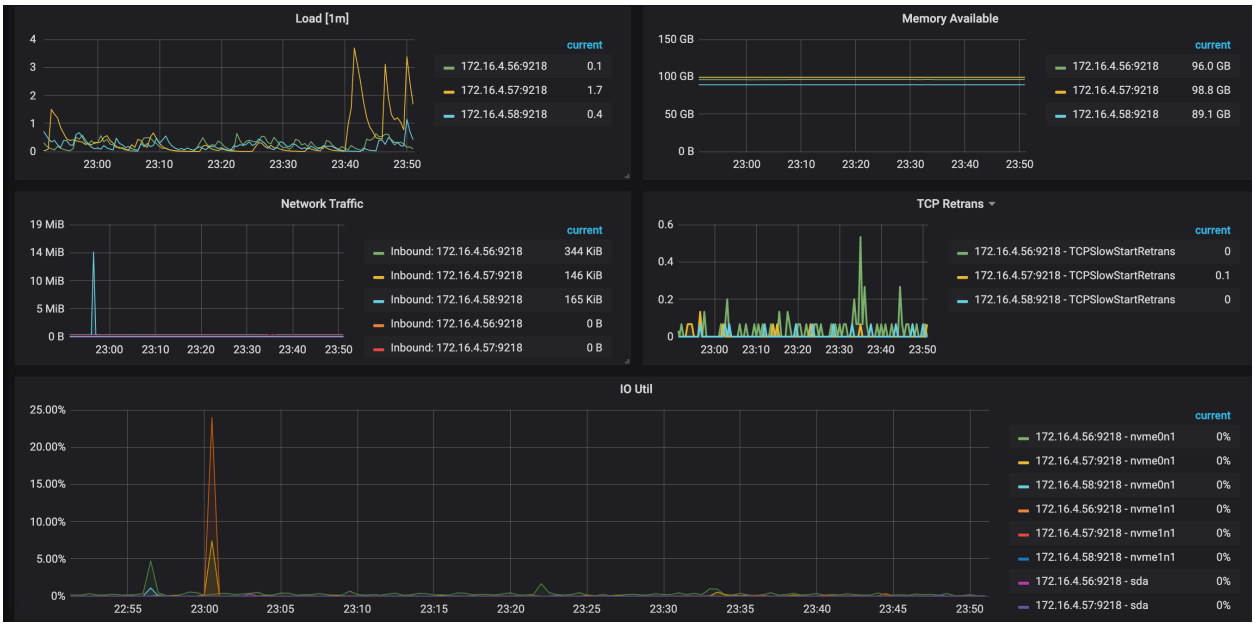


Figure 90: Overview panel

You can view the load, memory available, network traffic, and I/O utilities. When a bottleneck is found, it is recommended to scale out the capacity, or to optimize the cluster topology, SQL, and cluster parameters.

8.6.1.8 Exceptions

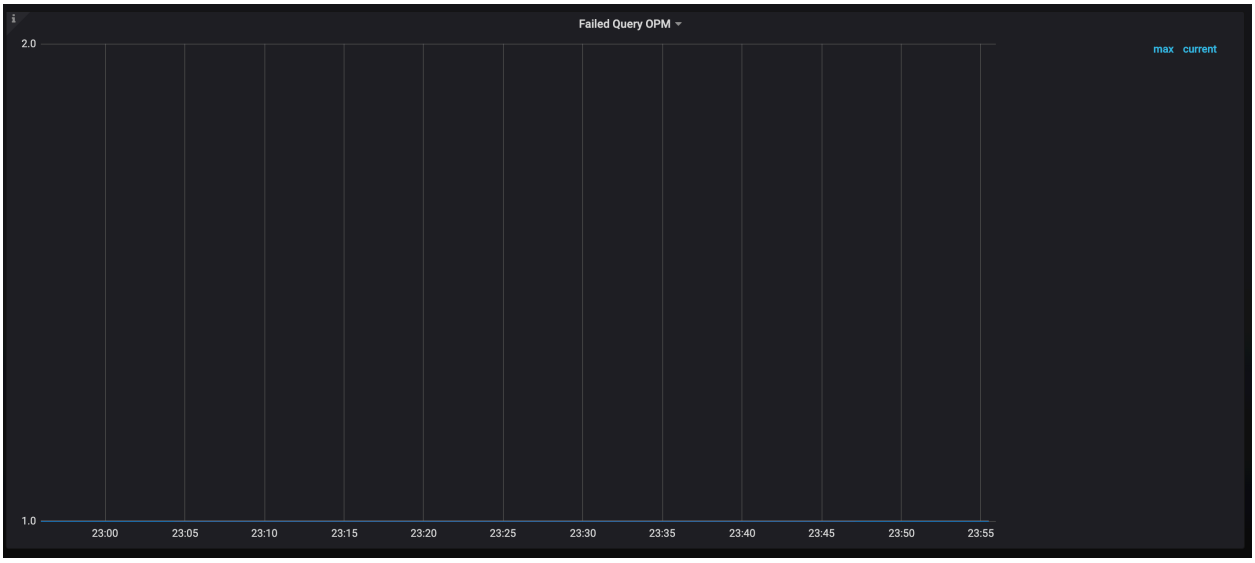


Figure 91: Exceptions

You can view the errors triggered by the execution of SQL statements on each TiDB instance. These include syntax error and primary key conflicts.

8.6.1.9 GC status

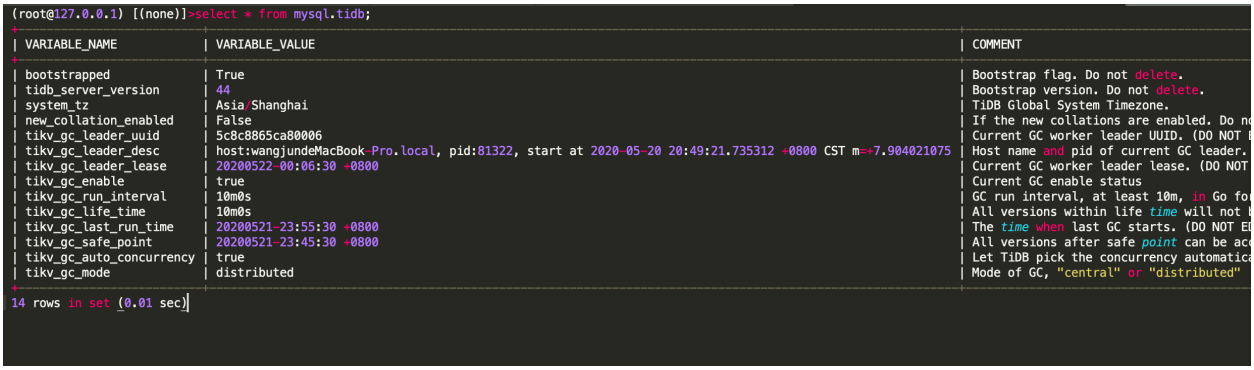


Figure 92: GC status

You can check whether the GC (Garbage Collection) status is normal by viewing the time when the last GC happens. If the GC is abnormal, it might lead to excessive historical data, thereby decreasing the access efficiency.

8.7 Maintain a TiFlash Cluster

This document describes how to perform common operations when you maintain a TiFlash cluster, including checking the TiFlash version. This document also introduces critical logs and a system table of TiFlash.

8.7.1 Check the TiFlash version

There are two ways to check the TiFlash version:

- If the binary file name of TiFlash is `tiflash`, you can check the version by executing the `./tiflash version` command.

  However, to execute the above command, you need to add the directory path which includes the `libtiflash_proxy.so` dynamic library to the `LD_LIBRARY_PATH` environment variable. This is because the running of TiFlash relies on the `libtiflash_proxy.so` dynamic library.

  For example, when `tiflash` and `libtiflash_proxy.so` are in the same directory, you can first switch to this directory, and then use the following command to check the TiFlash version:

```
LD_LIBRARY_PATH=./ ./tiflash version
```

- Check the TiFlash version by referring to the TiFlash log. For the log path, see the `[logger]` part in the `tiflash.toml` file. For example:

```
<information>: TiFlash version: TiFlash 0.2.0 master-375035282451103999
    ↪ f3863c691e2fc2
```

8.7.2 TiFlash critical logs

Log Information	Log Description
`[INFO] [<` ↪ `unknown>]` ↪ `["KVStore:` ↪ `Start to` ↪ `persist [` ↪ `region 47,` ↪ `applied:` ↪ `term 6` ↪ `index 10]"]` ↪ `[thread_id` ↪ `=23]`	Data starts to be replicated (the number in the square brackets at the start of the log refers to the thread ID

Log Information	Log Description
`[DEBUG] [<unknown>] ["CoprocessorHandler : grpc::Status DB::CoprocessorHandler::execute() : Handling DAG request"] [thread_id=30]`	Handling DAG request, that is, TiFlash starts to handle a Coprocessor request
`[DEBUG] [<unknown>] ["CoprocessorHandler : grpc::Status DB::CoprocessorHandler::execute() : Handle DAG request done"] [thread_id=30]`	Handling DAG request done, that is, TiFlash finishes handling a Coprocessor request

You can find the beginning or the end of a Coprocessor request, and then locate the related logs of the Coprocessor request through the thread ID printed at the start of the log.

8.7.3 TiFlash system table

The column names and their descriptions of the `information_schema.tiflash_replica` system table are as follows:

Column Name	Description
TABLE_SCHEMA	Database name
TABLE_NAME	Table name
TABLE_ID	Table ID
REPLICA_COUNT	Number of TiFlash replicas

Column Name	Description
LOCATION_LABELS	The hint for PD, based on which multiple replicas in a Region are scattered
AVAILABLE	Available or not (0/1)
PROGRESS	Replication progress [0.0~1.0]

8.8 TiUP Common Operations

This document describes the following common operations when you operate and maintain a TiDB cluster using TiUP.

- View the cluster list
- Start the cluster
- View the cluster status
- Modify the configuration
- Stop the cluster
- Destroy the cluster

8.8.1 View the cluster list

You can manage multiple TiDB clusters using the TiUP cluster component. When a TiDB cluster is deployed, the cluster appears in the TiUP cluster list.

To view the list, run the following command:

```
tiup cluster list
```

8.8.2 Start the cluster

The components in the TiDB cluster are started in the following order:

PD > TiKV > Pump > TiDB > TiFlash > Drainer > TiCDC > Prometheus > Grafana > Alertmanager

To start the cluster, run the following command:

```
tiup cluster start ${cluster-name}
```

> **Note:**
>
> Replace `${cluster-name}` with the name of your cluster. If you forget the cluster name, check it by running `tiup cluster list`.

You can start only some of the components by adding the `-R` or `-N` parameters in the command. For example:

- This command starts only the PD component:

```
tiup cluster start ${cluster-name} -R pd
```

- This command starts only the PD components on the `1.2.3.4` and `1.2.3.5` hosts:

```
tiup cluster start ${cluster-name} -N 1.2.3.4:2379,1.2.3.5:2379
```

> **Note:**
>
> If you start the specified component by using the `-R` or `-N` parameters, make sure the starting order is correct. For example, start the PD component before the TiKV component. Otherwise, the start might fail.

8.8.3 View the cluster status

After starting the cluster, check the status of each component to ensure that they work normally. TiUP provides the `display` command, so you do not have to log in to every machine to view the component status.

```
tiup cluster display ${cluster-name}
```

8.8.4 Modify the configuration

When the cluster is in operation, if you need to modify the parameters of a component, run the `edit-config` command. The detailed steps are as follows:

1. Open the configuration file of the cluster in the editing mode:

```
tiup cluster edit-config ${cluster-name}
```

2. Configure the parameters:

    - If the configuration is globally effective for a component, edit `server_configs`:

    ```
    server_configs:
      tidb:
        log.slow-threshold: 300
    ```

    - If the configuration takes effect on a specific node, edit the configuration in `config` of the node:

    ```
    tidb_servers:
    - host: 10.0.1.11
      port: 4000
      config:
          log.slow-threshold: 300
    ```

    For the parameter format, see the TiUP parameter template.

    Use `.` to represent the hierarchy of the configuration items.

    For more information on the configuration parameters of components, refer to TiDB `config.toml.example`, TiKV `config.toml.example`, and PD `config.toml.example` ↪ .

3. Rolling update the configuration and restart the corresponding components by running the `reload` command:

    ```
    tiup cluster reload ${cluster-name} [-N <nodes>] [-R <roles>]
    ```

8.8.4.1 Example

If you want to set the transaction size limit parameter (`txn-total-size-limit` in the performance module) to `1G` in tidb-server, edit the configuration as follows:

```
server_configs:
  tidb:
    performance.txn-total-size-limit: 1073741824
```

Then, run the `tiup cluster reload ${cluster-name} -R tidb` command to rolling restart the TiDB component.

8.8.5 Replace with a hotfix package

For normal upgrade, see Upgrade TiDB Using TiUP. But in some scenarios, such as debugging, you might need to replace the currently running component with a temporary package. To achieve this, use the `patch` command:

```
tiup cluster patch --help
```

```
Replace the remote package with a specified package and restart the service

Usage:
  cluster patch <cluster-name> <package-path> [flags]

Flags:
  -h, --help                help for patch
  -N, --node strings        Specify the nodes
      --overwrite           Use this package in the future scale-out
          ↪ operations
  -R, --role strings        Specify the role
      --transfer-timeout int Timeout in seconds when transferring PD and
          ↪ TiKV store leaders (default 300)

Global Flags:

      --native-ssh      Use the system's native SSH client
      --wait-timeout int Timeout of waiting the operation
      --ssh-timeout int Timeout in seconds to connect host via SSH, ignored
          ↪ for operations that don't need an SSH connection. (default 5)
  -y, --yes             Skip all confirmations and assumes 'yes'
```

If a TiDB hotfix package is in `/tmp/tidb-hotfix.tar.gz` and you want to replace all the TiDB packages in the cluster, run the following command:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -R tidb
```

You can also replace only one TiDB package in the cluster:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -N 172.16.4.5:4000
```

8.8.6 Rename the cluster

After deploying and starting the cluster, you can rename the cluster using the `tiup ↪ cluster rename` command:

```
tiup cluster rename ${cluster-name} ${new-name}
```

Note:

- The operation of renaming a cluster restarts the monitoring system (Prometheus and Grafana).
- After a cluster is renamed, some panels with the old cluster name might remain on Grafana. You need to delete them manually.

8.8.7 Stop the cluster

The components in the TiDB cluster are stopped in the following order (The monitoring component is also stopped):

Alertmanager > Grafana > Prometheus > TiCDC > Drainer > TiFlash > TiDB > Pump > TiKV > PD

To stop the cluster, run the following command:

```
tiup cluster stop ${cluster-name}
```

Similar to the `start` command, the `stop` command supports stopping some of the components by adding the `-R` or `-N` parameters. For example:

- This command stops only the TiDB component:

  ```
  tiup cluster stop ${cluster-name} -R tidb
  ```

- This command stops only the TiDB components on the `1.2.3.4` and `1.2.3.5` hosts:

  ```
  tiup cluster stop ${cluster-name} -N 1.2.3.4:4000,1.2.3.5:4000
  ```

8.8.8 Clean up cluster data

The operation of cleaning up cluster data stops all the services and cleans up the data directory or/and log directory. The operation cannot be reverted, so proceed **with caution**.

- Clean up the data of all services in the cluster, but keep the logs:

  ```
  tiup cluster clean ${cluster-name} --data
  ```

- Clean up the logs of all services in the cluster, but keep the data:

  ```
  tiup cluster clean ${cluster-name} --log
  ```

- Clean up the data and logs of all services in the cluster:

  ```
  tiup cluster clean ${cluster-name} --all
  ```

- Clean up the logs and data of all services except Prometheus:

```
tiup cluster clean ${cluster-name} --all --ignore-role prometheus
```

- Clean up the logs and data of all services except the `172.16.13.11:9000` instance:

```
tiup cluster clean ${cluster-name} --all --ignore-node
    ↪ 172.16.13.11:9000
```

- Clean up the logs and data of all services except the `172.16.13.12` node:

```
tiup cluster clean ${cluster-name} --all --ignore-node 172.16.13.12
```

8.8.9 Destroy the cluster

The destroy operation stops the services and clears the data directory and deployment directory. The operation cannot be reverted, so proceed **with caution**.

```
tiup cluster destroy ${cluster-name}
```

8.9 Modify Configuration Dynamically

This document describes how to dynamically modify the cluster configuration.

You can dynamically update the configuration of components (including TiDB, TiKV, and PD) using SQL statements, without restarting the cluster components. Currently, the method of changing TiDB instance configuration is different from that of changing configuration of other components (such TiKV and PD).

8.9.1 Common Operations

This section describes the common operations of dynamically modifying configuration.

8.9.1.1 View instance configuration

To view the configuration of all instances in the cluster, use the `show config` statement. The result is as follows:

```
show config;
```

```
+--
    ↪ ----+-----------------+-------------------------------------------------------------
    ↪
```

```
| Type | Instance     | Name                                                          |
   ↪  Value
   ↪
   ↪ |
+--
   ↪ ----+----------------+---------------------------------------------------------
   ↪
| tidb | 127.0.0.1:4001 | advertise-address                                          |
   ↪  127.0.0.1
   ↪
   ↪ |
| tidb | 127.0.0.1:4001 | binlog.binlog-socket                                       |
   ↪
   ↪
   ↪ |
| tidb | 127.0.0.1:4001 | binlog.enable                                              |
   ↪  false
   ↪
   ↪ |
| tidb | 127.0.0.1:4001 | binlog.ignore-error                                        |
   ↪  false
   ↪
   ↪ |
| tidb | 127.0.0.1:4001 | binlog.strategy                                            |
   ↪  range
   ↪
   ↪ |
| tidb | 127.0.0.1:4001 | binlog.write-timeout                                       |
   ↪  15s
   ↪
   ↪ |
| tidb | 127.0.0.1:4001 | check-mb4-value-in-utf8                                    |
   ↪  true
   ↪
   ↪ |

...
```

You can filter the result by fields. For example:

```
show config where type='tidb'
show config where instance in (...)
show config where name like '%log%'
show config where type='tikv' and name='log.level'
```

8.9.1.2 Modify TiKV configuration dynamically

> **Note:**
>
> - After dynamically changing TiKV configuration items, the TiKV configuration file is automatically updated. However, you also need to modify the corresponding configuration items by executing `tiup edit-config`; otherwise, operations such as `upgrade` and `reload` will overwrite your changes. For details of modifying configuration items, refer to Modify configuration using TiUP.
> - After executing `tiup edit-config`, you do not need to execute `tiup reload`.

When using the `set config` statement, you can modify the configuration of a single instance or of all instances according to the instance address or the component type.

- Modify the configuration of all TiKV instances:

> **Note:**
>
> It is recommended to wrap variable names in backticks.

```
set config tikv `split.qps-threshold`=1000;
```

- Modify the configuration of a single TiKV instance:

```
set config "127.0.0.1:20180" `split.qps-threshold`=1000;
```

If the modification is successful, `Query OK` is returned:

```
Query OK, 0 rows affected (0.01 sec)
```

If an error occurs during the batch modification, a warning is returned:

```
set config tikv `log-level`='warn';
```

```
Query OK, 0 rows affected, 1 warning (0.04 sec)
```

```
show warnings;
```

```
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
| Level   | Code | Message
    ↪
    ↪ |
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
| Warning | 1105 | bad request to http://127.0.0.1:20180/config: fail to
    ↪ update, error: "config log-level can not be changed" |
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

The batch modification does not guarantee atomicity. The modification might succeed on some instances, while failing on others. If you modify the configuration of the entire TiKV cluster using `set tikv key=val`, your modification might fail on some instances. You can use `show warnings` to check the result.

If some modifications fail, you need to re-execute the corresponding statement or modify each failed instance. If some TiKV instances cannot be accessed due to network issues or machine failure, modify these instances after they are recovered.

If a configuration item is successfully modified, the result is persisted in the configuration file, which will prevail in the subsequent operations. The names of some configuration items might conflict with TiDB reserved words, such as `limit` and `key`. For these configuration items, use backtick ` to enclose them. For example, `` `raftstore.raft-log-gc-size-limit` ``.

The following TiKV configuration items can be modified dynamically:

Configuration item	Description
`raftstore.raft-max-inflight-msgs`	The number of Raft logs to be confirmed. If this number is exceeded, the Raft state machine slows down log sending.
`raftstore.raft-log-gc-tick-interval`	The time interval at which the polling task of deleting Raft logs is scheduled

Configuration item	Description
`raftstore.raft-soft-limit`	The soft limit on the maximum allowable number of residual Raft logs
`raftstore.raft-hard-limit`	The hard limit on the allowable number of residual Raft logs

Configuration item	Description
`raftstore.raft-log-gc-size-limit`	The hard limit on the allowable size of residual Raft logs
`raftstore.raft-max-size-per-msg`	The soft limit on the size of a single message packet that is allowed to be generated

Configuration item	Description
`raftstore.raft-entry-max-size`	The hard limit on the maximum size of a single Raft log
`raftstore.raft-entry-cache-life-time`	The maximum remaining time allowed for the log cache in memory

Configuration item	Description
`raftstore.split-region-check-tick-interval`	The time interval for checking whether the region split is needed
`raftstore.region-max-size`	The maximum value for the Region data that is allowed to exceed before Region split

Configuration item	Description
`raftstore.region-compact-check-interval`	The time interval between which to check whether it is necessary to manually trigger RocksDB compaction
`raftstore.region-compact-check-step`	The number of Regions checked at one time for each round of manual compaction

Configuration item	Description
`raftstore.region-compact-min-tombstones`	The minimum number of tombstones required to trigger RocksDB compaction
`raftstore.region-compact-tombstones-percent`	The proportion of tombstones required to trigger RocksDB compaction

Configuration item	Description
`raftstore.pd-heartbeat-tick-interval`	The time interval for a Region's heartbeat to PD is triggered
`raftstore.store-heartbeat-tick-interval`	The time interval for which store's heartbeat to PD is triggered

Configuration item	Description
`raftstore.snap-mgr-gc-tick-interval`	The time interval at which the recycle of expired snapshot files is triggered
`raftstore.snap-gc-timeout`	The longest time for which a snapshot file is saved

Configuration item	Description
`raftstore.lock-cf-compact-interval`	The time interval at which TiKV triggers a manual compaction for the Lock Column Family
`raftstore.lock-cf-compact-bytes-threshold`	The size at which TiKV triggers a manual compaction for the Lock Column Family

Configuration item	Description
`raftstore.messages-per-tick`	The maximum number of messages processed per batch
`raftstore.max-peer-down-duration`	The longest inactive duration allowed for a peer

Configuration item	Description
`raftstore.max-leader-missing-duration`	The longest duration for a peer to be without a leader. If this value is exceeded, the peer verifies with PD whether it has been deleted.

Configuration item	Description
`raftstore.abnormal-leader-missing-duration`	The normal abnormal duration time allowed for a peer to be without a leader. If this value is exceeded, the peer is seen as abnormal and marked in metrics and logs.

Configuration item	Description
`raftstore.peer-stale-state-check-interval`	The time interval to check whether a peer is out a leader
`raftstore.consistency-check-interval`	The time interval for consistency check (**NOT** recommended because it is not compatible with the garbage collection in TiDB)

Configuration item	Description
`raftstore.raft-max-leader-lease`	The longest trusted period of a Raft leader lease
`raftstore.merge-check-tick-interval`	The time interval for the merge check
`raftstore.cleanup-import-sst-interval`	The time interval for check expired SST files

Configuration item	Description
`raftstore.local-read-batch-size`	The maximum number of read requests processed in one batch

Configuration item	Description
`raftstore.hibernate-wait-timeout`	The shortest wait duration before entering hibernation upon start. Within this duration, TiKV does not hibernate (not released).

Configuration item	Description
`raftstore.apply-pool-size`	The number of threads in the pool that flushes data to the disk, which is the size of the Apply thread pool

Configuration item	Description
`raftstore.store-pool-size`	The number of threads in the pool that processes Raft, which is the size of the Raftstore thread pool

Configuration item	Description
`raftstore.apply-max-batch-size`	Raft state machines process data write requests in batches by the BatchSystem. This configuration item specifies the maximum number of Raft state machines that can execute the requests in one batch.

Configuration item	Description
`raftstore.store-max-batch-size`	Raft state machines process requests for flushing logs into the disk in batches by the BatchSystem. This configuration item specifies the maximum number of Raft state machines that can process the requests

Configuration item	Description
`readpool.unified.max-thread-count`	The maximum number of threads in the thread pool that uniformly processes read requests, which is the size of the UnifyReadPool thread pool
`coprocessor.split-region-on-table`	Enables to split Region by table

Configuration item	Description
coprocessor.batch-split-limit	The threshold of Region split in batches
coprocessor.region-max-size	The maximum size of a Region
coprocessor.region-split-size	The size of the newly split Region
coprocessor.region-max-keys	The maximum number of keys allowed in a Region

Configuration item	Description
coprocessor.region-split-keys	The number of keys in the newly split Region
pessimistic-txn.wait-for-lock-timeout	The longest duration that a pessimistic transaction waits for the lock
pessimistic-txn.wake-up-delay-duration	The duration after which a pessimistic transaction is woken up

Configuration item	Description
`pessimistic-txn.pipelined`	Determines whether to enable the pipelined pessimistic locking process
`pessimistic-txn.in-memory`	Determines whether to enable the in-memory pessimistic lock

Configuration item	Description
`quota.foreground-cpu-time`	The soft limit on the CPU resources used by TiKV foreground to process read and write requests
`quota.foreground-write-bandwidth`	The soft limit on the bandwidth with which transactions write data

Configuration item	Description
`quota-foreground-bandwidth`	The soft limit on the read bandwidth with which transactions and the Coprocessor read data

Configuration item	Description
`quota.max-delay-duration`	The maximum time that a single read or write request is forced to wait before it is processed in the foreground

Configuration item	Description
`gc.ratio-threshold`	The threshold which Region GC is skipped (the number of GC versions/the number of keys)
`gc.batch-keys`	The number of keys processed in one batch

Configuration item	Description
gc.max-write-bytes-per-sec	The maximum bytes that can be written into RocksDB per second
gc.enable-compaction-filter	Whether to enable compaction filter
gc.compaction-filter-skip-version-check	Whether to skip the cluster version check of compaction filter (not released)

Configuration item	Description
{db-name}.max-total-wal-size	The maximum size of total WAL
{db-name}.max-background-jobs	The number of background threads in RocksDB
{db-name}.max-background-flushes	The maximum number of background flush threads in RocksDB
{db-name}.max-open-files	The total number of files that RocksDB can open

Configuration item	Description
{db-name}.compaction-readahead-size	The size of pre-reading during compaction
{db-name}.bytes-per-sync	The rate at which OS incrementally synchronizes files to disk while these files are being written asynchronously

Configuration item	Description
{db-name}.wal_bytes_per_sync	The rate at which OS incrementally synchronizes WAL files to disk while the WAL files are being written
{db-name}.writable_file_max_buffer_size	The maximum buffer size used in WritableFileWriter buffer size

Configuration item	Description
{db-name}.{cf-name}.block-cache-size	The size of a block cache
{db-name}.{cf-name}.write-buffer-size	The size of a memtable
{db-name}.{cf-name}.max-write-buffer-number	The maximum number of memtables

Configuration item	Description
{db-name}.{cf-name}.max-bytes-for-level-base	The maximum number of bytes at base level (L1) for level-base
{db-name}.{cf-name}.target-file-size-base	The size of the target file at base level

Configuration item	Description
{db-name}.{cf-name}.{level-0-file-num-compaction-trigger}	The maximum number of files at L0 that trigger compaction
{db-name}.{cf-name}.{level-0-slowdown-writes-trigger}	The maximum number of files at L0 that trigger write stall

Configuration item	Description
{db-name}.{cf-name}.level0-stop-writes-trigger	The maximum number of files at L0 that completely block write
{db-name}.{cf-name}.max-compaction-bytes	The maximum number of bytes written by one compaction

Configuration item	Description
`{db-name}.{cf-name}.max-bytes-for-level-multiplier`	The default amplification multiple for each layer
`{db-name}.{cf-name}.disable-auto-compactions`	Enables or disables automatic compaction

Configuration item	Description
{db-name}.{cf-name}.soft-pending-compaction-bytes-limit	The soft limit on the pending compaction bytes
{db-name}.{cf-name}.hard-pending-compaction-bytes-limit	The hard limit on the pending compaction bytes

Configuration item	Description
`{db-name}.{cf-name}.titan.blob_run_mode`	The mode of processing blob files
`server.grpc-memory-pool-quota`	Limits the memory size that can be used by gRPC
`server.max-grpc-send-msg-len`	Sets the maximum length of a gRPC message that can be sent

Configuration item	Description
`server.snapshot-max-write-bytes-per-sec`	Sets the maximum allowable disk bandwidth when processing snapshots
`server.concurrent-send-snapshot-limit`	Sets the maximum number of snapshots sent at the same time

Configuration item	Description
`server.concurrent-recv-snap-limit`	Sets the maximum number of snapshots received at the same time
`server.raft-msg-max-batch-size`	Sets the maximum number of Raft messages that are contained in a single gRPC message

Configuration item	Description
`storage.block-cache.capacity`	The size of shared block cache (supported since v4.0.3)
`storage.scheduler-worker-pool-size`	The number of threads in the Scheduler thread pool
`backup.num-threads`	The number of backup threads (supported since v4.0.3)

Configuration item	Description
`split-qps-threshold`	The threshold to execute `load-base-split` on a Region. If the QPS of read requests for a Region exceeds `qps-threshold` for a consecutive period of time, this Region should be split.

Configuration item	Description
`split-byte-threshold`	The threshold to execute `split-load-base-split` on a Region. If the traffic of read requests for a Region exceeds the `byte-threshold` for a consecutive period of time, this Region should be split.

Configuration item	Description
`split.split-balance-score`	The parameter of `balance-load` split. The smaller the value is, the more balanced the load is. But setting it too small might cause split

Configuration item	Description
`split.split-contained-score-base`	The parameter of the load base split. The smaller the value, the fewer cross-Region visits after Region split.
`cdc.min-ts-interval`	The time interval at which Resolved TS is forwarded

Configuration item	Description
`cdc.old-value.memory.quota`	The upper limit of memory occupied by the TiCDC Old Value entries
`cdc.sink-memory.quota`	The upper limit of memory occupied by TiCDC data change events

Configuration item	Description
`cdc.incremental-scan-speed-limit`	The rate limit on the speed of incremental scanning for historical data
`cdc.incremental-scan-concurrency`	The maximum number of concurrent incremental scanning tasks for historical data

In the table above, parameters with the `{db-name}` or `{db-name}.{cf-name}` prefix are configurations related to RocksDB. The optional values of `db-name` are `rocksdb` and `raftdb`.

- When `db-name` is `rocksdb`, the optional values of `cf-name` are `defaultcf`, `writecf`, `lockcf`, and `raftcf`.
- When `db-name` is `raftdb`, the value of `cf-name` can be `defaultcf`.

For detailed parameter description, refer to TiKV Configuration File.

8.9.1.3 Modify PD configuration dynamically

Currently, PD does not support the separate configuration for each instance. All PD instances share the same configuration.

You can modify the PD configurations using the following statement:

```
set config pd `log.level`='info';
```

If the modification is successful, `Query OK` is returned:

```
Query OK, 0 rows affected (0.01 sec)
```

If a configuration item is successfully modified, the result is persisted in etcd instead of in the configuration file; the configuration in etcd will prevail in the subsequent operations. The names of some configuration items might conflict with TiDB reserved words. For these configuration items, use backtick ` to enclose them. For example, `` `schedule.leader-schedule-limit` ``.

The following PD configuration items can be modified dynamically:

Configuration item	Description
`log.level`	The log level
`cluster-version`	The cluster version
`schedule.max-merge-region-size`	Controls the size limit of Region Merge (in MiB)

Configuration item	Description
`schedule-max-merge-region-keys`	Specifies the maximum numbers of the Region Merge keys
`schedule-patrol-region-interval`	Determines the frequency which ReplicaChecker checks the health state of a Region

Configuration item	Description
`schedule-split-interval`	Determines the time interval of performing split and merge operations on the same Region
`schedule-max-snapshot-count`	Determines the maximum number of snapshots that a single store can send or receive at the same time

Configuration item	Description
`schedule-max-pending-peer-count`	Determines the maximum number of pending peers in a single store
`schedule-max-store-down-time`	The down-time after which PD judges that the disconnected store cannot be recovered
`schedule-leader-schedule-policy`	Determines the policy of leader scheduling

Configuration item	Description
`schedule` ↪ `.` ↪ `leader` ↪ `-` ↪ `schedule` ↪ `-` ↪ `limit` ↪	The number of leader scheduling tasks performed at the same time
`schedule` ↪ `.` ↪ `region` ↪ `-` ↪ `schedule` ↪ `-` ↪ `limit` ↪	The number of Region scheduling tasks performed at the same time

Configuration item	Description
`schedule-replica-schedule-limit`	The number of Replica scheduling tasks performed at the same time
`schedule-merge-schedule-limit`	The number of Region Merge scheduling tasks performed at the same time

Configuration item	Description
`schedule-hot-region-schedule-limit`	The number of hot Region schedule scheduling tasks performed at the same time
`schedule-hot-region-cache-hits-threshold`	Determines the threshold which a Region is considered a hot spot

Configuration item	Description
`schedule.high-space-ratio`	The threshold ratio of low space ratio, which the capacity of the store is sufficient
`schedule.low-space-ratio`	The threshold ratio of high space ratio, which the capacity of the store is insufficient
`schedule.tolerant-size-ratio`	Controls the balance buffer size ratio

Configuration item	Description
`schedule-enable-down-replica`	Determines whether to enable the feature that automatically removes `DownReplica`
`schedule-enable-replace-offline-replica`	Determines whether to enable the feature that migrates `OfflineReplica`

Configuration item	Description
`schedule-enable-make-up-replica`	Determines whether to enable the feature that automatically supplements replicas
`schedule-enable-remove-extra-replica`	Determines whether to enable the feature that removes extra replicas
`schedule-enable-location-replacement`	Determines whether to enable location replacement check

Configuration item	Description
`schedule.enable-cross-table-merge`	Determines whether to enable cross-table merge
`schedule.enable-one-way-merge`	Enables one-way merge, which only allows merging with the next adjacent Region
`replication.max-replicas`	Sets the maximum number of replicas

Configuration item	Description
`replication.location-labels`	The topology information of a TiKV cluster
`replication.enable-placement-rules`	Enables Placement Rules
`replication.strictly-match-label`	Enables the label check
`pd-server.use-region-storage`	Enables independent Region storage

Configuration item	Description
pd-server.max-gap-reset-ts	Sets the maximum interval of resetting timestamp (BR)
pd-server.key-type	Sets the cluster key type
pd-server.metric-storage	Sets the storage address of the cluster metrics
pd-server.dashboard-address	Sets the dashboard address

Configuration item	Description
`replication`	Sets
↪ `-`	the
↪ `mode`	backup
↪ `.`	mode
↪ `replication`	
↪ `-`	
↪ `mode`	
↪	

For detailed parameter description, refer to PD Configuration File.

8.9.1.4 Modify TiDB configuration dynamically

Currently, the method of changing TiDB configuration is different from that of changing TiKV and PD configurations. You can modify TiDB configuration by using system variables.

The following example shows how to dynamically modify `slow-threshold` by using the `tidb_slow_log_threshold` variable.

The default value of `slow-threshold` is 300 ms. You can set it to 200 ms by using `tidb_slow_log_threshold`.

```
set tidb_slow_log_threshold = 200;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select @@tidb_slow_log_threshold;
```

```
+---------------------------+
| @@tidb_slow_log_threshold |
+---------------------------+
| 200                    |
+---------------------------+
1 row in set (0.00 sec)
```

The following TiDB configuration items can be modified dynamically:

Configuration item	SQL variable
`instance.tidb_enable_slow_log`	`tidb_enable_slow_log`
`instance.tidb_slow_log_threshold`	`tidb_slow_log_threshold`
`instance.tidb_expensive_query_time_threshold`	`tidb_expensive_query_time_threshold`

8.9.1.5 Modify TiFlash configuration dynamically

Currently, you can modify the TiFlash configuration `max_threads` by using the system variable `tidb_max_tiflash_threads`, which specifies the maximum concurrency for TiFlash to execute a request.

The default value of `tidb_max_tiflash_threads` is `-1`, indicating that this system variable is invalid and depends on the setting of the TiFlash configuration file. You can set `max_threads` to 10 by using `tidb_max_tiflash_threads`:

```
set tidb_max_tiflash_threads = 10;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select @@tidb_max_tiflash_threads;
```

```
+----------------------------+
| @@tidb_max_tiflash_threads |
+----------------------------+
| 10                         |
+----------------------------+
1 row in set (0.00 sec)
```

8.10 Online Unsafe Recovery

> **Warning:**
>
> Online Unsafe Recovery is a type of lossy recovery. If you use this feature, the integrity of data and data indexes cannot be guaranteed.

When permanently damaged replicas cause part of data on TiKV to be unreadable and unwritable, you can use the Online Unsafe Recovery feature to perform a lossy recovery operation.

8.10.1 Feature description

In TiDB, the same data might be stored in multiple stores at the same time according to the replica rules defined by users. This guarantees that data is still readable and writable even if a single or a few stores are temporarily offline or damaged. However, when most or all replicas of a Region go offline during a short period of time, the Region becomes temporarily unavailable and cannot be read or written.

Suppose that multiple replicas of a data range encounter issues like permanent damage (such as disk damage), and these issues cause the stores to stay offline. In this case, this data range is temporarily unavailable. If you want the cluster back in use and also accept data rewind or data loss, in theory, you can re-form the majority of replicas by manually removing the failed replicas from the group. This allows application-layer services to read and write this data range (might be stale or empty) again.

In this case, if some stores with loss-tolerant data are permanently damaged, you can perform a lossy recovery operation by using Online Unsafe Recovery. When you use this feature, PD automatically pauses Region scheduling (including split and merge), collects the metadata of data shards from all stores, and then, under its global perspective, generates a real-time and complete recovery plan. Then, PD distributes the plan to all surviving stores to make them perform data recovery tasks. In addition, once the data recovery plan is distributed, PD periodically monitors the recovery progress and re-send the plan when necessary.

8.10.2 User scenarios

The Online Unsafe Recovery feature is suitable for the following scenarios:

- The data for application services is unreadable and unwritable, because permanently damaged stores cause the stores to fail to restart.
- You can accept data loss and want the affected data to be readable and writable.
- You want to perform a one-stop online data recovery operation.

8.10.3 Usage

8.10.3.1 Prerequisites

Before using Online Unsafe Recovery, make sure that the following requirements are met:

- The offline stores indeed cause some pieces of data to be unavailable.
- The offline stores cannot be automatically recovered or restarted.

8.10.3.2 Step 1. Specify the stores that cannot be recovered

To trigger automatic recovery, use PD Control to execute `unsafe remove-failed-stores <store_id>[,<store_id>,...]` and specify **all** the TiKV nodes that cannot be recovered, seperated by commas.

```
pd-ctl -u <pd_addr> unsafe remove-failed-stores <store_id1,store_id2,...>
```

If the command returns `Success`, PD Control has successfully registered the task to PD. This only means that the request has been accepted, not that the recovery has been successfully performed. The recovery task is performed in the background. To see the recovery progress, use `show`.

If the command returns `Failed`, PD Control has failed to register the task to PD. The possible errors are as follows:

- `unsafe recovery is running`: There is already an ongoing recovery task.
- `invalid input store x doesn't exist`: The specified store ID does not exist.
- `invalid input store x is up and connected`: The specified store with the ID is still healthy and should not be recovered.

To specify the longest allowable duration of a recovery task, use the `--timeout <seconds >` option. If this option is not specified, the longest duration is 5 minutes by default. When the timeout occurs, the recovery is interrupted and returns an error.

Note:

- Because this command needs to collect information from all peers, it might cause an increase in memory usage (100,000 peers are estimated to use 500 MiB of memory).
- If PD restarts when the command is running, the recovery is interrupted and you need to trigger the task again.
- Once the command is running, the specified stores will be set to the Tombstone status, and you cannot restart these stores.
- When the command is running, all scheduling tasks and split/merge are paused and will be resumed automatically after the recovery is successful or fails.

8.10.3.3 Step 2. Check the recovery progress and wait for the completion

When the above store removal command runs successfully, you can use PD Control to check the removal progress by running `unsafe remove-failed-stores show`.

```
pd-ctl -u <pd_addr> unsafe remove-failed-stores show
```

The recovery process has multiple possible stages:

- `collect report`: The initial stage in which PD collects reports from TiKV and gets global information.
- `tombstone tiflash learner`: Among the unhealthy Regions, delete the TiFlash learners that are newer than other healthy peers, to prevent such an extreme situation and the possible panic.
- `force leader for commit merge`: A special stage. When there is an uncompleted commit merge, `force leader` is first performed on the Regions with commit merge, in case of extreme situations.

- `force leader`: Forces unhealthy Regions to assign a Raft leader among the remaining healthy peers.
- `demote failed voter`: Demotes the Region's failed voters to learners, and then the Regions can select a Raft leader as normal.
- `create empty region`: Creates an empty Region to fill in the space in the key range. This is to resolve the case that the stores with all replicas of some Regions have been damaged.

Each of the above stages is output in the JSON format, including information, time, and a detailed recovery plan. For example:

```
[
    {
        "info": "Unsafe recovery enters collect report stage: failed stores
            ↪ 4, 5, 6",
        "time": "......"
    },
    {
        "info": "Unsafe recovery enters force leader stage",
        "time": "......",
        "actions": {
            "store 1": [
                "force leader on regions: 1001, 1002"
            ],
            "store 2": [
                "force leader on regions: 1003"
            ]
        }
    },
    {
        "info": "Unsafe recovery enters demote failed voter stage",
        "time": "......",
        "actions": {
            "store 1": [
                "region 1001 demotes peers { id:101 store_id:4 }, { id:102
                    ↪ store_id:5 }",
                "region 1002 demotes peers { id:103 store_id:5 }, { id:104
                    ↪ store_id:6 }",
            ],
            "store 2": [
                "region 1003 demotes peers { id:105 store_id:4 }, { id:106
                    ↪ store_id:6 }",
            ]
        }
    },
```

```
    {
        "info": "Collecting reports from alive stores(1/3)",
        "time": "......",
        "details": [
            "Stores that have not dispatched plan: ",
            "Stores that have reported to PD: 4",
            "Stores that have not reported to PD: 5, 6",
        ]
    }
]
```

After PD has successfully dispatched the recovery plan, it waits for TiKV to report the execution results. As you can see in `Collecting reports from alive stores`, the last stage of the above output, this part of the output shows the detailed statuses of PD dispatching recovery plan and receiving reports from TiKV.

The whole recovery process takes multiple stages and one stage might be retried multiple times. Usually, the estimated duration is 3 to 10 periods of store heartbeat (one period of store heartbeat is 10 seconds by default). After the recovery is completed, the last stage in the command output shows `"Unsafe recovery finished"`, the table IDs to which the affected Regions belong (if there is none or RawKV is used, the output does not show the table IDs), and the affected SQL meta Regions. For example:

```
{
    "info": "Unsafe recovery finished",
    "time": "......",
    "details": [
        "Affected table ids: 64, 27",
        "Affected meta regions: 1001",
    ]
}
```

After you get the affected table IDs, you can query `INFORMATION_SCHEMA.TABLES` to view the affected table names.

```
SELECT TABLE_SCHEMA, TABLE_NAME, TIDB_TABLE_ID FROM INFORMATION_SCHEMA.
    ↪ TABLES WHERE TIDB_TABLE_ID IN (64, 27);
```

Note:

- The recovery operation has turned some failed voters to failed learners. Then PD scheduling needs some time to remove these failed learners.
- It is recommended to add new stores in time.

If an error occurs during the task, the last stage in the output shows `"Unsafe recovery failed"` and the error message. For example:

```
{
    "info": "Unsafe recovery failed: <error>",
    "time": "......"
}
```

8.10.3.4 Step 3. Check the consistency of data and index (not required for RawKV)

> **Note:**
>
> Although the data can be read and written, it does not mean that there is no data loss.

After the recovery is completed, the data and index might be inconsistent. Use the SQL command `ADMIN CHECK` to check the data and index consistency of the affected tables

```
ADMIN CHECK TABLE table_name;
```

If there are inconsistent indexes, you can fix the index inconsistency by renaming the old index, creating a new index, and then droping the old index.

1. Rename the old index:

    ```
    ALTER TABLE table_name RENAME INDEX index_name TO index_name_lame_duck
        ↪ ;
    ```

2. Create a new index:

    ```
    ALTER TABLE table_name ADD INDEX index_name (column_name);
    ```

3. Drop the old index:

    ```
    ALTER TABLE table_name DROP INDEX index_name_lame_duck;
    ```

8.10.3.5 Step 4: Remove unrecoverable stores (optional)

1. Remove the unrecoverable nodes:

    ```
    tiup cluster scale-in <cluster-name> -N <host> --force
    ```

2. Clean up Tombstone nodes:

```
tiup cluster prune <cluster-name>
```

1. Delete the `PersistentVolumeClaim`.

```
kubectl delete -n ${namespace} pvc ${pvc_name} --wait=false
```

2. Delete the TiKV Pod and wait for newly created TiKV Pods to join the cluster.

```
kubectl delete -n ${namespace} pod ${pod_name}
```

8.11 Replicate Data Between Primary and Secondary Clusters

This document describes how to configure a TiDB primary (upstream) cluster and a TiDB or MySQL secondary (downstream) cluster, and replicate incremental data from the primary cluster to the secondary cluster. The process includes the following steps:

1. Configure a TiDB primary cluster and a TiDB or MySQL secondary cluster.
2. Replicate incremental data from the primary cluster to the secondary cluster.
3. Recover data consistently by using Redo log when the primary cluster is down.

To replicate incremental data from a running TiDB cluster to its secondary cluster, you can use Backup & Restore BR and TiCDC.

8.11.1 Step 1. Set up the environment

1. Deploy TiDB clusters.

   Deploy two TiDB clusters, one upstream and the other downstream by using TiUP Playground. For production environments, deploy the clusters by referring to Deploy and Maintain an Online TiDB Cluster Using TiUP.

   In this document, we deploy the two clusters on two machines:

   - Node A: 172.16.6.123, for deploying the upstream TiDB cluster
   - Node B: 172.16.6.124, for deploying the downstream TiDB cluster

```
# Create an upstream cluster on Node A
tiup --tag upstream playground --host 0.0.0.0 --db 1 --pd 1 --kv 1 --
    ↪ tiflash 0 --ticdc 1
# Create a downstream cluster on Node B
tiup --tag downstream playground --host 0.0.0.0 --db 1 --pd 1 --kv 1 --
    ↪ tiflash 0 --ticdc 0
# View cluster status
tiup status
```

2. Initialize data.

   By default, test databases are created in the newly deployed clusters. Therefore, you can use sysbench to generate test data and simulate data in real scenarios.

```
sysbench oltp_write_only --config-file=./tidb-config --tables=10 --
    ↪ table-size=10000 prepare
```

   In this document, we use sysbench to run the `oltp_write_only` script. This script generates 10 tables in the upstream database, each with 10,000 rows. The tidb-config is as follows:

```
mysql-host=172.16.6.122 # Replace it with the IP address of your
    ↪ upstream cluster
mysql-port=4000
mysql-user=root
mysql-password=
db-driver=mysql      # Set database driver to MySQL
mysql-db=test        # Set the database as a test database
report-interval=10   # Set data collection period to 10s
threads=10           # Set the number of worker threads to 10
time=0               # Set the time required for executing the script.
    ↪ 0 indicates time unlimited
rate=100             # Set average TPS to 100
```

3. Simulate service workload.

   In real scenarios, service data is continuously written to the upstream cluster. In this document, we use sysbench to simulate this workload. Specifically, run the following command to enable 10 workers to continuously write data to three tables, sbtest1, sbtest2, and sbtest3, with a total TPS not exceeding 100.

```
sysbench oltp_write_only --config-file=./tidb-config --tables=3 run
```

4. Prepare external storage.

   In full data backup, both the upstream and downstream clusters need to access backup files. It is recommended that you use External storage to store backup files. In this example, Minio is used to simulate an S3-compatible storage service.

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
# Configure access-key access-screct-id to access minio
export HOST_IP='172.16.6.123' # Replace it with the IP address of your
    ↪ upstream cluster
export MINIO_ROOT_USER='minio'
export MINIO_ROOT_PASSWORD='miniostorage'
# Create the redo and backup directories. `backup` and `redo` are
    ↪ bucket names.
```

```
mkdir -p data/redo
mkdir -p data/backup
# Start minio at port 6060
nohup ./minio server ./data --address :6060 &
```

The preceding command starts a minio server on one node to simulate S3 services. Parameters in the command are configured as follows:

- Endpoint: `http://${HOST_IP}:6060/`
- Access-key: `minio`
- Secret-access-key: `miniostorage`
- Bucket: `redo`

The link is as follows:

```
s3://backup?access-key=minio&secret-access-key=miniostorage&endpoint=
    ↪ http://${HOST_IP}:6060&force-path-style=true
```

8.11.2 Step 2. Migrate full data

After setting up the environment, you can use the backup and restore functions of BR) to migrate full data. BR can be started in three ways. In this document, we use the SQL statements, `BACKUP` and `RESTORE`.

Note:

- `BACKUP` and `RESTORE` SQL statements are experimental. It is not recommended that you use them in the production environment. They might be changed or removed without prior notice. If you find a bug, you can report an issue on GitHub.
- In production clusters, performing a backup with GC disabled might affect cluster performance. It is recommended that you back up data in off-peak hours, and set RATE_LIMIT to a proper value to avoid performance degradation.
- If the versions of the upstream and downstream clusters are different, you should check BR compatibility. In this document, we assume that the upstream and downstream clusters are the same version.

1. Disable GC.

   To ensure that newly written data is not deleted during incremental migration, you should disable GC for the upstream cluster before backup. In this way, history data is not deleted.

Run the following command to disable GC:

```
MySQL [test]> SET GLOBAL tidb_gc_enable=FALSE;
```

```
Query OK, 0 rows affected (0.01 sec)
```

To verify that the change takes effect, query the value of `tidb_gc_enable`:

```
MySQL [test]> SELECT @@global.tidb_gc_enable;
```

```
+-------------------------+
| @@global.tidb_gc_enable |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set (0.00 sec)
```

2. Back up data.

   Run the `BACKUP` statement in the upstream cluster to back up data:

```
MySQL [(none)]> BACKUP DATABASE * TO 's3://backup?access-key=minio&
    ↪ secret-access-key=miniostorage&endpoint=http://${HOST_IP}:6060&
    ↪ force-path-style=true' RATE_LIMIT = 120 MB/SECOND;
```

```
+---------------------+----------+--------------------+--------------------+-----
    ↪
| Destination         | Size     | BackupTS           | Queue Time         |
    ↪ Execution Time    |
+---------------------+----------+--------------------+--------------------+-----
    ↪
| local:///tmp/backup/ | 10315858 | 431434047157698561 | 2022-02-25
    ↪ 19:57:59 | 2022-02-25 19:57:59 |
+---------------------+----------+--------------------+--------------------+-----
    ↪
1 row in set (2.11 sec)
```

   After the `BACKUP` command is executed, TiDB returns metadata about the backup data. Pay attention to `BackupTS`, because data generated before it is backed up. In this document, we use `BackupTS` as **the end of data check** and **the start of incremental migration scanning by TiCDC**.

3. Restore data.

   Run the `RESTORE` command in the downstream cluster to restore data:

```
mysql> RESTORE DATABASE * FROM 's3://backup?access-key=minio&secret-
    ↪ access-key=miniostorage&endpoint=http://${HOST_IP}:6060&force-
    ↪ path-style=true';
```

```
+---------------------+---------+-------------------+---------------------+-----
   ↪
| Destination         | Size    | BackupTS          | Queue Time          |
   ↪ Execution Time    |
+---------------------+---------+-------------------+---------------------+-----
   ↪
| local:///tmp/backup/ | 10315858 | 431434141450371074 | 2022-02-25
   ↪ 20:03:59 | 2022-02-25 20:03:59 |
+---------------------+---------+-------------------+---------------------+-----
   ↪
1 row in set (41.85 sec)
```

4. (Optional) Validate data.

   Use sync-diff-inspector to check data consistency between upstream and downstream at a certain time. The preceding `BACKUP` output shows that the upstream cluster finishes backup at 431434047157698561. The preceding `RESTORE` output shows that the downstream finishes restoration at 431434141450371074.

```
sync_diff_inspector -C ./config.yaml
```

   For details about how to configure the sync-diff-inspector, see Configuration file description. In this document, the configuration is as follows:

```
# Diff Configuration.
######################### Global config #########################
check-thread-count = 4
export-fix-sql = true
check-struct-only = false

######################### Datasource config #########################
[data-sources]
[data-sources.upstream]
       host = "172.16.6.123" # Replace it with the IP address of your
          ↪ upstream cluster
       port = 4000
       user = "root"
       password = ""
       snapshot = "431434047157698561" # Set snapshot to the actual
          ↪ backup time
[data-sources.downstream]
       host = "172.16.6.124" # Replace the value with the IP address of
          ↪  your downstream cluster
       port = 4000
       user = "root"
       password = ""
```

```
        snapshot = "431434141450371074" # Set snapshot to the actual
            ↪ restore time

########################## Task config ###########################
[task]
        output-dir = "./output"
        source-instances = ["upstream"]
        target-instance = "downstream"
        target-check-tables = ["*.*"]
```

8.11.3 Step 3. Migrate incremental data

1. Deploy TiCDC.

   After finishing full data migration, deploy and configure a TiCDC to replicate incremental data. In production environments, deploy TiCDC as instructed in Deploy TiCDC. In this document, a TiCDC node has been started upon the creation of the test clusters. Therefore, we skip the step of deploying TiCDC and proceed with changefeed configuration.

2. Create a changefeed.

   Create a changefeed configuration file `changefeed.toml`.

   ```
   [consistent]
   # Consistency level, eventual means enabling consistent replication
   level = "eventual"
   # Use S3 to store redo logs. Other options are local and nfs.
   storage = "s3://redo?access-key=minio&secret-access-key=miniostorage&
       ↪ endpoint=http://172.16.6.125:6060&force-path-style=true"
   ```

   In the upstream cluster, run the following command to create a changefeed from the upstream to the downstream clusters:

   ```
   tiup cdc cli changefeed create --pd=http://172.16.6.122:2379 --sink-uri
       ↪ ="mysql://root:@172.16.6.125:4000" --changefeed-id="primary-to-
       ↪ secondary" --start-ts="431434047157698561"
   ```

   In this command, the parameters are as follows:

   - `--pd`: PD address of the upstream cluster
   - `--sink-uri`: URI of the downstream cluster
   - `--start-ts`: start timestamp of the changefeed, must be the backup time (or BackupTS mentioned in Step 2. Migrate full data)

   For more information about the changefeed configurations, see Task configuration file.

3. Enable GC.

   In incremental migration using TiCDC, GC only removes history data that is replicated. Therefore, after creating a changefeed, you need to run the following command to enable GC. For details, see What is the complete behavior of TiCDC garbage collection (GC) safepoint?.

   To enable GC, run the following command:

```
MySQL [test]> SET GLOBAL tidb_gc_enable=TRUE;
```

```
Query OK, 0 rows affected (0.01 sec)
```

   To verify that the change takes effect, query the value of `tidb_gc_enable`:

```
MySQL [test]> SELECT @@global.tidb_gc_enable;
```

```
+-------------------------+
| @@global.tidb_gc_enable |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)
```

8.11.4 Step 4. Simulate a disaster in the upstream cluster

Create a disastrous event in the upstream cluster while it is running. For example, you can terminate the tiup playground process by pressing Ctrl+C.

8.11.5 Step 5. Use redo log to ensure data consistency

Normally, TiCDC concurrently writes transactions to downstream to increase throughout. When a changefeed is interrupted unexpectedly, the downstream may not have the latest data as it is in the upstream. To address inconsistency, run the following command to ensure that the downstream data is consistent with the upstream data.

```
tiup cdc redo apply --storage "s3://redo?access-key=minio&secret-access-key=
    ↪ miniostorage&endpoint=http://172.16.6.123:6060&force-path-style=true"
    ↪  --tmp-dir /tmp/redo --sink-uri "mysql://root:@172.16.6.124:4000"
```

- `--storage`: Location and credential of the redo log in S3
- `--tmp-dir`: Cache directory of the redo log downloaded from S3
- `--sink-uri`: URI of the downstream cluster

8.11.6 Step 6. Recover the primary cluster and its services

After the previous step, the downstream (secondary) cluster has data that is consistent with the upstream (primary) cluster at a specific time. You need to set up new primary and secondary clusters to ensure data reliability.

1. Deploy a new TiDB cluster on Node A as the new primary cluster.

```
tiup --tag upstream playground v5.4.0 --host 0.0.0.0 --db 1 --pd 1 --kv
    ↪  1 --tiflash 0 --ticdc 1
```

2. Use BR to back up and restore data fully from the secondary cluster to the primary cluster.

```
# Back up full data of the secondary cluster
tiup br --pd http://172.16.6.124:2379 backup full --storage ./backup
# Restore full data of the secondary cluster
tiup br --pd http://172.16.6.123:2379 restore full --storage ./backup
```

3. Create a new changefeed to back up data from the primary cluster to the secondary cluster.

```
# Create a changefeed
tiup cdc cli changefeed create --pd=http://172.16.6.122:2379 --sink-uri
    ↪ ="mysql://root:@172.16.6.125:4000" --changefeed-id="primary-to-
    ↪ secondary"
```

9 Monitor and Alert

9.1 TiDB Monitoring Framework Overview

The TiDB monitoring framework adopts two open source projects: Prometheus and Grafana. TiDB uses Prometheus to store the monitoring and performance metrics and Grafana to visualize these metrics.

9.1.1 About Prometheus in TiDB

As a time series database, Prometheus has a multi-dimensional data model and flexible query language. As one of the most popular open source projects, Prometheus has been adopted by many companies and organizations and has a very active community. PingCAP is one of the active developers and adopters of Prometheus for monitoring and alerting in TiDB, TiKV and PD.

Prometheus consists of multiple components. Currently, TiDB uses the following of them:

- The Prometheus Server to scrape and store time series data
- The client libraries to customize necessary metrics in the application
- An Alertmanager for the alerting mechanism

The diagram is as follows:

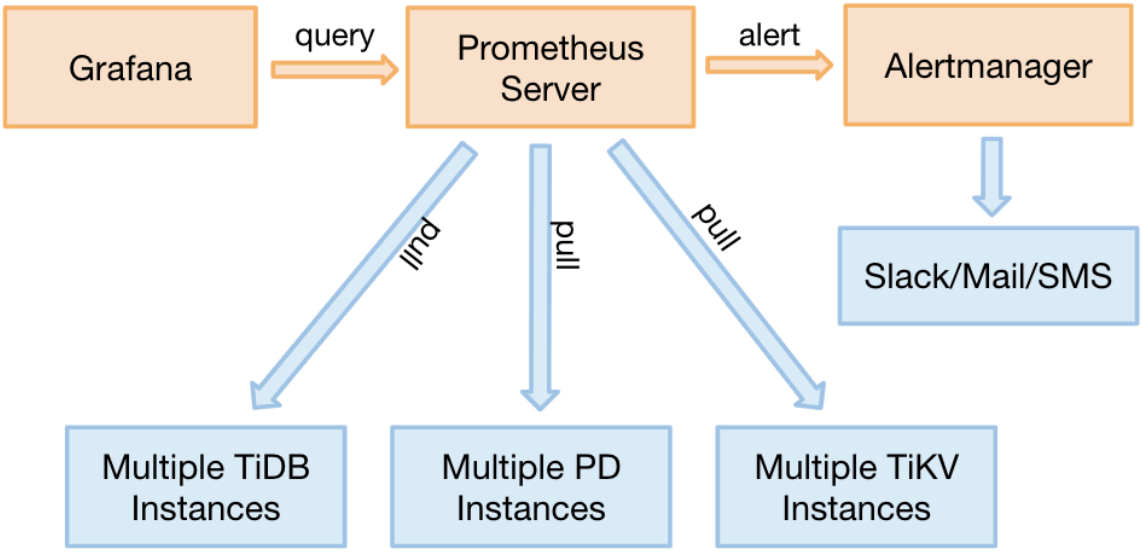


Figure 93: diagram

9.1.2 About Grafana in TiDB

Grafana is an open source project for analyzing and visualizing metrics. TiDB uses Grafana to display the performance metrics as follows:

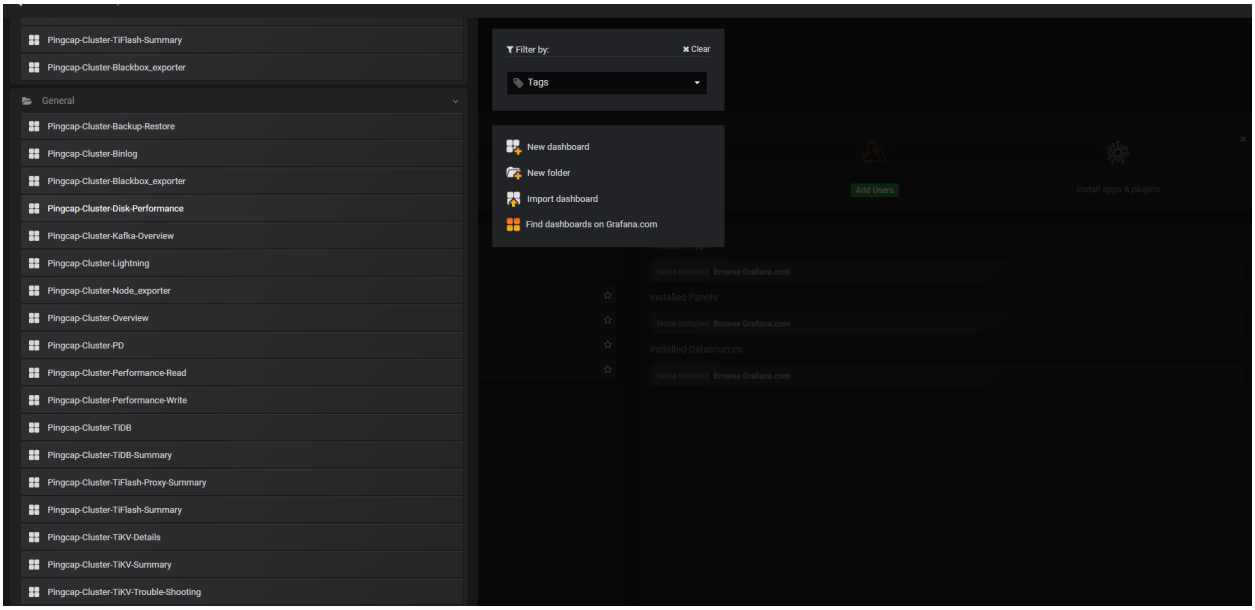


Figure 94: Grafana monitored_groups

- {TiDB_Cluster_name}-Backup-Restore: Monitoring metrics related to backup and restore.
- {TiDB_Cluster_name}-Binlog: Monitoring metrics related to TiDB Binlog.
- {TiDB_Cluster_name}-Blackbox_exporter: Monitoring metrics related to network probe.
- {TiDB_Cluster_name}-Disk-Performance: Monitoring metrics related to disk performance.
- {TiDB_Cluster_name}-Kafka-Overview: Monitoring metrics related to Kafka.
- {TiDB_Cluster_name}-Lightning: Monitoring metrics related to TiDB Lightning.
- {TiDB_Cluster_name}-Node_exporter: Monitoring metrics related to the operating system.
- {TiDB_Cluster_name}-Overview: Monitoring overview related to important components.
- {TiDB_Cluster_name}-PD: Monitoring metrics related to the PD server.
- {TiDB_Cluster_name}-Performance-Read: Monitoring metrics related to read performance.
- {TiDB_Cluster_name}-Performance-Write: Monitoring metrics related to write performance.
- {TiDB_Cluster_name}-TiDB: Detailed monitoring metrics related to the TiDB server.
- {TiDB_Cluster_name}-TiDB-Summary: Monitoring overview related to TiDB.
- {TiDB_Cluster_name}-TiFlash-Proxy-Summary: Monitoring overview of the proxy server that is used to replicate data to TiFlash.
- {TiDB_Cluster_name}-TiFlash-Summary: Monitoring overview related to TiFlash.
- {TiDB_Cluster_name}-TiKV-Details: Detailed monitoring metrics related to the

TiKV server.

- {TiDB_Cluster_name}-TiKV-Summary: Monitoring overview related to the TiKV server.
- {TiDB_Cluster_name}-TiKV-Trouble-Shooting: Monitoring metrics related to the TiKV error diagnostics.
- {TiDB_Cluster_name}-TiCDC: Detailed monitoring metrics related to TiCDC.

Each group has multiple panel labels of monitoring metrics, and each panel contains detailed information of multiple monitoring metrics. For example, the **Overview** monitoring group has five panel labels, and each labels corresponds to a monitoring panel. See the following UI:



Figure 95: Grafana Overview

9.2 TiDB Monitoring API

You can use the following types of interfaces to monitor the TiDB cluster status:

- The status interface: this interface uses the HTTP interface to get the component information. Using this interface, you can get the running status of the current TiDB server and the storage information of a table.
- The metrics interface: this interface uses Prometheus to record the detailed information of the various operations in components and views these metrics using Grafana.

9.2.1 Use the status interface

The status interface monitors the basic information of a specific component in the TiDB cluster. It can also act as the monitor interface for Keepalive messages. In addition, the status interface for the Placement Driver (PD) can get the details of the entire TiKV cluster.

9.2.1.1 TiDB server

- TiDB API address: `http://${host}:${port}`
- Default port: `10080`

9.2.1.2 Running status

The following example uses `http://${host}:${port}/status` to get the current status of the TiDB server and to determine whether the server is alive. The result is returned in **JSON** format.

```
curl http://127.0.0.1:10080/status
{
    connections: 0, # The current number of clients connected to the TiDB
        ↪ server.
    version: "5.7.25-TiDB-v3.0.0-beta-250-g778c3f4a5", # The TiDB version
        ↪ number.
    git_hash: "778c3f4a5a716880bcd1d71b257c8165685f0d70" # The Git Hash of
        ↪ the current TiDB code.
}
```

9.2.1.2.1 Storage information

The following example uses `http://${host}:${port}/schema_storage/${db}/${table}` to get the storage information of the specific data table. The result is returned in **JSON** format.

```
curl http://127.0.0.1:10080/schema_storage/mysql/stats_histograms
```

```
{
    "table_schema": "mysql",
    "table_name": "stats_histograms",
    "table_rows": 0,
```

```
    "avg_row_length": 0,
    "data_length": 0,
    "max_data_length": 0,
    "index_length": 0,
    "data_free": 0
}
```

```
curl http://127.0.0.1:10080/schema_storage/test
```

```
[
    {
        "table_schema": "test",
        "table_name": "test",
        "table_rows": 0,
        "avg_row_length": 0,
        "data_length": 0,
        "max_data_length": 0,
        "index_length": 0,
        "data_free": 0
    }
]
```

9.2.1.3 PD server

- PD API address: `http://${host}:${port}/pd/api/v1/${api_name}`
- Default port: 2379
- Details about API names: see PD API doc

The PD interface provides the status of all the TiKV servers and the information about load balancing. See the following example for the information about a single-node TiKV cluster:

```
curl http://127.0.0.1:2379/pd/api/v1/stores
{
  "count": 1, # The number of TiKV nodes.
  "stores": [ # The list of TiKV nodes.
    # The details about the single TiKV node.
    {
      "store": {
        "id": 1,
        "address": "127.0.0.1:20160",
        "version": "3.0.0-beta",
        "state_name": "Up"
      },
```

```
    "status": {
      "capacity": "20 GiB", # The total capacity.
      "available": "16 GiB", # The available capacity.
      "leader_count": 17,
      "leader_weight": 1,
      "leader_score": 17,
      "leader_size": 17,
      "region_count": 17,
      "region_weight": 1,
      "region_score": 17,
      "region_size": 17,
      "start_ts": "2019-03-21T14:09:32+08:00", # The starting timestamp.
      "last_heartbeat_ts": "2019-03-21T14:14:22.961171958+08:00", # The
          ↪ timestamp of the last heartbeat.
      "uptime": "4m50.961171958s"
    }
  }
]
```

9.2.2 Use the metrics interface

The metrics interface monitors the status and performance of the entire TiDB cluster.

- If you use other deployment ways, deploy Prometheus and Grafana before using this interface.

After Prometheus and Grafana are successfully deployed, configure Grafana.

9.3 Deploy Monitoring Services for the TiDB Cluster

This document is intended for users who want to manually deploy TiDB monitoring and alert services.

If you deploy the TiDB cluster using TiUP, the monitoring and alert services are automatically deployed, and no manual deployment is needed.

9.3.1 Deploy Prometheus and Grafana

Assume that the TiDB cluster topology is as follows:

Name	Host IP	Services
Node1	192.168.199.113	PD1, TiDB, node_export, Prometheus, Grafana
Node2	192.168.199.114	PD2, node_export

Name	Host IP	Services
Node3	192.168.199.115	PD3, node_export
Node4	192.168.199.116	TiKV1, node_export
Node5	192.168.199.117	TiKV2, node_export
Node6	192.168.199.118	TiKV3, node_export

9.3.1.1 Step 1: Download the binary package

```
## Downloads the package.
wget https://download.pingcap.org/prometheus-2.27.1.linux-amd64.tar.gz
wget https://download.pingcap.org/node_exporter-0.17.0.linux-amd64.tar.gz
wget https://download.pingcap.org/grafana-7.5.11.linux-amd64.tar.gz
```

```
## Extracts the package.
tar -xzf prometheus-2.27.1.linux-amd64.tar.gz
tar -xzf node_exporter-0.17.0.linux-amd64.tar.gz
tar -xzf grafana-7.5.11.linux-amd64.tar.gz
```

9.3.1.2 Step 2: Start `node_exporter` on Node1, Node2, Node3, and Node4

```
cd node_exporter-0.17.0.linux-amd64

## Starts the node_exporter service.
$ ./node_exporter --web.listen-address=":9100" \
    --log.level="info" &
```

9.3.1.3 Step 3: Start Prometheus on Node1

Edit the Prometheus configuration file:

```
cd prometheus-2.27.1.linux-amd64 &&
vi prometheus.yml
```

```
...

global:
  scrape_interval:  15s # By default, scrape targets every 15 seconds.
  evaluation_interval: 15s # By default, scrape targets every 15 seconds.
  # scrape_timeout is set to the global default value (10s).
  external_labels:
    cluster: 'test-cluster'
    monitor: "prometheus"

scrape_configs:
```

```
  - job_name: 'overwritten-nodes'
    honor_labels: true # Do not overwrite job & instance labels.
    static_configs:
    - targets:
      - '192.168.199.113:9100'
      - '192.168.199.114:9100'
      - '192.168.199.115:9100'
      - '192.168.199.116:9100'
      - '192.168.199.117:9100'
      - '192.168.199.118:9100'

  - job_name: 'tidb'
    honor_labels: true # Do not overwrite job & instance labels.
    static_configs:
    - targets:
      - '192.168.199.113:10080'

  - job_name: 'pd'
    honor_labels: true # Do not overwrite job & instance labels.
    static_configs:
    - targets:
      - '192.168.199.113:2379'
      - '192.168.199.114:2379'
      - '192.168.199.115:2379'

  - job_name: 'tikv'
    honor_labels: true # Do not overwrite job & instance labels.
    static_configs:
    - targets:
      - '192.168.199.116:20180'
      - '192.168.199.117:20180'
      - '192.168.199.118:20180'

...
```

Start the Prometheus service:

```
$ ./prometheus \
    --config.file="./prometheus.yml" \
    --web.listen-address=":9090" \
    --web.external-url="http://192.168.199.113:9090/" \
    --web.enable-admin-api \
    --log.level="info" \
    --storage.tsdb.path="./data.metrics" \
    --storage.tsdb.retention="15d" &
```

9.3.1.4 Step 4: Start Grafana on Node1

Edit the Grafana configuration file:

```
cd grafana-7.5.11 &&
vi conf/grafana.ini

...

[paths]
data = ./data
logs = ./data/log
plugins = ./data/plugins
[server]
http_port = 3000
domain = 192.168.199.113
[database]
[session]
[analytics]
check_for_updates = true
[security]
admin_user = admin
admin_password = admin
[snapshots]
[users]
[auth.anonymous]
[auth.basic]
[auth.ldap]
[smtp]
[emails]
[log]
mode = file
[log.console]
[log.file]
level = info
format = text
[log.syslog]
[event_publisher]
[dashboards.json]
enabled = false
path = ./data/dashboards
[metrics]
[grafana_net]
url = https://grafana.net

...
```

Start the Grafana service:

```
./bin/grafana-server \
    --config="./conf/grafana.ini" &
```

9.3.2 Configure Grafana

This section describes how to configure Grafana.

9.3.2.1 Step 1: Add a Prometheus data source

1. Log in to the Grafana Web interface.

    - Default address: http://localhost:3000
    - Default account: admin
    - Default password: admin

> **Note:**
> For the **Change Password** step, you can choose **Skip**.

2. In the Grafana sidebar menu, click **Data Source** within the **Configuration**.
3. Click **Add data source**.
4. Specify the data source information.

    - Specify a **Name** for the data source.
    - For **Type**, select **Prometheus**.
    - For **URL**, specify the Prometheus address.
    - Specify other fields as needed.

5. Click **Add** to save the new data source.

9.3.2.2 Step 2: Import a Grafana dashboard

To import a Grafana dashboard for the PD server, the TiKV server, and the TiDB server, take the following steps respectively:

1. Click the Grafana logo to open the sidebar menu.
2. In the sidebar menu, click **Dashboards** -> **Import** to open the **Import Dashboard** window.

3. Click **Upload .json File** to upload a JSON file (Download TiDB Grafana configuration files from pingcap/tidb, tikv/tikv, and tikv/pd).

> **Note:**
> For the TiKV, PD, and TiDB dashboards, the corresponding JSON files are `tikv_summary.json`, `tikv_details.json`, `tikv_trouble_shooting.json`, `pd.json`, `tidb.json`, and `tidb_summary.json`.

4. Click **Load**.
5. Select a Prometheus data source.
6. Click **Import**. A Prometheus dashboard is imported.

9.3.3 View component metrics

Click **New dashboard** in the top menu and choose the dashboard you want to view.

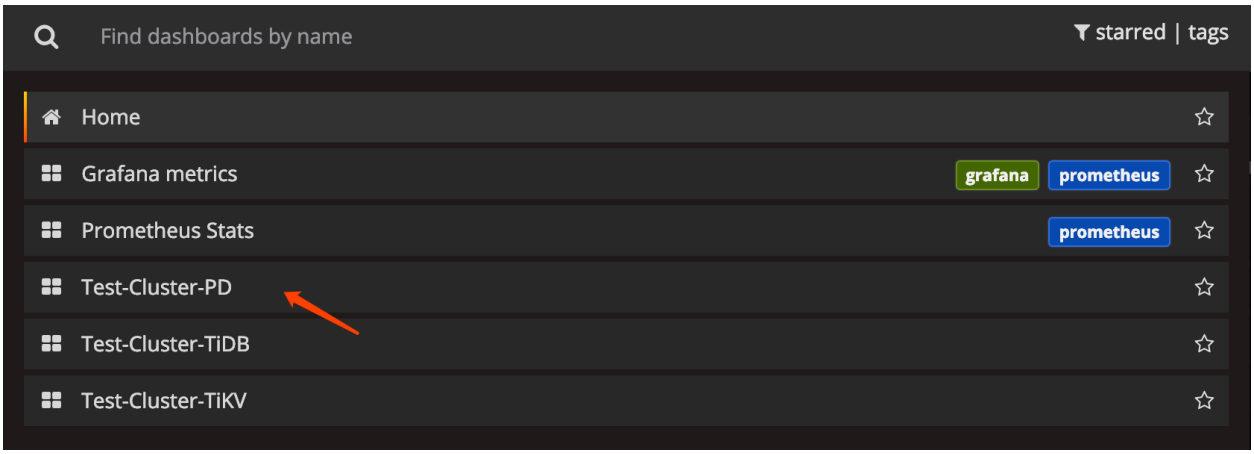


Figure 96: view dashboard

You can get the following metrics for cluster components:

- **TiDB server:**
  - Query processing time to monitor the latency and throughput
  - The DDL process monitoring
  - TiKV client related monitoring
  - PD client related monitoring
- **PD server:**

- The total number of times that the command executes
- The total number of times that a certain command fails
- The duration that a command succeeds
- The duration that a command fails
- The duration that a command finishes and returns result

- **TiKV server:**

  - Garbage Collection (GC) monitoring
  - The total number of times that the TiKV command executes
  - The duration that Scheduler executes commands
  - The total number of times of the Raft propose command
  - The duration that Raft executes commands
  - The total number of times that Raft commands fail
  - The total number of times that Raft processes the ready state

9.4 Upgrade TiDB Cluster Monitoring Services

When deploying a TiDB cluster, TiUP automatically deploys monitoring services (such as Prometheus, Grafana, and Alertmanager) for the cluster. If you scale out this cluster, TiUP also automatically adds monitoring configurations for newly added nodes during the scaling. The monitoring services automatically deployed by TiUP are usually not the latest versions of these third-party monitoring services. To use the latest versions, you can follow this document to upgrade the monitoring services.

When managing a cluster, TiUP uses its own configurations to override the configurations of the monitoring services. If you directly upgrade the monitoring services by replacing their configuration files, any subsequent TiUP operations such as `deploy`, `scale-out`, `scale-in`, and `reload` on the cluster might overwrite your upgrade, leading to errors. To upgrade Prometheus, Grafana, and Alertmanager, follow the steps in this document rather than directly replacing configuration files.

Note:

- If your monitoring services are deployed manually instead of using TiUP, you can directly upgrade them without referring to this document.
- The TiDB compatibility with newer versions of monitoring services has not been tested, so some features might not work as expected after the upgrade. For any issues, create an issue on GitHub.
- The upgrade steps in this document are applicable for TiUP version 1.9.0 and later. Therefore, check your TiUP version before the upgrade.
- When you use TiUP to upgrade the TiDB cluster, TiUP will redeploy the monitoring services to the default version. You need to redo the upgrade for monitoring services after the TiDB upgrade.

9.4.1 Upgrade Prometheus

For better compatibility with TiDB, it is recommended to use the Prometheus installation package provided in the TiDB installation package. The version of Prometheus in the TiDB installation package is fixed. If you want to use a newer Prometheus version, refer to Prometheus Release Notes for new features of each version and choose a suitable version for your production environment. You can also consult with PingCAP technical staff for a recommended version.

In the following upgrade steps, you need to download the Prometheus installation package of your desired version from the Prometheus website, and then use it to create a Prometheus package that TiUP can use.

9.4.1.1 Step 1. Download a new Prometheus installation package from the Prometheus website

Download a new installation package from the Prometheus download page and extract it.

9.4.1.2 Step 2. Download the Prometheus installation package provided by TiDB

1. Download the TiDB server package and extract it. Note that your downloading means you agree to the Privacy Policy.

```
https://download.pingcap.org/tidb-community-server-{version}-linux-{
    ↪ arch}.tar.gz
```

> **Tip:**
>
> `{version}` in the link indicates the version number of TiDB and `{arch ↪ }` indicates the architecture of the system, which can be `amd64` or `arm64`. For example, the download link for `v6.1.0` in the `amd64` architecture is `https://download.pingcap.org/tidb-community-toolkit-v6 ↪ .1.0-linux-amd64.tar.gz`.

2. In the extracted files, locate `prometheus-v{version}-linux-amd64.tar.gz` and extract it.

```
tar -xzf prometheus-v{version}-linux-amd64.tar.gz
```

9.4.1.3 Step 3. Create a new Prometheus package that TiUP can use

1. Copy the files extracted in Step 1, and then use the copied files to replace the files in the `./prometheus-v{version}-linux-amd64/prometheus` directory extracted in Step 2.

2. Recompress the `./prometheus-v{version}-linux-amd64` directory and name the new compressed package as `prometheus-v{new-version}.tar.gz`, where `{new-version}` can be specified according to your need.

```
cd prometheus-v{version}-linux-amd64
tar -zcvf ../prometheus-v{new-version}.tar.gz ./
```

9.4.1.4 Step 4. Upgrade Prometheus using the newly created Prometheus package

Execute the following command to upgrade Prometheus:

```
tiup cluster patch <cluster-name> prometheus-v{new-version}.tar.gz -R
    ↪ prometheus --overwrite
```

After the upgrade, you can go to the home page of the Prometheus server (usually at `http://<Prometheus-server-host-name>:9090`), click **Status** in the top navigation menu, and then open the **Runtime & Build Information** page to check the Prometheus version and confirm whether the upgrade is successful.

9.4.2 Upgrade Grafana

For better compatibility with TiDB, it is recommended to use the Grafana installation package provided in the TiDB installation package. The version of Grafana in the TiDB installation package is fixed. If you want to use a newer Grafana version, refer to Grafana Release Notes for new features of each version and choose a suitable version for your production environment. You can also consult with PingCAP technical staff for a recommended version.

In the following upgrade steps, you need to download the Grafana installation package of your desired version from the Grafana website, and then use it to create a Grafana package that TiUP can use.

9.4.2.1 Step 1. Download a new Grafana installation package from the Grafana website

1. Download a new installation package from the Grafana download page. You can choose either the `OSS` or `Enterprise` edition according to your needs.
2. Extract the downloaded package.

9.4.2.2 Step 2. Download the Grafana installation package provided by TiDB

1. Download the TiDB server package and extract it. Note that your downloading means you agree to the Privacy Policy.

```
https://download.pingcap.org/tidb-community-server-{version}-linux-{
    ↪ arch}.tar.gz
```

> **Tip:**
>
> `{version}` in the link indicates the version number of TiDB and `{arch}` indicates the architecture of the system, which can be `amd64` or `arm64`. For example, the download link for `v6.1.0` in the `amd64` architecture is `https://download.pingcap.org/tidb-community-toolkit-v6.1.0-linux-amd64.tar.gz`.

2. In the extracted files, locate `grafana-v{version}-linux-amd64.tar.gz` and extract it.

```
tar -xzf grafana-v{version}-linux-amd64.tar.gz
```

9.4.2.3 Step 3. Create a new Grafana package that TiUP can use

1. Copy the files extracted in Step 1, and then use the copied files to replace the files in the `./grafana-v{version}-linux-amd64/` directory extracted in Step 2.

2. Recompress the `./grafana-v{version}-linux-amd64` directory and name the new compressed package as `grafana-v{new-version}.tar.gz`, where `{new-version}` can be specified according to your need.

```
cd grafana-v{version}-linux-amd64
tar -zcvf ../grafana-v{new-version}.tar.gz ./
```

9.4.2.4 Step 4. Upgrade Grafana using the newly created Grafana package

Execute the following command to upgrade Grafana:

```
tiup cluster patch <cluster-name> grafana-v{new-version}.tar.gz -R grafana
    ↪ --overwrite
```

After the upgrade, you can go to the home page of the Grafana server (usually at `http://<Grafana-server-host-name>:3000`), and then check the Grafana version on the page to confirm whether the upgrade is successful.

9.4.3 Upgrade Alertmanager

The Alertmanager package in the TiDB installation package is directly from the Prometheus website. Therefore, when upgrading Alertmanager, you only need to download and install a new version of Alertmanager from the Prometheus website.

9.4.3.1 Step 1. Download a new Alertmanager installation package from the Prometheus website

Download the `alertmanager` installation package from the Prometheus download page.

9.4.3.2 Step 2. Upgrade Alertmanager using the downloaded installation package

Execute the following command to upgrade Alertmanager:

```
tiup cluster patch <cluster-name> alertmanager-v{new-version}-linux-amd64.
    ↪ tar.gz -R alertmanager --overwrite
```

After the upgrade, you can go to the home page of the Alertmanager server (usually at `http://<Alertmanager-server-host-name>:9093`), click **Status** in the top navigation menu, and then check the Alertmanager version to confirm whether the upgrade is successful.

Warning:

- Since TiDB v6.0.0, PingCAP no longer maintains MetricsTool. Since v6.1.0, PingCAP no longer maintains the MetricsTool document.
- To export monitoring metrics data, use the PingCAP Clinic diagnostic service to get the information required for diagnosing a TiDB cluster, including the monitoring metrics, logs, cluster topology, configuration, and parameters.

9.5 Export Grafana Snapshots

Note:

Currently, MetricsTool can only be used with Grafana v6.x.x.

Metrics data is important in troubleshooting. When you request remote assistance, sometimes the support staff need to view the Grafana dashboards to diagnose problems. MetricsTool can help export snapshots of Grafana dashboards as local files and visualize these snapshots. You can share these snapshots with outsiders and allow them to accurately read out the graphs, without giving out access to other sensitive information on the Grafana server.

9.5.1 Usage

MetricsTool can be accessed from https://metricstool.pingcap.net/. It consists of three sets of tools:

- **Export**: A user script running on the browser's Developer Tool, allowing you to download a snapshot of all visible panels in the current dashboard on any Grafana v6.x.x server.

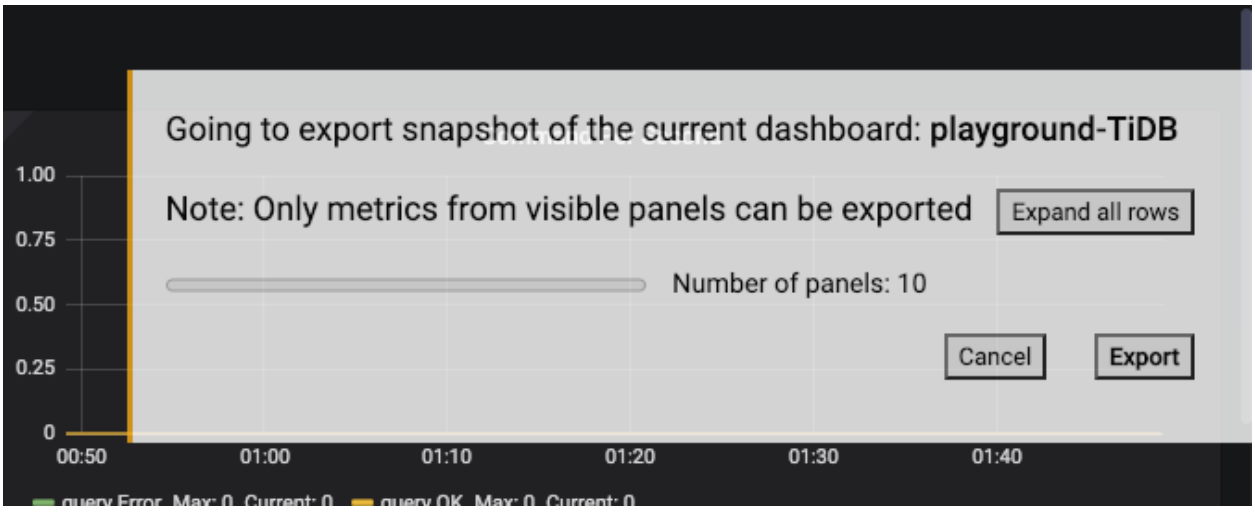


Figure 97: Screenshot of MetricsTool Exporter after running the user script

- **Visualize**: A web page visualizing the exported snapshot files. The visualized snapshots can be operated in the same way as live Grafana dashboards.

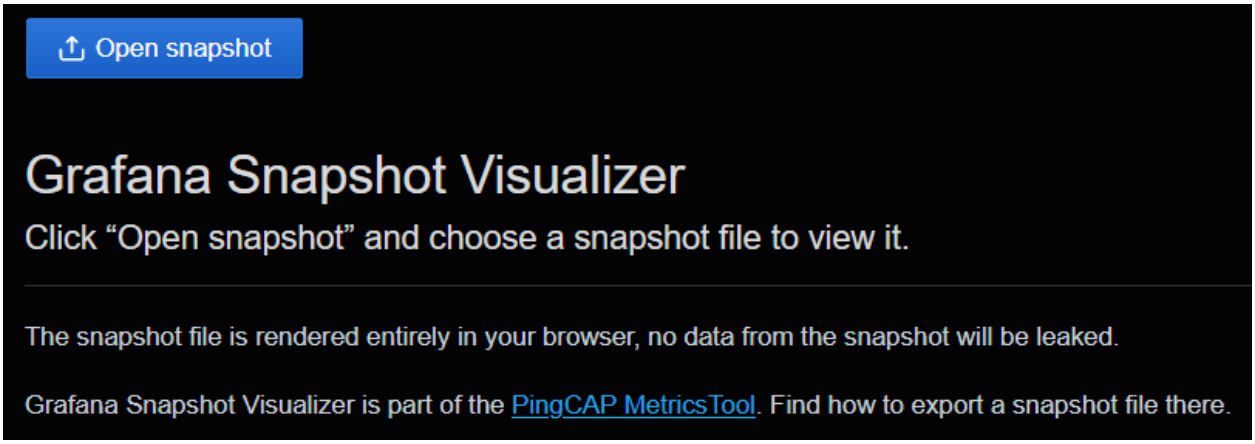


Figure 98: Screenshot of MetricsTool Visualizer

- **Import**: Instructions to import the exported snapshot back into an actual Grafana instance.

9.5.2 FAQs

9.5.2.1 What is the advantage of this tool compared with screenshot or PDF printing?

The snapshot files exported by MetricsTool contain the actual values when they are taken. And the Visualizer allows you to interact with the rendered graphs as if it is a live Grafana dashboard, supporting operations like toggling series, zooming into a smaller time range, and checking the precise value at a given time. This makes MetricsTool much more powerful than images or PDFs.

9.5.2.2 What are included in the snapshot file?

The snapshot file contains the values of all graphs and panels in the selected time range. It does not save the original metrics from the data sources (and thus you cannot edit the query expression in the Visualizer).

9.5.2.3 Will the Visualizer save the uploaded snapshot files in PingCAP's servers?

No, the Visualizer parses the snapshot files entirely inside your browser. Nothing will be sent to PingCAP. You are free to view snapshot files received from sensitive sources, and no need to worry about these leaking to third parties through the Visualizer.

9.5.2.4 Will there be problems to execute the script before all metrics are loaded?

No, the script UI will notify you to wait for all metrics to be loaded. However, you can manually skip waiting and export the snapshot in case of some metrics loading for too long.

9.5.2.5 Can we share a link to a visualized snapshot?

No, but you can share the snapshot file, with instruction on how to use the Visualizer to view it. If you truly need a world-readable URL, you may also try the public `snapshot.raintank.io` service built into Grafana, but make sure all privacy concerns are cleared before doing so.

9.6 TiDB Cluster Alert Rules

This document describes the alert rules for different components in a TiDB cluster, including the rule descriptions and solutions of the alert items in TiDB, TiKV, PD, TiFlash, TiDB Binlog, TiCDC, Node_exporter and Blackbox_exporter.

According to the severity level, alert rules are divided into three categories (from high to low): emergency-level, critical-level, and warning-level. This division of severity levels applies to all alert items of each component below.

Severity level	Description
Emergency-level	The highest severity level at which the service is unavailable. Emergency-level alerts are often caused by a service or node failure. **Manual intervention is required immediately**.

Severity level	Description
Critical-level	Decreased service availability. For the critical-level alerts, a close watch on the abnormal metrics is required.
Warning-level	Warning-level alerts are a reminder for an issue or error.

9.6.1 TiDB alert rules

This section gives the alert rules for the TiDB component.

9.6.1.1 Emergency-level alerts

9.6.1.1.1 `TiDB_schema_error`

- Alert rule:

```
increase(tidb_session_schema_lease_error_total{type="outdated"}[15m])>
↪ 0
```

- Description:

The latest schema information is not reloaded in TiDB within one lease. When TiDB fails to continue providing services, an alert is triggered.

- Solution:

  It is often caused by an unavailable Region or a TiKV timeout. You need to locate the issue by checking the TiKV monitoring items.

9.6.1.1.2 TiDB_tikvclient_region_err_total

- Alert rule:

  `increase(tidb_tikvclient_region_err_total[10m])> 6000`

- Description:

  When TiDB accesses TiKV, a Region error occurs. When the error is reported over 6000 times in 10 minutes, an alert is triggered.

- Solution:

  View the monitoring status of TiKV.

9.6.1.1.3 TiDB_domain_load_schema_total

- Alert rule:

  `increase(tidb_domain_load_schema_total{type="failed"}[10m])> 10`

- Description:

  The total number of failures to reload the latest schema information in TiDB. If the reloading failure occurs over 10 times in 10 minutes, an alert is triggered.

- Solution:

  Same as TiDB_schema_error.

9.6.1.1.4 TiDB_monitor_keep_alive

- Alert rule:

  `increase(tidb_monitor_keep_alive_total[10m])< 100`

- Description:

  Indicates whether the TiDB process still exists. If the number of times for `tidb_monitor_keep_alive_total` increases less than 100 in 10 minutes, the TiDB process might already exit and an alert is triggered.

- Solution:

  - Check whether the TiDB process is out of memory.
  - Check whether the machine has restarted.

9.6.1.2 Critical-level alerts

9.6.1.2.1 TiDB_server_panic_total

- Alert rule:

  `increase(tidb_server_panic_total[10m])> 0`

- Description:

  The number of panicked TiDB threads. When a panic occurs, an alert is triggered. The thread is often recovered, otherwise, TiDB will frequently restart.

- Solution:

  Collect the panic logs to locate the issue.

9.6.1.3 Warning-level alerts

9.6.1.3.1 TiDB_memory_abnormal

- Alert rule:

  `go_memstats_heap_inuse_bytes{job="tidb"} > 1e+10`

- Description:

  The monitoring on the TiDB memory usage. If the usage exceeds 10 G, an alert is triggered.

- Solution:

  Use the HTTP API to troubleshoot the goroutine leak issue.

9.6.1.3.2 TiDB_query_duration

- Alert rule:

  ```
  histogram_quantile(0.99, sum(rate(tidb_server_handle_query_duration_seconds_bucket
  ↪ [1m]))BY (le, instance))> 1
  ```

- Description:

  The latency of handling a request in TiDB. If the ninety-ninth percentile latency exceeds 1 second, an alert is triggered.

- Solution:

  View TiDB logs and search for the `SLOW_QUERY` and `TIME_COP_PROCESS` keywords to locate the slow SQL queries.

9.6.1.3.3 TiDB_server_event_error

- Alert rule:

```
increase(tidb_server_event_total{type=~"server_start|server_hang"}[15m
↪ ])> 0
```

- Description:

  The number of events that happen in the TiDB service. An alert is triggered when the following events happen:

  1. start: The TiDB service starts.
  2. hang: When a critical-level event (currently there is only one scenario: TiDB cannot write binlog) happens, TiDB enters the `hang` mode and waits to be killed manually.

- Solution:

  - Restart TiDB to recover the service.
  - Check whether the TiDB Binlog service is normal.

9.6.1.3.4 TiDB_tikvclient_backoff_seconds_count

- Alert rule:

```
increase(tidb_tikvclient_backoff_seconds_count[10m])> 10
```

- Description:

  The number of retries when TiDB fails to access TiKV. When the retry times is over 10 in 10 minutes, an alert is triggered.

- Solution:

  View the monitoring status of TiKV.

9.6.1.3.5 TiDB_monitor_time_jump_back_error

- Alert rule:

```
increase(tidb_monitor_time_jump_back_total[10m])> 0
```

- Description:

  When the time of the machine that holds TiDB rewinds, an alert is triggered.

- Solution:

  Troubleshoot the NTP configurations.

9.6.1.3.6 `TiDB_ddl_waiting_jobs`

- Alert rule:

  ```
  sum(tidb_ddl_waiting_jobs)> 5
  ```

- Description:

  When the number of DDL tasks pending for execution in TiDB exceeds 5, an alert is triggered.

- Solution:

  Check whether there is any time-consuming `add index` operation that is being executed by running `admin show ddl`.

9.6.2 PD alert rules

This section gives the alert rules for the PD component.

9.6.2.1 Emergency-level alerts

9.6.2.1.1 `PD_cluster_down_store_nums`

- Alert rule:

  ```
  (sum(pd_cluster_status{type="store_down_count"})by (instance)> 0)and (
  ↪ sum(etcd_server_is_leader)by (instance)> 0)
  ```

- Description:

  PD has not received a TiKV/TiFlash heartbeat for a long time (the default configuration is 30 minutes).

- Solution:

  - Check whether the TiKV/TiFlash process is normal, the network is isolated or the load is too high, and recover the service as much as possible.
  - If the TiKV/TiFlash instance cannot be recovered, you can make it offline.

9.6.2.2 Critical-level alerts

9.6.2.2.1 PD_etcd_write_disk_latency

- Alert rule:

```
histogram_quantile(0.99, sum(rate(etcd_disk_wal_fsync_duration_seconds_bucket
↪ [1m]))by (instance, job, le))> 1
```

- Description:

  If the latency of the fsync operation exceeds 1 second, it indicates that etcd writes data to disk at a lower speed than normal. It might lead to PD leader timeout or failure to store TSO on disk in time, which will shut down the service of the entire cluster.

- Solution:

  - Find the cause of slow writes. It might be other services that overload the system. You can check whether PD itself occupies a large amount of CPU or I/O resources.
  - Try to restart PD or manually transfer leader to another PD to recover the service.
  - If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

9.6.2.2.2 PD_miss_peer_region_count

- Alert rule:

```
(sum(pd_regions_status{type="miss-peer-region-count"})by (instance)>
↪ 100)and (sum(etcd_server_is_leader)by (instance)> 0)
```

- Description:

  The number of Region replicas is smaller than the value of `max-replicas`.

- Solution:

  - Find the cause of the issue by checking whether there is any TiKV machine that is down or being made offline.
  - Watch the Region health panel and see whether `miss-peer-region-count` is continuously decreasing.

9.6.2.3 Warning-level alerts

9.6.2.3.1 PD_cluster_lost_connect_store_nums

- Alert rule:

```
(sum(pd_cluster_status{type="store_disconnected_count"})by (instance)>
↪ 0)and (sum(etcd_server_is_leader)by (instance)> 0)
```

- Description:

  PD does not receive a TiKV/TiFlash heartbeat within 20 seconds. Normally a TiKV/TiFlash heartbeat comes in every 10 seconds.

- Solution:

  - Check whether the TiKV/TiFlash instance is being restarted.
  - Check whether the TiKV/TiFlash process is normal, the network is isolated, and the load is too high, and recover the service as much as possible.
  - If you confirm that the TiKV/TiFlash instance cannot be recovered, you can make it offline.
  - If you confirm that the TiKV/TiFlash instance can be recovered, but not in the short term, you can consider increasing the value of `max-down-time`. It will prevent the TiKV/TiFlash instance from being considered as irrecoverable and the data from being removed from the TiKV/TiFlash.

9.6.2.3.2 PD_cluster_low_space

- Alert rule:

```
(sum(pd_cluster_status{type="store_low_space_count"})by (instance)> 0)
↪ and (sum(etcd_server_is_leader)by (instance)> 0)
```

- Description:

  Indicates that there is no sufficient space on the TiKV/TiFlash node.

- Solution:

  - Check whether the space in the cluster is generally insufficient. If so, increase its capacity.
  - Check whether there is any issue with Region balance scheduling. If so, it will lead to uneven data distribution.
  - Check whether there is any file that occupies a large amount of disk space, such as the log, snapshot, and core dump.
  - Lower the Region weight of the node to reduce the data volume.
  - When it is not possible to release the space, consider proactively making the node offline. This prevents insufficient disk space that leads to downtime.

9.6.2.3.3 PD_etcd_network_peer_latency

- Alert rule:

```
histogram_quantile(0.99, sum(rate(etcd_network_peer_round_trip_time_seconds_bucket
↪ [1m]))by (To, instance, job, le))> 1
```

- Description:

  The network latency between PD nodes is high. It might lead to the leader timeout and TSO disk storage timeout, which impacts the service of the cluster.

- Solution:

  - Check the network and system load status.
  - If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

9.6.2.3.4 PD_tidb_handle_requests_duration

- Alert rule:

```
histogram_quantile(0.99, sum(rate(pd_client_request_handle_requests_duration_second
↪ {type="tso"}[1m]))by (instance, job, le))> 0.1
```

- Description:

  It takes a longer time for PD to handle the TSO request. It is often caused by a high load.

- Solution:

  - Check the load status of the server.
  - Use pprof to analyze the CPU profile of PD.
  - Manually switch the PD leader.
  - If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

9.6.2.3.5 PD_down_peer_region_nums

- Alert rule:

```
(sum(pd_regions_status{type="down-peer-region-count"})by (instance)> 0)
↪ and (sum(etcd_server_is_leader)by (instance)> 0)
```

- Description:

  The number of Regions with an unresponsive peer reported by the Raft leader.

- Solution:

  - Check whether there is any TiKV that is down, or that was just restarted, or that is busy.
  - Watch the Region health panel and see whether `down_peer_region_count` is continuously decreasing.
  - Check the network between TiKV servers.

9.6.2.3.6 PD_pending_peer_region_count

- Alert rule:

```
(sum(pd_regions_status{type="pending-peer-region-count"})by (instance)>
↪  100)and (sum(etcd_server_is_leader)by (instance)> 0)
```

- Description:

  There are too many Regions that have lagged Raft logs. It is normal that scheduling leads to a small number of pending peers, but if the number remains high, there might be an issue.

- Solution:

  - Watch the Region health panel and see whether `pending_peer_region_count` is continuously decreasing.
  - Check the network between TiKV servers, especially whether there is enough bandwidth.

9.6.2.3.7 PD_leader_change

- Alert rule:

```
count(changes(pd_tso_events{type="save"}[10m])> 0)>= 2
```

- Description:

  The PD leader is recently switched.

- Solution:

  - Exclude the human factors, such as restarting PD, manually transferring leader, and adjusting leader priority.
  - Check the network and system load status.
  - If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

9.6.2.3.8 TiKV_space_used_more_than_80%

- Alert rule:

```
sum(pd_cluster_status{type="storage_size"})/ sum(pd_cluster_status{type
↪ ="storage_capacity"})* 100 > 80
```

- Description:

  Over 80% of the cluster space is occupied.

- Solution:

- Check whether it is needed to increase capacity.
- Check whether there is any file that occupies a large amount of disk space, such as the log, snapshot, and core dump.

9.6.2.3.9 PD_system_time_slow

- Alert rule:

```
changes(pd_tso_events{type="system_time_slow"}[10m])>= 1
```

- Description:

  The system time rewind might happen.

- Solution:

  Check whether the system time is configured correctly.

9.6.2.3.10 PD_no_store_for_making_replica

- Alert rule:

```
increase(pd_checker_event_count{type="replica_checker", name="no_target_store
↪ "}[1m])> 0
```

- Description:

  There is no appropriate store for additional replicas.

- Solution:

  - Check whether there is enough space in the store.
  - Check whether there is any store for additional replicas according to the label configuration if it is configured.

9.6.2.3.11 PD_cluster_slow_tikv_nums

- Alert rule:

```
sum(pd_cluster_status{type="store_slow_count"})by (instance)> 0)and (
↪ sum(etcd_server_is_leader)by (instance)> 0
```

- Description:

  There is a slow TiKV node. `raftstore.inspect-interval` controls the detection of TiKV slow nodes. For more information, see raftstore.inspect-interval.

- Solution:

  - Check whether the performance of the store is proper.
  - Set the `raftstore.inspect-interval` configuration item to a larger value to increase the timeout limit of latency.

9.6.3 TiKV alert rules

This section gives the alert rules for the TiKV component.

9.6.3.1 Emergency-level alerts

9.6.3.1.1 TiKV_memory_used_too_fast

- Alert rule:

```
process_resident_memory_bytes{job=~"tikv",instance=~".*"} - (process_resident_memor
↪ {job=~"tikv",instance=~".*"} offset 5m)> 5*1024*1024*1024
```

- Description:

  Currently, there are no TiKV monitoring items about memory. You can monitor the memory usage of the machines in the cluster by Node_exporter. The above rule indicates that when the memory usage exceeds 5 GB within 5 minutes (the memory is occupied too fast in TiKV), an alert is triggered.

- Solution:

  Adjust the `block-cache-size` value of both `rocksdb.defaultcf` and `rocksdb.writecf`.

9.6.3.1.2 TiKV_GC_can_not_work

- Alert rule:

```
sum(increase(tikv_gcworker_gc_tasks_vec{task="gc"}[1d]))< 1 and (sum(
↪ increase(tikv_gc_compaction_filter_perform[1d]))< 1 and sum(increase
↪ (tikv_engine_event_total{db="kv", cf="write", type="compaction"}[1d
↪ ]))>= 1)
```

- Description:

  GC is not performed successfully on a TiKV instance within 24 hours, which indicates that GC is not working properly. If GC does not run in a short term, it will not cause much trouble; but if GC keeps down, more and more versions are retained, which slows down the query.

- Solution:

  1. Perform `SELECT VARIABLE_VALUE FROM mysql.tidb WHERE VARIABLE_NAME = "tikv_gc_leader_desc"` to locate the `tidb-server` corresponding to the GC leader;
  2. View the log of the `tidb-server`, and grep gc_worker tidb.log;

3. If you find that the GC worker has been resolving locks (the last log is "start resolve locks") or deleting ranges (the last log is "start delete {number} ranges") during this time, it means the GC process is running normally. Otherwise, get support from PingCAP or the community.

9.6.3.2 Critical-level alerts

9.6.3.2.1 `TiKV_server_report_failure_msg_total`

- Alert rule:

```
sum(rate(tikv_server_report_failure_msg_total{type="unreachable"}[10m])
↪ )BY (store_id)> 10
```

- Description:

  Indicates that the remote TiKV cannot be connected.

- Solution:

  1. Check whether the network is clear.
  2. Check whether the remote TiKV is down.
  3. If the remote TiKV is not down, check whether the pressure is too high. Refer to the solution in `TiKV_channel_full_total`.

9.6.3.2.2 `TiKV_channel_full_total`

- Alert rule:

```
sum(rate(tikv_channel_full_total[10m]))BY (type, instance)> 0
```

- Description:

  This issue is often caused by the stuck Raftstore thread and high pressure on TiKV.

- Solution:

  1. Watch the Raft Propose monitor, and see whether the alerted TiKV node has a much higher Raft propose than other TiKV nodes. If so, it means that there are one or more hot spots on this TiKV. You need to check whether the hot spot scheduling can work properly.
  2. Watch the Raft I/O monitor, and see whether the latency increases. If the latency is high, it means a bottleneck might exist in the disk. One feasible but unsafe solution is setting `sync-log` to `false`.
  3. Watch the Raft Process monitor, and see whether the tick duration is high. If so, you need to add `raft-base-tick-interval = "2s"` under the `[raftstore]` configuration.

9.6.3.2.3 TiKV_write_stall

- Alert rule:

  ```
  delta(tikv_engine_write_stall[10m])> 0
  ```

- Description:

  The write pressure on RocksDB is too high, and a stall occurs.

- Solution:

  1. View the disk monitor, and troubleshoot the disk issues;
  2. Check whether there is any write hot spot on the TiKV;
  3. Set `max-sub-compactions` to a larger value under the `[rocksdb]` and `[raftdb]` configurations.

9.6.3.2.4 TiKV_raft_log_lag

- Alert rule:

  ```
  histogram_quantile(0.99, sum(rate(tikv_raftstore_log_lag_bucket[1m]))by
  ↪  (le, instance))> 5000
  ```

- Description:

  If this value is relatively large, it means Follower has lagged far behind Leader, and Raft cannot be replicated normally. It is possibly because the TiKV machine where Follower is located is stuck or down.

9.6.3.2.5 TiKV_async_request_snapshot_duration_seconds

- Alert rule:

  ```
  histogram_quantile(0.99, sum(rate(tikv_storage_engine_async_request_duration_second
  ↪ {type="snapshot"}[1m]))by (le, instance, type))> 1
  ```

- Description:

  If this value is relatively large, it means the load pressure on Raftstore is too high, and it might be stuck already.

- Solution:

  Refer to the solution in TiKV_channel_full_total.

9.6.3.2.6 TiKV_async_request_write_duration_seconds

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_storage_engine_async_request_duration_second
↪ {type="write"}[1m]))by (le, instance, type))> 1
```

- Description:

  If this value is relatively large, it means the Raft write takes a long time.

- Solution:

  1. Check the pressure on Raftstore. See the solution in TiKV_channel_full_total.
  2. Check the pressure on the apply worker thread.

9.6.3.2.7 TiKV_coprocessor_request_wait_seconds

- Alert rule:

```
histogram_quantile(0.9999, sum(rate(tikv_coprocessor_request_wait_seconds_bucket
↪ [1m]))by (le, instance, req))> 10
```

- Description:

  If this value is relatively large, it means the pressure on the Coprocessor worker is high. There might be a slow task that makes the Coprocessor thread stuck.

- Solution:

  1. View the slow query log from the TiDB log to see whether the index or full table scan is used in a query, or see whether it is needed to analyze;
  2. Check whether there is a hot spot;
  3. View the Coprocessor monitor and see whether `total` and `process` in `coprocessor table/index scan` match. If they differ a lot, it indicates too many invalid queries are performed. You can see whether there is `over seek bound`. If so, there are too many versions that GC does not handle in time. Then you need to increase the number of parallel GC threads.

9.6.3.2.8 TiKV_raftstore_thread_cpu_seconds_total

- Alert rule:

```
sum(rate(tikv_thread_cpu_seconds_total{name=~"raftstore_.*"}[1m]))by (
↪ instance)> 1.6
```

- Description:

  This rule monitors CPU usage by Raftstore. If the value is high, it indicates pressure on Raftstore threads is heavy.

  The alert threshold is 80% of the `raftstore.store-pool-size` value. `raftstore.store-pool-size` is 2 by default, so the alert threshold is 1.6.

- Solution:

  Refer to the solution in `TiKV_channel_full_total`.

9.6.3.2.9 TiKV_raft_append_log_duration_secs

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_raftstore_append_log_duration_seconds_bucket
↪ [1m]))by (le, instance))> 1
```

- Description:

  Indicates the time cost of appending Raft log. If it is high, it usually means I/O is too busy.

9.6.3.2.10 TiKV_raft_apply_log_duration_secs

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_raftstore_apply_log_duration_seconds_bucket
↪ [1m]))by (le, instance))> 1
```

- Description:

  Indicates the time cost of applying Raft log. If it is high, it usually means I/O is too busy.

9.6.3.2.11 TiKV_scheduler_latch_wait_duration_seconds

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_scheduler_latch_wait_duration_seconds_bucket
↪ [1m]))by (le, instance, type))> 1
```

- Description:

  The waiting time for the write operations to obtain the memory lock in Scheduler. If it is high, there might be many write conflicts, or that some operations that lead to conflicts take a long time to finish and block other operations that wait for the same lock.

- Solution:

1. View the scheduler command duration in the Scheduler-All monitor and see which command is most time-consuming;
2. View the scheduler scan details in the Scheduler-All monitor and see whether `total` and `process` match. If they differ a lot, there are many invalid scans. You can also see whether there is `over seek bound`. If there is too much, it indicates GC does not work in time;
3. View the storage async snapshot/write duration in the Storage monitor and see whether the Raft operation is performed in time.

9.6.3.2.12 `TiKV_thread_apply_worker_cpu_seconds`

- Alert rule:

```
max(rate(tikv_thread_cpu_seconds_total{name=~"apply_.*"}[1m]))by (
↪ instance)> 0.9
```

- Description:

  The apply Raft log thread is under great pressure and is approaching or has exceeded its limit. This is often caused by a burst of writes.

9.6.3.3 Warning-level alerts

9.6.3.3.1 `TiKV_leader_drops`

- Alert rule:

```
delta(tikv_pd_heartbeat_tick_total{type="leader"}[30s])< -10
```

- Description:

  It is often caused by a stuck Raftstore thread.

- Solution:

  1. Refer to TiKV_channel_full_total.
  2. It there is low pressure on TiKV, consider whether the PD scheduling is too frequent. You can view the Operator Create panel on the PD page, and check the types and number of the PD scheduling.

9.6.3.3.2 `TiKV_raft_process_ready_duration_secs`

- Alert rule:

```
histogram_quantile(0.999, sum(rate(tikv_raftstore_raft_process_duration_secs_bucket
↪ {type='ready'}[1m]))by (le, instance, type))> 2
```

- Description:

  Indicates the time cost of handling Raft ready. If this value is large, it is often caused by the stuck appending log task.

9.6.3.3.3 TiKV_raft_process_tick_duration_secs

- Alert rule:

```
histogram_quantile(0.999, sum(rate(tikv_raftstore_raft_process_duration_secs_bucket
↪ {type='tick'}[1m]))by (le, instance, type))> 2
```

- Description:

  Indicates the time cost of handling Raft tick. If this value is large, it is often caused by too many Regions.

- Solution:

  1. Consider using a higher-level log such as `warn` or `error`.
  2. Add `raft-base-tick-interval = "2s"` under the `[raftstore]` configuration.

9.6.3.3.4 TiKV_scheduler_context_total

- Alert rule:

```
abs(delta( tikv_scheduler_context_total[5m]))> 1000
```

- Description:

  The number of write commands that are being executed by Scheduler. If this value is large, it means the task is not finished timely.

- Solution:

  Refer to TiKV_scheduler_latch_wait_duration_seconds.

9.6.3.3.5 TiKV_scheduler_command_duration_seconds

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_scheduler_command_duration_seconds_bucket
↪ [1m]))by (le, instance, type))> 1
```

- Description:

  Indicates the time cost of executing the Scheduler command.

- Solution:

  Refer to TiKV_scheduler_latch_wait_duration_seconds.

9.6.3.3.6 TiKV_coprocessor_outdated_request_wait_seconds

- Alert rule:

  ```
  delta(tikv_coprocessor_outdated_request_wait_seconds_count[10m])> 0
  ```

- Description:

  The waiting time of the expired requests by Coprocessor. If this value is large, it means there is high pressure on Coprocessor.

- Solution:

  Refer to TiKV_coprocessor_request_wait_seconds.

9.6.3.3.7 TiKV_coprocessor_pending_request

- Alert rule:

  ```
  delta(tikv_coprocessor_pending_request[10m])> 5000
  ```

- Description:

  The queuing requests of Coprocessor.

- Solution:

  Refer to TiKV_coprocessor_request_wait_seconds.

9.6.3.3.8 TiKV_batch_request_snapshot_nums

- Alert rule:

  ```
  sum(rate(tikv_thread_cpu_seconds_total{name=~"cop_.*"}[1m]))by (instance
  ↪ )/ (count(tikv_thread_cpu_seconds_total{name=~"cop_.*"})* 0.9)/
  ↪ count(count(tikv_thread_cpu_seconds_total)by (instance))> 0
  ```

- Description:

  The Coprocessor CPU usage of a TiKV machine exceeds 90%.

9.6.3.3.9 TiKV_pending_task

- Alert rule:

  ```
  sum(tikv_worker_pending_task_total)BY (instance,name)> 1000
  ```

- Description:

  The number of pending tasks of TiKV.

- Solution:

  Check which kind of tasks has a higher value. You can normally find a solution to the Coprocessor and apply worker tasks from other metrics.

9.6.3.3.10 TiKV_low_space

- Alert rule:

```
sum(tikv_store_size_bytes{type="available"})by (instance)/ sum(tikv_store_size_byte
↪ {type="capacity"})by (instance)< 0.2
```

- Description:

  The data volume of TiKV exceeds 80% of the configured node capacity or the disk capacity of the machine.

- Solution:

  - Check the balance condition of node space.
  - Make a plan to increase the disk capacity or delete some data or increase cluster node depending on different situations.

9.6.3.3.11 TiKV_approximate_region_size

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_raftstore_region_size_bucket[1m
↪ ]))by (le))> 1073741824
```

- Description:

  The maximum Region approximate size that is scanned by the TiKV split checker is continually larger than 1 GB within one minute.

- Solution:

  The speed of splitting Regions is slower than the write speed. To alleviate this issue, you'd better update TiDB to a version that supports batch-split (>= 2.1.0-rc1). If it is not possible to update temporarily, you can use `pd-ctl operator add split-region <region_id> --policy=approximate` to manually split Regions.

9.6.4 TiFlash alert rules

For the detailed descriptions of TiFlash alert rules, see TiFlash Alert Rules.

9.6.5 TiDB Binlog alert rules

For the detailed descriptions of TiDB Binlog alert rules, see TiDB Binlog monitoring document.

9.6.6 TiCDC Alert rules

For the detailed descriptions of TiCDC alert rules, see TiCDC Alert Rules.

9.6.7 Node_exporter host alert rules

This section gives the alert rules for the Node_exporter host.

9.6.7.1 Emergency-level alerts

9.6.7.1.1 NODE_disk_used_more_than_80%

- Alert rule:

```
node_filesystem_avail_bytes{fstype=~"(ext.|xfs)", mountpoint!~"/boot"}
↪  / node_filesystem_size_bytes{fstype=~"(ext.|xfs)", mountpoint!~"/
↪ boot"} * 100 <= 20
```

- Description:

  The disk space usage of the machine exceeds 80%.

- Solution:

  - Log in to the machine, run the `df -h` command to check the disk space usage.
  - Make a plan to increase the disk capacity or delete some data or increase cluster node depending on different situations.

9.6.7.1.2 NODE_disk_inode_more_than_80%

- Alert rule:

```
node_filesystem_files_free{fstype=~"(ext.|xfs)"} / node_filesystem_files
↪ {fstype=~"(ext.|xfs)"} * 100 < 20
```

- Description:

  The inode usage of the filesystem on the machine exceeds 80%.

- Solution:

  - Log in to the machine and run the `df -i` command to view the node usage of the filesystem.
  - Make a plan to increase the disk capacity or delete some data or increase cluster node depending on different situations.

9.6.7.1.3 NODE_disk_readonly

- Alert rule:

  ```
  node_filesystem_readonly{fstype=~"(ext.|xfs)"} == 1
  ```

- Description:

  The filesystem is read-only and data cannot be written in it. It is often caused by disk failure or filesystem corruption.

- Solution:

  - Log in to the machine and create a file to test whether it is normal.
  - Check whether the disk LED is normal. If not, replace the disk and repair the filesystem of the machine.

9.6.7.2 Critical-level alerts

9.6.7.2.1 NODE_memory_used_more_than_80%

- Alert rule:

  ```
  (((node_memory_MemTotal_bytes-node_memory_MemFree_bytes-node_memory_Cached_bytes
  ↪ )/(node_memory_MemTotal_bytes)*100))>= 80
  ```

- Description:

  The memory usage of the machine exceeds 80%.

- Solution:

  - View the Memory panel of the host in the Grafana Node Exporter dashboard, and see whether Used memory is too high and Available memory is too low.
  - Log in to the machine and run the `free -m` command to view the memory usage. You can run `top` to check whether there is any abnormal process that has an overly high memory usage.

9.6.7.3 Warning-level alerts

9.6.7.3.1 NODE_node_overload

- Alert rule:

  ```
  (node_load5 / count without (cpu, mode)(node_cpu_seconds_total{mode="
  ↪ system"}))> 1
  ```

- Description:

  The CPU load on the machine is relatively high.

- Solution:

  - View the CPU Usage and Load Average of the host in the Grafana Node Exporter dashboard to check whether they are too high.
  - Log in to the machine and run `top` to check the load average and the CPU usage, and see whether there is any abnormal process that has an overly high CPU usage.

9.6.7.3.2 `NODE_cpu_used_more_than_80%`

- Alert rule:

```
avg(irate(node_cpu_seconds_total{mode="idle"}[5m]))by(instance)* 100 <=
↪ 20
```

- Description:

  The CPU usage of the machine exceeds 80%.

- Solution:

  - View the CPU Usage and Load Average of the host on the Grafana Node Exporter dashboard to check whether they are too high.
  - Log in to the machine and run `top` to check the Load Average and the CPU Usage, and see whether there is any abnormal process that has an overly high CPU usage.

9.6.7.3.3 `NODE_tcp_estab_num_more_than_50000`

- Alert rule:

```
node_netstat_Tcp_CurrEstab > 50000
```

- Description:

  There are more than 50,000 TCP links in the "establish" status on the machine.

- Solution:

  - Log in to the machine and run `ss -s` to check the number of TCP links in the "estab" status in the current system.
  - Run `netstat` to check whether there is any abnormal link.

9.6.7.3.4 NODE_disk_read_latency_more_than_32ms

- Alert rule:

```
((rate(node_disk_read_time_seconds_total{device=~".+"}[5m])/ rate(
↪ node_disk_reads_completed_total{device=~".+"}[5m]))or (irate(node_disk_read_time
↪ {device=~".+"}[5m])/ irate(node_disk_reads_completed_total{device
↪ =~".+"}[5m])))* 1000 > 32
```

- Description:

  The read latency of the disk exceeds 32 ms.

- Solution:

  - Check the disk status by viewing the Grafana Disk Performance dashboard.
  - Check the read latency of the disk by viewing the Disk Latency panel.
  - Check the I/O usage by viewing the Disk I/O Utilization panel.

9.6.7.3.5 NODE_disk_write_latency_more_than_16ms

- Alert rule:

```
((rate(node_disk_write_time_seconds_total{device=~".+"}[5m])/ rate
↪ (node_disk_writes_completed_total{device=~".+"}[5m]))or (irate(
↪ node_disk_write_time_seconds_total{device=~".+"}[5m])/ irate(node_disk_writes_co
↪ {device=~".+"}[5m])))> 16
```

- Description:

  The write latency of the disk exceeds 16ms.

- Solution:

  - Check the disk status by viewing the Grafana Disk Performance dashboard.
  - Check the write latency of the disk by viewing the Disk Latency panel.
  - Check the I/O usage by viewing the Disk I/O Utilization panel.

9.6.8 Blackbox_exporter TCP, ICMP, and HTTP alert rules

This section gives the alert rules for the Blackbox_exporter TCP, ICMP, and HTTP.

9.6.8.1 Emergency-level alerts

9.6.8.1.1 `TiDB_server_is_down`

- Alert rule:

  `probe_success{group="tidb"} == 0`

- Description:

  Failure to probe the TiDB service port.

- Solution:

  - Check whether the machine that provides the TiDB service is down.
  - Check whether the TiDB process exists.
  - Check whether the network between the monitoring machine and the TiDB machine is normal.

9.6.8.1.2 `TiFlash_server_is_down`

- Alert rule:

  `probe_success{group="tiflash"} == 0`

- Description:

  Failure to probe the TiFlash service port.

- Solution:

  - Check whether the machine that provides the TiFlash service is down.
  - Check whether the TiFlash process exists.
  - Check whether the network between the monitoring machine and the TiFlash machine is normal.

9.6.8.1.3 `Pump_server_is_down`

- Alert rule:

  `probe_success{group="pump"} == 0`

- Description:

  Failure to probe the pump service port.

- Solution:

  - Check whether the machine that provides the pump service is down.
  - Check whether the pump process exists.
  - Check whether the network between the monitoring machine and the pump machine is normal.

9.6.8.1.4 Drainer_server_is_down

- Alert rule:

  ```
  probe_success{group="drainer"} == 0
  ```

- Description:

  Failure to probe the Drainer service port.

- Solution:

  - Check whether the machine that provides the Drainer service is down.
  - Check whether the Drainer process exists.
  - Check whether the network between the monitoring machine and the Drainer machine is normal.

9.6.8.1.5 TiKV_server_is_down

- Alert rule:

  ```
  probe_success{group="tikv"} == 0
  ```

- Description:

  Failure to probe the TiKV service port.

- Solution:

  - Check whether the machine that provides the TiKV service is down.
  - Check whether the TiKV process exists.
  - Check whether the network between the monitoring machine and the TiKV machine is normal.

9.6.8.1.6 PD_server_is_down

- Alert rule:

  ```
  probe_success{group="pd"} == 0
  ```

- Description:

  Failure to probe the PD service port.

- Solution:

  - Check whether the machine that provides the PD service is down.
  - Check whether the PD process exists.
  - Check whether the network between the monitoring machine and the PD machine is normal.

9.6.8.1.7 Node_exporter_server_is_down

- Alert rule:

  ```
  probe_success{group="node_exporter"} == 0
  ```

- Description:

  Failure to probe the Node_exporter service port.

- Solution:

  - Check whether the machine that provides the Node_exporter service is down.
  - Check whether the Node_exporter process exists.
  - Check whether the network between the monitoring machine and the Node_exporter machine is normal.

9.6.8.1.8 Blackbox_exporter_server_is_down

- Alert rule:

  ```
  probe_success{group="blackbox_exporter"} == 0
  ```

- Description:

  Failure to probe the Blackbox_Exporter service port.

- Solution:

  - Check whether the machine that provides the Blackbox_Exporter service is down.
  - Check whether the Blackbox_Exporter process exists.
  - Check whether the network between the monitoring machine and the Blackbox_Exporter machine is normal.

9.6.8.1.9 Grafana_server_is_down

- Alert rule:

  ```
  probe_success{group="grafana"} == 0
  ```

- Description:

  Failure to probe the Grafana service port.

- Solution:

  - Check whether the machine that provides the Grafana service is down.
  - Check whether the Grafana process exists.
  - Check whether the network between the monitoring machine and the Grafana machine is normal.

9.6.8.1.10 Pushgateway_server_is_down

- Alert rule:

  ```
  probe_success{group="pushgateway"} == 0
  ```

- Description:

  Failure to probe the Pushgateway service port.

- Solution:

  - Check whether the machine that provides the Pushgateway service is down.
  - Check whether the Pushgateway process exists.
  - Check whether the network between the monitoring machine and the Pushgateway machine is normal.

9.6.8.1.11 Kafka_exporter_is_down

- Alert rule:

  ```
  probe_success{group="kafka_exporter"} == 0
  ```

- Description:

  Failure to probe the Kafka_Exporter service port.

- Solution:

  - Check whether the machine that provides the Kafka_Exporter service is down.
  - Check whether the Kafka_Exporter process exists.
  - Check whether the network between the monitoring machine and the Kafka_Exporter machine is normal.

9.6.8.1.12 Pushgateway_metrics_interface

- Alert rule:

  ```
  probe_success{job="blackbox_exporter_http"} == 0
  ```

- Description:

  Failure to probe the Pushgateway service http interface.

- Solution:

  - Check whether the machine that provides the Pushgateway service is down.
  - Check whether the Pushgateway process exists.
  - Check whether the network between the monitoring machine and the Pushgateway machine is normal.

9.6.8.2 Warning-level alerts

9.6.8.2.1 BLACKER_ping_latency_more_than_1s

- Alert rule:

```
max_over_time(probe_duration_seconds{job=~"blackbox_exporter.*_icmp"}[1
↪ m])> 1
```

- Description:

  The ping latency exceeds 1 second.

- Solution:

  - View the ping latency between the two nodes on the Grafana Blackbox Exporter page to check whether it is too high.
  - Check the TCP panel on the Grafana Node Exporter page to check whether there is any packet loss.

9.7 TiFlash Alert Rules

This document introduces the alert rules of the TiFlash cluster.

9.7.1 TiFlash_schema_error

- Alert rule:

```
increase(tiflash_schema_apply_count{type="failed"}[15m])> 0
```

- Description:

  When the schema apply error occurs, an alert is triggered.

- Solution:

  The error might be caused by some wrong logic. Get support from PingCAP or the community.

9.7.2 TiFlash_schema_apply_duration

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tiflash_schema_apply_duration_seconds_bucket
↪ [1m]))BY (le, instance))> 20
```

- Description:

  When the probability that the apply duration exceeds 20 seconds is over 99%, an alert is triggered.

- Solution:

  It might be caused by the internal problems of the TiFlash storage engine. Get support from PingCAP or the community.

9.7.3 `TiFlash_raft_read_index_duration`

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tiflash_raft_read_index_duration_seconds_bucket
↪ [1m]))BY (le, instance))> 3
```

- Description:

  When the probability that the read index duration exceeds 3 seconds is over 99%, an alert is triggered.

> **Note:**
>
> `read index` is the kvproto request sent to the TiKV leader. TiKV region retries, busy store, or network problems might lead to long request time of `read index`.

- Solution:

  The frequent retries might be caused by frequent splitting or migration of the TiKV cluster. You can check the TiKV cluster status to identify the retry reason.

9.7.4 `TiFlash_raft_wait_index_duration`

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tiflash_raft_wait_index_duration_seconds_bucket
↪ [1m]))BY (le, instance))> 2
```

- Description:

  When the probability that the waiting time for Region Raft Index in TiFlash exceeds 2 seconds is over 99%, an alert is triggered.

- Solution:

  It might be caused by a communication error between TiKV and the proxy. Get support from PingCAP or the community.

9.8 Customize Configurations of Monitoring Servers

When you deploy a TiDB cluster using TiUP, TiUP also deploys monitoring servers, such as Prometheus, Grafana, and Alertmanager. In the meantime, if you scale out this cluster, TiUP also adds the new nodes into monitoring scope.

To customize the configurations of the monitoring servers mentioned above, you can follow the instructions below to add related configuration items in the topology.yaml of the TiDB cluster.

Note:

- Do not modify the configurations files of the monitoring server directly. Because these modifications will be overwritten by later TiUP operations such as deployment, scaling out, scaling in, and reloading.
- If your monitoring servers are not deployed and managed by TiUP, you can directly modify the configuration files of the monitoring servers instead of referring to this document.
- This feature is supported in TiUP v1.9.0 and above. Therefore, check the TiUP version before using this feature.

9.8.1 Customize Prometheus configurations

Currently, TiUP supports customizing Prometheus rule and scrape configuration files.

9.8.1.1 Customize Prometheus rule configuration

1. Customize the rule configuration file and place it under a directory of the machine where TiUP locates.
2. In the topology.yaml file, set `rule_dir` to the directory of the customized rule configuration file.

   The following is a configuration example of monitoring_servers in the topology.yaml file:

```
# # Server configs are used to specify the configuration of Prometheus
    ↪ Server.
monitoring_servers:
  # # The ip address of the Monitoring Server.
- host: 127.0.0.1
  rule_dir: /home/tidb/prometheus_rule # prometheus rule dir on TiUP
      ↪ machine
```

After the preceding configuration is done, when you deploy, scale out, scale in, or reload a TiDB cluster, TiUP loads the customized rule configurations from `rule_dir` (for example, `/home/tidb/prometheus_rule`) and sends them to the Prometheus Server to replace the default rule configuration.

9.8.1.2 Customize Prometheus scrape configuration

1. Open the topology.yaml file of the TiDB cluster.
2. In the `monitoring_servers` configuration, add the `additional_scrape_conf` field.

   The following is a configuration example of monitoring_servers in the topology.yaml file:

```
monitoring_servers:
- host: xxxxxxx
ssh_port: 22
port: 9090
deploy_dir: /tidb-deploy/prometheus-9090
data_dir: /tidb-data/prometheus-9090
log_dir: /tidb-deploy/prometheus-9090/log
external_alertmanagers: []
arch: amd64
os: linux
additional_scrape_conf:
  metric_relabel_configs:
    - source_labels: [__name__]
        separator: ;
        regex: tikv_thread_nonvoluntary_context_switches|
            ↪ tikv_thread_voluntary_context_switches|
            ↪ tikv_threads_io_bytes_total
        action: drop
    - source_labels: [__name__,name]
        separator: ;
        regex: tikv_thread_cpu_seconds_total;(tokio|rocksdb).+
        action: drop
```

After the preceding configuration is done, when you deploy, scale out, scale in, or reload a TiDB cluster, TiUP adds the `additional_scrape_conf` field to the corresponding parameters of the Prometheus configuration file.

9.8.2 Customize Grafana configurations

Currently, TiUP supports customizing Grafana Dashboard and other configurations.

9.8.2.1 Customize Grafana Dashboard

1. Customize the configuration file of the Grafana Dashboard and place it under a directory of the machine where TiUP locates.

2. In the topology.yaml file, set `dashboard_dir` to the directory of the customized Dashboard configuration file.

   The following is a configuration example of grafana_servers in the topology.yaml file:

```
# # Server configs are used to specify the configuration of Grafana
   ↪ Servers.
grafana_servers:
  # # The ip address of the Grafana Server.
 - host: 127.0.0.1
 dashboard_dir: /home/tidb/dashboards # grafana dashboard dir on TiUP
    ↪ machine
```

After the preceding configuration is done, when you deploy, scale out, scale in, or reload a TiDB cluster, TiUP loads the customized Dashboard configurations from `dashboard_dir` (for example, `/home/tidb/dashboards`) and sends the configurations to the Grafana Server to replace the default Dashboard configuration.

9.8.2.2 Customize other Grafana configurations

1. Open the topology.yaml file of the TiDB cluster.

2. Add other configuration items in the `grafana_servers` configuration.

   The following is a configuration example of the `[log.file] level` and `smtp` fields in the topology.yaml file:

```
# # Server configs are used to specify the configuration of Grafana
   ↪ Servers.
grafana_servers:
# # The ip address of the Grafana Server.
- host: 127.0.0.1
    config:
    log.file.level: warning
    smtp.enabled: true
    smtp.host: {IP}:{port}
    smtp.user: example@pingcap.com
    smtp.password: {password}
    smtp.skip_verify: true
```

After the preceding configuration is done, when you deploy, scale out, scale in, or reload a TiDB cluster, TiUP adds the `config` field to the Grafana configuration file `grafana.ini`.

9.8.3 Customize Alertmanager configurations

Currently, TiUP supports customizing the listening address of Alertmanager.

Alertmanager deployed by TiUP listens to `alertmanager_servers.host` by default. You cannot access Alertmanager if you use a proxy. To address this issue, you can specify the listening address by adding `listen_host` to the cluster configuration file topology.yaml. The recommended value is 0.0.0.0.

The following example sets the `listen_host` field to 0.0.0.0.

```
alertmanager_servers:
  # # The ip address of the Alertmanager Server.
  - host: 172.16.7.147
    listen_host: 0.0.0.0
    # # SSH port of the server.
    ssh_port: 22
```

After the preceding configuration is done, when you deploy, scale out, scale in, or reload a TiDB cluster, TiUP adds the `listen_host` field to `--web.listen-address` in Alertmanager startup parameters.

10 Troubleshoot

10.1 TiDB Troubleshooting Map

This document summarizes common issues in TiDB and other components. You can use this map to diagnose and solve issues when you encounter related problems.

10.1.1 1. Service Unavailable

10.1.1.1 1.1 The client reports `Region is Unavailable` error

- 1.1.1 The `Region is Unavailable` error is usually because a Region is not available for a period of time. You might encounter `TiKV server is busy`, or the request to TiKV fails due to `not leader` or `epoch not match`, or the request to TiKV time out. In such cases, TiDB performs a `backoff` retry mechanism. When the `backoff` exceeds a threshold (20s by default), the error will be sent to the client. Within the `backoff` threshold, this error is not visible to the client.
- 1.1.2 Multiple TiKV instances are OOM at the same time, which causes no Leader during the OOM period. See case-991 in Chinese.
- 1.1.3 TiKV reports `TiKV server is busy`, and exceeds the `backoff` time. For more details, refer to 4.3. `TiKV server is busy` is a result of the internal flow control mechanism and should not be counted in the `backoff` time. This issue will be fixed.

- 1.1.4 Multiple TiKV instances failed to start, which causes no Leader in a Region. When multiple TiKV instances are deployed in a physical machine, the failure of the physical machine can cause no Leader in a Region if the label is not properly configured. See case-228 in Chinese.
- 1.1.5 When a Follower apply is lagged in a previous epoch, after the Follower becomes a Leader, it rejects the request with `epoch not match`. See case-958 in Chinese (TiKV needs to optimize its mechanism).

10.1.1.2 1.2 PD errors cause service unavailable

Refer to 5 PD issues.

10.1.2 2. Latency increases significantly

10.1.2.1 2.1 Transient increase

- 2.1.1 Wrong TiDB execution plan causes latency increase. Refer to 3.3.
- 2.1.2 PD Leader election issue or OOM. Refer to 5.2 and 5.3.
- 2.1.3 A significant number of Leader drops in some TiKV instances. Refer to 4.4.

10.1.2.2 2.2 Persistent and significant increase

- 2.2.1 TiKV single thread bottleneck
  - Too many Regions in a TiKV instance causes a single gRPC thread to be the bottleneck (Check the **Grafana** -> **TiKV-details** -> **Thread CPU/gRPC CPU Per Thread** metric). In v3.x or later versions, you can enable `Hibernate Region` to resolve the issue. See case-612 in Chinese.
  - For versions earlier than v3.0, when the raftstore thread or the apply thread becomes the bottleneck (**Grafana** -> **TiKV-details** -> **Thread CPU/raft store CPU** and **Async apply CPU** metrics exceed `80%`), you can scale out TiKV (v2.x) instances or upgrade to v3.x with multi-threading.
- 2.2.2 CPU load increases.
- 2.2.3 TiKV slow write. Refer to 4.5.
- 2.2.4 TiDB wrong execution plan. Refer to 3.3.

10.1.3 3. TiDB issues

10.1.3.1 3.1 DDL

- 3.1.1 An error `ERROR 1105 (HY000): unsupported modify decimal column precision` is reported when you modify the length of the `decimal` field. TiDB does not support changing the length of the `decimal` field.
- 3.1.2 TiDB DDL job hangs or executes slowly (use `admin show ddl jobs` to check DDL progress)
  - Cause 1: Network issue with other components (PD/TiKV).
  - Cause 2: Early versions of TiDB (earlier than v3.0.8) have heavy internal load because of a lot of goroutine at high concurrency.
  - Cause 3: In early versions (v2.1.15 & versions < v3.0.0-rc1), PD instances fail to delete TiDB keys, which causes every DDL change to wait for two leases.
  - For other unknown causes, report a bug.
  - Solution:
    - For cause 1, check the network connection between TiDB and TiKV/PD.
    - For cause 2 and 3, the issues are already fixed in later versions. You can upgrade TiDB to a later version.
    - For other causes, you can use the following solution of migrating the DDL owner.
  - DDL owner migration:
    - If you can connect to the TiDB server, execute the owner election command again: `curl -X POST http://{TiDBIP}:10080/ddl/owner/resign`
    - If you cannot connect to the TiDB server, use `tidb-ctl` to delete the DDL owner from the etcd of the PD cluster to trigger re-election: `tidb-ctl etcd delowner [LeaseID] [flags] + ownerKey`
- 3.1.3 TiDB reports `information schema is changed` error in log
  - For the detailed causes and solution, see Why the `Information schema is changed` error is reported.
  - Background: The increased number of `schema version` is consistent with the number of `schema state` of each DDL change operation. For example, the `create table` operation has 1 version change, and the `add column` operation has 4 version changes. Therefore, too many column change operations might cause `schema version` to increase fast. For details, refer to online schema change.
- 3.1.4 TiDB reports `information schema is out of date` in log

- Cause 1: The TiDB server that is executing the DML statement is stopped by `graceful kill` and prepares to exit. The execution time of the transaction that contains the DML statement exceeds one DDL lease. An error is reported when the transaction is committed.
- Cause 2: The TiDB server cannot connect to PD or TiKV when it is executing the DML statement. As a result, the TiDB server did not load the new schema within one DDL lease (`45s` by default), or the TiDB server disconnects from PD with the `keep alive` setting.
- Cause 3: TiKV has high load or network timed out. Check the node loads in **Grafana** -> **TiDB** and **TiKV**.
- Solution:
  - For cause 1, retry the DML operation when TiDB is started.
  - For cause 2, check the network between the TiDB server and PD/TiKV.
  - For cause 3, investigate why TiKV is busy. Refer to 4 TiKV issues.

10.1.3.2 3.2 OOM issues

- 3.2.1 Symptom
  - Client: The client reports the error `ERROR 2013 (HY000): Lost connection to MySQL server during query`.
  - Check the log
    - Execute `dmesg -T | grep tidb-server`. The result shows the OOM-killer log around the time point when the error occurs.
    - Grep the "Welcome to TiDB" log in `tidb.log` around the time point after the error occurs (namely, the time when tidb-server restarts).
    - Grep `fatal error: runtime: out of memory` or `cannot allocate memory` in `tidb_stderr.log`.
    - In v2.1.8 or earlier versions, you can grep `fatal error: stack overflow` in the `tidb_stderr.log`.
  - Monitor: The memory usage of tidb-server instances increases sharply in a short period of time.
- 3.2.2 Locate the SQL statement that causes OOM. (Currently all versions of TiDB cannot locate SQL accurately. You still need to analyze whether OOM is caused by the SQL statement after you locate one.)
  - For versions >= v3.0.0, grep "expensive_query" in `tidb.log`. That log message records SQL queries that timed out or exceed memory quota.
  - For versions < v3.0.0, grep "memory exceeds quota" in `tidb.log` to locate SQL queries that exceed memory quota.

Note:

The default threshold for a single SQL memory usage is `1GB`. You can set this parameter by configuring the system variable `tidb_mem_quota_query`.

- 3.2.3 Mitigate OOM issues
    - By enabling `SWAP`, you can mitigate the OOM issue caused by overuse of memory by large queries. When the memory is insufficient, this method can have impact on the performance of large queries due to the I/O overhead. The degree to which the performance is affected depends on the remaining memory space and the disk I/O speed.
- 3.2.4 Typical reasons for OOM
    - The SQL query has `join`. If you view the SQL statement by using `explain`, you can find that the `join` operation selects the `HashJoin` algorithm and the `inner` table is large.
    - The data volume of a single `UPDATE/DELETE` query is too large. See case-882 in Chinese.
    - The SQL contains multiple sub-queries connected by `Union`. See case-1828 in Chinese.

For more information about troubleshooting OOM, see Troubleshoot TiDB OOM Issues.

10.1.3.3 3.3 Wrong execution plan

- 3.3.1 Symptom
    - SQL query execution time is much longer compared with that of previous executions, or the execution plan suddenly changes. If the execution plan is logged in the slow log, you can directly compare the execution plans.
    - SQL query execution time is much longer compared with that of other databases such as MySQL. Compare the execution plan with other databases to see the differences, such as `Join Order`.
    - In slow log, the number of SQL execution time `Scan Keys` is large.
- 3.3.2 Investigate the execution plan

- `explain analyze {SQL}`. When the execution time is acceptable, compare `count` in the result of `explain analyze` and the number of `row` in `execution info`. If a large difference is found in the `TableScan/IndexScan` row, it is likely that the statistics is incorrect. If a large difference is found in other rows, the problem might not be in the statistics.
- `select count(*)`. When the execution plan contains a `join` operation, `explain analyze` might take a long time. You can check whether the problem is in the statistics by executing `select count(*)` for the conditions on `TableScan/IndexScan` and comparing the `row count` information in the `explain` result.

- 3.3.3 Mitigation

  - For v3.0 and later versions, use the `SQL Bind` feature to bind the execution plan.
  - Update the statistics. If you are roughly sure that the problem is caused by the statistics, dump the statistics. If the cause is outdated statistics, such as the `modify count/row count` in `show stats_meta` is greater than a certain value (for example, 0.3), or the table has an index of time column, you can try recovering by using `analyze table`. If `auto analyze` is configured, check whether the `tidb_auto_analyze_ratio` system variable is too large (for example, greater than 0.3), and whether the current time is between `tidb_auto_analyze_start_time` and `tidb_auto_analyze_end_time`.
  - For other situations, report a bug.

10.1.3.4 3.4 SQL execution error

- 3.4.1 The client reports the `ERROR 1265(01000)Data Truncated` error. This is because the way TiDB internally calculates the precision of `Decimal` type is incompatible with that of MySQL. This issue has been fixed in v3.0.10 (#14438).

  - Cause:

    In MySQL, if two large-precision `Decimal` are divided and the result exceeds the maximum decimal precision (`30`), only `30` digits are reserved and no error is reported;

    In TiDB, the calculation result is the same as in MySQL, but inside the data structure that represents `Decimal`, a field for decimal precision still retains the actual precision.

    Take `(0.1^30)/ 10` as an example. The results in TiDB and MySQL are both `0`, because the precision is `30` at most. However, in TiDB, the field for decimal precision is still `31`.

    After multiple `Decimal` divisions, even though the result is correct, this precision field could grow larger and larger, and eventually exceeds the threshold in TiDB (`72`), and the `Data Truncated` error is reported.

    The multiplication of `Decimal` does not have this issue, because the out-of-bounds is bypassed, and the precision is set to the maximum precision limit.

- Solution: You can bypass this issue by manually adding `Cast(xx as decimal(a , b))`, in which `a` and `b` are the target precisions.

10.1.4 4. TiKV issues

10.1.4.1 4.1 TiKV panics and fails to start

- 4.1.1 `sync-log = false`. The `unexpected raft log index: last_index X < applied_index Y` error is returned after the machine is powered off.

  This issue is expected. You can restore the Region using `tikv-ctl`.

- 4.1.2 If TiKV is deployed on a virtual machine, when the virtual machine is killed or the physical machine is powered off, the `entries[X, Y] is unavailable from storage` error is reported.

  This issue is expected. The `fsync` of virtual machines is not reliable, so you need to restore the Region using `tikv-ctl`.

- 4.1.3 For other unexpected causes, report a bug.

10.1.4.2 4.2 TiKV OOM

- 4.2.1 If the `block-cache` configuration is too large, it might cause OOM.

  To verify the cause of the problem, check the `block cache size` of RocksDB by selecting the corresponding instance in the monitor **Grafana -> TiKV-details**.

  Meanwhile, check whether the `[storage.block-cache] capacity = # "1GB"` parameter is set properly. By default, TiKV's `block-cache` is set to `45%` of the total memory of the machine. You need to explicitly specify this parameter when you deploy TiKV in the container, because TiKV obtains the memory of the physical machine, which might exceed the memory limit of the container.

- 4.2.2 Coprocessor receives many large queries and returns a large volume of data. gRPC fails to send data as quickly as the coprocessor returns data, which results in OOM.

  To verify the cause, you can check whether `response size` exceeds the `network outbound` traffic by viewing the monitor **Grafana -> TiKV-details -> coprocessor overview**.

- 4.2.3 Other components occupy too much memory.

  This issue is unexpected. You can report a bug.

10.1.4.3 4.3 The client reports the `server is busy` error

Check the specific cause for busy by viewing the monitor **Grafana -> TiKV -> errors**. `server is busy` is caused by the flow control mechanism of TiKV, which informs `tidb/ti` ↪ `-client` that TiKV is currently under too much pressure and will retry later.

- 4.3.1 TiKV RocksDB encounters `write stall`.

  A TiKV instance has two RocksDB instances, one in `data/raft` to save the Raft log, another in `data/db` to save the real data. You can check the specific cause for stall by running `grep "Stalling" RocksDB` in the log. The RocksDB log is a file starting with `LOG`, and `LOG` is the current log.

  - Too many `level0 sst` causes stall. You can add the `[rocksdb] max-sub-` ↪ `compactions = 2` (or 3) parameter to speed up `level0 sst` compaction. The compaction task from level0 to level1 is divided into several subtasks (the max number of subtasks is the value of `max-sub-compactions`) to be executed concurrently. See case-815 in Chinese.
  - Too many `pending compaction bytes` causes stall. The disk I/O fails to keep up with the write operations in business peaks. You can mitigate this problem by increasing the `soft-pending-compaction-bytes-limit` and `hard-pending-` ↪ `compaction-bytes-limit` of the corresponding CF.
    - The default value of `[rocksdb.defaultcf] soft-pending-compaction` ↪ `-bytes-limit` is `64GB`. If the pending compaction bytes reaches the threshold, RocksDB slows down the write speed. You can set `[rocksdb.` ↪ `defaultcf] soft-pending-compaction-bytes-limit` to `128GB`.
    - The default value of `hard-pending-compaction-bytes-limit` is `256GB`. If the pending compaction bytes reaches the threshold (this is not likely to happen, because RocksDB slows down the write after the pending compaction bytes reaches `soft-pending-compaction-bytes-limit`), RocksDB stops the write operation. You can set `hard-pending-compaction-bytes-limit` to `512GB`.
    - If the disk I/O capacity fails to keep up with the write for a long time, it is recommended to scale up your disk. If the disk throughput reaches the upper limit and causes write stall (for example, the SATA SSD is much lower than NVME SSD), while the CPU resources is sufficient, you may apply a compression algorithm of higher compression ratio. This way, the CPU resources is traded for disk resources, and the pressure on the disk is eased.
    - If the default CF compaction sees a high pressure, change the `[rocksdb.` ↪ `defaultcf] compression-per-level` parameter from `["no", "no", "` ↪ `lz4", "lz4", "lz4", "zstd", "zstd"]` to `["no", "no", "zstd", "` ↪ `zstd", "zstd", "zstd", "zstd"]`.
  - Too many memtables causes stall. This usually occurs when the amount of instant writes is large and the memtables flush to the disk slowly. If the disk write speed

cannot be improved, and this issue only occurs during business peaks, you can mitigate it by increasing the `max-write-buffer-number` of the corresponding CF.

- For example, set `[rocksdb.defaultcf] max-write-buffer-number` to `8` (`5` by default). Note that this might cause more memory usage in the peak, because more memtables might be in the memory.

- 4.3.2 `scheduler too busy`

  - Serious write conflict. `latch wait duration` is high. You can view `latch wait duration` in the monitor **Grafana -> TiKV-details -> scheduler prewrite/scheduler commit**. When the write tasks pile up in the scheduler, the pending write tasks exceed the threshold set in `[storage] scheduler-pending-write-threshold` (100MB). You can verify the cause by viewing the metric corresponding to `MVCC_CONFLICT_COUNTER`.
  - Slow write causes write tasks to pile up. The data being written to TiKV exceeds the threshold set by `[storage] scheduler-pending-write-threshold` (100MB). Refer to 4.5.

- 4.3.3 `raftstore is busy`. The processing of messages is slower than the receiving of messages. The short-term `channel full` status does not affect the service, but if the error persists for a long time, it might cause Leader switch.

  - `append log` encounters stall. Refer to 4.3.1.
  - `append log duration` is high, which causes slow processing of messages. You can refer to 4.5 to analyze why `append log duration` is high.
  - raftstore receives a large batch of messages in an instant (check in the TiKV Raft messages dashboard), and fails to process them. Usually the short-term `channel full` status does not affect the service.

- 4.3.4 TiKV coprocessor is in a queue. The number of piled up tasks exceeds `coprocessor threads * readpool.coprocessor.max-tasks-per-worker-[normal|low|high]`. Too many large queries leads to the tasks piling up in coprocessor. You need to check whether a execution plan change causes a large number of table scan operations. Refer to 3.3.

10.1.4.4 4.4 Some TiKV nodes drop Leader frequently

- 4.4.1 Re-election because TiKV is restarted

  - After TiKV panics, it is pulled up by systemd and runs normally. You can check whether panic has occurred by viewing the TiKV log. Because this issue is unexpected, report a bug if it happens.
  - TiKV is stopped or killed by a third party and then pulled up by systemd. Check the cause by viewing `dmesg` and the TiKV log.

  - TiKV is OOM, which causes restart. Refer to 4.2.
  - TiKV is hung because of dynamically adjusting `THP` (Transparent Hugepage). See case case-500 in Chinese.

- 4.4.2 TiKV RocksDB encounters write stall and thus results in re-election. You can check if the monitor **Grafana** -> **TiKV-details** -> **errors** shows `server is busy`. Refer to 4.3.1.

- 4.4.3 Re-election because of network isolation.

10.1.4.5 4.5 TiKV write is slow

- 4.5.1 Check whether the TiKV write is low by viewing the `prewrite/commit/raw-put` duration of TiKV gRPC (only for RawKV clusters). Generally, you can locate the slow phase according to the performance-map. Some common situations are listed as follows.

- 4.5.2 The scheduler CPU is busy (only for transaction kv).

  The `scheduler command duration` of prewrite/commit is longer than the sum of `scheduler latch wait duration` and `storage async write duration`. The scheduler worker has a high CPU demand, such as over 80% of `scheduler-worker-pool-size` * 100%, or the CPU resources of the entire machine are relatively limited. If the write workload is large, check if `[storage] scheduler-worker-pool-size` is set too small.

  For other situations, report a bug.

- 4.5.3 Append log is slow.

  The **Raft IO**/`append log duration` in TiKV Grafana is high, usually because the disk write operation is slow. You can verify the cause by checking the `WAL Sync Duration max` value of RocksDB - raft.

  For other situations, report a bug.

- 4.5.4 The raftstore thread is busy.

  The **Raft Propose**/`propose wait duration` is significantly larger than the append log duration in TiKV Grafana. Take the following methods:

  - Check whether the `[raftstore] store-pool-size` configuration value is too small. It is recommended to set the value between `1` and `5` and not too large.
  - Check whether the CPU resources on the machine are insufficient.

- 4.5.5 Apply is slow.

  The **Raft IO**/`apply log duration` in TiKV Grafana is high, which usually comes with a high **Raft Propose**/`apply wait duration`. The possible causes are as follows:

- `[raftstore] apply-pool-size` is too small (it is recommended to set the value between `1` and `5` and not too large), and the **Thread CPU**/`apply CPU` is large.
- The CPU resources on the machine are insufficient.
- Region write hot spot. A single apply thread has high CPU usage. Currently, we cannot properly address the hot spot problem on a single Region, which is being improved. To view the CPU usage of each thread, modify the Grafana expression and add `by (instance, name)`.
- RocksDB write is slow. **RocksDB kv**/`max write duration` is high. A single Raft log might contain multiple KVs. When writing into RocksDB, 128 KVs are written into RocksDB in a write batch. Therefore, an apply log might be associated with multiple writes in RocksDB.
- For other situations, report a bug.

- 4.5.6 Raft commit log is slow.

  The **Raft IO**/`commit log duration` in TiKV Grafana is high (this metric is only supported in Grafana after v4.x). Every Region corresponds to an independent Raft group. Raft has a flow control mechanism, similar to the sliding window mechanism of TCP. You can control the size of the sliding window by configuring the `[raftstore]` ↪ `raft-max-inflight-msgs = 256` parameter. If there is a write hot spot and the `commit log duration` is high, you can adjust the parameter, such as increasing it to `1024`.

- 4.5.7 For other situations, refer to the write path on performance-map and analyze the cause.

10.1.5 5. PD issues

10.1.5.1 5.1 PD scheduling

- 5.1.1 Merge
  - Empty Regions across tables cannot be merged. You need to modify the `[` ↪ `coprocessor] split-region-on-table` parameter in TiKV, which is set to `false` in v4.x by default. See case-896 in Chinese.
  - Region merge is slow. You can check whether the merged operator is generated by accessing the monitor dashboard in **Grafana** -> **PD** -> **operator**. To accelerate the merge, increase the value of `merge-schedule-limit`.

- 5.1.2 Add replicas or take replicas online/offline
  - The TiKV disk uses 80% of the capacity, and PD does not add replicas. In this situation, the number of miss peers increases, so TiKV needs to be scaled out. See case-801 in Chinese.

- When a TiKV node is taken offline, some Region cannot be migrated to other nodes. This issue has been fixed in v3.0.4 (#5526). See case-870 in Chinese.

- 5.1.3 Balance

  - The Leader/Region count is not evenly distributed. See case-394 and case-759 in Chinese. The major cause is that the balance performs scheduling based on the size of Region/Leader, so this might result in the uneven distribution of the count. In TiDB 4.0, the `[leader-schedule-policy]` parameter is introduced, which enables you to set the scheduling policy of Leader to be `count`-based or `size`-based.

10.1.5.2 5.2 PD election

- 5.2.1 PD switches Leader.

  - Cause 1: Disk. The disk where the PD node is located has full I/O load. Investigate whether PD is deployed with other components with high I/O demand and the health of the disk. You can verify the cause by viewing the monitor metrics in **Grafana -> disk performance -> latency/load**. You can also use the FIO tool to run a check on the disk if necessary. See case-292 in Chinese.
  - Cause 2: Network. The PD log shows `lost the TCP streaming connection`. You need to check whether there is a problem with the network between PD nodes and verify the cause by viewing `round trip` in the monitor **Grafana -> PD -> etcd**. See case-177 in Chinese.
  - Cause 3: High system load. The log shows `server is likely overloaded`. See case-214 in Chinese.

- 5.2.2 PD cannot elect a Leader or the election is slow.

  - PD cannot elect a Leader: The PD log shows `lease is not expired`. This issue has been fixed in v3.0.x and v2.1.19. See case-875 in Chinese.
  - The election is slow: The Region loading duration is long. You can check this issue by running `grep "regions cost"` in the PD log. If the result is in seconds, such as `load 460927 regions cost 11.77099s`, it means the Region loading is slow. You can enable the `region storage` feature in v3.0 by setting `use-region -storage` to `true`, which significantly reduce the Region loading duration. See case-429 in Chinese.

- 5.2.3 PD timed out when TiDB executes SQL statements.

  - PD doesn't have a Leader or switches Leader. Refer to 5.2.1 and 5.2.2.
  - Network issue. Check whether the network from TiDB to PD Leader is running normally by accessing the monitor **Grafana -> blackbox_exporter -> ping latency**.

- PD panics. Report a bug.
- PD is OOM. Refer to 5.3.
- If the issue has other causes, get goroutine by running `curl http://127.0.0.1:2379/debug/pprof/goroutine?debug=2` and report a bug.

- 5.2.4 Other issues
  - PD reports the `FATAL` error, and the log shows `range failed to find revision pair`. This issue has been fixed in v3.0.8 (#2040). For details, see case-947 in Chinese.
  - For other situations, report a bug.

10.1.5.3 5.3 PD OOM

- 5.3.1 When the `/api/v1/regions` interface is used, too many Regions might cause PD OOM. This issue has been fixed in v3.0.8 (#1986).
- 5.3.2 PD OOM during the rolling upgrade. The size of gRPC messages is not limited, and the monitor shows that TCP InSegs is relatively large. This issue has been fixed in v3.0.6 (#1952).

10.1.5.4 5.4 Grafana display

- 5.4.1 The monitor in **Grafana** -> **PD** -> **cluster** -> **role** displays follower. The Grafana expression issue has been fixed in v3.0.8.

10.1.6 6. Ecosystem tools

10.1.6.1 6.1 TiDB Binlog

- 6.1.1 TiDB Binlog is a tool that collects changes from TiDB and provides backup and replication to downstream TiDB or MySQL platforms. For details, see TiDB Binlog on GitHub.
- 6.1.2 The `Update Time` in Pump/Drainer Status is updated normally, and no anomaly shows in the log, but no data is written to the downstream.
  - Binlog is not enabled in the TiDB configuration. Modify the `[binlog]` configuration in TiDB.
- 6.1.3 `sarama` in Drainer reports the `EOF` error.
  - The Kafka client version in Drainer is inconsistent with the version of Kafka. You need to modify the `[syncer.to] kafka-version` configuration.

- 6.1.4 Drainer fails to write to Kafka and panics, and Kafka reports the `Message was too large` error.

    - The binlog data is too large, so the single message written to Kafka is too large. You need to modify the following configuration of Kafka:

    ```
    message.max.bytes=1073741824
    replica.fetch.max.bytes=1073741824
    fetch.message.max.bytes=1073741824
    ```

    For details, see case-789 in Chinese.

- 6.1.5 Inconsistent data in upstream and downstream

    - Some TiDB nodes do not enable binlog. For v3.0.6 or later versions, you can check the binlog status of all the nodes by accessing the http://127.0.0.1:10080/info/all interface. For versions earlier than v3.0.6, you can check the binlog status by viewing the configuration file.
    - Some TiDB nodes go into the `ignore binlog` status. For v3.0.6 or later versions, you can check the binlog status of all the nodes by accessing the http://127.0.0.1:10080/info/all interface. For versions earlier than v3.0.6, check the TiDB log to see whether it contains the `ignore binlog` keyword.
    - The value of the timestamp column is inconsistent in upstream and downstream.
        - This is caused by different time zones. You need to ensure that Drainer is in the same time zone as the upstream and downstream databases. Drainer obtains its time zone from `/etc/localtime` and does not support the `TZ` environment variable. See case-826 in Chinese.
        - In TiDB, the default value of timestamp is `null`, but the same default value in MySQL 5.7 (not including MySQL 8) is the current time. Therefore, when the timestamp in upstream TiDB is `null` and the downstream is MySQL 5.7, the data in the timestamp column is inconsistent. You need to run `set @@global.explicit_defaults_for_timestamp=on;` in the upstream before enabling binlog.
    - For other situations, report a bug.

- 6.1.6 Slow replication

    - The downstream is TiDB/MySQL, and the upstream performs frequent DDL operations. See case-1023 in Chinese.
    - The downstream is TiDB/MySQL, and the table to be replicated has no primary key and no unique index, which causes reduced performance in binlog. It is recommended to add the primary key or unique index.
    - If the downstream outputs to files, check whether the output disk or network disk is slow.

    - For other situations, report a bug.

- 6.1.7 Pump cannot write binlog and reports the `no space left on device` error.

    - The local disk space is insufficient for Pump to write binlog data normally. You need to clean up the disk space and then restart Pump.

- 6.1.8 Pump reports the `fail to notify all living drainer` error when it is started.

    - Cause: When Pump is started, it notifies all Drainer nodes that are in the `online` state. If it fails to notify Drainer, this error log is printed.
    - Solution: Use the binlogctl tool to check whether each Drainer node is normal or not. This is to ensure that all Drainer nodes in the `online` state are working normally. If the state of a Drainer node is not consistent with its actual working status, use the binlogctl tool to change its state and then restart Pump. See the case fail-to-notify-all-living-drainer.

- 6.1.9 Drainer reports the `gen update sqls failed: table xxx: row data is corruption []` error.

    - Trigger: The upstream performs DML operations on this table while performing `DROP COLUMN` DDL. This issue has been fixed in v3.0.6. See case-820 in Chinese.

- 6.1.10 Drainer replication is hung. The process remains active but the checkpoint is not updated.

    - This issues has been fixed in v3.0.4. See case-741 in Chinese.

- 6.1.11 Any component panics.

    - Report a bug.

10.1.6.2 6.2 Data Migration

- 6.2.1 TiDB Data Migration (DM) is a migration tool that supports data migration from MySQL/MariaDB into TiDB. For details, see DM on GitHub.

- 6.2.2 `Access denied for user 'root'@'172.31.43.27' (using password: YES)` shows when you run `query status` or check the log.

    - The database related passwords in all the DM configuration files should be encrypted by `dmctl`. If a database password is empty, it is unnecessary to encrypt the password. Cleartext passwords can be used since v1.0.6.

    - During DM operation, the user of the upstream and downstream databases must have the corresponding read and write privileges. Data Migration also prechecks the corresponding privileges automatically while starting the data replication task.
    - To deploy different versions of DM-worker/DM-master/dmctl in a DM cluster, see the case study on AskTUG in Chinese.

- 6.2.3 A replication task is interrupted with the `driver: bad connection` error returned.

    - The `driver: bad connection` error indicates that an anomaly has occurred in the connection between DM and the downstream TiDB database (such as network failure and TiDB restart), and that the data of the current request has not yet been sent to TiDB.
        - For versions earlier than DM 1.0.0 GA, stop the task by running `stop-task` and then restart the task by running `start-task`.
        - For DM 1.0.0 GA or later versions, an automatic retry mechanism for this type of error is added. See #265.

- 6.2.4 A replication task is interrupted with the `invalid connection` error.

    - The `invalid connection` error indicates that an anomaly has occurred in the connection between DM and the downstream TiDB database (such as network failure, TiDB restart, and TiKV busy), and that a part of the data for the current request has been sent to TiDB. Because DM has the feature of concurrently replicating data to the downstream in replication tasks, several errors might occur when a task is interrupted. You can check these errors by running `query-status` or `query-error`.
        - If only the `invalid connection` error occurs during the incremental replication process, DM retries the task automatically.
        - If DM does not retry or fails to retry automatically because of version problems (automatic retry is introduced in v1.0.0-rc.1), use `stop-task` to stop the task and then use `start-task` to restart the task.

- 6.2.5 The relay unit reports the error `event from * in * diff from passed-in` ↪ `event *`, or a replication task is interrupted with an error that fails to get or parse binlog, such as `get binlog error ERROR 1236 (HY000)and binlog checksum` ↪ `mismatch, data may be corrupted returned`

    - During the process that DM pulls relay log or the incremental replication, this two errors might occur if the size of the upstream binlog file exceeds 4 GB.
    - Cause: When writing relay logs, DM needs to perform event verification based on binlog positions and the binlog file size, and store the replicated binlog positions as checkpoints. However, the official MySQL uses uint32 to store binlog positions, which means the binlog position for a binlog file over 4 GB overflows, and then the errors above occur.

- Solution:
  - For relay processing units, manually recover replication.
  - For binlog replication processing units, manually recover replication.

- 6.2.6 The DM replication is interrupted, and the log returns `ERROR 1236 (HY000) ↪ The slave is connecting using CHANGE MASTER TO MASTER_AUTO_POSITION ↪ = 1, but the master has purged binary logs containing GTIDs that the ↪ slave requires.`
  - Check whether the master binlog is purged.
  - Check the position information recorded in `relay.meta`.
    - `relay.meta` has recorded the empty GTID information. DM-worker saves the GTID information in memory to `relay.meta` when it exits or in every 30s. When DM-worker does not obtain the upstream GTID information, it saves the empty GTID information to `relay.meta`. See case-772 in Chinese.
    - The binlog event recorded in `relay.meta` triggers the incomplete recover process and records the wrong GTID information. This issue is fixed in v1.0.2, and might occur in earlier versions.

- 6.2.7 The DM replication process returns an error `Error 1366: incorrect utf8 ↪ value eda0bdedb29d(\ufffd\ufffd\ufffd\ufffd\ufffd\ufffd)`.
  - This value cannot be successfully written into MySQL 8.0 or TiDB, but can be written into MySQL 5.7. You can skip the data format check by enabling the `tidb_skip_utf8_check` parameter.

10.1.6.3 6.3 TiDB Lightning

- 6.3.1 TiDB Lightning is a tool for fast full import of large amounts of data into a TiDB cluster. See TiDB Lightning on GitHub.

- 6.3.2 Import speed is too slow.
  - `region-concurrency` is set too high, which causes thread contention and reduces performance. Three ways to troubleshoot:
    - The setting can be found from the start of the log by searching `region- ↪ concurrency`.
    - If TiDB Lightning shares a server with other services (for example, Importer), you must manually set `region-concurrency` to 75% of the total number of CPU cores on that server.
    - If there is a quota on CPU (for example, limited by Kubernetes settings), TiDB Lightning might not be able to read this out. In this case, `region- ↪ concurrency` must also be manually reduced.

- Every additional index introduces a new KV pair for each row. If there are N indices, the actual size to be imported would be approximately (N+1) times the size of the Mydumper output. If the indices are negligible, you may first remove them from the schema, and add them back via `CREATE INDEX` after the import is complete.
- The version of TiDB Lightning is old. Try the latest version, which might improve the import speed.

- 6.3.3 `checksum failed: checksum mismatched remote vs local`.
  - Cause 1: The table might already have data. These old data can affect the final checksum.
  - Cause 2: If the checksum of the target database is 0, which means nothing is imported, it is possible that the cluster is too hot and fails to take in any data.
  - Cause 3: If the data source is generated by the machine and not backed up by Mydumper, ensure it respects the constrains of the table. For example:
    - `AUTO_INCREMENT` columns need to be positive, and do not contain the value "0".
    - UNIQUE and PRIMARY KEYs must not have duplicate entries.
  - Solution: See Troubleshooting Solution.

- 6.3.4 `Checkpoint for ··· has invalid status:(error code)`
  - Cause: Checkpoint is enabled, and Lightning/Importer has previously abnormally exited. To prevent accidental data corruption, TiDB Lightning will not start until the error is addressed. The error code is an integer less than 25, with possible values as `0, 3, 6, 9, 12, 14, 15, 17, 18, 20 and 21`. The integer indicates the step where the unexpected exit occurs in the import process. The larger the integer is, the later the exit occurs.
  - Solution: See Troubleshooting Solution.

- 6.3.5 `ResourceTemporarilyUnavailable("Too many open engines ···: 8")`
  - Cause: The number of concurrent engine files exceeds the limit specified by tikv-importer. This could be caused by misconfiguration. In addition, even when the configuration is correct, if tidb-lightning has exited abnormally before, an engine file might be left at a dangling open state, which could cause this error as well.
  - Solution: See Troubleshooting Solution.

- 6.3.6 `cannot guess encoding for input file, please convert to UTF-8` ↪ `manually`

- Cause: TiDB Lightning only supports the UTF-8 and GB-18030 encodings. This error means the file is not in any of these encodings. It is also possible that the file has mixed encoding, such as containing a string in UTF-8 and another string in GB-18030, due to historical ALTER TABLE executions.
- Solution: See Troubleshooting Solution.

- 6.3.7 `[sql2kv] sql encode error = [types:1292]invalid time format: ↪ '{1970 1 1 0 45 0 0}'`

  - Cause: A timestamp type entry has a time value that does not exist. This is either because of DST changes or because the time value has exceeded the supported range (from Jan 1, 1970 to Jan 19, 2038).
  - Solution: See Troubleshooting Solution.

10.1.7 7. Common log analysis

10.1.7.1 7.1 TiDB

- 7.1.1 `GC life time is shorter than transaction duration`.

  The transaction duration exceeds the GC lifetime (10 minutes by default).

  You can increase the GC lifetime by modifying the `tidb_gc_life_time` system variable. Generally, it is not recommended to modify this parameter, because changing it might cause many old versions to pile up if this transaction has a large number of `UPDATE` and `DELETE` statements.

- 7.1.2 `txn takes too much time`.

  This error is returned when you commit a transaction that has not been committed for a long time (over 590 seconds).

  If your application needs to execute a transaction of such a long time, you can increase the `[tikv-client] max-txn-time-use = 590` parameter and the GC lifetime to avoid this issue. It is recommended to check whether your application needs such a long transaction time.

- 7.1.3 `coprocessor.go` reports `request outdated`.

  This error is returned when the coprocessor request sent to TiKV waits in a queue at TiKV for over 60 seconds.

  You need to investigate why the TiKV coprocessor is in a long queue.

- 7.1.4 `region_cache.go` reports a large number of `switch region peer to next due ↪ to send request fail`, and the error message is `context deadline exceeded`.

  The request for TiKV timed out and triggers the region cache to switch the request to other nodes. You can continue to run the `grep "<addr> cancelled` command on the `addr` field in the log and take the following steps according to the `grep` results:

- `send request is cancelled`: The request timed out during the sending phase. You can investigate the monitoring **Grafana** -> **TiDB** -> **Batch Client**/`Pending Request Count by TiKV` and see whether the Pending Request Count is greater than 128:
  - If the value is greater than 128, the sending goes beyond the processing capacity of KV, so the sending piles up.
  - If the value is not greater than 128, check the log to see if the report is caused by the operation and maintenance changes of the corresponding KV; otherwise, this error is unexpected, and you need to report a bug.
- `wait response is cancelled`: The request timed out after it is sent to TiKV. You need to check the response time of the corresponding TiKV address and the Region logs in PD and KV at that time.

- 7.1.5 `distsql.go` reports `inconsistent index`.

  The data index seems to be inconsistent. Run the `admin check table <TableName>` command on the table where the reported index is. If the check fails, disable garbage collection by running the following command, and report a bug:

```
SET GLOBAL tidb_gc_enable = 0;
```

10.1.7.2 7.2 TiKV

- 7.2.1 `key is locked`.

  The read and write have conflict. The read request encounters data that has not been committed and needs to wait until the data is committed.

  A small number of this error has no impact on the business, but a large number of this error indicates that the read-write conflict is severe in your business.

- 7.2.2 `write conflict`.

  This is the write-write conflict in optimistic transactions. If multiple transactions modify the same key, only one transaction succeed and other transactions automatically obtain the timestamp again and retry the operation, with no impact on the business.

  If the conflict is severe, it might cause transaction failure after multiple retries. In this case, it is recommended to use the pessimistic lock.

- 7.2.3 `TxnLockNotFound`.

  This transaction commit is too slow, causing it to be rolled back by other transactions after Time To Live (TTL). This transaction will automatically retry, so the business is usually not affected. For a transaction with a size of 0.25 MB or smaller, the default TTL is 3 seconds.

- 7.2.4 `PessimisticLockNotFound`.

  Similar to `TxnLockNotFound`. The pessimistic transaction commit is too slow and thus rolled back by other transactions.

- 7.2.5 `stale_epoch`.

  The request epoch is outdated, so TiDB re-sends the request after updating the routing. The business is not affected. Epoch changes when Region has a split/merge operation or a replica is migrated.

- 7.2.6 `peer is not leader`.

  The request is sent to a replica that is not Leader. If the error response indicates which replica is the latest Leader, TiDB updates the local routing according the error and sends a new request to the latest Leader. Usually, the business is not affected.

  In v3.0 and later versions, TiDB tries other peers if the request to the previous Leader fails, which might lead to frequent `peer is not leader` in TiKV log. You can check the `switch region peer to next due to send request fail` log of the corresponding Region in TiDB to determine the root cause of the sending failure. For details, refer to 7.1.4.

  This error might also be returned if a Region has no Leader due to other reasons. For details, see 4.4.

10.2 Identify Slow Queries

To help users identify slow queries, analyze and improve the performance of SQL execution, TiDB outputs the statements whose execution time exceeds slow-threshold (The default value is 300 milliseconds) to slow-query-file (The default value is "tidb-slow.log").

TiDB enables the slow query log by default. You can enable or disable the feature by modifying the configuration `enable-slow-log`.

10.2.1 Usage example

```
## Time: 2019-08-14T09:26:59.487776265+08:00
## Txn_start_ts: 410450924122144769
## User@Host: root[root] @ localhost [127.0.0.1]
## Conn_ID: 3086
## Exec_retry_time: 5.1 Exec_retry_count: 3
## Query_time: 1.527627037
## Parse_time: 0.000054933
## Compile_time: 0.000129729
## Rewrite_time: 0.000000003 Preproc_subqueries: 2 Preproc_subqueries_time:
   ↪ 0.000000002
## Optimize_time: 0.00000001
## Wait_TS: 0.00001078
```

```
## Process_time: 0.07 Request_count: 1 Total_keys: 131073 Process_keys:
    ↪ 131072 Prewrite_time: 0.335415029 Commit_time: 0.032175429
    ↪ Get_commit_ts_time: 0.000177098 Local_latch_wait_time: 0.106869448
    ↪ Write_keys: 131072 Write_size: 3538944 Prewrite_region: 1
## DB: test
## Is_internal: false
## Digest: 50a2e32d2abbd6c1764b1b7f2058d428ef2712b029282b776beb9506a365c0f1
## Stats: t:pseudo
## Num_cop_tasks: 1
## Cop_proc_avg: 0.07 Cop_proc_p90: 0.07 Cop_proc_max: 0.07 Cop_proc_addr:
    ↪ 172.16.5.87:20171
## Cop_wait_avg: 0 Cop_wait_p90: 0 Cop_wait_max: 0 Cop_wait_addr:
    ↪ 172.16.5.87:20171
## Cop_backoff_regionMiss_total_times: 200 Cop_backoff_regionMiss_total_time
    ↪ : 0.2 Cop_backoff_regionMiss_max_time: 0.2
    ↪ Cop_backoff_regionMiss_max_addr: 127.0.0.1
    ↪ Cop_backoff_regionMiss_avg_time: 0.2 Cop_backoff_regionMiss_p90_time:
    ↪  0.2
## Cop_backoff_rpcPD_total_times: 200 Cop_backoff_rpcPD_total_time: 0.2
    ↪ Cop_backoff_rpcPD_max_time: 0.2 Cop_backoff_rpcPD_max_addr: 127.0.0.1
    ↪  Cop_backoff_rpcPD_avg_time: 0.2 Cop_backoff_rpcPD_p90_time: 0.2
## Cop_backoff_rpcTiKV_total_times: 200 Cop_backoff_rpcTiKV_total_time: 0.2
    ↪ Cop_backoff_rpcTiKV_max_time: 0.2 Cop_backoff_rpcTiKV_max_addr:
    ↪ 127.0.0.1 Cop_backoff_rpcTiKV_avg_time: 0.2
    ↪ Cop_backoff_rpcTiKV_p90_time: 0.2
## Mem_max: 525211
## Disk_max: 65536
## Prepared: false
## Plan_from_cache: false
## Succ: true
## Plan: tidb_decode_plan('
    ↪ ZJAwCTMyXzcJMAkyMAlkYXRhOlRhYmxlU2Nhbl82CjEJMTBfNgkxAR0AdAEY1DpOLCByYW5nZTpbLWluZi
    ↪ ==')
use test;
insert into t select * from t;
```

10.2.2 Fields description

> **Note:**
>
> The unit of all the following time fields in the slow query log is **"second"**.

Slow query basics:

- `Time`: The print time of log.
- `Query_time`: The execution time of a statement.
- `Parse_time`: The parsing time for the statement.
- `Compile_time`: The duration of the query optimization.
- `Optimize_time`: The time consumed for optimizing the execution plan.
- `Wait_TS`: The waiting time of the statement to get transaction timestamps.
- `Query`: A SQL statement. `Query` is not printed in the slow log, but the corresponding field is called `Query` after the slow log is mapped to the memory table.
- `Digest`: The fingerprint of the SQL statement.
- `Txn_start_ts`: The start timestamp and the unique ID of a transaction. You can use this value to search for the transaction-related logs.
- `Is_internal`: Whether a SQL statement is TiDB internal. `true` indicates that a SQL statement is executed internally in TiDB and `false` indicates that a SQL statement is executed by the user.
- `Index_names`: The index names used by the statement.
- `Stats`: The health state of the involved tables. `pseudo` indicates that the state is unhealthy.
- `Succ`: Whether a statement is executed successfully.
- `Backoff_time`: The waiting time before retry when a statement encounters errors that require a retry. The common errors as such include: `lock occurs`, `Region split`, and `tikv server is busy`.
- `Plan`: The execution plan of the statement. Use the `select tidb_decode_plan('xxx ...')` statement to parse the specific execution plan.
- `Prepared`: Whether this statement is a `Prepare` or `Execute` request or not.
- `Plan_from_cache`: Whether this statement hits the execution plan cache.
- `Plan_from_binding`: Whether this statement uses the bound execution plans.
- `Has_more_results`: Whether this statement has more results to be fetched by users.
- `Rewrite_time`: The time consumed for rewriting the query of this statement.
- `Preproc_subqueries`: The number of subqueries (in the statement) that are executed in advance. For example, the `where id in (select if from t)` subquery might be executed in advance.
- `Preproc_subqueries_time`: The time consumed for executing the subquery of this statement in advance.
- `Exec_retry_count`: The retry times of this statement. This field is usually for pessimistic transactions in which the statement is retried when the lock is failed.
- `Exec_retry_time`: The execution retry duration of this statement. For example, if a statement has been executed three times in total (failed for the first two times), `Exec_retry_time` means the total duration of the first two executions. The duration of the last execution is `Query_time` minus `Exec_retry_time`.
- `KV_total`: The time spent on all the RPC requests on TiKV or TiFlash by this statement.
- `PD_total`: The time spent on all the RPC requests on PD by this statement.

- `Backoff_total`: The time spent on all the backoff during the execution of this statement.
- `Write_sql_response_total`: The time consumed for sending the results back to the client by this statement.
- `Result_rows`: The row count of the query results.
- `IsExplicitTxn`: Whether this statement is in an explicit transaction. If the value is `false`, the transaction is `autocommit=1` and the statement is automatically committed after execution.

The following fields are related to transaction execution:

- `Prewrite_time`: The duration of the first phase (prewrite) of the two-phase transaction commit.
- `Commit_time`: The duration of the second phase (commit) of the two-phase transaction commit.
- `Get_commit_ts_time`: The time spent on getting `commit_ts` during the second phase (commit) of the two-phase transaction commit.
- `Local_latch_wait_time`: The time that TiDB spends on waiting for the lock before the second phase (commit) of the two-phase transaction commit.
- `Write_keys`: The count of keys that the transaction writes to the Write CF in TiKV.
- `Write_size`: The total size of the keys or values to be written when the transaction commits.
- `Prewrite_region`: The number of TiKV Regions involved in the first phase (prewrite) of the two-phase transaction commit. Each Region triggers a remote procedure call.

Memory usage fields:

- `Mem_max`: The maximum memory space used during the execution period of a SQL statement (the unit is byte).

Hard disk fields:

- `Disk_max`: The maximum disk space used during the execution period of a SQL statement (the unit is byte).

User fields:

- `User`: The name of the user who executes this statement.
- `Host`: The host name of this statement.
- `Conn_ID`: The Connection ID (session ID). For example, you can use the keyword `con:3` to search for the log whose session ID is `3`.
- `DB`: The current database.

TiKV Coprocessor Task fields:

- `Request_count`: The number of Coprocessor requests that a statement sends.
- `Total_keys`: The number of keys that Coprocessor has scanned.
- `Process_time`: The total processing time of a SQL statement in TiKV. Because data is sent to TiKV concurrently, this value might exceed `Query_time`.
- `Wait_time`: The total waiting time of a statement in TiKV. Because the Coprocessor of TiKV runs a limited number of threads, requests might queue up when all threads of Coprocessor are working. When a request in the queue takes a long time to process, the waiting time of the subsequent requests increases.
- `Process_keys`: The number of keys that Coprocessor has processed. Compared with `total_keys`, `processed_keys` does not include the old versions of MVCC. A great difference between `processed_keys` and `total_keys` indicates that many old versions exist.
- `Num_cop_tasks`: The number of Coprocessor tasks sent by this statement.
- `Cop_proc_avg`: The average execution time of cop-tasks, including some waiting time that cannot be counted, such as the mutex in RocksDB.
- `Cop_proc_p90`: The P90 execution time of cop-tasks.
- `Cop_proc_max`: The maximum execution time of cop-tasks.
- `Cop_proc_addr`: The address of the cop-task with the longest execution time.
- `Cop_wait_avg`: The average waiting time of cop-tasks, including the time of request queueing and getting snapshots.
- `Cop_wait_p90`: The P90 waiting time of cop-tasks.
- `Cop_wait_max`: The maximum waiting time of cop-tasks.
- `Cop_wait_addr`: The address of the cop-task whose waiting time is the longest.
- `Cop_backoff_{backoff-type}_total_times`: The total times of backoff caused by an error.
- `Cop_backoff_{backoff-type}_total_time`: The total time of backoff caused by an error.
- `Cop_backoff_{backoff-type}_max_time`: The longest time of backoff caused by an error.
- `Cop_backoff_{backoff-type}_max_addr`: The address of the cop-task that has the longest backoff time caused by an error.
- `Cop_backoff_{backoff-type}_avg_time`: The average time of backoff caused by an error.
- `Cop_backoff_{backoff-type}_p90_time`: The P90 percentile backoff time caused by an error.

10.2.3 Related system variables

- `tidb_slow_log_threshold`: Sets the threshold for the slow log. The SQL statement whose execution time exceeds this threshold is recorded in the slow log. The default value is 300 (ms).

- `tidb_query_log_max_len`: Sets the maximum length of the SQL statement recorded in the slow log. The default value is 4096 (byte).
- tidb_redact_log: Determines whether to desensitize user data using `?` in the SQL statement recorded in the slow log. The default value is `0`, which means to disable the feature.
- `tidb_enable_collect_execution_info`: Determines whether to record the physical execution information of each operator in the execution plan. The default value is `1`. This feature impacts the performance by approximately 3%. After enabling this feature, you can view the `Plan` information as follows:

```
> select tidb_decode_plan('
   ↪ jAOIMAk1XzE3CTAJMQlmdW5jczpjb3VudChDb2x1bW4jNyktPkMJC/
   ↪ BMNQkxCXRpbWU6MTAuOTMxNTA1bXMsIGxvb3BzOjIJMzcyIEJ5dGVzCU4vQQoxCTMyXzE4CTAJMQl
   ↪ ');
+--
   ↪ ------------------------------------------------------------------------------
   ↪
| tidb_decode_plan('
   ↪ jAOIMAk1XzE3CTAJMQlmdW5jczpjb3VudChDb2x1bW4jNyktPkMJC/
   ↪ BMNQkxCXRpbWU6MTAuOTMxNTA1bXMsIGxvb3BzOjIJMzcyIEJ5dGVzCU4vQQoxCTMyXzE4CTAJMQl
   ↪  |
+------------------------------------------------------------------------------------
   ↪
|    id                  task   estRows            operator info
   ↪                                          actRows   execution info
   ↪                                                         memory
   ↪     disk                          |
|    StreamAgg_17        root   1                  funcs:count(Column
   ↪ #7)->Column#5                        1         time:10.931505ms,
   ↪ loops:2                                             372 Bytes   N
   ↪ /A                            |
|     ─IndexReader_18    root   1                  index:StreamAgg_9
   ↪                                        1         time:10.927685ms,
   ↪ loops:2, rpc num: 1, rpc time:10.884355ms, proc keys:25007 206
   ↪ Bytes N/A                      |
|       ─StreamAgg_9     cop    1                  funcs:count(1)->
   ↪ Column#7                               1         time:11ms, loops
   ↪ :25                                                  N/A
   ↪         N/A                            |
|         ─IndexScan_16  cop    31281.857819905217 table:t, index:idx(
   ↪ a), range:[-inf,50000), keep order:false 25007 time:11ms, loops
   ↪ :25                                                 N/A         N
   ↪ /A                            |
+------------------------------------------------------------------------------------
```

```
    ↪
```

If you are conducting a performance test, you can disable the feature of automatically collecting the execution information of operators:

```
set @@tidb_enable_collect_execution_info=0;
```

The returned result of the `Plan` field has roughly the same format with that of `EXPLAIN` or `EXPLAIN ANALYZE`. For more details of the execution plan, see `EXPLAIN` or `EXPLAIN ANALYZE`.

For more information, see TiDB specific variables and syntax.

10.2.4 Memory mapping in slow log

You can query the content of the slow query log by querying the `INFORMATION_SCHEMA.` ↪ `SLOW_QUERY` table. Each column name in the table corresponds to one field name in the slow log. For table structure, see the introduction to the `SLOW_QUERY` table in Information Schema.

> **Note:**
>
> Every time you query the `SLOW_QUERY` table, TiDB reads and parses the current slow query log.

For TiDB 4.0, `SLOW_QUERY` supports querying the slow log of any period of time, including the rotated slow log file. You need to specify the `TIME` range to locate the slow log files that need to be parsed. If you don't specify the `TIME` range, TiDB only parses the current slow log file. For example:

- If you don't specify the time range, TiDB only parses the slow query data that TiDB is writing to the slow log file:

```
select count(*),
     min(time),
     max(time)
from slow_query;
```

```
+----------+----------------------------+----------------------------+
| count(*) | min(time)                  | max(time)                  |
+----------+----------------------------+----------------------------+
| 122492   | 2020-03-11 23:35:20.908574 | 2020-03-25 19:16:38.229035 |
+----------+----------------------------+----------------------------+
```

- If you specify the time range, for example, from `2020-03-10 00:00:00` to `2020-03-11 00:00:00`, TiDB first locates the slow log files of the specified time range, and then parses the slow query information:

```
select count(*),
     min(time),
     max(time)
from slow_query
where time > '2020-03-10 00:00:00'
  and time < '2020-03-11 00:00:00';
```

```
+----------+----------------------------+----------------------------+
| count(*) | min(time)                  | max(time)                  |
+----------+----------------------------+----------------------------+
| 2618049  | 2020-03-10 00:00:00.427138 | 2020-03-10 23:00:22.716728 |
+----------+----------------------------+----------------------------+
```

> **Note:**
>
> If the slow log files of the specified time range are removed, or there is no slow query, the query returns NULL.

TiDB 4.0 adds the `CLUSTER_SLOW_QUERY` system table to query the slow query information of all TiDB nodes. The table schema of the `CLUSTER_SLOW_QUERY` table differs from that of the `SLOW_QUERY` table in that an `INSTANCE` column is added to `CLUSTER_SLOW_QUERY`. The `INSTANCE` column represents the TiDB node address of the row information on the slow query. You can use `CLUSTER_SLOW_QUERY` the way you do with `SLOW_QUERY`.

When you query the `CLUSTER_SLOW_QUERY` table, TiDB pushes the computation and the judgment down to other nodes, instead of retrieving all slow query information from other nodes and executing the operations on one TiDB node.

10.2.5 `SLOW_QUERY` / `CLUSTER_SLOW_QUERY` usage examples

10.2.5.1 Top-N slow queries

Query the Top 2 slow queries of users. `Is_internal=false` means excluding slow queries inside TiDB and only querying slow queries of users.

```
select query_time, query
from information_schema.slow_query
where is_internal = false
order by query_time desc
limit 2;
```

Output example:

```
+-------------+-------------------------------------------------------------------------+
   ↪
| query_time | query                                                                    |
+-------------+-------------------------------------------------------------------------+
   ↪
| 12.77583857 | select * from t_slim, t_wide where t_slim.c0=t_wide.c0;                  |
|  0.734982725 | select t0.c0, t1.c1 from t_slim t0, t_wide t1 where t0.c0=
   ↪ t1.c0;                                                                            |
+-------------+-------------------------------------------------------------------------+
   ↪
```

10.2.5.2 Query the Top-N slow queries of the `test` user

In the following example, the slow queries executed by the `test` user are queried, and the first two results are displayed in reverse order of execution time.

```
select query_time, query, user
from information_schema.slow_query
where is_internal = false
  and user = "test"
order by query_time desc
limit 2;
```

Output example:

```
+------------+-------------------------------------------------------------------------+------
   ↪
| Query_time | query                                                                    |
   ↪ user         |
+------------+-------------------------------------------------------------------------+------
   ↪
| 0.676408014 | select t0.c0, t1.c1 from t_slim t0, t_wide t1 where t0.c0=t1
   ↪ .c1; | test |
+------------+-------------------------------------------------------------------------+------
   ↪
```

10.2.5.3 Query similar slow queries with the same SQL fingerprints

After querying the Top-N SQL statements, continue to query similar slow queries using the same fingerprints.

1. Acquire Top-N slow queries and the corresponding SQL fingerprints.

```
select query_time, query, digest
from information_schema.slow_query
where is_internal = false
order by query_time desc
limit 1;
```

Output example:

```
+-------------+----------------------------+-------------------------------------
   ↪
| query_time | query                    | digest
   ↪                                                    |
+-------------+----------------------------+-------------------------------------
   ↪
| 0.302558006 | select * from t1 where a=1; | 4751
   ↪ cb6008fda383e22dacb601fde85425dc8f8cf669338d55d944bafb46a6fa |
+-------------+----------------------------+-------------------------------------
   ↪
```

2. Query similar slow queries with the fingerprints.

```
select query, query_time
from information_schema.slow_query
where digest = "4751
   ↪ cb6008fda383e22dacb601fde85425dc8f8cf669338d55d944bafb46a6fa";
```

Output example:

```
+----------------------------+-------------+
| query                    | query_time |
+----------------------------+-------------+
| select * from t1 where a=1; | 0.302558006 |
| select * from t1 where a=2; | 0.401313532 |
+----------------------------+-------------+
```

10.2.6 Query slow queries with pseudo stats

```
select query, query_time, stats
from information_schema.slow_query
where is_internal = false
  and stats like '%pseudo%';
```

Output example:

```
+----------------------------+------------+--------------------------------+
   ↪
| query                      | query_time | stats                          |
+----------------------------+------------+--------------------------------+
   ↪
| select * from t1 where a=1; | 0.302558006 | t1:pseudo                     |
| select * from t1 where a=2; | 0.401313532 | t1:pseudo                     |
| select * from t1 where a>2; | 0.602011247 | t1:pseudo                     |
| select * from t1 where a>3; | 0.50077719 | t1:pseudo                      |
| select * from t1 join t2; | 0.931260518 | t1:407872303825682445,t2:pseudo
   ↪ |
+----------------------------+------------+--------------------------------+
   ↪
```

10.2.6.1 Query slow queries whose execution plan is changed

When the execution plan of SQL statements of the same category is changed, the execution slows down, because the statistics is outdated, or the statistics is not accurate enough to reflect the real data distribution. You can use the following SQL statement to query SQL statements with different execution plans.

```
select count(distinct plan_digest) as count,
      digest,
      min(query)
from cluster_slow_query
group by digest
having count > 1
limit 3\G
```

Output example:

```
***************************[ 1. row ]***************************
count     | 2
digest    | 17b4518fde82e32021877878bec2bb309619d384fca944106fcaf9c93b536e94
min(query) | SELECT DISTINCT c FROM sbtest25 WHERE id BETWEEN ? AND ? ORDER
   ↪ BY c [arguments: (291638, 291737)];
***************************[ 2. row ]***************************
count     | 2
digest    | 9337865f3e2ee71c1c2e740e773b6dd85f23ad00f8fa1f11a795e62e15fc9b23
min(query) | SELECT DISTINCT c FROM sbtest22 WHERE id BETWEEN ? AND ? ORDER
   ↪ BY c [arguments: (215420, 215519)];
***************************[ 3. row ]***************************
count     | 2
digest    | db705c89ca2dfc1d39d10e0f30f285cbbadec7e24da4f15af461b148d8ffb020
```

```
min(query) | SELECT DISTINCT c FROM sbtest11 WHERE id BETWEEN ? AND ? ORDER
    ↪ BY c [arguments: (303359, 303458)];
```

Then you can query the different plans using the SQL fingerprint in the query result above:

```
select min(plan),
       plan_digest
from cluster_slow_query
where digest='17
    ↪ b4518fde82e32021877878bec2bb309619d384fca944106fcaf9c93b536e94'
group by plan_digest\G
```

Output example:

```
*************************** 1. row ***************************
  min(plan):  Sort_6                root    100.00131380758702   sbtest.
     ↪ sbtest25.c:asc
        ─HashAgg_10          root    100.00131380758702   group by:sbtest.
          ↪ sbtest25.c, funcs:firstrow(sbtest.sbtest25.c)->sbtest.sbtest25
          ↪ .c
          ─TableReader_15    root    100.00131380758702   data:
            ↪ TableRangeScan_14
            ─TableScan_14    cop     100.00131380758702   table:sbtest25,
              ↪ range:[502791,502890], keep order:false
plan_digest: 6
    ↪ afbbd21f60ca6c6fdf3d3cd94f7c7a49dd93c00fcf8774646da492e50e204ee
*************************** 2. row ***************************
  min(plan):  Sort_6                root    1                    sbtest.
     ↪ sbtest25.c:asc
        ─HashAgg_12          root    1                    group by:sbtest.
          ↪ sbtest25.c, funcs:firstrow(sbtest.sbtest25.c)->sbtest.sbtest25
          ↪ .c
          ─TableReader_13    root    1                    data:HashAgg_8
            ─HashAgg_8       cop     1                    group by:sbtest.
              ↪ sbtest25.c,
              ─TableScan_11  cop     1.2440069558121831   table:sbtest25,
                ↪ range:[472745,472844], keep order:false
```

10.2.6.2 Query the number of slow queries for each TiDB node in a cluster

```
select instance, count(*) from information_schema.cluster_slow_query where
    ↪ time >= "2020-03-06 00:00:00" and time < now() group by instance;
```

Output example:

```
+---------------+----------+
| instance      | count(*) |
+---------------+----------+
| 0.0.0.0:10081 | 124      |
| 0.0.0.0:10080 | 119771   |
+---------------+----------+
```

10.2.6.3 Query slow logs occurring only in abnormal time period

If you find problems such as decreased QPS or increased latency for the time period from `2020-03-10 13:24:00` to `2020-03-10 13:27:00`, the reason might be that a large query crops up. Run the following SQL statement to query slow logs that occur only in abnormal time period. The time range from `2020-03-10 13:20:00` to `2020-03-10 13:23:00` refers to the normal time period.

```
SELECT * FROM
    (SELECT /*+ AGG_TO_COP(), HASH_AGG() */ count(*),
        min(time),
        sum(query_time) AS sum_query_time,
        sum(Process_time) AS sum_process_time,
        sum(Wait_time) AS sum_wait_time,
        sum(Commit_time),
        sum(Request_count),
        sum(process_keys),
        sum(Write_keys),
        max(Cop_proc_max),
        min(query),min(prev_stmt),
        digest
    FROM information_schema.CLUSTER_SLOW_QUERY
    WHERE time >= '2020-03-10 13:24:00'
           AND time < '2020-03-10 13:27:00'
           AND Is_internal = false
    GROUP BY digest) AS t1
WHERE t1.digest NOT IN
    (SELECT /*+ AGG_TO_COP(), HASH_AGG() */ digest
    FROM information_schema.CLUSTER_SLOW_QUERY
    WHERE time >= '2020-03-10 13:20:00'
           AND time < '2020-03-10 13:23:00'
    GROUP BY digest)
ORDER BY t1.sum_query_time DESC limit 10\G
```

Output example:

```
***************************[ 1. row ]***************************
count(*)         | 200
```

```
min(time)        | 2020-03-10 13:24:27.216186
sum_query_time   | 50.114126194
sum_process_time | 268.351
sum_wait_time    | 8.476
sum(Commit_time) | 1.044304306
sum(Request_count) | 6077
sum(process_keys) | 202871950
sum(Write_keys)  | 319500
max(Cop_proc_max) | 0.263
min(query)       | delete from test.tcs2 limit 5000;
min(prev_stmt)   |
digest           | 24
    ↪ bd6d8a9b238086c9b8c3d240ad4ef32f79ce94cf5a468c0b8fe1eb5f8d03df
```

10.2.6.4 Parse other TiDB slow log files

TiDB uses the session variable `tidb_slow_query_file` to control the files to be read and parsed when querying `INFORMATION_SCHEMA.SLOW_QUERY`. You can query the content of other slow query log files by modifying the value of the session variable.

```
set tidb_slow_query_file = "/path-to-log/tidb-slow.log"
```

10.2.6.5 Parse TiDB slow logs with `pt-query-digest`

Use `pt-query-digest` to parse TiDB slow logs.

> **Note:**
>
> It is recommended to use `pt-query-digest` 3.0.13 or later versions.

For example:

```
pt-query-digest --report tidb-slow.log
```

Output example:

```
## 320ms user time, 20ms system time, 27.00M rss, 221.32M vsz
## Current date: Mon Mar 18 13:18:51 2019
## Hostname: localhost.localdomain
## Files: tidb-slow.log
## Overall: 1.02k total, 21 unique, 0 QPS, 0x concurrency _________________
## Time range: 2019-03-18-12:22:16 to 2019-03-18-13:08:52
## Attribute       total    min    max    avg    95% stddev median
```

```
## ============   ======= ======= ======= ======= ======= ======= =======
## Exec time        218s   10ms    13s   213ms   30ms     1s    19ms
## Query size     175.37k     9   2.01k 175.89 158.58 122.36 158.58
## Commit time      46ms    2ms    7ms    3ms    7ms    1ms    3ms
## Conn ID            71      1     16   8.88  15.25   4.06   9.83
## Process keys   581.87k     2 103.15k 596.43 400.73  3.91k 400.73
## Process time      31s    1ms    10s   32ms   19ms  334ms   16ms
## Request coun    1.97k      1     10   2.02   1.96   0.33   1.96
## Total keys     636.43k     2 103.16k 652.35 793.42  3.97k 400.73
## Txn start ts   374.38E     0  16.00E 375.48P 1.25P 89.05T  1.25P
## Wait time        943ms    1ms   19ms    1ms    2ms    1ms   972us
.
.
.
```

10.2.7 Identify problematic SQL statements

Not all of the `SLOW_QUERY` statements are problematic. Only those whose `process_time` is very large increase the pressure on the entire cluster.

The statements whose `wait_time` is very large and `process_time` is very small are usually not problematic. This is because the statement is blocked by real problematic statements and it has to wait in the execution queue, which leads to a much longer response time.

10.2.7.1 `ADMIN SHOW SLOW` command

In addition to the TiDB log file, you can identify slow queries by running the `ADMIN SHOW SLOW` command:

```
ADMIN SHOW SLOW recent N
ADMIN SHOW SLOW TOP [internal | all] N
```

`recent N` shows the recent N slow query records, for example:

```
ADMIN SHOW SLOW recent 10
```

`top N` shows the slowest N query records recently (within a few days). If the `internal` option is provided, the returned results would be the inner SQL executed by the system; If the `all` option is provided, the returned results would be the user's SQL combinated with inner SQL; Otherwise, this command would only return the slow query records from the user's SQL.

```
ADMIN SHOW SLOW top 3
ADMIN SHOW SLOW top internal 3
ADMIN SHOW SLOW top all 5
```

TiDB stores only a limited number of slow query records because of the limited memory. If the value of `N` in the query command is greater than the records count, the number of returned records is smaller than `N`.

The following table shows output details:

Column name	Description
start	The starting time of the SQL execution
duration	The duration of the SQL execution
details	The details of the SQL execution

Column name	Description
succ	Whether the SQL statement is executed successfully. `1` means success and `0` means failure.
conn_id	The connection ID for the session
transaction_ts	The `commit ↪ ↪ ts` for a transaction commit

Column name	Description
user	The user name for the execution of the statement
db	The database involved when the statement is executed
table_ids	The ID of the table involved when the SQL statement is executed

Column name	Description
index_ids	The ID of the index involved when the SQL statement is executed
internal	This is a TiDB internal SQL statement
digest	The fingerprint of the SQL statement
sql	The SQL statement that is being executed or has been executed

10.3 Analyze Slow Queries

To address the issue of slow queries, you need to take the following two steps:

1. Among many queries, identify which type of queries are slow.
2. Analyze why this type of queries are slow.

You can easily perform step 1 using the slow query log and the statement summary table features. It is recommended to use TiDB Dashboard, which integrates the two features and directly displays the slow queries in your browser.

This document focuses on how to perform step 2 - analyze why this type of queries are slow.

Generally, slow queries have the following major causes:

- Optimizer issues, such as wrong index selected, wrong join type or sequence selected.
- System issues. All issues not caused by the optimizer are system issues. For example, a busy TiKV instance processes requests slowly; outdated Region information causes slow queries.

In actual situations, optimizer issues might cause system issues. For example, for a certain type of queries, the optimizer uses a full table scan instead of the index. As a result, the SQL queries consume many resources, which causes the CPU usage of some TiKV instances to soar. This seems like a system issue, but in essence, it is an optimizer issue.

To identify system issues is relatively simple. To analyze optimizer issues, you need to determine whether the execution plan is reasonable or not. Therefore, it is recommended to analyze slow queries by following these procedures:

1. Identify the performance bottleneck of the query, that is, the time-consuming part of the query process.
2. Analyze the system issues: analyze the possible causes according to the query bottleneck and the monitoring/log information of that time.
3. Analyze the optimizer issues: analyze whether there is a better execution plan.

The procedures above are explained in the following sections.

10.3.1 Identify the performance bottleneck of the query

First, you need to have a general understanding of the query process. The key stages of the query execution process in TiDB are illustrated in TiDB performance map.

You can get the duration information using the following methods:

- Slow log. It is recommended to view the slow log in TiDB Dashboard.
- `EXPLAIN ANALYZE` statement.

The methods above are different in the following aspects:

- The slow log records the duration of almost all stages of a SQL execution, from parsing to returning results, and is relatively comprehensive (you can query and analyze the slow log in TiDB Dashboard in an intuitive way).
- By executing `EXPLAIN ANALYZE`, you can learn the time consumption of each operator in an actual SQL execution. The results have more detailed statistics of the execution duration.

In summary, the slow log and `EXPLAIN ANALYZE` statements help you determine the SQL query is slow in which component (TiDB or TiKV) at which stage of the execution. Therefore, you can accurately identify the performance bottleneck of the query.

In addition, since v4.0.3, the `Plan` field in the slow log also includes the SQL execution information, which is the result of `EXPLAIN ANALYZE`. So you can find all information of SQL duration in the slow log.

10.3.2 Analyze system issues

System issues can be divided into the following types according to different execution stages of a SQL statement:

1. TiKV is slow in data processing. For example, the TiKV coprocessor processes data slowly.
2. TiDB is slow in execution. For example, a `Join` operator processes data slowly.
3. Other key stages are slow. For example, getting the timestamp takes a long time.

For each slow query, first determine to which type the query belongs, and then analyze it in detail.

10.3.2.1 TiKV is slow in data processing

If TiKV is slow in data processing, you can easily identify it in the result of `EXPLAIN ANALYZE`. In the following example, `StreamAgg_8` and `TableFullScan_15`, two `tikv-tasks` (as indicated by `cop[tikv]` in the `task` column), take `170ms` to execute. After subtracting `170ms`, the execution time of TiDB operators account for a very small proportion of the total execution time. This indicates that the bottleneck is in TiKV.

```
+--
    ↪ ------------------------+---------+---------+-----------+-----------------+-----
    ↪
```

```
| id                    | estRows | actRows | task   | access object |
    ↪ execution info                                                  |
    ↪ operator info               | memory   | disk |
+--
    ↪ ----------------------+---------+---------+----------+---------------+-----
    ↪
| StreamAgg_16            | 1.00    | 1       | root     |               | time
    ↪ :170.08572ms, loops:2                                             |
    ↪ funcs:count(Column#5)->Column#3 | 372 Bytes | N/A |
|  ─TableReader_17        | 1.00    | 1       | root     |               | time
    ↪ :170.080369ms, loops:2, rpc num: 1, rpc time:17.023347ms, proc keys
    ↪ :28672 | data:StreamAgg_8  | 202 Bytes | N/A |
|    ─StreamAgg_8         | 1.00    | 1       | cop[tikv] |              | time
    ↪ :170ms, loops:29                                                  |
    ↪ funcs:count(1)->Column#5    | N/A      | N/A |
|      ─TableFullScan_15  | 7.00    | 28672   | cop[tikv] | table:t    | time
    ↪ :170ms, loops:29                                                  |
    ↪ keep order:false, stats:pseudo | N/A   | N/A |
+--
    ↪ ----------------------+---------+---------+----------+---------------+-----
    ↪
```

In addition, the `Cop_process` and `Cop_wait` fields in the slow log can also help your analysis. In the following example, the total duration of the query is around `180.85ms`, and the largest `coptask` takes `171ms`. This indicates that the bottleneck of this query is on the TiKV side.

For the description of each field in the slow log, see fields description.

```
## Query_time: 0.18085
...
## Num_cop_tasks: 1
## Cop_process: Avg_time: 170ms P90_time: 170ms Max_time: 170ms Max_addr:
    ↪ 10.6.131.78
## Cop_wait: Avg_time: 1ms P90_time: 1ms Max_time: 1ms Max_Addr: 10.6.131.78
```

After identifying that TiKV is the bottleneck, you can find out the cause as described in the following sections.

10.3.2.1.1 TiKV instance is busy

During the execution of a SQL statement, TiDB might fetch data from multiple TiKV instances. If one TiKV instance responds slowly, the overall SQL execution speed is slowed down.

The `Cop_wait` field in the slow log can help you determine this cause.

```
## Cop_wait: Avg_time: 1ms P90_time: 2ms Max_time: 110ms Max_Addr:
   ↪ 10.6.131.78
```

The log above shows that a `cop-task` sent to the `10.6.131.78` instance waits `110ms` ↪ before being executed. It indicates that this instance is busy. You can check the CPU monitoring of that time to confirm the cause.

10.3.2.1.2 Too many outdated keys

A TiKV instance has much outdated data, which needs to be cleaned up for data scan. This impacts the processing speed.

Check `Total_keys` and `Processed_keys`. If they are greatly different, the TiKV instance has too many keys of the older versions.

```
...
## Total_keys: 2215187529 Processed_keys: 1108056368
...
```

10.3.2.2 Other key stages are slow

10.3.2.2.1 Slow in getting timestamps

You can compare `Wait_TS` and `Query_time` in the slow log. The timestamps are prefetched, so generally `Wait_TS` should be low.

```
## Query_time: 0.0300000
...
## Wait_TS: 0.02500000
```

10.3.2.2.2 Outdated Region information

Region information on the TiDB side might be outdated. In this situation, TiKV might return the `regionMiss` error. Then TiDB gets the Region information from PD again, which is reflected in the `Cop_backoff` information. Both the failed times and the total duration are recorded.

```
## Cop_backoff_regionMiss_total_times: 200 Cop_backoff_regionMiss_total_time
   ↪ : 0.2 Cop_backoff_regionMiss_max_time: 0.2
   ↪ Cop_backoff_regionMiss_max_addr: 127.0.0.1
   ↪ Cop_backoff_regionMiss_avg_time: 0.2 Cop_backoff_regionMiss_p90_time:
   ↪  0.2
## Cop_backoff_rpcPD_total_times: 200 Cop_backoff_rpcPD_total_time: 0.2
   ↪ Cop_backoff_rpcPD_max_time: 0.2 Cop_backoff_rpcPD_max_addr: 127.0.0.1
   ↪  Cop_backoff_rpcPD_avg_time: 0.2 Cop_backoff_rpcPD_p90_time: 0.2
```

10.3.2.2.3 Subqueries are executed in advance

For statements with non-correlated subqueries, the subquery part might be executed in advance. For example, in `select * from t1 where a = (select max(a)from t2)`, the `select max(a)from t2` part might be executed in advance in the optimization stage. The result of `EXPLAIN ANALYZE` does not show the duration of this type of subqueries.

```
mysql> explain analyze select count(*) from t where a=(select max(t1.a)
    ↪ from t t1, t t2 where t1.a=t2.a);
+--
    ↪ ---------------------------+----------+---------+-----------+---------------+--
    ↪
| id                          | estRows | actRows | task    | access object |
    ↪ execution info      | operator info                 | memory   | disk |
+--
    ↪ ---------------------------+----------+---------+-----------+---------------+--
    ↪
| StreamAgg_59                | 1.00    | 1       | root      |               | time
    ↪ :4.69267ms, loops:2 | funcs:count(Column#10)->Column#8 | 372 Bytes |
    ↪ N/A |
|  ─TableReader_60            | 1.00    | 1       | root      |               |
    ↪ time:4.690428ms, loops:2 | data:StreamAgg_48          | 141 Bytes | N/A
    ↪  |
|    ─StreamAgg_48            | 1.00    |         | cop[tikv] |               |
    ↪ time:0ns, loops:0     | funcs:count(1)->Column#10    | N/A      | N/A
    ↪ |
|      ─Selection_58          | 16384.00 |       | cop[tikv] |               |
    ↪ time:0ns, loops:0     | eq(test.t.a, 1)              | N/A      | N/A
    ↪ |
|        ─TableFullScan_57    | 16384.00 | -1    | cop[tikv] | table:t       |
    ↪ time:0s, loops:0      | keep order:false             | N/A      | N/A
    ↪ |
+--
    ↪ ---------------------------+----------+---------+-----------+---------------+--
    ↪
5 rows in set (7.77 sec)
```

But you can identify this type of subquery execution in the slow log:

```
## Query_time: 7.770634843
...
## Rewrite_time: 7.765673663 Preproc_subqueries: 1 Preproc_subqueries_time:
    ↪ 7.765231874
```

From log record above, you can see that a subquery is executed in advance and takes `7.76s`.

10.3.2.3 TiDB is slow in execution

Assume that the execution plan in TiDB is correct but the execution is slow. To solve this type of issue, you can adjust parameters or use the hint according to the result of `EXPLAIN ANALYZE` for the SQL statement.

If the execution plan is incorrect, see the Analyze optimizer issues section.

10.3.2.3.1 Low concurrency

If the bottleneck is in the operator with concurrency, speed up the execution by adjusting the concurrency. For example:

```
mysql> explain analyze select sum(t1.a) from t t1, t t2 where t1.a=t2.a;
+--
   ↪ ------------------------------+-------------+----------+----------+-------
   ↪
| id                            | estRows     | actRows  | task     | access
   ↪ object | execution info
   ↪                                                                  | operator
   ↪ info                                | memory         | disk   |
+--
   ↪ ------------------------------+-------------+----------+----------+-------
   ↪
| HashAgg_11                    | 1.00        | 1        | root     |
   ↪             | time:9.666832189s, loops:2, PartialConcurrency:4,
   ↪ FinalConcurrency:4     | funcs:sum(Column#6)->Column#5            |
   ↪ 322.125 KB     | N/A    |
|  └Projection_24               | 268435456.00 | 268435456 | root |
   ↪             | time:9.098644711s, loops:262145, Concurrency:4
   ↪                               | cast(test.t.a, decimal(65,0) BINARY)->
   ↪ Column#6 | 199 KB   | N/A    |
|    └HashJoin_14               | 268435456.00 | 268435456 | root |
   ↪             | time:6.616773501s, loops:262145, Concurrency:5, probe
   ↪ collision:0, build:881.404µs | inner join, equal:[eq(test.t.a, test.t
   ↪ .a)] | 131.75 KB | 0 Bytes |
|      └TableReader_21(Build)   | 16384.00    | 16384    | root     |
   ↪             | time:6.553717ms, loops:17
   ↪                                                  | data:Selection_20
   ↪                         | 33.6318359375 KB | N/A  |
|        └Selection_20          | 16384.00    |          | cop[tikv] |
   ↪           | time:0ns, loops:0
   ↪                                                             | not(isnull(
   ↪ test.t.a))                      | N/A            | N/A    |
|          └TableFullScan_19    | 16384.00    | -1       | cop[tikv] | table:
   ↪ t2   | time:0s, loops:0
   ↪                                                              | keep order:
```

```
    ↪ false                          | N/A            | N/A    |
|     └─TableReader_18(Probe)   | 16384.00   | 16384    | root     |
    ↪             | time:6.880923ms, loops:17
    ↪                                                  | data:Selection_17
    ↪                         | 33.6318359375 KB | N/A |
|       └─Selection_17          | 16384.00   |          | cop[tikv] |
    ↪           | time:0ns, loops:0
    ↪                                                        | not(isnull(
    ↪ test.t.a))                     | N/A            | N/A    |
|         └─TableFullScan_16    | 16384.00   | -1       | cop[tikv] | table:
    ↪ t1   | time:0s, loops:0
    ↪                                                         | keep order:
    ↪ false                          | N/A            | N/A    |
+--
    ↪ ------------------------------+-------------+----------+----------+--------
    ↪
9 rows in set (9.67 sec)
```

As shown above, `HashJoin_14` and `Projection_24` consume much of the execution time. Consider increasing their concurrency using SQL variables to speed up execution.

All system variables are documented in system-variables. To increase the concurrency of `HashJoin_14`, you can modify the `tidb_hash_join_concurrency` system variable.

10.3.2.3.2 Data is spilled to disk

Another cause of slow execution is disk spill that occurs during execution if the memory limit is reached. You can find out this cause in the execution plan and the slow log:

```
+--
    ↪ ----------------------+----------+---------+----------+--------------+------
    ↪
| id                    | estRows  | actRows | task    | access object |
    ↪ execution info          | operator info     | memory             |
    ↪ disk         |
+--
    ↪ ----------------------+----------+---------+----------+--------------+------
    ↪
| Sort_4                | 462144.00 | 462144 | root   |              | time
    ↪ :2.02848898s, loops:453 | test.t.a       | 149.68795776367188 MB |
    ↪ 219.3203125 MB |
|  └─TableReader_8      | 462144.00 | 462144 | root   |              | time
    ↪ :616.211272ms, loops:453 | data:TableFullScan_7 | 197.49601364135742
    ↪ MB | N/A |
|    └─TableFullScan_7  | 462144.00 | -1     | cop[tikv] | table:t    | time:0
    ↪ s, loops:0          | keep order:false  | N/A                | N/A
```

```
    ↪          |
+--
    ↪ ---------------------+----------+--------+----------+-----------+------
    ↪
```

```
...
## Disk_max: 229974016
...
```

10.3.2.3.3 Join operations with Cartesian product

Join operations with Cartesian product generate data volume as large as `left child row count * right child row count`. This is inefficient and should be avoided.

This type of join operations is marked `CARTESIAN` in the execution plan. For example:

```
mysql> explain select * from t t1, t t2 where t1.a>t2.a;
+--
    ↪ --------------------------+------------+----------+-------------+--------
    ↪
| id                          | estRows    | task     | access object | operator
    ↪  info                                  |
+--
    ↪ --------------------------+------------+----------+-------------+--------
    ↪
| HashJoin_8                  | 99800100.00 | root   |             | CARTESIAN
    ↪ inner join, other cond:gt(test.t.a, test.t.a) |
|  ─TableReader_15(Build)   | 9990.00    | root     |             | data:
    ↪ Selection_14                                 |
|    ─Selection_14          | 9990.00    | cop[tikv] |            | not(
    ↪ isnull(test.t.a))                             |
|      ─TableFullScan_13    | 10000.00   | cop[tikv] | table:t2   | keep
    ↪ order:false, stats:pseudo                     |
|  ─TableReader_12(Probe)   | 9990.00    | root     |             | data:
    ↪ Selection_11                                 |
|    ─Selection_11          | 9990.00    | cop[tikv] |            | not(
    ↪ isnull(test.t.a))                             |
|      ─TableFullScan_10    | 10000.00   | cop[tikv] | table:t1   | keep
    ↪ order:false, stats:pseudo                     |
+--
    ↪ --------------------------+------------+----------+-------------+--------
    ↪
```

10.3.3 Analyze optimizer issues

To analyze optimizer issues, you need to determine whether the execution plan is reasonable or not. You need to have some understanding of the optimization process and each operator.

For the following examples, assume that the table schema is `create table t (id int , a int, b int, c int, primary key(id), key(a), key(b, c))`.

1. `select * from t`: There is no filter condition and a full table scan is performed. So the `TableFullScan` operator is used to read data.
2. `select a from t where a=2`: There is a filter condition and only the index columns are read, so the `IndexReader` operator is used to read data.
3. `select * from t where a=2`: There is a filter condition for `a` but the `a` index cannot fully cover the data to be read, so the `IndexLookup` operator is used.
4. `select b from t where c=3`: Without the prefix condition, the multi-column index cannot be used. So the `IndexFullScan` is used.
5. …

The examples above are operators used for data reads. For more operators, see Understand TiDB Execution Plan.

In addition, reading SQL Tuning Overview helps you better understand the TiDB optimizer and determine whether the execution plan is reasonable or not.

Most optimizer issues are explained in SQL Tuning Overview. For the solutions, see the following documents:

1. Wrong Index Solution
2. Wrong join order
3. Expressions are not pushed down

10.4 SQL Diagnostics

> **Warning:**
>
> SQL diagnostics is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

SQL diagnostics is a feature introduced in TiDB v4.0. You can use this feature to locate problems in TiDB with higher efficiency. Before TiDB v4.0, you need to use different tools to obtain different information.

The SQL diagnostic system has the following advantages:

- It integrates information from all components of the system as a whole.
- It provides a consistent interface to the upper layer through system tables.
- It provides monitoring summaries and automatic diagnostics.
- You will find it easier to query cluster information.

10.4.1 Overview

The SQL diagnostic system consists of three major parts:

- **Cluster information table**: The SQL diagnostics system introduces cluster information tables that provide a unified way to get the discrete information of each instance. This system fully integrates the cluster topology, hardware information, software information, kernel parameters, monitoring, system information, slow queries, statements, and logs of the entire cluster into the table. So you can query these information using SQL statements.
- **Cluster monitoring table**: The SQL diagnostic system introduces cluster monitoring tables. All of these tables are in `metrics_schema`, and you can query monitoring information using SQL statements. Compared to the visualized monitoring before v4.0, you can use this SQL-based method to perform correlated queries on all the monitoring information of the entire cluster, and compare the results of different time periods to quickly identify performance bottlenecks. Because the TiDB cluster has many monitoring metrics, the SQL diagnostic system also provides monitoring summary tables, so you can find abnormal monitoring items more easily.

Automatic diagnostics: Although you can manually execute SQL statements to query cluster information tables, cluster monitoring tables, and summary tables to locate issues, the automatic diagnostics allows you to quickly locate common issues. The SQL diagnostic system performs automatic diagnostics based on the existing cluster information tables and monitoring tables, and provides relevant diagnostic result tables and diagnostic summary tables.

10.4.2 Cluster information tables

The cluster information tables bring together the information of all instances and instances in a cluster. With these tables, you can query all cluster information using only one SQL statement. The following is a list of cluster information tables:

- From the cluster topology table `information_schema.cluster_info`, you can get the current topology information of the cluster, the version of each instance, the Git Hash corresponding to the version, the starting time of each instance, and the running time of each instance.

- From the cluster configuration table `information_schema.cluster_config`, you can get the configuration of all instances in the cluster. For versions earlier than 4.0, you need to access the HTTP API of each instance one by one to get these configuration information.
- On the cluster hardware table `information_schema.cluster_hardware`, you can quickly query the cluster hardware information.
- On the cluster load table `information_schema.cluster_load`, you can query the load information of different instances and hardware types of the cluster.
- On the kernel parameter table `information_schema.cluster_systeminfo`, you can query the kernel configuration information of different instances in the cluster. Currently, TiDB supports querying the sysctl information.
- On the cluster log table `information_schema.cluster_log`, you can query cluster logs. By pushing down query conditions to each instance, the impact of the query on cluster performance is less than that of the `grep` command.

On the system tables earlier than TiDB v4.0, you can only view the current instance. TiDB v4.0 introduces the corresponding cluster tables and you can have a global view of the entire cluster on a single TiDB instance. These tables are currently in `information_schema`, and the query method is the same as other `information_schema` system tables.

10.4.3 Cluster monitoring tables

To dynamically observe and compare cluster conditions in different time periods, the SQL diagnostic system introduces cluster monitoring system tables. All monitoring tables are in `metrics_schema`, and you can query the monitoring information using SQL statements. Using this method, you can perform correlated queries on all monitoring information of the entire cluster and compare the results of different time periods to quickly identify performance bottlenecks.

- `information_schema.metrics_tables`: Because many system tables exist now, you can query meta-information of these monitoring tables on the `information_schema.` ↪ `metrics_tables` table.

Because the TiDB cluster has many monitoring metrics, TiDB provides the following monitoring summary tables in v4.0:

- The monitoring summary table `information_schema.metrics_summary` summarizes all monitoring data to for you to check each monitoring metric with higher efficiency.
- `information_schema.metrics_summary_by_label` also summarizes all monitoring data. Particularly, this table aggregates statistics using different labels of each monitoring metric.

10.4.4 Automatic diagnostics

On the cluster information tables and cluster monitoring tables above, you need to manually execute SQL statements to troubleshoot the cluster. TiDB v4.0 supports the automatic diagnostics. You can use diagnostic-related system tables based on the existing basic information tables, so that the diagnostics is automatically executed. The following are the system tables related to the automatic diagnostics:

- The diagnostic result table `information_schema.inspection_result` displays the diagnostic result of the system. The diagnostics is passively triggered. Executing `select * from inspection_result` triggers all diagnostic rules to diagnose the system, and the faults or risks in the system are displayed in the results.
- The diagnostic summary table `information_schema.inspection_summary` summarizes the monitoring information of a specific link or module. You can troubleshoot and locate problems based on the context of the entire module or link.

10.5 TiDB Dashboard Top SQL Page

With Top SQL, you can monitor and visually explore the CPU overhead of each SQL statement in your database in real-time, which helps you optimize and resolve database performance issues. Top SQL continuously collects and stores CPU load data summarized by SQL statements at any seconds from all TiDB and TiKV instances. The collected data can be stored for up to 30 days. Top SQL presents you with visual charts and tables to quickly pinpoint which SQL statements are contributing the high CPU load of a TiDB or TiKV instance over a certain period of time.

Top SQL provides the following features:

- Visualize the top 5 types of SQL statements with the highest CPU overhead through charts and tables.
- Display detailed execution information such as queries per second, average latency, and query plan.
- Collect all SQL statements that are executed, including those that are still running.
- Allow viewing data of a specific TiDB and TiKV instance.

10.5.1 Recommended scenarios

Top SQL is suitable for analyzing performance issues. The following are some typical Top SQL scenarios:

- You discovered that an individual TiKV instance in the cluster has a very high CPU usage through the Grafana charts. You want to know which SQL statements cause the CPU hotspots so that you can optimize them and better leverage all of your distributed resources.

- You discovered that the cluster has a very high CPU usage overall and queries are slow. You want to quickly figure out which SQL statements are currently consuming the most CPU resources so that you can optimize them.
- The CPU usage of the cluster has drastically changed and you want to know the major cause.
- Analyze the most resource-intensive SQL statements in the cluster and optimize them to reduce hardware costs.

Top SQL cannot be used to pinpoint non-performance issues, such as incorrect data or abnormal crashes.

The Top SQL feature is still in an early stage and is being continuously enhanced. Here are some scenarios that are **not supported** at the moment:

- Analyzing the overhead of SQL statements outside of Top 5 (for example, when multiple business workloads are mixed).
- Analyzing the overhead of Top N SQL statements by various dimensions such as users and databases.
- Analyzing database performance issues that are not caused by high CPU load, such as transaction lock conflicts.

10.5.2 Access the page

You can access the Top SQL page using either of the following methods:

- After logging into TiDB Dashboard, click **Top SQL** on the left navigation bar.

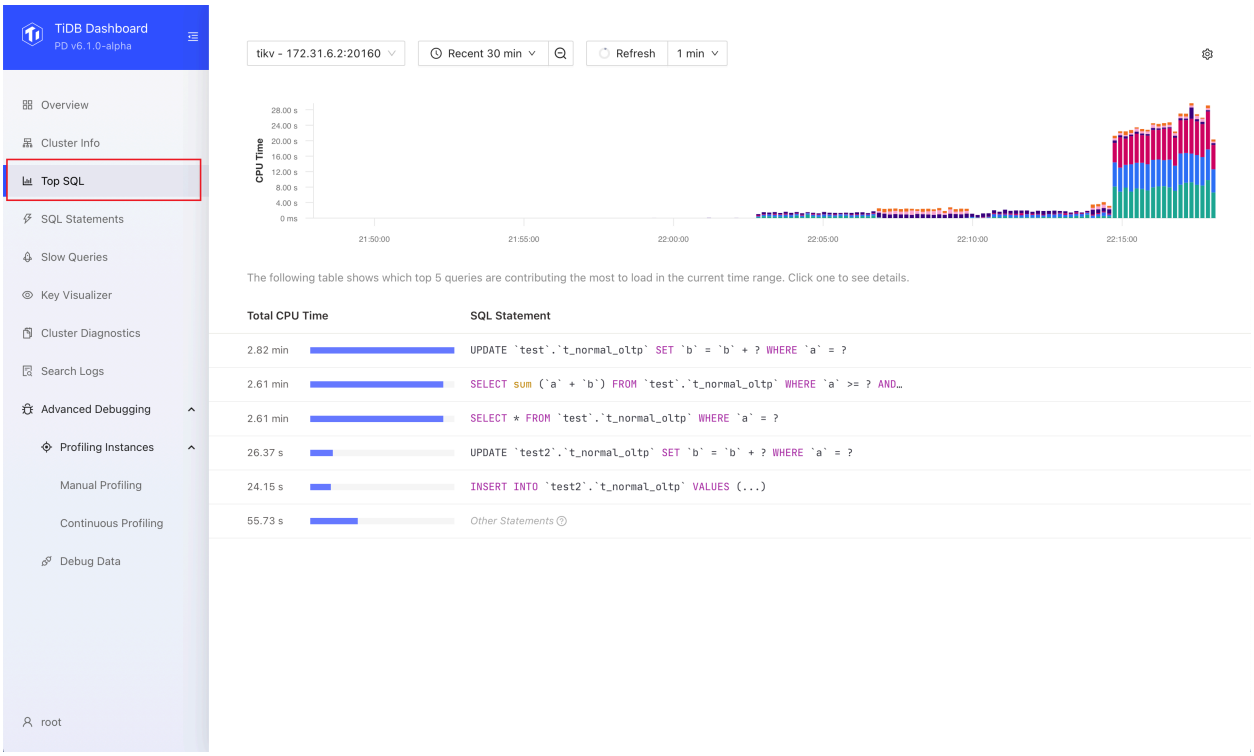


Figure 99: Top SQL

- Visit http://127.0.0.1:2379/dashboard/#/topsql in your browser. Replace `127.0.0.1:2379` ↪ with the actual PD instance address and port.

10.5.3 Enable Top SQL

> **Note:**
>
> To use Top SQL, your cluster should be deployed or upgraded with a recent version of TiUP (v1.9.0 or above) or TiDB Operator (v1.3.0 or above). If your cluster was upgraded using an earlier version of TiUP or TiDB Operator, see FAQ for instructions.

Top SQL is not enabled by default as it has a slight impact on cluster performance (within 3% on average) when enabled. You can enable Top SQL by the following steps:

1. Visit the Top SQL page.
2. Click **Open Settings**. On the right side of the **Settings** area, switch on **Enable Feature**.

3. Click **Save**.

After enabling the feature, wait up to 1 minute for Top SQL to load the data. Then you can see the CPU load details.

In addition to the UI, you can also enable the Top SQL feature by setting the TiDB system variable `tidb_enable_top_sql`:

```
SET GLOBAL tidb_enable_top_sql = 1;
```

10.5.4 Use Top SQL

The following are the common steps to use Top SQL.

1. Visit the Top SQL page.
2. Select a particular TiDB or TiKV instance that you want to observe the load.

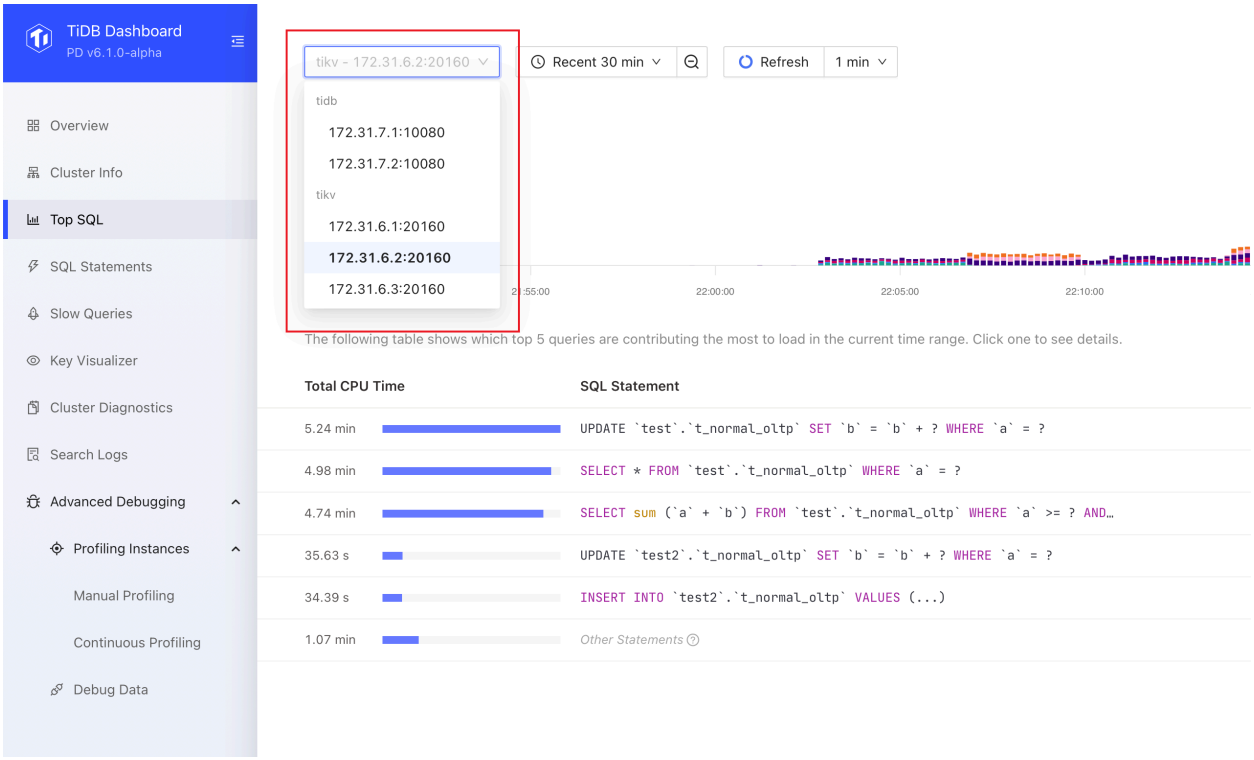


Figure 100: Select Instance

If you are unsure of which TiDB or TiKV instance to observe, you can select an arbitrary instance. Also, when the cluster CPU load is extremely unbalanced, you can first use Grafana charts to determine the specific instance you want to observe.

3. Observe the charts and tables presented by Top SQL.

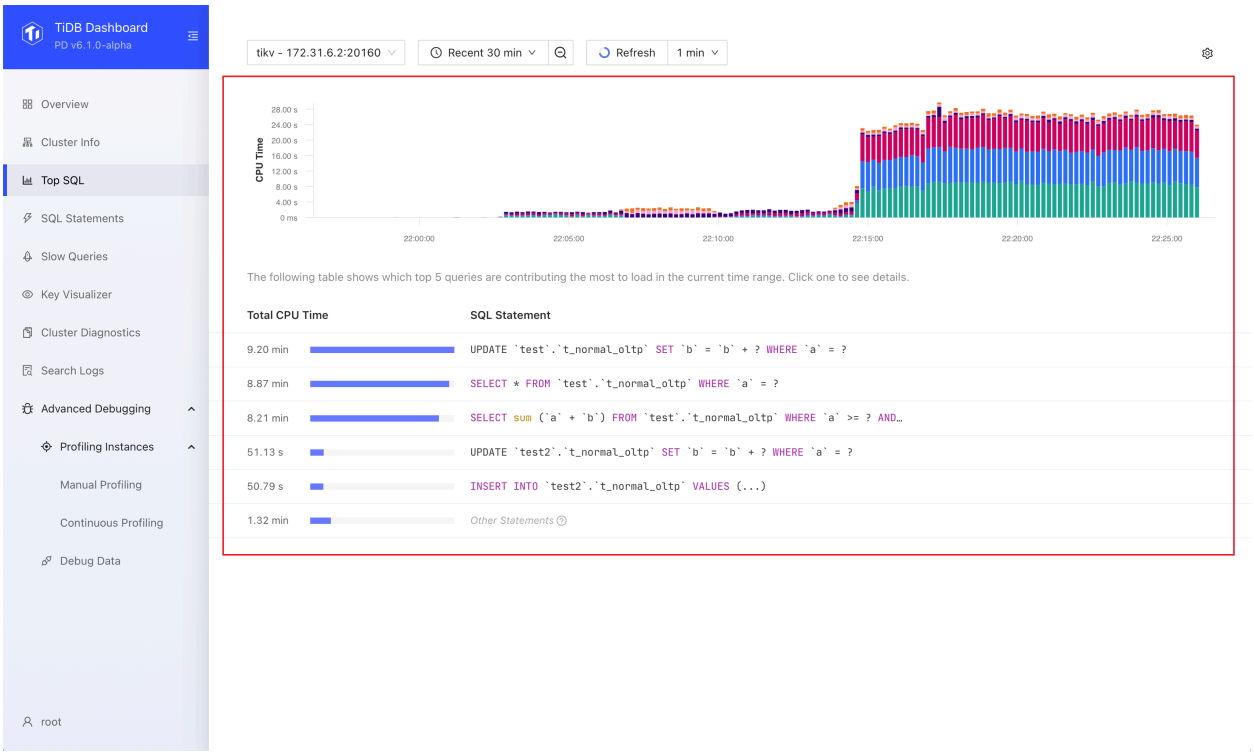


Figure 101: Chart and Table

The size of the bars in the bar chart represents the size of CPU resources consumed by the SQL statement at that moment. Different colors distinguish different types of SQL statements. In most cases, you only need to focus on the SQL statements that have a higher CPU resource overhead in the corresponding time range in the chart.

4. Click a SQL statement in the table to show more information. You can see detailed execution metrics of different plans of that statement, such as Call/sec (average queries per second) and Scan Indexes/sec (average number of index rows scanned per second).

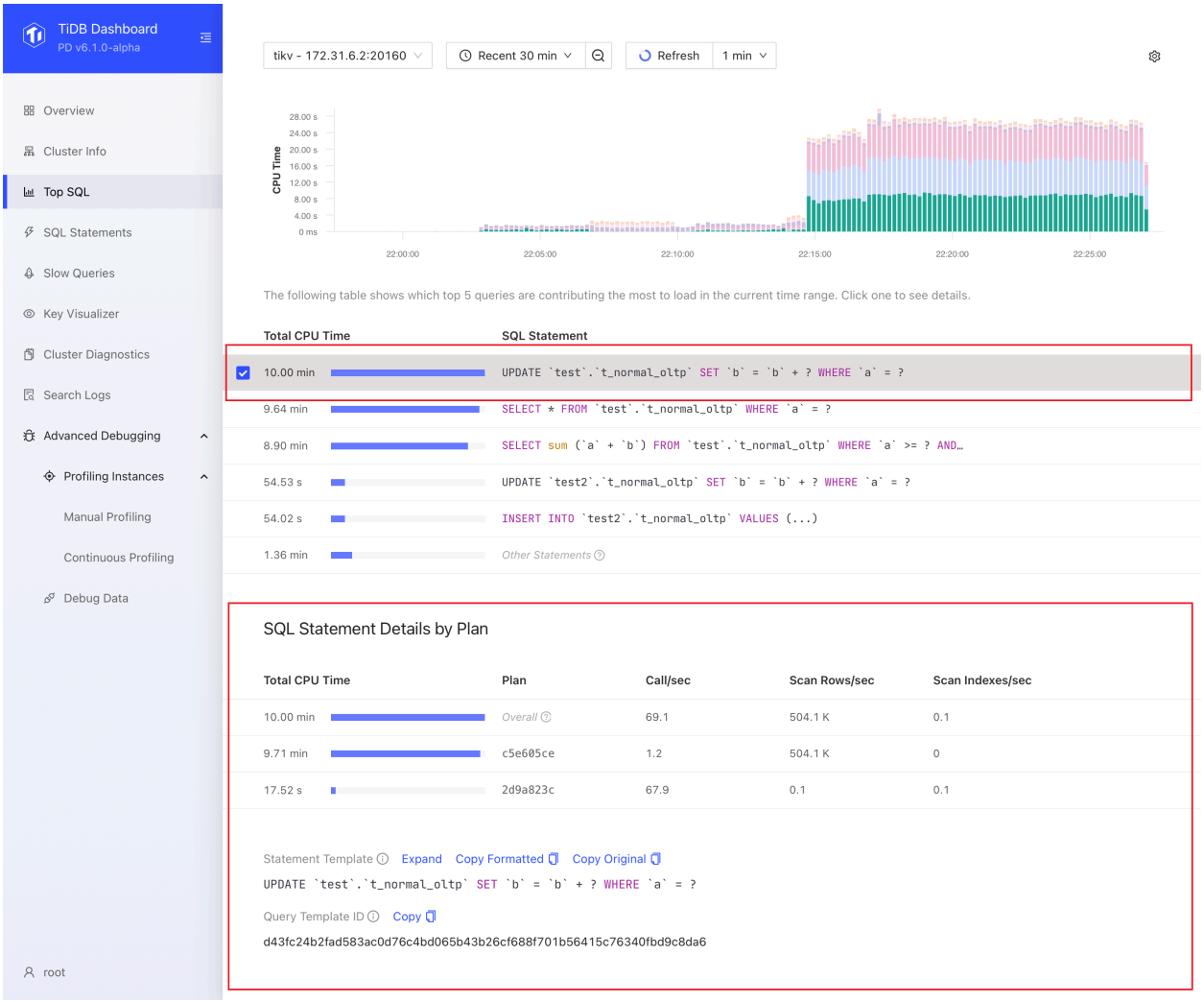


Figure 102: Details

5. Based on these initial clues, you can further explore the SQL Statement or Slow Queries page to find the root cause of high CPU consumption or large data scans of the SQL statement.

Additionally, you can configure Top SQL as follows:

- You can adjust the time range in the time picker or select a time range in the chart to get a more precise and detailed look at the problem. A smaller time range can provide more detailed data, with precision of up to 1 second.



Figure 103: Change time range

- If the chart is out of date, you can click the **Refresh** button or select Auto Refresh options from the **Refresh** drop-down list.

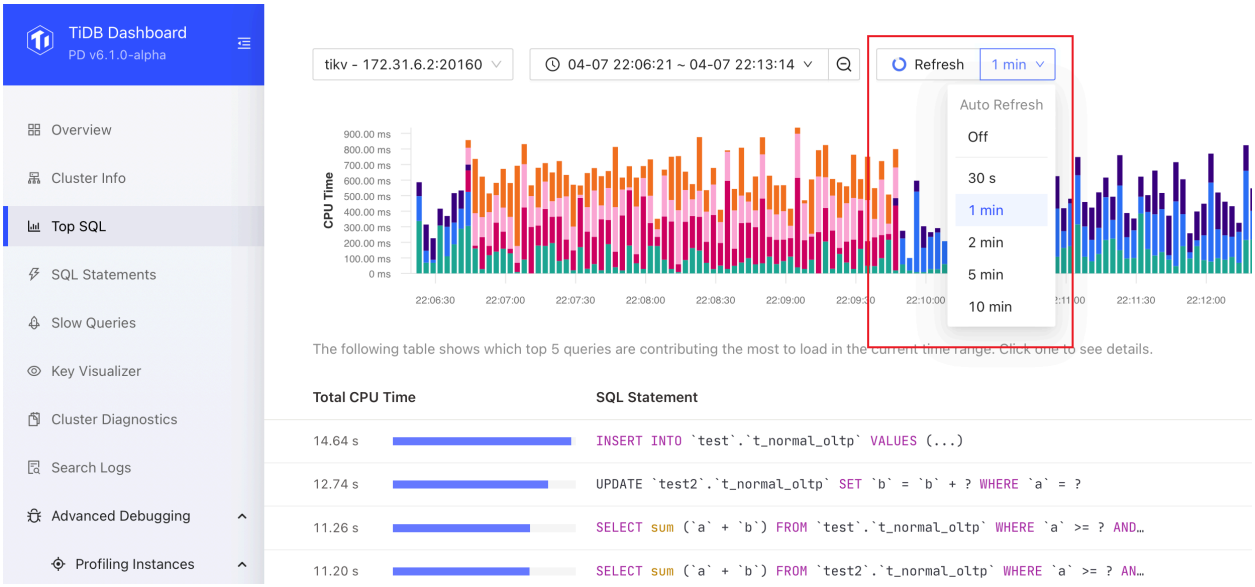


Figure 104: Refresh

10.5.5 Disable Top SQL

You can disable this feature by following these steps:

1. Visit Top SQL page.
2. Click the gear icon in the upper right corner to open the settings screen and switch off **Enable Feature**.
3. Click **Save**.
4. In the popped-up dialog box, click **Disable**.

In addition to the UI, you can also disable the Top SQL feature by setting the TiDB system variable `tidb_enable_top_sql`:

```
SET GLOBAL tidb_enable_top_sql = 0;
```

10.5.6 Frequently asked questions

1. Top SQL cannot be enabled and the UI displays "required component NgMonitoring is not started".

See TiDB Dashboard FAQ.

2. Will performance be affected after enabling Top SQL?

This feature has a slight impact on cluster performance. According to our benchmark, the average performance impact is usually less than 3% when the feature is enabled.

3. What is the status of this feature?

It is now a generally available (GA) feature and can be used in production environments.

4. What is the meaning of "Other Statements"?

"Other Statement" counts the total CPU overhead of all non-Top 5 statements. With this information, you can learn the CPU overhead contributed by the Top 5 statements compared with the overall.

5. What is the relationship between the CPU overhead displayed by Top SQL and the actual CPU usage of the process?

Their correlation is strong but they are not exactly the same thing. For example, the cost of writing multiple replicas is not counted in the TiKV CPU overhead displayed by Top SQL. In general, SQL statements with higher CPU usage result in higher CPU overhead displayed in Top SQL.

6. What is the meaning of the Y-axis of the Top SQL chart?

It represents the size of CPU resources consumed. The more resources consumed by a SQL statement, the higher the value is. In most cases, you do not need to care about the meaning or unit of the specific value.

7. Does Top SQL collect running (unfinished) SQL statements?

Yes. The bars displayed in the Top SQL chart at each moment indicate the CPU overhead of all running SQL statements at that moment.

10.6 Identify Expensive Queries

TiDB allows you to identify expensive queries during SQL execution, so you can diagnose and improve the performance of SQL execution. Specifically, TiDB prints the information about statements whose execution time exceeds `tidb_expensive_query_time_threshold` (60 seconds by default) or memory usage exceeds `tidb_mem_quota_query` (1 GB by default) to the tidb-server log file ("tidb.log" by default).

> **Note:**
>
> The expensive query log differs from the slow query log in this way: TiDB prints statement information to the expensive query log **as soon as** the statement exceeds the threshold of resource usage (execution time or memory usage); while TiDB prints statement information to the slow query log **after** the statement execution.

10.6.1 Expensive query log example

```
[2020/02/05 15:32:25.096 +08:00] [WARN] [expensivequery.go:167] [
   ↪ expensive_query] [cost_time=60.008338935s] [wait_time=0s] [
   ↪ request_count=1] [total_keys=70] [process_keys=65] [num_cop_tasks=1]
   ↪ [process_avg_time=0s] [process_p90_time=0s] [process_max_time=0s] [
   ↪ process_max_addr=10.0.1.9:20160] [wait_avg_time=0.002s] [
   ↪ wait_p90_time=0.002s] [wait_max_time=0.002s] [wait_max_addr
   ↪ =10.0.1.9:20160] [stats=t:pseudo] [conn_id=60026] [user=root] [
   ↪ database=test] [table_ids="[122]"] [txn_start_ts=414420273735139329]
   ↪ [mem_max="1035 Bytes (1.0107421875 KB)"] [sql="insert into t select
   ↪ sleep(1) from t"]
```

10.6.2 Fields description

Basic fields:

- `cost_time`: The execution time of a statement when the log is printed.
- `stats`: The version of statistics used by the tables or indexes involved in a statement. If the value is `pseudo`, it means that there are no available statistics. In this case, you need to analyze the tables or indexes.

- `table_ids`: The IDs of the tables involved in a statement.
- `txn_start_ts`: The start timestamp and the unique ID of a transaction. You can use this value to search for the transaction-related logs.
- `sql`: The sql statement.

Memory usage related fields:

- `mem_max`: Memory usage of a statement when the log is printed. This field has two kinds of units to measure memory usage: byte and other readable and adaptable units (such as MB and GB).

User related fields:

- `user`: The name of the user who executes the statement.
- `conn_id`: The connection ID (session ID). For example, you can use the keyword `con:60026` to search for the log whose session ID is `60026`.
- `database`: The database where the statement is executed.

TiKV Coprocessor task related fields:

- `wait_time`: The total waiting time of all Coprocessor requests of a statement in TiKV. Because the Coprocessor of TiKV runs a limited number of threads, requests might queue up when all threads of Coprocessor are working. When a request in the queue takes a long time to process, the waiting time of the subsequent requests increases.
- `request_count`: The number of Coprocessor requests that a statement sends.
- `total_keys`: The number of keys that Coprocessor has scanned.
- `processed_keys`: The number of keys that Coprocessor has processed. Compared with `total_keys`, `processed_keys` does not include the old versions of MVCC. A great difference between `processed_keys` and `total_keys` indicates that many old versions exist.
- `num_cop_tasks`: The number of Coprocessor requests that a statement sends.
- `process_avg_time`: The average execution time of Coprocessor tasks.
- `process_p90_time`: The P90 execution time of Coprocessor tasks.
- `process_max_time`: The maximum execution time of Coprocessor tasks.
- `process_max_addr`: The address of the Coprocessor task with the longest execution time.
- `wait_avg_time`: The average waiting time of Coprocessor tasks.
- `wait_p90_time`: The P90 waiting time of Coprocessor tasks.
- `wait_max_time`: The maximum waiting time of Coprocessor tasks.
- `wait_max_addr`: The address of the Coprocessor task with the longest waiting time.

10.7 Statement Summary Tables

To better handle SQL performance issues, MySQL has provided statement summary tables in `performance_schema` to monitor SQL with statistics. Among these tables, `events_statements_summary_by_digest` is very useful in locating SQL problems with its abundant fields such as latency, execution times, rows scanned, and full table scans.

Therefore, starting from v4.0.0-rc.1, TiDB provides system tables in `information_schema` (*not* `performance_schema`) that are similar to `events_statements_summary_by_digest` in terms of features.

- `statements_summary`
- `statements_summary_history`
- `cluster_statements_summary`
- `cluster_statements_summary_history`
- `statements_summary_evicted`

This document details these tables and introduces how to use them to troubleshoot SQL performance issues.

10.7.1 `statements_summary`

`statements_summary` is a system table in `information_schema`. `statements_summary` groups the SQL statements by the SQL digest and the plan digest, and provides statistics for each SQL category.

The "SQL digest" here means the same as used in slow logs, which is a unique identifier calculated through normalized SQL statements. The normalization process ignores constant, blank characters, and is case insensitive. Therefore, statements with consistent syntaxes have the same digest. For example:

```
SELECT * FROM employee WHERE id IN (1, 2, 3) AND salary BETWEEN 1000 AND
    ↪ 2000;
select * from EMPLOYEE where ID in (4, 5) and SALARY between 3000 and 4000;
```

After normalization, they are both of the following category:

```
select * from employee where id in (...) and salary between ? and ?;
```

The "plan digest" here refers to the unique identifier calculated through normalized execution plan. The normalization process ignores constants. The same SQL statements might be grouped into different categories because the same statements might have different execution plans. SQL statements of the same category have the same execution plan.

`statements_summary` stores the aggregated results of SQL monitoring metrics. In general, each of the monitoring metrics includes the maximum value and average value. For

example, the execution latency metric corresponds to two fields: `AVG_LATENCY` (average latency) and `MAX_LATENCY` (maximum latency).

To make sure that the monitoring metrics are up to date, data in the `statements_summary` table is periodically cleared, and only recent aggregated results are retained and displayed. The periodical data clearing is controlled by the `tidb_stmt_summary_refresh_interval` system variable. If you happen to make a query right after the clearing, the data displayed might be very little.

The following is a sample output of querying `statements_summary`:

```
  SUMMARY_BEGIN_TIME: 2020-01-02 11:00:00
    SUMMARY_END_TIME: 2020-01-02 11:30:00
          STMT_TYPE: Select
        SCHEMA_NAME: test
             DIGEST: 0611
                 ↪ cc2fe792f8c146cc97d39b31d9562014cf15f8d41f23a4938ca341f54182
                 ↪
        DIGEST_TEXT: select * from employee where id = ?
        TABLE_NAMES: test.employee
        INDEX_NAMES: NULL
        SAMPLE_USER: root
         EXEC_COUNT: 3
        SUM_LATENCY: 1035161
        MAX_LATENCY: 399594
        MIN_LATENCY: 301353
        AVG_LATENCY: 345053
    AVG_PARSE_LATENCY: 57000
    MAX_PARSE_LATENCY: 57000
  AVG_COMPILE_LATENCY: 175458
  MAX_COMPILE_LATENCY: 175458
  ............
             AVG_MEM: 103
             MAX_MEM: 103
             AVG_DISK: 65535
             MAX_DISK: 65535
    AVG_AFFECTED_ROWS: 0
          FIRST_SEEN: 2020-01-02 11:12:54
           LAST_SEEN: 2020-01-02 11:25:24
    QUERY_SAMPLE_TEXT: select * from employee where id=3100
     PREV_SAMPLE_TEXT:
          PLAN_DIGEST:
             ↪ f415b8d52640b535b9b12a9c148a8630d2c6d59e419aad29397842e32e8e5de3
             ↪
                PLAN: Point_Get_1    root    1       table:employee, handle:3100
```

Note:

In TiDB, the time unit of fields in statement summary tables is nanosecond (ns), whereas in MySQL the time unit is picosecond (ps).

10.7.2 `statements_summary_history`

The table schema of `statements_summary_history` is identical to that of `statements_summary` ↪ . `statements_summary_history` saves the historical data of a time range. By checking historical data, you can troubleshoot anomalies and compare monitoring metrics of different time ranges.

The fields `SUMMARY_BEGIN_TIME` and `SUMMARY_END_TIME` represent the start time and the end time of the historical time range.

10.7.3 `statements_summary_evicted`

The `tidb_stmt_summary_max_stmt_count` variable controls the maximum number of statements that the `statement_summary` table stores in memory. The `statement_summary` ↪ table uses the LRU algorithm. Once the number of SQL statements exceeds the `tidb_stmt_summary_max_stmt_count` value, the longest unused record is evicted from the table. The number of evicted SQL statements during each period is recorded in the `statements_summary_evicted` table.

The `statements_summary_evicted` table is updated only when a SQL record is evicted from the `statement_summary` table. The `statements_summary_evicted` only records the period during which the eviction occurs and the number of evicted SQL statements.

10.7.4 The `cluster` tables for statement summary

The `statements_summary`, `statements_summary_history`, and `statements_summary_evicted` ↪ tables only show the statement summary of a single TiDB server. To query the data of the entire cluster, you need to query the `cluster_statements_summary` ↪ , `cluster_statements_summary_history`, or `cluster_statements_summary_evicted` tables.

`cluster_statements_summary` displays the `statements_summary` data of each TiDB server. `cluster_statements_summary_history` displays the `statements_summary_history` ↪ data of each TiDB server. `cluster_statements_summary_evicted` displays the `statements_summary_evicted` data of each TiDB server. These tables use the `INSTANCE` field to represent the address of the TiDB server. The other fields are the same as those in `statements_summary`, `statements_summary_history`, and `statements_summary_evicted` ↪ .

10.7.5 Parameter configuration

The following system variables are used to control the statement summary:

- `tidb_enable_stmt_summary`: Determines whether to enable the statement summary feature. `1` represents `enable`, and `0` means `disable`. The feature is enabled by default. The statistics in the system table are cleared if this feature is disabled. The statistics are re-calculated next time this feature is enabled. Tests have shown that enabling this feature has little impact on performance.
- `tidb_stmt_summary_refresh_interval`: The interval at which the `statements_summary` ↪ table is refreshed. The time unit is second (s). The default value is `1800`.
- `tidb_stmt_summary_history_size`: The size of each SQL statement category stored in the `statements_summary_history` table, which is also the maximum number of records in the `statement_summary_evicted` table. The default value is `24`.
- `tidb_stmt_summary_max_stmt_count`: Limits the number of SQL statements that can be stored in statement summary tables. The default value is `3000`. If the limit is exceeded, those SQL statements that recently remain unused are cleared. These cleared SQL statements are recorded in the `statement_summary_evicted` table.
- `tidb_stmt_summary_max_sql_length`: Specifies the longest display length of `DIGEST_TEXT` and `QUERY_SAMPLE_TEXT`. The default value is `4096`.
- `tidb_stmt_summary_internal_query`: Determines whether to count the TiDB SQL statements. `1` means to count, and `0` means not to count. The default value is `0`.

> **Note:**
>
> When a category of SQL statement needs to be removed because the `tidb_stmt_summary_max_stmt_count` limit is exceeded, TiDB removes the data of that SQL statement category of all time ranges from the `statement_summary_history` table. Therefore, even if the number of SQL statement categories in a certain time range does not reach the limit, the number of SQL statements stored in the `statement_summary_history` table is less than the actual number of SQL statements. If this situation occurs and affects performance, you are recommended to increase the value of `tidb_stmt_summary_max_stmt_count`.

An example of the statement summary configuration is shown as follows:

```
set global tidb_enable_stmt_summary = true;
set global tidb_stmt_summary_refresh_interval = 1800;
set global tidb_stmt_summary_history_size = 24;
```

After the configuration above takes effect, every 30 minutes the `statements_summary` table is cleared. The `statements_summary_history` table stores data generated over the recent 12 hours.

The `statements_summary_evicted` table records the recent 24 periods during which SQL statements are evicted from the statement summary. The `statements_summary_evicted` table is updated every 30 minutes.

> **Note:**
>
> The `tidb_stmt_summary_history_size`, `tidb_stmt_summary_max_stmt_count`, and `tidb_stmt_summary_max_sql_length` configuration items affect memory usage. It is recommended that you adjust these configurations based on your needs, the SQL size, SQL count, and machine configuration. It is not recommended to set them too large values. You can calculate the memory usage using `tidb_stmt_summary_history_size * tidb_stmt_summary_max_stmt_count * tidb_stmt_summary_max_sql_length * 3`.

10.7.5.1 Set a proper size for statement summary

After the system has run for a period of time (depending on the system load), you can check the `statement_summary` table to see whether SQL eviction has occurred. For example:

```
select @@global.tidb_stmt_summary_max_stmt_count;
select count(*) from information_schema.statements_summary;
```

```
+-----------------------------------------+
| @@global.tidb_stmt_summary_max_stmt_count |
+-----------------------------------------+
| 3000                                    |
+-----------------------------------------+
1 row in set (0.001 sec)

+----------+
| count(*) |
+----------+
|     3001 |
+----------+
1 row in set (0.001 sec)
```

You can see that the `statements_summary` table is full of records. Then check the evicted data from the `statements_summary_evicted` table:

```
select * from information_schema.statements_summary_evicted;
```

```
+--------------------+--------------------+---------------+
| BEGIN_TIME         | END_TIME           | EVICTED_COUNT |
+--------------------+--------------------+---------------+
| 2020-01-02 16:30:00 | 2020-01-02 17:00:00 |     59 |
+--------------------+--------------------+---------------+
| 2020-01-02 16:00:00 | 2020-01-02 16:30:00 |     45 |
+--------------------+--------------------+---------------+
2 row in set (0.001 sec)
```

From the preceding result, you can see that a maximum of 59 SQL categories are evicted. In this case, it is recommended that you increase the size of the `statement_summary` table by at least 59 records, which means increasing the size to at least 3059 records.

10.7.6 Limitation

The statement summary tables have the following limitation:

All data of the statement summary tables above will be lost when the TiDB server is restarted. This is because statement summary tables are all memory tables, and the data is cached in memory instead of being persisted on storage.

10.7.7 Troubleshooting examples

This section provides two examples to show how to use the statement summary feature to troubleshoot SQL performance issues.

10.7.7.1 Could high SQL latency be caused by the server end?

In this example, the client shows slow performance with point queries on the `employee` table. You can perform a fuzzy search on SQL texts:

```
SELECT avg_latency, exec_count, query_sample_text
    FROM information_schema.statements_summary
    WHERE digest_text LIKE 'select * from employee%';
```

`1ms` and `0.3ms` are considered within the normal range of `avg_latency`. Therefore, it can be concluded that the server end is not the cause. You can troubleshoot with the client or the network.

```
+-------------+-----------+-----------------------------------------+
| avg_latency | exec_count | query_sample_text                      |
+-------------+-----------+-----------------------------------------+
|     1042040 |         2 | select * from employee where name='eric' |
```

```
|     345053 |         3 | select * from employee where id=3100 |
+------------+-----------+---------------------------------------+
2 rows in set (0.00 sec)
```

10.7.7.2 Which categories of SQL statements consume the longest total time?

If the QPS decrease significantly from 10:00 to 10:30, you can find out the three categories of SQL statements with the longest time consumption from the history table:

```
SELECT sum_latency, avg_latency, exec_count, query_sample_text
    FROM information_schema.statements_summary_history
    WHERE summary_begin_time='2020-01-02 10:00:00'
    ORDER BY sum_latency DESC LIMIT 3;
```

The result shows that the following three categories of SQL statements consume the longest time in total, which need to be optimized with high priority.

```
+--
    ↪ -----------+-------------+-----------+-----------------------------------
    ↪
| sum_latency | avg_latency | exec_count | query_sample_text
    ↪                                        |
+--
    ↪ -----------+-------------+-----------+-----------------------------------
    ↪
|    7855660 |    1122237 |         7 | select avg(salary) from employee
    ↪ where company_id=2013       |
|    7241960 |    1448392 |         5 | select * from employee join company
    ↪ on employee.company_id=company.id |
|    2084081 |    1042040 |         2 | select * from employee where name='
    ↪ eric'                     |
+--
    ↪ -----------+-------------+-----------+-----------------------------------
    ↪
3 rows in set (0.00 sec)
```

10.7.8 Fields description

10.7.8.1 `statements_summary` fields description

The following are descriptions of fields in the `statements_summary` table.

Basic fields:

- `STMT_TYPE`: SQL statement type.

- `SCHEMA_NAME`: The current schema in which SQL statements of this category are executed.
- `DIGEST`: The digest of SQL statements of this category.
- `DIGEST_TEXT`: The normalized SQL statement.
- `QUERY_SAMPLE_TEXT`: The original SQL statements of the SQL category. Only one original statement is taken.
- `TABLE_NAMES`: All tables involved in SQL statements. If there is more than one table, each is separated by a comma.
- `INDEX_NAMES`: All SQL indexes used in SQL statements. If there is more than one index, each is separated by a comma.
- `SAMPLE_USER`: The users who execute SQL statements of this category. Only one user is taken.
- `PLAN_DIGEST`: The digest of the execution plan.
- `PLAN`: The original execution plan. If there are multiple statements, the plan of only one statement is taken.
- `PLAN_CACHE_HITS`: The total number of times that SQL statements of this category hit the plan cache.
- `PLAN_IN_CACHE`: Indicates whether the previous execution of SQL statements of this category hit the plan cache.

Fields related to execution time:

- `SUMMARY_BEGIN_TIME`: The beginning time of the current summary period.
- `SUMMARY_END_TIME`: The ending time of the current summary period.
- `FIRST_SEEN`: The time when SQL statements of this category are seen for the first time.
- `LAST_SEEN`: The time when SQL statements of this category are seen for the last time.

Fields related to TiDB server:

- `EXEC_COUNT`: Total execution times of SQL statements of this category.
- `SUM_ERRORS`: The sum of errors occurred during execution.
- `SUM_WARNINGS`: The sum of warnings occurred during execution.
- `SUM_LATENCY`: The total execution latency of SQL statements of this category.
- `MAX_LATENCY`: The maximum execution latency of SQL statements of this category.
- `MIN_LATENCY`: The minimum execution latency of SQL statements of this category.
- `AVG_LATENCY`: The average execution latency of SQL statements of this category.
- `AVG_PARSE_LATENCY`: The average latency of the parser.
- `MAX_PARSE_LATENCY`: The maximum latency of the parser.
- `AVG_COMPILE_LATENCY`: The average latency of the compiler.
- `MAX_COMPILE_LATENCY`: The maximum latency of the compiler.
- `AVG_MEM`: The average memory (byte) used.
- `MAX_MEM`: The maximum memory (byte) used.
- `AVG_DISK`: The average disk space (byte) used.

- `MAX_DISK`: The maximum disk space (byte) used.

Fields related to TiKV Coprocessor task:

- `SUM_COP_TASK_NUM`: The total number of Coprocessor requests sent.
- `MAX_COP_PROCESS_TIME`: The maximum execution time of Coprocessor tasks.
- `MAX_COP_PROCESS_ADDRESS`: The address of the Coprocessor task with the maximum execution time.
- `MAX_COP_WAIT_TIME`: The maximum waiting time of Coprocessor tasks.
- `MAX_COP_WAIT_ADDRESS`: The address of the Coprocessor task with the maximum waiting time.
- `AVG_PROCESS_TIME`: The average processing time of SQL statements in TiKV.
- `MAX_PROCESS_TIME`: The maximum processing time of SQL statements in TiKV.
- `AVG_WAIT_TIME`: The average waiting time of SQL statements in TiKV.
- `MAX_WAIT_TIME`: The maximum waiting time of SQL statements in TiKV.
- `AVG_BACKOFF_TIME`: The average waiting time before retry when a SQL statement encounters an error that requires a retry.
- `MAX_BACKOFF_TIME`: The maximum waiting time before retry when a SQL statement encounters an error that requires a retry.
- `AVG_TOTAL_KEYS`: The average number of keys that Coprocessor has scanned.
- `MAX_TOTAL_KEYS`: The maximum number of keys that Coprocessor has scanned.
- `AVG_PROCESSED_KEYS`: The average number of keys that Coprocessor has processed. Compared with `avg_total_keys`, `avg_processed_keys` does not include the old versions of MVCC. A great difference between `avg_total_keys` and `avg_processed_keys` indicates that many old versions exist.
- `MAX_PROCESSED_KEYS`: The maximum number of keys that Coprocessor has processed.

Transaction-related fields:

- `AVG_PREWRITE_TIME`: The average time of the prewrite phase.
- `MAX_PREWRITE_TIME`: The longest time of the prewrite phase.
- `AVG_COMMIT_TIME`: The average time of the commit phase.
- `MAX_COMMIT_TIME`: The longest time of the commit phase.
- `AVG_GET_COMMIT_TS_TIME`: The average time of getting `commit_ts`.
- `MAX_GET_COMMIT_TS_TIME`: The longest time of getting `commit_ts`.
- `AVG_COMMIT_BACKOFF_TIME`: The average waiting time before retry when a SQL statement encounters an error that requires a retry during the commit phase.
- `MAX_COMMIT_BACKOFF_TIME`: The maximum waiting time before retry when a SQL statement encounters an error that requires a retry during the commit phase.
- `AVG_RESOLVE_LOCK_TIME`: The average time for resolving lock conflicts occurred between transactions.
- `MAX_RESOLVE_LOCK_TIME`: The longest time for resolving lock conflicts occurred between transactions.
- `AVG_LOCAL_LATCH_WAIT_TIME`: The average waiting time of the local transaction.

- `MAX_LOCAL_LATCH_WAIT_TIME`: The maximum waiting time of the local transaction.
- `AVG_WRITE_KEYS`: The average count of written keys.
- `MAX_WRITE_KEYS`: The maximum count of written keys.
- `AVG_WRITE_SIZE`: The average amount of written data (in byte).
- `MAX_WRITE_SIZE`: The maximum amount of written data (in byte).
- `AVG_PREWRITE_REGIONS`: The average number of Regions involved in the prewrite phase.
- `MAX_PREWRITE_REGIONS`: The maximum number of Regions during the prewrite phase.
- `AVG_TXN_RETRY`: The average number of transaction retries.
- `MAX_TXN_RETRY`: The maximum number of transaction retries.
- `SUM_BACKOFF_TIMES`: The sum of retries when SQL statements of this category encounter errors that require a retry.
- `BACKOFF_TYPES`: All types of errors that require retries and the number of retries for each type. The format of the field is `type:number`. If there is more than one error type, each is separated by a comma, like `txnLock:2,pdRPC:1`.
- `AVG_AFFECTED_ROWS`: The average number of rows affected.
- `PREV_SAMPLE_TEXT`: When the current SQL statement is `COMMIT`, `PREV_SAMPLE_TEXT` is the previous statement to `COMMIT`. In this case, SQL statements are grouped by the digest and `prev_sample_text`. This means that `COMMIT` statements with different `prev_sample_text` are grouped to different rows. When the current SQL statement is not `COMMIT`, the `PREV_SAMPLE_TEXT` field is an empty string.

10.7.8.2 `statements_summary_evicted` fields description

- `BEGIN_TIME`: Records the starting time.
- `END_TIME`: Records the ending time.
- `EVICTED_COUNT`: The number of SQL categories that are evicted during the record period.

10.8 Troubleshoot Hotspot Issues

This document describes how to locate and resolve the problem of read and write hotspots.

As a distributed database, TiDB has a load balancing mechanism to distribute the application loads as evenly as possible to different computing or storage nodes, to make better use of server resources. However, in certain scenarios, some application loads cannot be well distributed, which can affect the performance and form a single point of high load, also known as a hotspot.

TiDB provides a complete solution to troubleshooting, resolving or avoiding hotspots. By balancing load hotspots, overall performance can be improved, including improving QPS and reducing latency.

10.8.1 Common hotspots

This section describes TiDB encoding rules, table hotspots, and index hotspots.

10.8.1.1 TiDB encoding rules

TiDB assigns a TableID to each table, an IndexID to each index, and a RowID to each row. By default, if the table uses an integer primary key, the value of the primary key is treated as the RowID. Among these IDs, TableID is unique in the entire cluster, while IndexID and RowID are unique in the table. The type of all these IDs is int64.

Each row of data is encoded as a key-value pair according to the following rule:

```
Key: tablePrefix{tableID}_recordPrefixSep{rowID}
Value: [col1, col2, col3, col4]
```

The `tablePrefix` and `recordPrefixSep` of the key are specific string constants, used to distinguish from other data in the KV space.

For Index data, the key-value pair is encoded according to the following rule:

```
Key: tablePrefix{tableID}_indexPrefixSep{indexID}_indexedColumnsValue
Value: rowID
```

Index data has two types: the unique index and the non-unique index.

- For unique indexes, you can follow the coding rules above.
- For non-unique indexes, a unique key cannot be constructed through this encoding, because the `tablePrefix{tableID}_indexPrefixSep{indexID}` of the same index is the same and the `ColumnsValue` of multiple rows might be the same. The encoding rule for non-unique indexes is as follows:

```
Key: tablePrefix{tableID}_indexPrefixSep{indexID}
    ↪ _indexedColumnsValue_rowID
Value: null
```

10.8.1.2 Table hotspots

According to TiDB coding rules, the data of the same table is in a range prefixed by the beginning of the TableID, and the data is arranged in the order of RowID values. When RowID values are incremented during table inserting, the inserted line can only be appended to the end. The Region will split after it reaches a certain size, and then it still can only be appended to the end of the range. The `INSERT` operation can only be executed on one Region, forming a hotspot.

The common auto-increment primary key is sequentially increasing. When the primary key is of the integer type, the value of the primary key is used as the RowID by default. At

this time, the RowID is sequentially increasing, and a write hotspot of the table forms when a large number of `INSERT` operations exist.

Meanwhile, the RowID in TiDB is also sequentially auto-incremental by default. When the primary key is not an integer type, you might also encounter the problem of write hotspots.

In addition, when hotspots occur during the process of data writes (on a newly created table or partition) or data reads (periodic read hotspots in read-only scenarios), you can control the Region merge behavior using table attributes. For details, see Control the Region merge behavior using table attributes.

10.8.1.3 Index hotspots

Index hotspots are similar to table hotspots. Common index hotspots appear in fields that are monotonously increasing in time order, or `INSERT` scenarios with a large number of repeated values.

10.8.2 Identify hotspot issues

Performance problems are not necessarily caused by hotspots and might be caused by multiple factors. Before troubleshooting issues, confirm whether it is related to hotspots.

- To judge write hotspots, open **Hot Write** in the **TiKV-Trouble-Shooting** monitoring panel to check whether the Raftstore CPU metric value of any TiKV node is significantly higher than that of other nodes.
- To judge read hotspots, open **Thread_CPU** in the **TiKV-Details** monitoring panel to check whether the coprocessor CPU metric value of any TiKV node is particularly high.

10.8.2.1 Use TiDB Dashboard to locate hotspot tables

The **Key Visualizer** feature in TiDB Dashboard helps users narrow down hotspot troubleshooting scope to the table level. The following is an example of the thermal diagram shown by **Key Visualizer**. The horizontal axis of the graph is time, and the vertical axis are various tables and indexes. The brighter the color, the greater the load. You can switch the read or write flow in the toolbar.

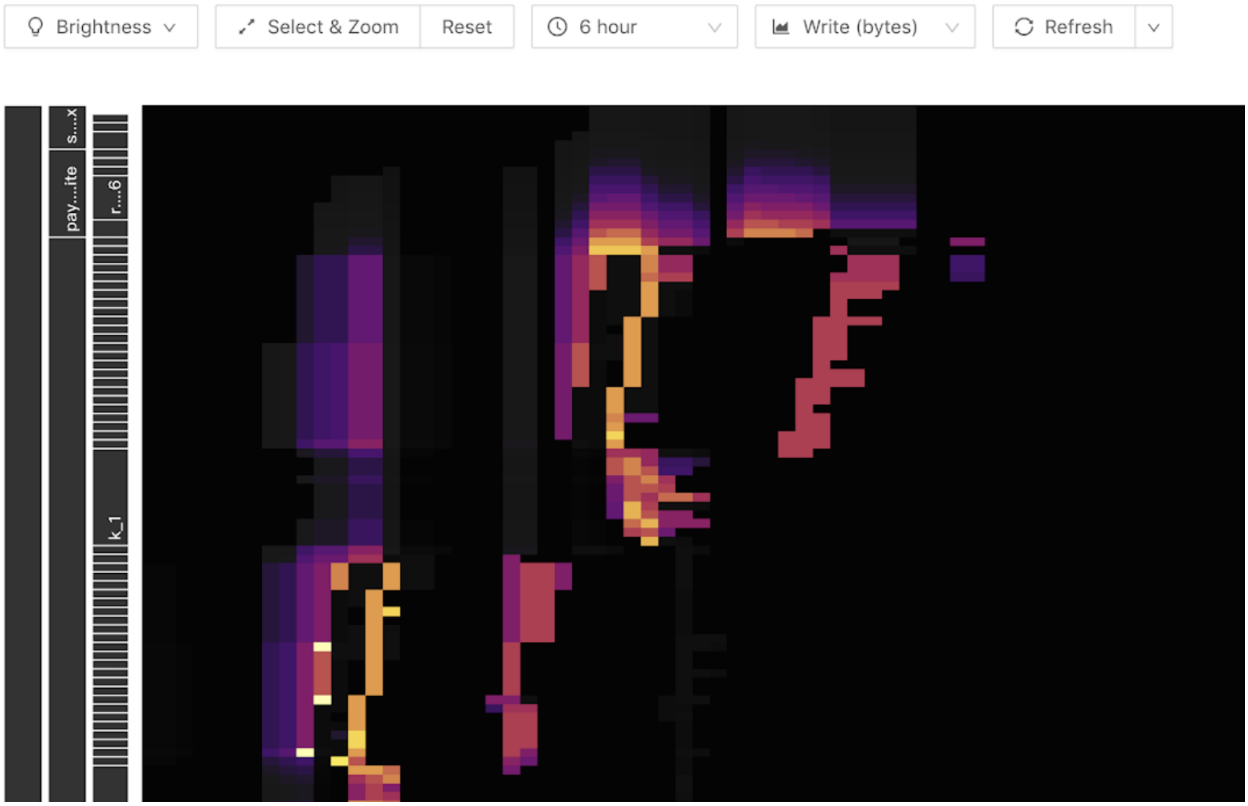


Figure 105: Dashboard Example 1

The following bright diagonal lines (oblique upward or downward) can appear in the write flow graph. Because the write only appears at the end, as the number of table Regions becomes larger, it appears as a ladder. This indicates that a write hotspot shows in this table:

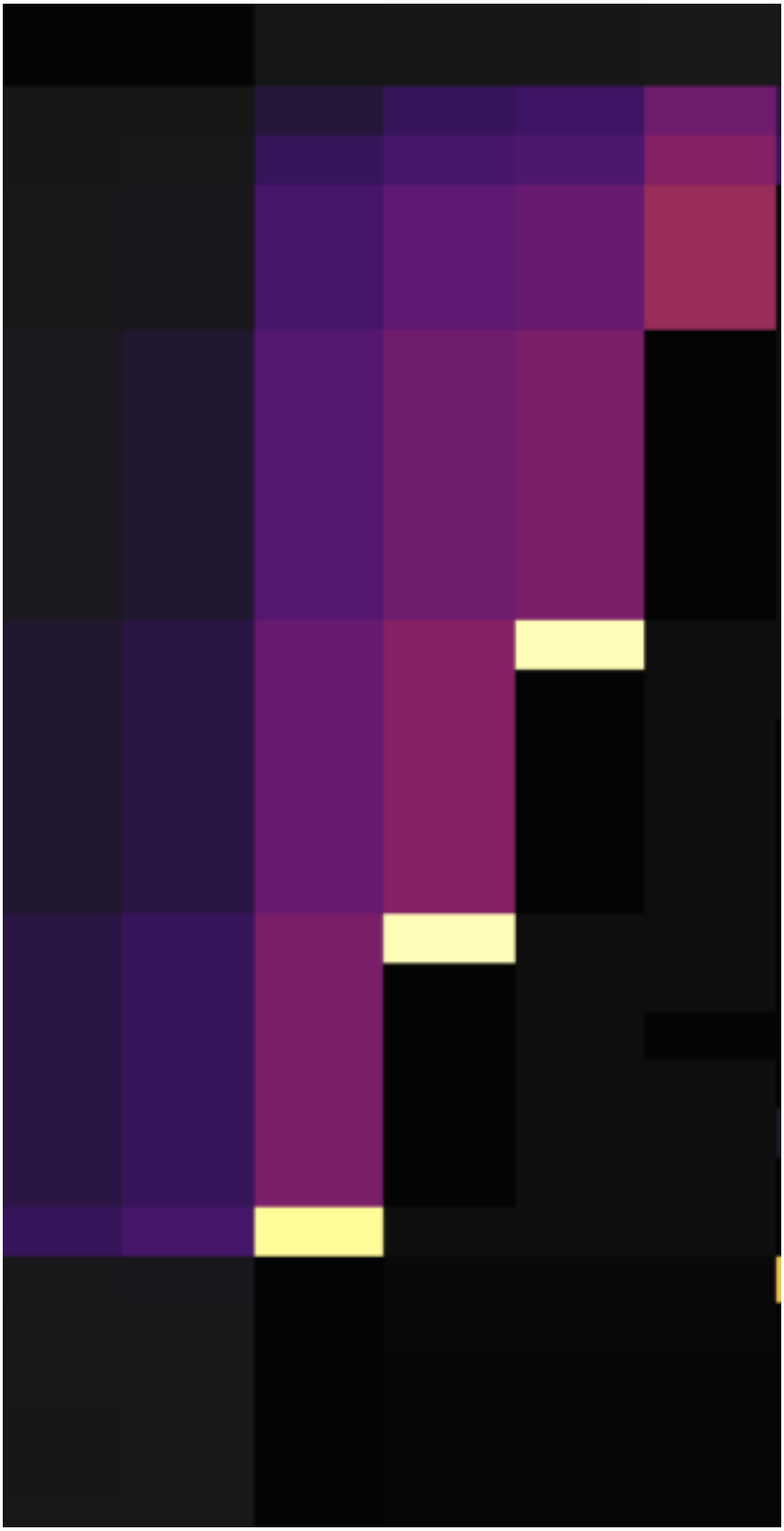


Figure 106: Dashboard Example 2

For read hotspots, a bright horizontal line is generally shown in the thermal diagram. Usually these are caused by small tables with a large number of accesses, shown as follows:

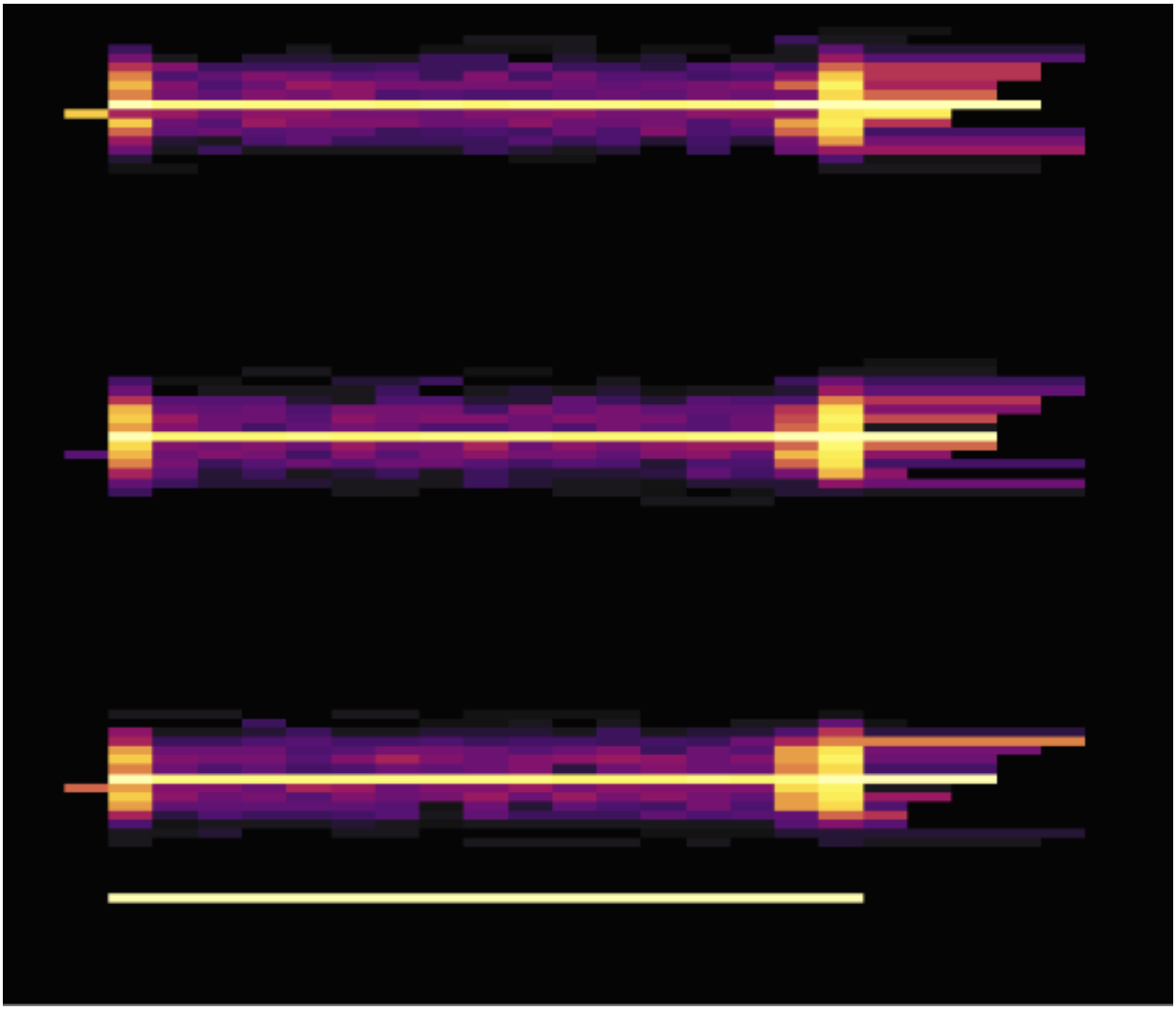


Figure 107: Dashboard Example 3

Hover over the bright block, you can see what table or index has a heavy load. For example:

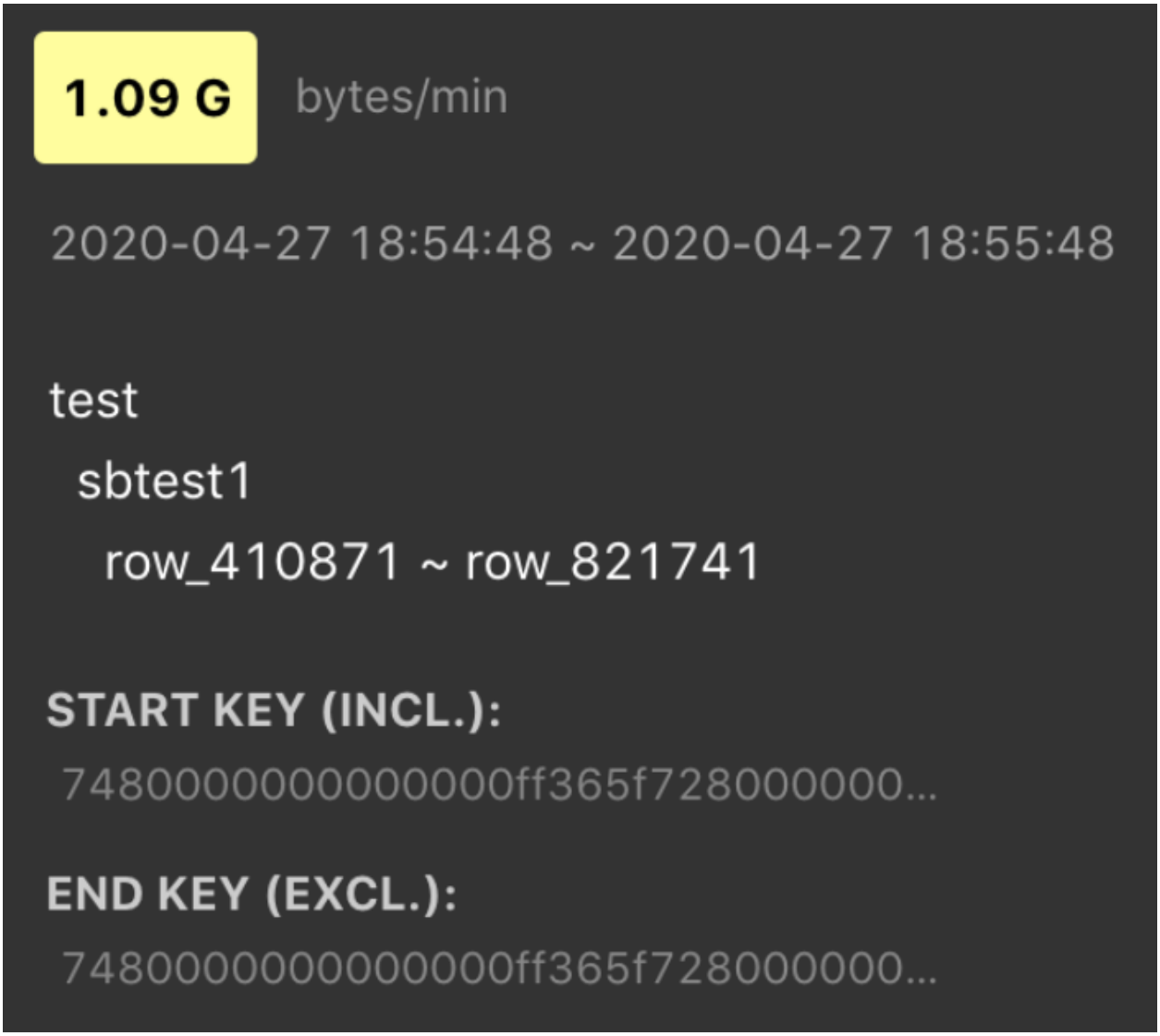


Figure 108: Dashboard Example 4

10.8.3 Use `SHARD_ROW_ID_BITS` to process hotspots

For a non-clustered primary key or a table without a primary key, TiDB uses an implicit auto-increment RowID. When a large number of `INSERT` operations exist, the data is written into a single Region, resulting in a write hotspot.

By setting `SHARD_ROW_ID_BITS`, row IDs are scattered and written into multiple Regions, which can alleviate the write hotspot issue.

```
SHARD_ROW_ID_BITS = 4 # Represents 16 shards.
SHARD_ROW_ID_BITS = 6 # Represents 64 shards.
SHARD_ROW_ID_BITS = 0 # Represents the default 1 shard.
```

Statement example:

```
CREATE TABLE: CREATE TABLE t (c int) SHARD_ROW_ID_BITS = 4;
ALTER TABLE: ALTER TABLE t SHARD_ROW_ID_BITS = 4;
```

The value of `SHARD_ROW_ID_BITS` can be dynamically modified. The modified value only takes effect for newly written data.

For the table with a primary key of the `CLUSTERED` type, TiDB uses the primary key of the table as the RowID. At this time, the `SHARD_ROW_ID_BITS` option cannot be used because it changes the RowID generation rules. For the table with the primary key of the `NONCLUSTERED` type, TiDB uses an automatically allocated 64-bit integer as the RowID. In this case, you can use the `SHARD_ROW_ID_BITS` feature. For more details about the primary key of the `CLUSTERED` type, refer to clustered index.

The following two load diagrams shows the case where two tables without primary keys use `SHARD_ROW_ID_BITS` to scatter hotspots. The first diagram shows the situation before scattering hotspots, while the second one shows the situation after scattering hotspots.

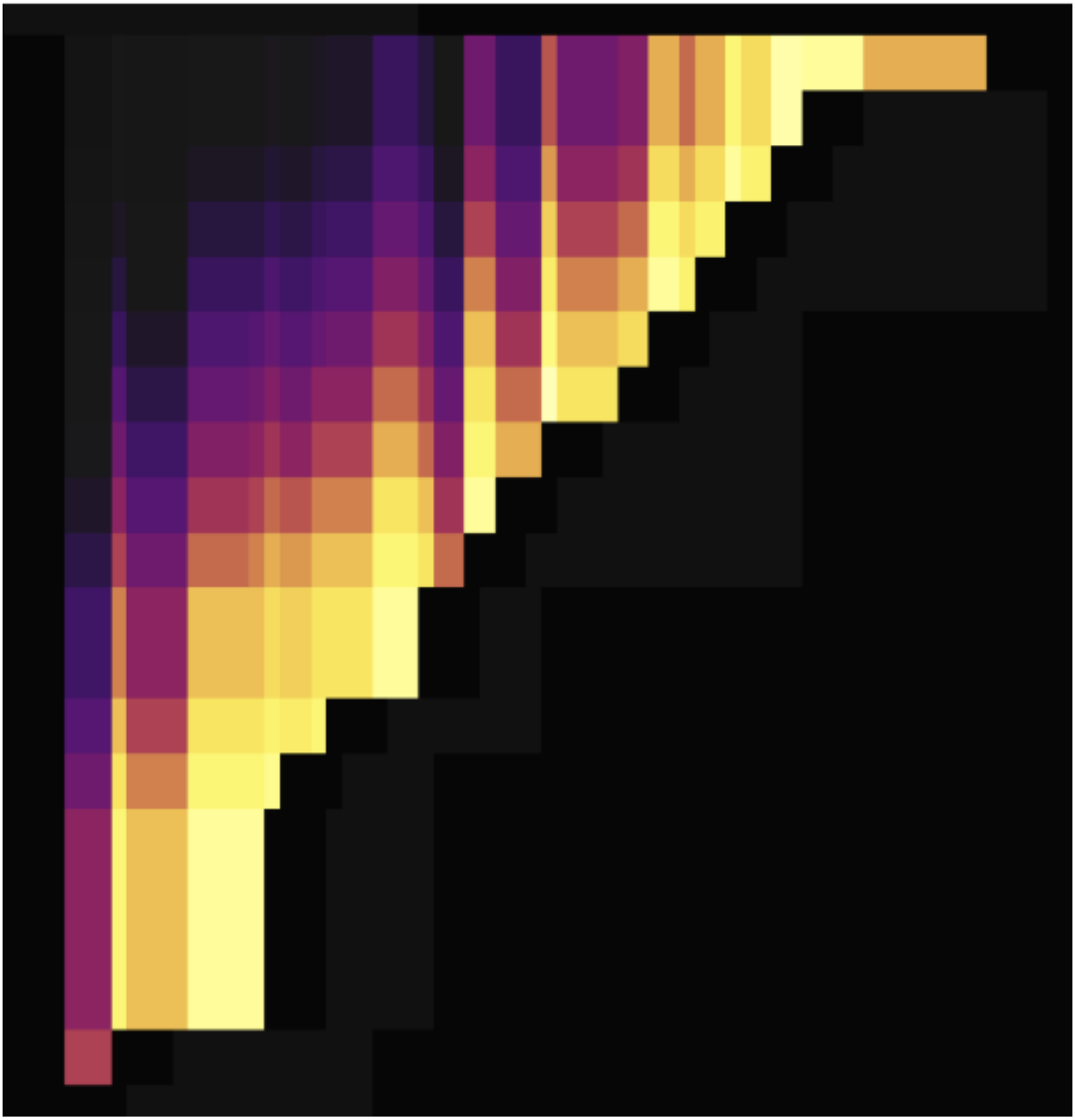


Figure 109: Dashboard Example 5

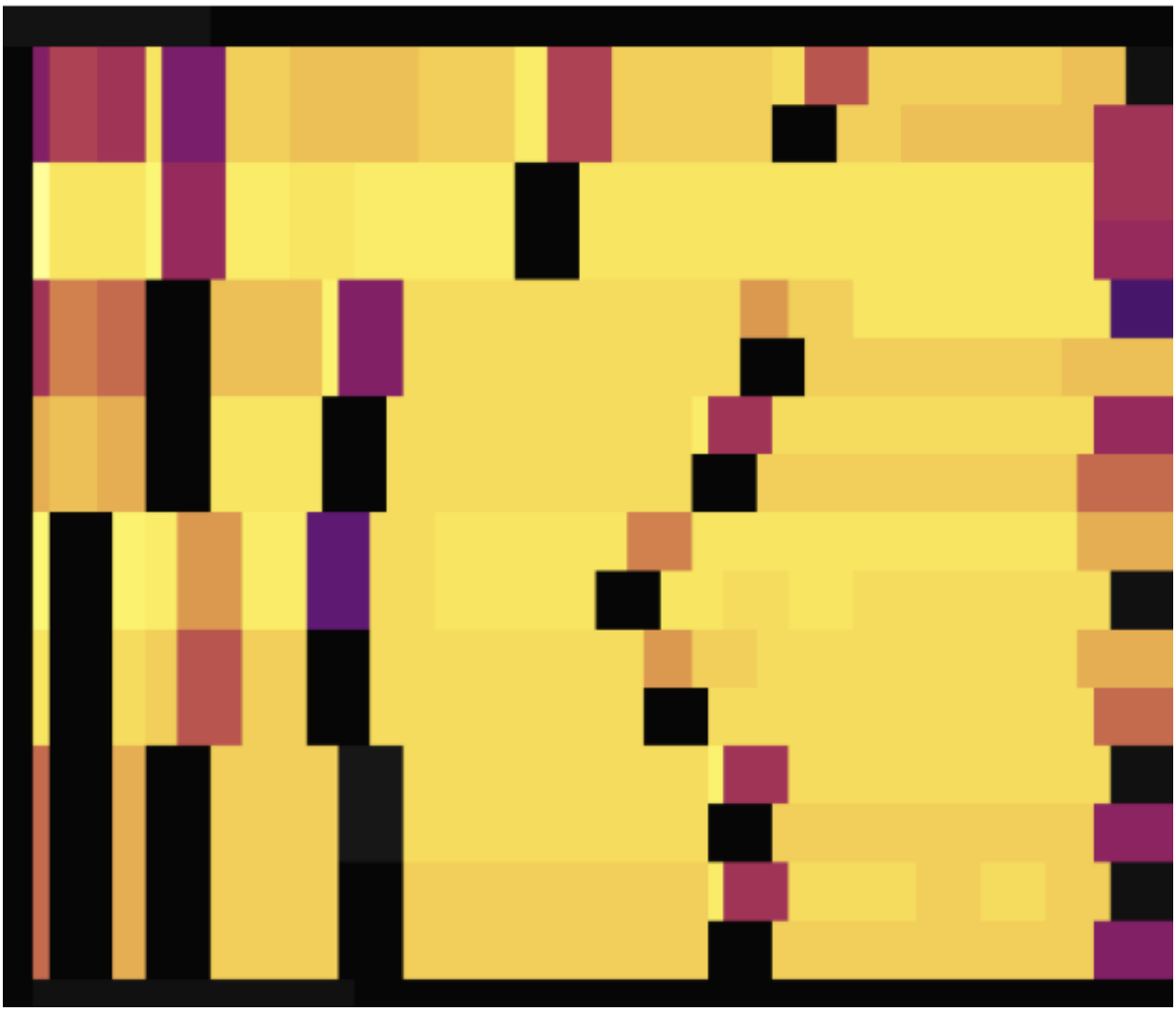


Figure 110: Dashboard Example 6

As shown in the load diagrams above, before setting `SHARD_ROW_ID_BITS`, load hotspots are concentrated on a single Region. After setting `SHARD_ROW_ID_BITS`, load hotspots become scattered.

10.8.4 Handle auto-increment primary key hotspot tables using `AUTO_RANDOM`

To resolve the write hotspots brought by auto-increment primary keys, use `AUTO_RANDOM` to handle hotspot tables that have auto-increment primary keys.

If this feature is enabled, TiDB generates randomly distributed and non-repeated (before the space is used up) primary keys to achieve the purpose of scattering write hotspots.

Note that the primary keys generated by TiDB are no longer auto-increment primary keys and you can use `LAST_INSERT_ID()` to obtain the primary key value assigned last time.

To use this feature, modify `AUTO_INCREMENT` to `AUTO_RANDOM` in the `CREATE TABLE` statement. This feature is suitable for non-application scenarios where the primary keys only need to guarantee uniqueness.

For example:

```
CREATE TABLE t (a BIGINT PRIMARY KEY AUTO_RANDOM, b varchar(255));
INSERT INTO t (b) VALUES ("foo");
SELECT * FROM t;
```

```
+------------+---+
| a          | b |
+------------+---+
| 1073741825 | b |
+------------+---+
```

```
SELECT LAST_INSERT_ID();
```

```
+------------------+
| LAST_INSERT_ID() |
+------------------+
| 1073741825       |
+------------------+
```

The following two load diagrams shows the situations both before and after modifying `AUTO_INCREMENT` to `AUTO_RANDOM` to scatter hotspots. The first one uses `AUTO_INCREMENT`, while the second one uses `AUTO_RANDOM`.

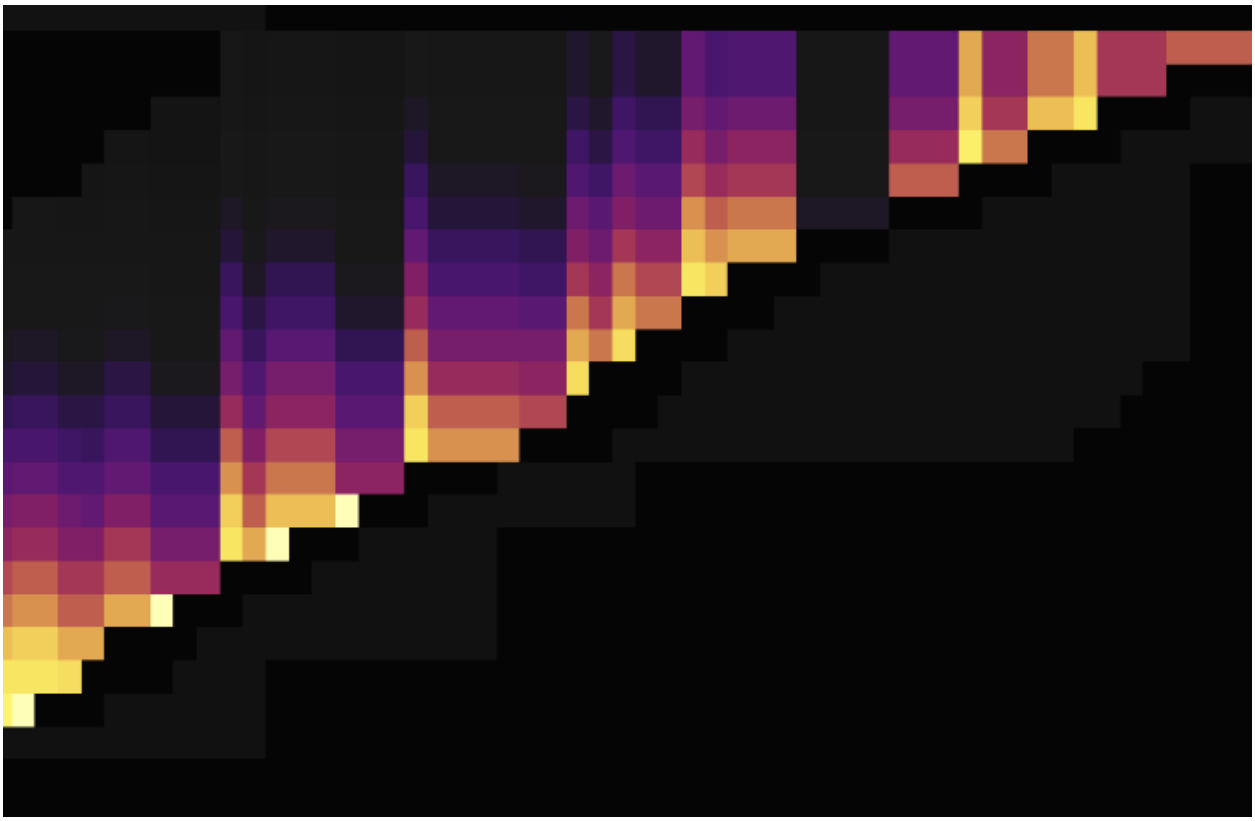


Figure 111: Dashboard Example 7

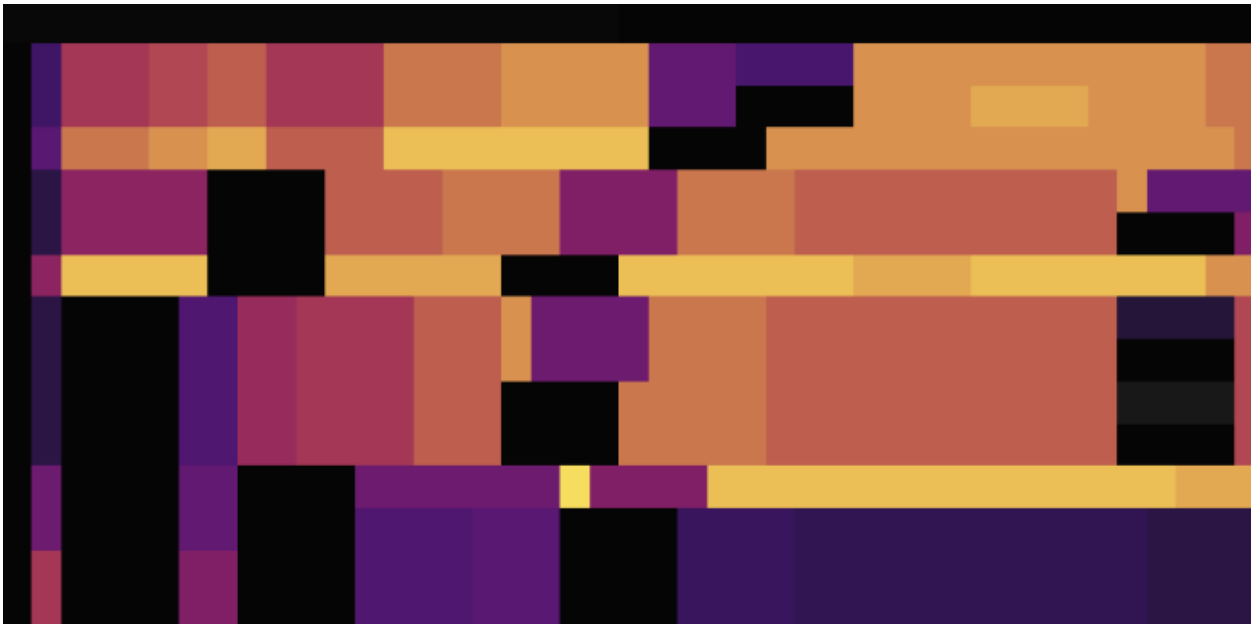


Figure 112: Dashboard Example 8

As shown in the load diagrams above, using `AUTO_RANDOM` to replace `AUTO_INCREMENT` can well scatter hotspots.

For more details, see AUTO_RANDOM.

10.8.5 Optimization of small table hotspots

The Coprocessor Cache feature of TiDB supports pushing down computing result caches. After this feature is enabled, TiDB caches the computing results that will be pushed down to TiKV. This feature works well for read hotspots of small tables.

For more details, see Coprocessor Cache.

See also:

- Highly Concurrent Write Best Practices
- Split Region

10.9 Troubleshoot Increased Read and Write Latency

This document introduces the possible causes of read and write latency and jitters, and how to troubleshoot these issues.

10.9.1 Common causes

10.9.1.1 Incorrect TiDB execution plan

The execution plan of queries is unstable and might select the incorrect index, which causes higher latency.

10.9.1.1.1 Phenomenon

- If the query execution plan is output in the slow log, you can directly view the plan. Execute the `select tidb_decode_plan('xxx...')` statement to parse the detailed execution plan.
- The number of scanned keys in the monitor abnormally increases; in the slow log, the number of `Scan Keys` are large.
- The SQL execution duration in TiDB is greatly different than that in other databases such as MySQL. You can compare the execution plan of other databases (for example, whether `Join Order` is different).

10.9.1.1.2 Possible reason

The statistics is inaccurate.

10.9.1.1.3 Troubleshooting methods

- Update the statistical information
  - Execute `analyze table` manually and execute `analyze` periodically with the `crontab` command to keep the statistics accurate.
  - Execute `auto analyze` automatically. Lower the threshold value of `analyze ratio`, increase the frequency of information collection, and set the start and end time of the execution. See the following examples:
    - `set global tidb_auto_analyze_ratio=0.2;`
    - `set global tidb_auto_analyze_start_time='00:00 +0800';`
    - `set global tidb_auto_analyze_end_time='06:00 +0800';`
- Bind the execution plan
  - Modify the application SQL statements and execute `use index` to consistently use the index of the column.
  - In 3.0 versions, you do not need to modify the application SQL statements. Use `create global binding` to create the binding SQL statement of `force index`.
  - In 4.0 versions, SQL Plan Management is supported, which avoids the performance decrease caused by unstable execution plans.

10.9.1.2 PD anomalies

10.9.1.2.1 Phenomenon

There is an abnormal increase of the `wait duration` metric for the PD TSO. This metric represents the duration of waiting for PD to return requests.

10.9.1.2.2 Possible reasons

- Disk issue. The disk where the PD node is located has full I/O load. Investigate whether PD is deployed with other components with high I/O demand and the health of the disk. You can verify the cause by viewing the monitor metrics in **Grafana -> disk performance -> latency/load**. You can also use the FIO tool to run a check on the disk if necessary.
- Network issues between PD peers. The PD log shows `lost the TCP streaming connection`. You need to check whether there is a problem with the network between PD nodes and verify the cause by viewing `round trip` in the monitor **Grafana -> PD -> etcd**.
- High server load. The log shows `server is likely overloaded`.
- PD cannot elect a Leader: The PD log shows `lease is not expired`. This issue has been fixed in v3.0.x and v2.1.19.

- The leader election is slow. The Region loading duration is long. You can check this issue by running `grep "regions cost"` in the PD log. If the result is in seconds, such as `load 460927 regions cost 11.77099s`, it means the Region loading is slow. You can enable the `region storage` feature in v3.0 by setting `use-region-storage` to `true`, which significantly reduce the Region loading duration.
- The network issue between TiDB and PD. Check whether the network from TiDB to PD Leader is running normally by accessing the monitor **Grafana -> black-box_exporter -> ping latency**.
- PD reports the `FATAL` error, and the log shows `range failed to find revision pair`. This issue has been fixed in v3.0.8 (#2040).
- When the `/api/v1/regions` interface is used, too many Regions might cause PD OOM. This issue has been fixed in v3.0.8 (#1986).
- PD OOM during the rolling upgrade. The size of gRPC messages is not limited, and the monitor shows that `TCP InSegs` is relatively large. This issue has been fixed in v3.0.6 (#1952).
- PD panics. Report a bug.
- Other causes. Get goroutine by running `curl http://127.0.0.1:2379/debug/pprof/goroutine?debug=2` and report a bug.

10.9.1.3 TiKV anomalies

10.9.1.3.1 Phenomenon

The `KV Cmd Duration` metric in the monitor increases abnormally. This metric represents the duration between the time that TiDB sends a request to TiKV and the time that TiDB receives the response.

10.9.1.3.2 Possible reasons

- Check the `gRPC duration` metric. This metric represents the total duration of a gRPC request in TiKV. You can find out the potential network issue by comparing `gRPC duration` of TiKV and `KV duration` of TiDB. For example, the gRPC duration is short but the KV duration of TiDB is long, which indicates that the network latency between TiDB and TiKV might be high, or that the NIC bandwidth between TiDB and TiKV is fully occupied.
- Re-election because TiKV is restarted.
    - After TiKV panics, it is pulled up by `systemd` and runs normally. You can check whether panic has occurred by viewing the TiKV log. Because this issue is unexpected, report a bug if it happens.

  - TiKV is stopped or killed by a third party and then pulled up by `systemd`. Check the cause by viewing `dmesg` and the TiKV log.
  - TiKV is OOM, which causes restart.
  - TiKV is hung because of dynamically adjusting `THP` (Transparent Hugepage).

- Check monitor: TiKV RocksDB encounters write stall and thus results in re-election. You can check if the monitor **Grafana** -> **TiKV-details** -> **errors** shows `server is busy`.

- Re-election because of network isolation.

- If the `block-cache` configuration is too large, it might cause TiKV OOM. To verify the cause of the problem, check the `block cache size` of RocksDB by selecting the corresponding instance in the monitor **Grafana** -> **TiKV-details**. Meanwhile, check whether the `[storage.block-cache] capacity = # "1GB"`parameter is set properly. By default, TiKV's `block-cache` is set to `45%` of the total memory of the machine. You need to explicitly specify this parameter when you deploy TiKV in the container, because TiKV obtains the memory of the physical machine, which might exceed the memory limit of the container.

- Coprocessor receives many large queries and returns a large volume of data. gRPC fails to send data as quickly as the coprocessor returns data, which results in OOM. To verify the cause, you can check whether `response size` exceeds the `network outbound` traffic by viewing the monitor **Grafana** -> **TiKV-details** -> **coprocessor overview**.

10.9.1.4 Bottleneck of a single TiKV thread

There are some single threads in TiKV that might become the bottleneck.

- Too many Regions in a TiKV instance causes a single gRPC thread to be the bottleneck (Check the **Grafana** -> **TiKV-details** -> **Thread CPU/gRPC CPU Per Thread** metric). In v3.x or later versions, you can enable `Hibernate Region` to resolve the issue.
- For versions earlier than v3.0, when the raftstore thread or the apply thread becomes the bottleneck (**Grafana** -> **TiKV-details** -> **Thread CPU/raft store CPU** and **Async apply CPU** metrics exceed `80%`), you can scale out TiKV (v2.x) instances or upgrade to v3.x with multi-threading.

10.9.1.5 CPU load increases

10.9.1.5.1 Phenomenon

The usage of CPU resources becomes the bottleneck.

10.9.1.5.2 Possible reasons

- Hotspot issue
- High overall load. Check the slow queries and expensive queries of TiDB. Optimize the executing queries by adding indexes or executing queries in batches. Another solution is to scale out the cluster.

10.9.2 Other causes

10.9.2.1 Cluster maintenance

Most of each online cluster has three or five nodes. If the machine to be maintained has the PD component, you need to determine whether the node is the leader or the follower. Disabling a follower has no impact on the cluster operation. Before disabling a leader, you need to switch the leadership. During the leadership change, performance jitter of about 3 seconds will occur.

10.9.2.2 Minority of replicas are offline

By default, each TiDB cluster has three replicas, so each Region has three replicas in the cluster. These Regions elect the leader and replicate data through the Raft protocol. The Raft protocol ensures that TiDB can still provide services without data loss even when the nodes (that are fewer than half of replicas) fail or are isolated. For the cluster with three replicas, the failure of one node might cause performance jitter but the usability and correctness in theory are not affected.

10.9.2.3 New indexes

Creating indexes consumes a huge amount of resources when TiDB scans tables and backfills indexes. Index creation might even conflict with the frequently updated fields, which affects the application. Creating indexes on a large table often takes a long time, so you must try to balance the index creation time and the cluster performance (for example, creating indexes at the off-peak time).

Parameter adjustment:

Currently, you can use `tidb_ddl_reorg_worker_cnt` and `tidb_ddl_reorg_batch_size` ↪ to dynamically adjust the speed of index creation. Usually, the smaller the values, the smaller the impact on the system, with longer execution time though.

In general cases, you can first keep their default values (`4` and `256`), observe the resource usage and response speed of the cluster, and then increase the value of `tidb_ddl_reorg_worker_cnt` to increase the concurrency. If no obvious jitter is observed in the monitor, increase the value of `tidb_ddl_reorg_batch_size`. If the columns involved in the index creation are frequently updated, the many resulting conflicts will cause the index creation to fail and be retried.

In addition, you can also set the value of `tidb_ddl_reorg_priority` to `PRIORITY_HIGH` to prioritize the index creation and speed up the process. But in the general OLTP system, it is recommended to keep its default value.

10.9.2.4 High GC pressure

The transaction of TiDB adopts the Multi-Version Concurrency Control (MVCC) mechanism. When the newly written data overwrites the old data, the old data is not replaced, and both versions of data are stored. Timestamps are used to mark different versions. The task of GC is to clear the obsolete data.

- In the phase of Resolve Locks, a large amount of `scan_lock` requests are created in TiKV, which can be observed in the gRPC-related metrics. These `scan_lock` requests call all Regions.
- In the phase of Delete Ranges, a few (or no) `unsafe_destroy_range` requests are sent to TiKV, which can be observed in the gRPC-related metrics and the **GC tasks** panel.
- In the phase of Do GC, each TiKV by default scans the leader Regions on the machine and performs GC to each leader, which can be observed in the **GC tasks** panel.

10.10 Use PLAN REPLAYER to Save and Restore the On-Site Information of a Cluster

When you locate and troubleshoot the issues of a TiDB cluster, you often need to provide information on the system and the execution plan. To help you get the information and troubleshoot cluster issues in a more convenient and efficient way, the `PLAN REPLAYER` ↪ command is introduced in TiDB v5.3.0. This command enables you to easily save and restore the on-site information of a cluster, improves the efficiency of troubleshooting, and helps you more easily archive the issue for management.

The features of `PLAN REPLAYER` are as follows:

- Exports the information of a TiDB cluster at an on-site troubleshooting to a ZIP-formatted file for storage.
- Imports into a cluster the ZIP-formatted file exported from another TiDB cluster. This file contains the information of the latter TiDB cluster at an on-site troubleshooting.

10.10.1 Use `PLAN REPLAYER` to export cluster information

You can use `PLAN REPLAYER` to save the on-site information of a TiDB cluster. The export interface is as follows:

```
PLAN REPLAYER DUMP EXPLAIN [ANALYZE] sql-statement;
```

Based on `sql-statement`, TiDB sorts out and exports the following on-site information:

- TiDB version
- TiDB configuration
- TiDB session variables
- TiDB SQL bindings
- The table schema in `sql-statement`
- The statistics of the table in `sql-statement`
- The result of `EXPLAIN [ANALYZE] sql-statement`

> **Note:**
>
> `PLAN REPLAYER` **DOES NOT** export any table data.

10.10.1.1 Examples of exporting cluster information

```
use test;
create table t(a int, b int);
insert into t values(1,1), (2, 2), (3, 3);
analyze table t;

plan replayer dump explain select * from t;
```

`PLAN REPLAYER DUMP` packages the table information above into a `ZIP` file and returns the file identifier as the execution result. This file is a one-time file. After the file is downloaded, TiDB will delete it.

> **Note:**
>
> The `ZIP` file is stored in a TiDB cluster for at most one hour. After one hour, TiDB will delete it.

```
MySQL [test]> plan replayer dump explain select * from t;
+------------------------------------------------------------------+
| Dump_link                                                        |
+------------------------------------------------------------------+
| replayer_single_JOGvpu4t7dssySqJfTtS4A==_1635750890568691080.zip |
+------------------------------------------------------------------+
1 row in set (0.015 sec)
```

Because the file cannot be downloaded on MySQL Client, you need to use the TiDB HTTP interface and the file identifier to download the file:

```
http://${tidb-server-ip}:${tidb-server-status-port}/plan_replayer/dump/${
    ↪ file_token}
```

`${tidb-server-ip}:${tidb-server-status-port}` is the address of any TiDB server in the cluster. For example:

```
curl http://127.0.0.1:10080/plan_replayer/dump/
    ↪ replayer_single_JOGvpu4t7dssySqJfTtS4A==_1635750890568691080.zip >
    ↪ plan_replayer.zip
```

10.10.2 Use `PLAN REPLAYER` to import cluster information

> **Warning:**
>
> When you import the on-site information of a TiDB cluster to another cluster, the TiDB session variables, SQL bindings, table schemas and statistics of the latter cluster are modified.

With an existing `ZIP` file exported using `PLAN REPLAYER`, you can use the `PLAN REPLAYER` import interface to restore the on-site information of a cluster to any other TiDB cluster. The syntax is as follows:

```
PLAN REPLAYER LOAD 'file_name';
```

In the statement above, `file_name` is the name of the `ZIP` file to be exported.

For example:

```
PLAN REPLAYER LOAD 'plan_replayer.zip';
```

After the cluster information is imported, the TiDB cluster is loaded with the required table schema, statistics and other information that affects the construction of the execution plan. You can view the execution plan and verify statistics in the following way:

```
mysql> desc t;
+-------+---------+------+------+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+------+---------+-------+
| a     | int(11) | YES |     | NULL    |       |
| b     | int(11) | YES |     | NULL    |       |
+-------+---------+------+------+---------+-------+
2 rows in set (0.01 sec)
```

```
mysql> explain select * from t where a = 1 or b =1;
+--
   ↪ ----------------------+---------+----------+--------------+-----------------
   ↪
| id                    | estRows | task    | access object | operator info
   ↪                  |
+--
   ↪ ----------------------+---------+----------+--------------+-----------------
   ↪
| TableReader_7         | 0.01    | root     |               | data:Selection_6
   ↪                 |
| └─Selection_6         | 0.01    | cop[tikv] |              | or(eq(test.t.a,
   ↪ 1), eq(test.t.b, 1)) |
|   └─TableFullScan_5   | 6.00    | cop[tikv] | table:t      | keep order:false,
   ↪  stats:pseudo  |
+--
   ↪ ----------------------+---------+----------+--------------+-----------------
   ↪
3 rows in set (0.00 sec)

mysql> show stats_meta;
+--
   ↪ -------+-----------+---------------+---------------------+--------------+-----
   ↪
| Db_name | Table_name | Partition_name | Update_time | Modify_count |
   ↪ Row_count |
+--
   ↪ -------+-----------+---------------+---------------------+--------------+-----
   ↪
| test   | t          |               | 2022-08-26 15:52:07 |        3 |
   ↪        6 |
+--
   ↪ -------+-----------+---------------+---------------------+--------------+-----
   ↪
1 row in set (0.04 sec)
```

After the scene is loaded and restored, you can diagnose and improve the execution plan for the cluster.

10.11 Troubleshoot TiDB OOM Issues

This document describes how to troubleshoot TiDB OOM (Out of Memory) issues, including phenomena, causes, solutions, and diagnostic information.

10.11.1 Typical OOM phenomena

The following are some typical OOM phenomena:

- The client side reports the following error: `SQL error, errno = 2013, state = '
  ↪ HY000': Lost connection to MySQL server during query`.
- The Grafana dashboard shows:
  - **TiDB** > **Server** > **Memory Usage** shows that the `process/heapInUse` metric keeps rising, and suddenly drops to zero after reaching the threshold.
  - **TiDB** > **Server** > **Uptime** suddenly drops to zero.
  - **TiDB-Runtime** > **Memory Usage** shows that the `estimate-inuse` metric keeps rising.
- Check `tidb.log`, and you can find the following log entries:
  - An alarm about OOM: `[WARN] [memory_usage_alarm.go:139] ["tidb-
    ↪ server has the risk of OOM. Running SQLs and heap profile will
    ↪ be recorded in record path"]`. For more information, see memory-usage
    ↪ -alarm-ratio.
  - A log entry about restart: `[INFO] [printer.go:33] ["Welcome to TiDB."]`.

10.11.2 Overall troubleshooting process

When you troubleshoot OOM issues, follow this process:

1. Confirm whether it is an OOM issue.

   Execute the following command to check the operating system logs. If there is an `oom-killer` log near the time when the problem occurs, you can confirm that it is an OOM issue.

   ```
   dmesg -T | grep tidb-server
   ```

   The following is an example of the log that contains `oom-killer`:

   ```
   ......
   Mar 14 16:55:03 localhost kernel: tidb-server invoked oom-killer:
       ↪ gfp_mask=0x201da, order=0, oom_score_adj=0
   Mar 14 16:55:03 localhost kernel: tidb-server cpuset=/ mems_allowed=0
   Mar 14 16:55:03 localhost kernel: CPU: 14 PID: 21966 Comm: tidb-server
       ↪ Kdump: loaded Not tainted 3.10.0-1160.el7.x86_64 #1
   Mar 14 16:55:03 localhost kernel: Hardware name: QEMU Standard PC (
       ↪ i440FX + PIIX, 1996), BIOS rel-1.14.0-0-g155821a1990b-prebuilt.
       ↪ qemu.org 04/01/2014
   ......
   ```

```
Mar 14 16:55:03 localhost kernel: Out of memory: Kill process 21945 (
    ↪ tidb-server) score 956 or sacrifice child
Mar 14 16:55:03 localhost kernel: Killed process 21945 (tidb-server),
    ↪ UID 1000, total-vm:33027492kB, anon-rss:31303276kB, file-rss:0kB,
    ↪  shmem-rss:0kB
Mar 14 16:55:07 localhost systemd: tidb-4000.service: main process
    ↪ exited, code=killed, status=9/KILL
......
```

2. After confirming that it is an OOM issue, you can further investigate whether the OOM is caused by deployment or the database.

    - If the OOM is caused by a deployment issue, you need to investigate the resource configuration and impact of hybrid deployment.
    - If the OOM is caused by a database issue, the following are some possible causes:
        - TiDB handles large data traffic, such as large queries, large writes, and data import.
        - TiDB is in a high concurrency scenario, where multiple SQL statements consume resources concurrently or operator concurrency is high.
        - TiDB has a memory leak and resources are not released.

    Refer to the following sections for specific troubleshooting methods.

10.11.3 Typical causes and solutions

OOM issues are usually caused by the following:

- Deployment issues
- Database issues
- Client side issues

10.11.3.1 Deployment issues

The following are some causes of OOM due to improper deployment:

- The memory capacity of the operating system is too small.
- The TiUP configuration `resource_control` is not appropriate.
- In the case of hybrid deployments (meaning that TiDB and other applications are deployed on the same server), TiDB is killed accidentally by `oom-killer` due to lack of resources.

10.11.3.2 Database issues

This section describes the causes and solutions for OOM caused by database issues.

> **Note:**
>
> If you have configured `tidb_mem_quota_query`, an error occurs: `ERROR 1105 (HY000): Out Of Memory Quota![conn_id=54]`. It is caused by the memory usage control behavior of the database. It is a normal behavior.

10.11.3.2.1 Executing SQL statements consumes too much memory

You can take the following measures to reduce the memory usage of SQL statements, depending on the different causes of OOM issues.

- If the execution plan of SQL is not optimal, for example, due to lack of proper indexes, outdated statistics, or optimizer bugs, a wrong execution plan of SQL might be selected. A huge intermediate result set will then be accumulated in the memory. In this case, consider the following measures:
  - Add appropriate indexes.
  - Use the disk spill feature for execution operators.
  - Adjust the JOIN order between tables.
  - Use hints to optimize SQL statements.
- Some operators and functions are not supported to be pushed down to the storage level, resulting in a huge accumulation of intermediate result sets. In this case, you need to refine the SQL statements or use hints to optimize, and use the functions or operators that support pushing down.
- The execution plan contains the operator HashAgg. HashAgg is executed concurrently by multiple threads, which is faster but consumes more memory. Instead, you can use `STREAM_AGG()`.
- Reduce the number of Regions to be read simultaneously or reduce the concurrency of operators to avoid memory problems caused by high concurrency. The corresponding system variables include:
  - `tidb_distsql_scan_concurrency`
  - `tidb_index_serial_scan_concurrency`
  - `tidb_executor_concurrency`
- The concurrency of sessions is too high near the time point when the problem occurs. In this case, consider scaling out the TiDB cluster by adding more TiDB nodes.

10.11.3.2.2 Large transactions or large writes consume too much memory

You need to plan for memory capacity. When a transaction is executed, the memory usage of the TiDB process is scaled up comparing with the transaction size, up to two to three times or more of the transaction size.

You can split a single large transaction to multiple smaller transactions.

10.11.3.2.3 The process of collecting and loading statistical information consumes too much memory

A TiDB node needs to load statistics into memory after it starts. TiDB consumes memory when collecting statistical information. You can control memory usage in the following ways:

- Specify a sampling rate, only collect statistics for specific columns, and reduce `ANALYZE` concurrency.
- Since TiDB v6.1.0, you can use the system variable `tidb_stats_cache_mem_quota` to control the memory usage for statistical information.
- Since TiDB v6.1.0, you can use the system variable `tidb_mem_quota_analyze` to control the maximum memory usage when TiDB updates statistics.

For more information, see Introduction to Statistics.

10.11.3.2.4 Prepared statements are overused

The client side keeps creating prepared statements but does not execute `deallocate prepare stmt`, which causes memory consumption to continue to rise and eventually triggers TiDB OOM. The reason is that the memory occupied by a prepared statement is not released until the session is closed. This is especially important for long-time connection sessions.

To solve the problem, consider the following measures:

- Adjust the session lifecycle.
- Adjust the `wait_timeout` and `max_execution_time` of the connection pool.
- Use the system variable `max_prepared_stmt_count` to control the maximum number of prepared statements in a session.

10.11.3.2.5 `tidb_enable_rate_limit_action` is not configured properly

The system variable `tidb_enable_rate_limit_action` controls memory usage effectively when an SQL statement only reads data. When this variable is enabled and computing operations (such as join or aggregation operations) are required, memory usage might not be under the control of `tidb_mem_quota_query`, which increases the risk of OOM.

It is recommended that you disable this system variable. Since TiDB v6.3.0, this system variable is disabled by default.

10.11.3.3 Client side issues

If OOM occurs on the client side, investigate the following:

- Check the trend and speed on **Grafana TiDB Details** > **Server** > **Client Data Traffic** to see if there is a network blockage.
- Check whether there is an application OOM caused by wrong JDBC configuration parameters. For example, if the `defaultFetchSize` parameter for streaming read is incorrectly configured, it can cause data to be heavily accumulated on the client side.

10.11.4 Diagnostic information to be collected to troubleshoot OOM issues

To locate the root cause of an OOM issue, you need to collect the following information:

- Collect the memory-related configurations of the operating system:
    - TiUP configuration: `resource_control.memory_limit`
    - Operating system configurations:
        - Memory information: `cat /proc/meminfo`
        - Kernel parameters: `vm.overcommit_memory`
    - NUMA information:
        - `numactl --hardware`
        - `numactl --show`
- Collect the version information and the memory-related configurations of the database:
    - TiDB version
    - `tidb_mem_quota_query`
    - `memory-usage-alarm-ratio`
    - `mem-quota-query`
    - `oom-action`
    - `tidb_enable_rate_limit_action`
    - `server-memory-quota`
    - `oom-use-tmp-storage`
    - `tmp-storage-path`
    - `tmp-storage-quota`
    - `tidb_analyze_version`
- Check the daily usage of TiDB memory on the Grafana dashboard: **TiDB** > **Server** > **Memory Usage**.
- Check the SQL statements that consume more memory.
    - View SQL statement analysis, slow queries, and memory usage on the TiDB Dashboard.

- Check `SLOW_QUERY` and `CLUSTER_SLOW_QUERY` in `INFORMATION_SCHEMA`.
- Check `tidb_slow_query.log` on each TiDB node.
- Run `grep "expensive_query" tidb.log` to check the corresponding log entries.
- Run `EXPLAIN ANALYZE` to check the memory usage of operators.
- Run `SELECT * FROM information_schema.processlist;` to check the value of the `MEM` column.

- Run the following command to collect the TiDB Profile information when memory usage is high:

```
curl -G "http://{TiDBIP}:10080/debug/zip?seconds=10" > profile.zip
```

- Run `grep "tidb-server has the risk of OOM" tidb.log` to check the path of the alert file collected by TiDB Server. The following is an example output:

```
["tidb-server has the risk of OOM. Running SQLs and heap profile will
    ↪ be recorded in record path"] ["is server-memory-quota set"=false]
    ↪  ["system memory total"=14388137984] ["system memory usage
    ↪ "=11897434112] ["tidb-server memory usage"=11223572312] [memory-
    ↪ usage-alarm-ratio=0.8] ["record path"="/tmp/0_tidb/
    ↪ MC4wLjAuMDo0MDAwLzAuMC4wLjA6MTAwODA=/tmp-storage/record"]
```

10.11.5 See also

- TiDB Memory Control
- Tune TiKV Memory Parameter Performance

10.12 TiDB Cluster Troubleshooting Guide

You can use this guide to help you diagnose and solve basic problems while using TiDB. If your problem is not resolved, please collect the following information and create an issue:

- The exact error message and the operations while the error occurs
- The state of all the components
- The `error/fatal/panic` information in the log of the component that reports the error
- The configuration and deployment topology
- The TiDB component related issue in `dmesg`

For other information, see Frequently Asked Questions (FAQ).

10.12.1 Cannot connect to the database

1. Make sure all the services are started, including `tidb-server`, `pd-server`, and `tikv-server`.
2. Use the `ps` command to check if all the processes are running.
    - If a certain process is not running, see the following corresponding sections to diagnose and solve the issue.
    - If all the processes are running, check the `tidb-server` log to see if the following messages are displayed:
        - InformationSchema is out of date: This message is displayed if the `tikv-server` cannot be connected. Check the state and log of `pd-server` and `tikv-server`.
        - panic: This message is displayed if there is an issue with the program. Please provide the detailed panic log and create an issue.
3. If the data is cleared and the services are re-deployed, make sure that:
    - All the data in `tikv-server` and `pd-server` are cleared. The specific data is stored in `tikv-server` and the metadata is stored in `pd-server`. If only one of the two servers is cleared, the data will be inconsistent.
    - After the data in `pd-server` and `tikv-server` are cleared and the `pd-server` and `tikv-server` are restarted, the `tidb-server` must be restarted too. The cluster ID is randomly allocated when the `pd-server` is initialized. So when the cluster is re-deployed, the cluster ID changes and you need to restart the `tidb-server` to get the new cluster ID.

10.12.2 Cannot start `tidb-server`

See the following for the situations when the `tidb-server` cannot be started:

- Error in the startup parameters.

  See the TiDB configuration and options.

- The port is occupied.

  Use the `lsof -i:port` command to show all the networking related to a given port and make sure the port to start the `tidb-server` is not occupied.

- Cannot connect to `pd-server`.
    - Check if the network between TiDB and PD is running smoothly, including whether the network can be pinged or if there is any issue with the Firewall configuration.
    - If there is no issue with the network, check the state and log of the `pd-server` process.

10.12.3 Cannot start `tikv-server`

See the following for the situations when the `tikv-server` cannot be started:

- Error in the startup parameters: See the TiKV configuration and options.
- The port is occupied: Use the `lsof -i:port` command to show all the networking related to a given port and make sure the port to start the `tikv-server` is not occupied.
- Cannot connect to `pd-server`.
    - Check if the network between TiDB and PD is running smoothly, including whether the network can be pinged or if there is any issue with the Firewall configuration.
    - If there is no issue with the network, check the state and log of the `pd-server` process.
- The file is occupied.

  Do not open two TiKV files on one database file directory.

10.12.4 Cannot start `pd-server`

See the following for the situations when the `pd-server` cannot be started:

- Error in the startup parameters.

  See the PD configuration and options.
- The port is occupied.

  Use the `lsof -i:port` command to show all the networking related to a given port and make sure the port to start the `pd-server` is not occupied.

10.12.5 The TiDB/TiKV/PD process aborts unexpectedly

- Is the process started on the foreground? The process might exit because the client aborts.
- Is `nohup+&` run in the command line? This might cause the process to abort because it receives the hup signal. It is recommended to write and run the startup command in a script.

10.12.6 TiDB panic

Please provide the panic log and create an issue.

10.12.7 The connection is rejected

Make sure the network parameters of the operating system are correct, including but not limited to:

- The port in the connection string is consistent with the `tidb-server` starting port.
- The firewall is configured correctly.

10.12.8 Open too many files

Before starting the process, make sure the result of `ulimit -n` is large enough. It is recommended to set the value to `unlimited` or larger than `1000000`.

10.12.9 Database access times out and the system load is too high

First, check the slow query log and see if it is because of some inappropriate SQL statement.

If you failed to solve the problem, provide the following information:

- The deployment topology
    - How many `tidb-server`/`pd-server`/`tikv-server` instances are deployed?
    - How are these instances distributed in the machines?
- The hardware configuration of the machines where these instances are deployed:
    - The number of CPU cores
    - The size of the memory
    - The type of the disk (SSD or Hard Drive Disk)
    - Are they physical machines or virtual machines?
- Are there other services besides the TiDB cluster?
- Are the `pd-server`s and `tikv-server`s deployed separately?
- What is the current operation?
- Check the CPU thread name using the `top -H` command.
- Are there any exceptions in the network or IO monitoring data recently?

10.13 Troubleshoot High Disk I/O Usage in TiDB

This document introduces how to locate and address the issue of high disk I/O usage in TiDB.

10.13.1 Check the current I/O metrics

If TiDB's response slows down after you have troubleshot the CPU bottleneck and the bottleneck caused by transaction conflicts, you need to check I/O metrics to help determine the current system bottleneck.

10.13.1.1 Locate I/O issues from monitor

The quickest way to locate I/O issues is to view the overall I/O status from the monitor, such as the Grafana dashboard which is deployed by default by TiUP. The dashboard panels related to I/O include **Overview**, **Node_exporter**, and **Disk-Performance**.

10.13.1.1.1 The first type of monitoring panels

In **Overview> System Info> IO Util**, you can see the I/O status of each machine in the cluster. This metric is similar to `util` in the Linux `iostat` monitor. The higher percentage represents higher disk I/O usage:

- If there is only one machine with high I/O usage in the monitor, currently there might be read and write hotspots on this machine.
- If the I/O usage of most machines in the monitor is high, the cluster now has high I/O loads.

For the first situation above (only one machine with high I/O usage), you can further observe I/O metrics from the **Disk-Performance Dashboard** such as `Disk Latency` and `Disk Load` to determine whether any anomaly exists. If necessary, use the fio tool to check the disk.

10.13.1.1.2 The second type of monitoring panels

The main storage component of the TiDB cluster is TiKV. One TiKV instance contains two RocksDB instances: one for storing Raft logs, located in `data/raft`, and the other for storing real data, located in `data/db`.

In **TiKV-Details** > **Raft IO**, you can see the metrics related to disk writes of these two instances:

- `Append log duration`: This metric indicates the response time of writes into RockDB that stores Raft logs. The `.99` response time should be within 50 ms.
- `Apply log duration`: This metric indicates the response time of writes into RockDB that stores real data. The `.99` response should be within 100 ms.

These two metrics also have the **.. per server** monitoring panel to help you view the write hotspots.

10.13.1.1.3 The third type of monitoring panels

In **TiKV-Details** > **Storage**, there are monitoring metrics related to storage:

- `Storage command total`: Indicates the number of different commands received.
- `Storage async write duration`: Includes monitoring metrics such as `disk sync duration`, which might be related to Raft I/O. If you encounter an abnormal situation, check the working statuses of related components by checking logs.

10.13.1.1.4 Other panels

In addition, some other panel metrics might help you determine whether the bottleneck is I/O, and you can try to set some parameters. By checking the prewrite/commit/raw-put (for raw key-value clusters only) of TiKV gRPC duration, you can determine that the bottleneck is indeed the slow TiKV write. The common situations of slow TiKV writes are as follows:

- `append log` is slow. TiKV Grafana's `Raft I/O` and `append log duration` metrics are relatively high, which is often due to slow disk writes. You can check the value of `WAL Sync Duration max` in **RocksDB-raft** to determine the cause of slow `append log`. Otherwise, you might need to report a bug.
- The `raftstore` thread is busy. In TiKV Grafana, `Raft Propose/propose wait duration` is significantly higher than `append log duration`. Check the following aspects for troubleshooting:
  - Whether the value of `store-pool-size` of `[raftstore]` is too small. It is recommended to set this value between `[1,5]` and not too large.
  - Whether the CPU resource of the machine is insufficient.
- `append log` is slow. TiKV Grafana's `Raft I/O` and `append log duration` metrics are relatively high, which might usually occur along with relatively high `Raft Propose /apply wait duration`. The possible causes are as follows:
  - The value of `apply-pool-size` of `[raftstore]` is too small. It is recommended to set this value between `[1, 5]` and not too large. The value of `Thread CPU /apply cpu` is also relatively high.
  - Insufficient CPU resources on the machine.
  - Write hotspot issue of a single Region (Currently, the solution to this issue is still on the way). The CPU usage of a single `apply` thread is high (which can be viewed by modifying the Grafana expression, appended with `by (instance, name)`).
  - Slow write into RocksDB, and `RocksDB kv/max write duration` is high. A single Raft log might contain multiple key-value pairs (kv). 128 kvs are written to RocksDB in a batch, so one `apply` log might involve multiple RocksDB writes.
  - For other causes, report them as bugs.

- `raft commit log` is slow. In TiKV Grafana, `Raft I/O` and `commit log duration` (only available in Grafana 4.x) metrics are relatively high. Each Region corresponds to an independent Raft group. Raft has a flow control mechanism similar to the sliding window mechanism of TCP. To control the size of a sliding window, adjust the `[raftstore] raft-max-inflight-msgs` parameter. If there is a write hotspot and `commit log duration` is high, you can properly set this parameter to a larger value, such as `1024`.

10.13.1.2 Locate I/O issues from log

- If the client reports errors such as `server is busy` or especially `raftstore is busy`, the errors might be related to I/O issues.

  You can check the monitoring panel (**Grafana** -> **TiKV** -> **errors**) to confirm the specific cause of the `busy` error. `server is busy` is TiKV's flow control mechanism. In this way, TiKV informs `tidb/ti-client` that the current pressure of TiKV is too high, and the client should try later.

- `Write stall` appears in TiKV RocksDB logs.

  It might be that too many level-0 SST files cause the write stall. To address the issue, you can add the `[rocksdb] max-sub-compactions = 2 (or 3)` parameter to speed up the compaction of level-0 SST files. This parameter means that the compaction tasks of level-0 to level-1 can be divided into `max-sub-compactions` subtasks for multi-threaded concurrent execution.

  If the disk's I/O capability fails to keep up with the write, it is recommended to scale up the disk. If the throughput of the disk reaches the upper limit (for example, the throughput of SATA SSD is much lower than that of NVMe SSD), which results in write stall, but the CPU resource is relatively sufficient, you can try to use a compression algorithm of higher compression ratio to relieve the pressure on the disk, that is, use CPU resources to make up for disk resources.

  For example, when the pressure of `default cf compaction` is relatively high, you can change the parameter `[rocksdb.defaultcf] compression-per-level = ["no", "no", "lz4", "lz4", "lz4", "zstd" , "zstd"]` to `compression-per-level = ["no", "no", "zstd", "zstd", "zstd", "zstd", "zstd"]`.

10.13.1.3 I/O issues found in alerts

The cluster deployment tool (TiUP) deploys the cluster with alert components by default that have built-in alert items and thresholds. The following alert items are related to I/O:

- TiKV_write_stall
- TiKV_raft_log_lag
- TiKV_async_request_snapshot_duration_seconds
- TiKV_async_request_write_duration_seconds
- TiKV_raft_append_log_duration_secs
- TiKV_raft_apply_log_duration_secs

10.13.2 Handle I/O issues

- When an I/O hotspot issue is confirmed to occur, you need to refer to Handle TiDB Hotspot Issues to eliminate the I/O hotspots.
- When it is confirmed that the overall I/O performance has become the bottleneck, and you can determine that the I/O performance will keep falling behind in the application side, then you can take advantage of the distributed database's capability of scaling and increase the number of TiKV nodes to have greater overall I/O throughput.
- Adjust some of the parameters as described above, and use computing/memory resources to make up for disk storage resources.

10.14 Troubleshoot Lock Conflicts

TiDB supports complete distributed transactions. Starting from v3.0, TiDB provides optimistic transaction mode and pessimistic transaction mode. This document describes how to use Lock View to troubleshoot lock issues and how to deal with common lock conflict issues in optimistic and pessimistic transactions.

10.14.1 Use Lock View to troubleshoot lock issues

Since v5.1, TiDB supports the Lock View feature. This feature has several system tables built in `information_schema` that provide more information about the lock conflicts and lock waitings.

> **Note:**
>
> Currently, the Lock View feature provides conflict and waiting information for pessimistic locks only.

For the detailed introduction of these tables, see the following documents:

- `TIDB_TRX` and `CLUSTER_TIDB_TRX`: Provides information of all running transactions on the current TiDB node or in the entire cluster, including whether the transaction is in the lock-waiting state, the lock-waiting time, and the digests of statements that have been executed in the transaction.
- `DATA_LOCK_WAITS`: Provides the pessimistic lock-waiting information in TiKV, including the `start_ts` of the blocking and blocked transaction, the digest of the blocked SQL statement, and the key on which the waiting occurs.
- `DEADLOCKS` and `CLUSTER_DEADLOCKS`: Provides the information of several deadlock events that have recently occurred on the current TiDB node or in the entire cluster, including the waiting relationship among transactions in the deadlock loops, the digest

of the statement currently being executed in the transaction, and the key on which the waiting occurs.

> **Note:**
>
> The SQL statements shown in the Lock View-related system tables are normalized SQL statements (that is, SQL statements without formats and arguments), which are obtained by internal queries according to SQL digests, so the tables cannot obtain the complete statements that include the format and arguments. For the detailed description of SQL digests and normalized SQL statement, see Statement Summary Tables.

The following sections show the examples of troubleshooting some issues using these tables.

10.14.1.1 Deadlock errors

To get the information of the recent deadlock errors, you can query the `DEADLOCKS` or `CLUSTER_DEADLOCKS` table.

For example, to query the `DEADLOCKS` table, you can execute the following SQL statement:

```
select * from information_schema.deadlocks;
```

The following is an example output:

```
+--
   ↪ -----------+----------------------------+-----------+--------------------+------
   ↪
| DEADLOCK_ID | OCCUR_TIME            | RETRYABLE | TRY_LOCK_TRX_ID |
   ↪ CURRENT_SQL_DIGEST                                          |
   ↪ CURRENT_SQL_DIGEST_TEXT             | KEY
   ↪                                 | KEY_INFO
   ↪
   ↪ | TRX_HOLDING_LOCK |
+--
   ↪ -----------+----------------------------+-----------+--------------------+------
   ↪
|          1 | 2021-08-05 11:09:03.230341 |    0 | 426812829645406216 |
   ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
   ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
   ↪ F728000000000000002 | {"db_id":1,"db_name":"test","table_id":53,"
   ↪ table_name":"t","handle_type":"int","handle_value":"2"} |
   ↪ 426812829645406217 |
```

```
|          1 | 2021-08-05 11:09:03.230341 |    0 | 426812829645406217 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000001 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"1"} |
    ↪ 426812829645406216 |
+--
    ↪ ----------+----------------------------+----------+------------------+------
    ↪
```

The query result above shows the waiting relationship among multiple transactions in the deadlock error, the normalized form of the SQL statements currently being executed in each transaction (statements without formats and arguments), the key on which the conflict occurs, and the information of the key.

For example, in the above example, the first row means that the transaction with the ID of `426812829645406216` is executing a statement like `update `t` set `v` =? Where `id ↪ ` =? ;` but is blocked by another transaction with the ID of `426812829645406217`. The transaction with the ID of `426812829645406217` is also executing a statement that is in the form of `update `t` set `v` =? Where `id` =? ;` but is blocked by the transaction with the ID of `426812829645406216`. The two transactions thus form a deadlock.

10.14.1.2 A few hot keys cause queueing locks

The `DATA_LOCK_WAITS` system table provides the lock-waiting status on the TiKV nodes. When you query this table, TiDB automatically obtains the real-time lock-waiting information from all TiKV nodes. If a few hot keys are frequently locked and block many transactions, you can query the `DATA_LOCK_WAITS` table and aggregate the results by key to try to find the keys on which issues frequently occur:

```
select `key`, count(*) as `count` from information_schema.data_lock_waits
    ↪ group by `key` order by `count` desc;
```

The following is an example output:

```
+----------------------------------------+-------+
| key                                    | count |
+----------------------------------------+-------+
| 748000000000000415F728000000000000001 | 2 |
| 748000000000000415F728000000000000002 | 1 |
+----------------------------------------+-------+
```

To avoid contingency, you might need to make multiple queries.

If you know the key that frequently has issues occurred, you can try to get the information of the transaction that tries to lock the key from the `TIDB_TRX` or `CLUSTER_TIDB_TRX` table.

Note that the information displayed in the `TIDB_TRX` and `CLUSTER_TIDB_TRX` tables is also the information of the transactions that are running at the time the query is performed. These tables do not display the information of the completed transactions. If there is a large number of concurrent transactions, the result set of the query might also be large. You can use the `limit` clause or the `where` clause to filter transactions with a long lock-waiting time. Note that when you join multiple tables in Lock View, the data in different tables might not be obtained at the same time, so the information in different tables might not be consistent.

For example, to filter transactions with a long lock-waiting time using the `where` clause, you can execute the following SQL statement:

```
select trx.* from information_schema.data_lock_waits as l left join
    ↪ information_schema.tidb_trx as trx on l.trx_id = trx.id where l.key =
    ↪  "748000000000000415F728000000000000001"\G
```

The following is an example output:

```
*************************** 1. row ***************************
                   ID: 426831815660273668
           START_TIME: 2021-08-06 07:16:00.081000
    CURRENT_SQL_DIGEST: 06
        ↪ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
CURRENT_SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ? ;
                 STATE: LockWaiting
    WAITING_START_TIME: 2021-08-06 07:16:00.087720
       MEM_BUFFER_KEYS: 0
      MEM_BUFFER_BYTES: 0
            SESSION_ID: 77
                  USER: root
                    DB: test
       ALL_SQL_DIGESTS: ["0
          ↪ fdc781f19da1c6078c9de7eadef8a307889c001e05f107847bee4cfc8f3cdf3
          ↪ ","06
          ↪ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
          ↪ "]
*************************** 2. row ***************************
                   ID: 426831818019569665
           START_TIME: 2021-08-06 07:16:09.081000
    CURRENT_SQL_DIGEST: 06
        ↪ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
CURRENT_SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ? ;
                 STATE: LockWaiting
    WAITING_START_TIME: 2021-08-06 07:16:09.290271
       MEM_BUFFER_KEYS: 0
      MEM_BUFFER_BYTES: 0
            SESSION_ID: 75
                  USER: root
```

```
                    DB: test
       ALL_SQL_DIGESTS: ["0
           ↪ fdc781f19da1c6078c9de7eadef8a307889c001e05f107847bee4cfc8f3cdf3
           ↪ ","06
           ↪ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
           ↪ "]
2 rows in set (0.00 sec)
```

10.14.1.3 A transaction is blocked for a long time

If a transaction is known to be blocked by another transaction (or multiple transactions) and the `start_ts` (transaction ID) of the current transaction is known, you can use the following method to obtain the information of the blocking transaction. Note that when you join multiple tables in Lock View, the data in different tables might not be obtained at the same time, so the information in different tables might not be consistent.

```
select l.key, trx.*, tidb_decode_sql_digests(trx.all_sql_digests) as sqls
   ↪ from information_schema.data_lock_waits as l join information_schema.
   ↪ cluster_tidb_trx as trx on l.current_holding_trx_id = trx.id where l.
   ↪ trx_id = 426831965449355272\G
```

The following is an example output:

```
*************************** 1. row ***************************
                    key: 7480000000000004D5F728000000000000001
             INSTANCE: 127.0.0.1:10080
                   ID: 426832040186609668
           START_TIME: 2021-08-06 07:30:16.581000
    CURRENT_SQL_DIGEST: 06
       ↪ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
CURRENT_SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ? ;
                STATE: LockWaiting
    WAITING_START_TIME: 2021-08-06 07:30:16.592763
       MEM_BUFFER_KEYS: 1
      MEM_BUFFER_BYTES: 19
            SESSION_ID: 113
                  USER: root
                    DB: test
       ALL_SQL_DIGESTS: ["0
           ↪ fdc781f19da1c6078c9de7eadef8a307889c001e05f107847bee4cfc8f3cdf3
           ↪ ","
           ↪ a4e28cc182bdd18288e2a34180499b9404cd0ba07e3cc34b6b3be7b7c2de7fe9
           ↪ ","06
           ↪ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
           ↪ "]
```

```
                sqls: ["begin ;","select * from `t` where `id` = ? for
                    ↪ update ;","update `t` set `v` = `v` + ? where `id` =
                    ↪ ? ;"]
1 row in set (0.01 sec)
```

In the above query, the TIDB_DECODE_SQL_DIGESTS function is used on the ALL_SQL_DIGESTS column of the CLUSTER_TIDB_TRX table. This function tries to convert this column (the value is a set of SQL digests) to the normalized SQL statements, which improves readability.

If the start_ts of the current transaction is unknown, you can try to find it out from the information in the TIDB_TRX / CLUSTER_TIDB_TRX table or in the PROCESSLIST / CLUSTER_PROCESSLIST table.

10.14.2 Troubleshoot optimistic lock conflicts

This section provides the solutions of common lock conflict issues in the optimistic transaction mode.

10.14.2.1 Read-write conflicts

As the TiDB server receives a read request from a client, it gets a globally unique and increasing timestamp at the physical time as the start_ts of the current transaction. The transaction needs to read the latest data before start_ts, that is, the target key of the latest commit_ts that is smaller than start_ts. When the transaction finds that the target key is locked by another transaction, and it cannot know which phase the other transaction is in, a read-write conflict happens. The diagram is as follows:

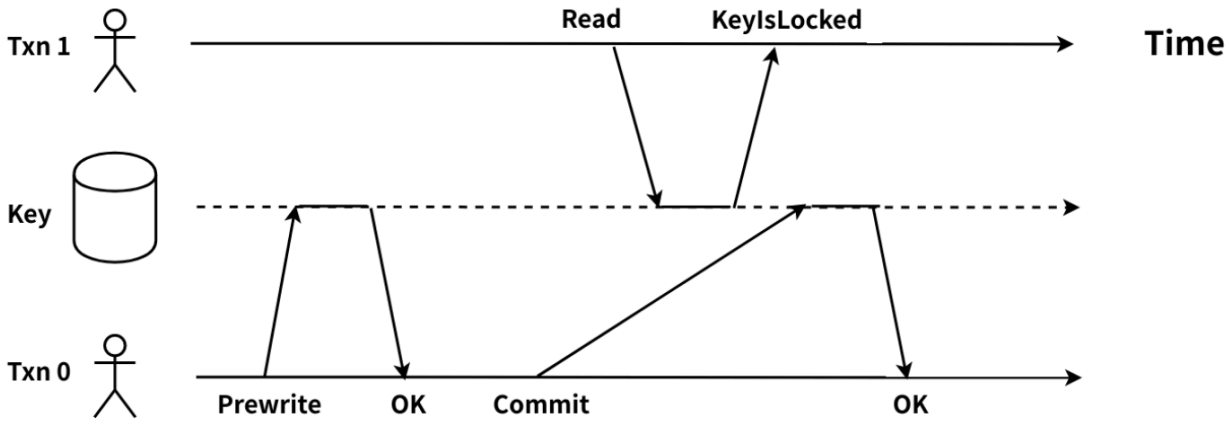


Figure 113: read-write conflict

Txn0 completes the Prewrite phase and enters the Commit phase. At this time, Txn1 requests to read the same target key. Txn1 needs to read the target key of the latest

commit_ts that is smaller than its start_ts. Because Txn1's start_ts is larger than Txn0's lock_ts, Txn1 must wait for the target key's lock to be cleared, but it hasn't been done. As a result, Txn1 cannot confirm whether Txn0 has been committed or not. Thus, a read-write conflict between Txn1 and Txn0 happens.

You can detect the read-write conflict in your TiDB cluster by the following ways:

1. Monitoring metrics and logs of the TiDB server

    - Monitoring data through Grafana

      In the `KV Errors` panel in the TiDB dashboard, `not_expired/resolve` in `Lock Resolve OPS` and `tikvLockFast` in `KV Backoff OPS` are monitoring metrics that can be used to check read-write conflicts in transactions. If the values of all the metrics increase, there might be many read-write conflicts. The `not_expired` item means that the transaction's lock has not timed out. The `resolve` item means that the other transaction tries to clean up the locks. The `tikvLockFast` item means that read-write conflicts occur.



    - Logs of the TiDB server

If there is any read-write conflict, you can see the following message in the TiDB log:

```
[INFO] [coprocessor.go:743] ["[TIME_COP_PROCESS] resp_time
    ↪ :406.038899ms txnStartTS:416643508703592451 region_id:8297
    ↪ store_addr:10.8.1.208:20160 backoff_ms:255 backoff_types:[
    ↪ txnLockFast,txnLockFast] kv_process_ms:333 scan_total_write
    ↪ :0 scan_processed_write:0 scan_total_data:0
    ↪ scan_processed_data:0 scan_total_lock:0 scan_processed_lock
    ↪ :0"]
```

- txnStartTS: The start_ts of the transaction that is sending the read request. In the above log, `416643508703592451` is the start_ts.
- backoff_types: If a read-write conflict happens, and the read request performs backoff and retry, the type of retry is `TxnLockFast`.
- backoff_ms: The time that the read request spends in the backoff and retry, and the unit is milliseconds. In the above log, the read request spends 255 milliseconds in the backoff and retry.
- region_id: Region ID corresponding to the target key of the read request.

2. Logs of the TiKV server

   If there is any read-write conflict, you can see the following message in the TiKV log:

```
[ERROR] [endpoint.rs:454] [error-response] [err=""locked primary_lock
    ↪ :7480000000000004
    ↪ D35F698000000000000010380000000004C788E0380000000004C0748
    ↪ lock_version: 411402933858205712 key: 7480000000000004
    ↪ D35F7280000000004C0748 lock_ttl: 3008 txn_size: 1""]
```

   This message indicates that a read-write conflict occurs in TiDB. The target key of the read request has been locked by another transaction. The locks are from the uncommitted optimistic transaction and the uncommitted pessimistic transaction after the prewrite phase.

   - primary_lock: Indicates that the target key is locked by the primary lock.
   - lock_version: The start_ts of the transaction that owns the lock.
   - key: The target key that is locked.
   - lock_ttl: The lock's TTL (Time To Live)
   - txn_size: The number of keys that are in the Region of the transaction that owns the lock.

Solutions:

- A read-write conflict triggers an automatic backoff and retry. As in the above example, Txn1 has a backoff and retry. The first time of the retry is 10 ms, the longest retry is 3000 ms, and the total time is 20000 ms at maximum.

- You can use the sub-command `decoder` of TiDB Control to view the table id and rowid of the row corresponding to the specified key:

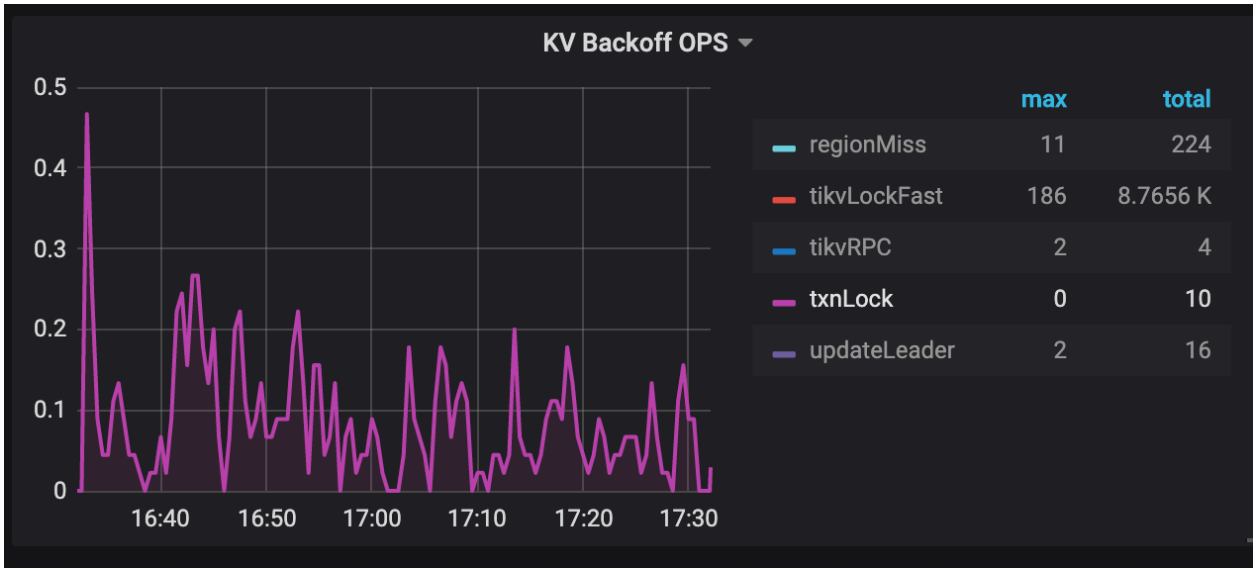
```
./tidb-ctl decoder "t\x00\x00\x00\x00\x00\x00\x1c_r\x00\x00\x00\x00
    ↪ \x00\x00\x00\xfa"
format: table_row
table_id: -9223372036854775780
row_id: -9223372036854775558
```

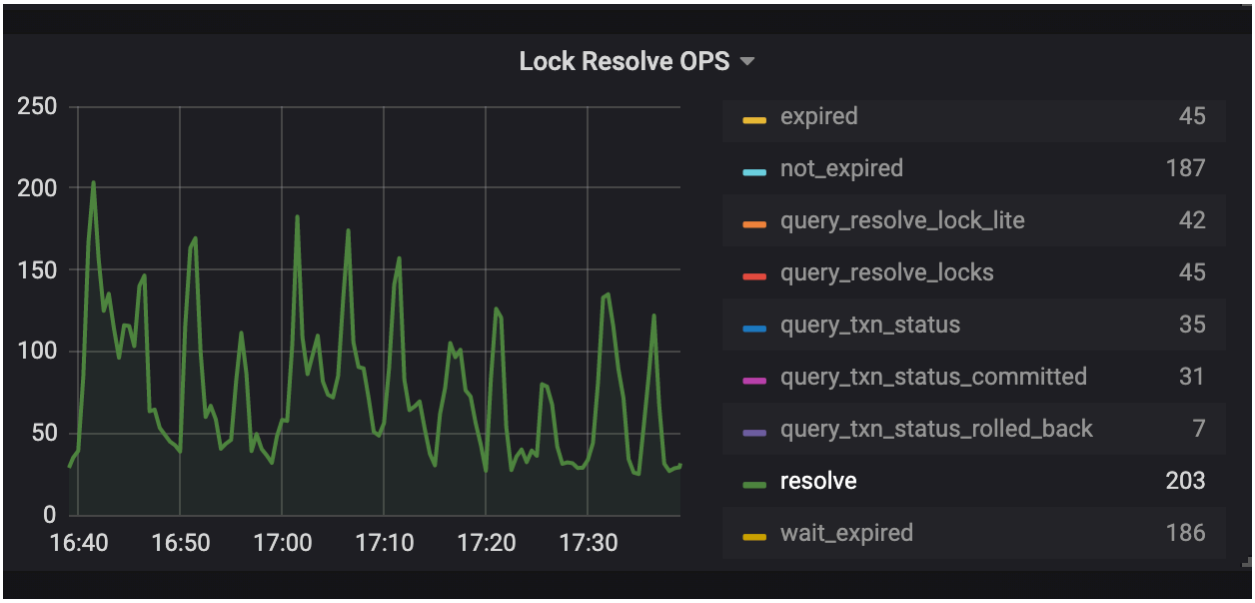
10.14.2.2 KeyIsLocked error

In the Prewrite phase of a transaction, TiDB checks whether there is any write-write conflict, and then checks whether the target key has been locked by another transaction. If the key is locked, the TiKV server outputs a "KeyIsLocked" error. At present, the error message is not printed in the logs of TiDB and TiKV. Same as read-write conflicts, when "KeyIsLocked" occurs, TiDB automatically performs backoff and retry for the transaction.

You can check whether there's any "KeyIsLocked" error in the TiDB monitoring on Grafana:

The `KV Errors` panel in the TiDB dashboard has two monitoring metrics `Lock Resolve OPS` and `KV Backoff OPS` which can be used to check write-write conflicts caused by a transaction. If the `resolve` item under `Lock Resolve OPS` and the `txnLock` item under `KV Backoff OPS` have a clear upward trend, a "KeyIsLocked" error occurs. `resolve` refers to the operation that attempts to clear the lock, and `txnLock` represents a write conflict.





Solutions:

- If there is a small amount of txnLock in the monitoring, no need to pay too much attention. The backoff and retry is automatically performed in the background. The first time of the retry is 100 ms and the maximum time is 3000 ms for a single retry.
- If there are too many "txnLock" operations in the `KV Backoff OPS`, it is recommended that you analyze the reasons to the write conflicts from the application side.
- If your application is a write-write conflict scenario, it is strongly recommended to use the pessimistic transaction mode.

10.14.2.3 LockNotFound error

The error log of "TxnLockNotFound" means that transaction commit time is longer than the the TTL time, and when the transaction is going to commit, its lock has been rolled back by other transactions. If the TiDB server enables transaction commit retry, this transaction is re-executed according to tidb_retry_limit. (Note about the difference between explicit and implicit transactions.)

You can check whether there is any "LockNotFound" error in the following ways:

1. View the logs of the TiDB server

   If a "TxnLockNotFound" error occurs, the TiDB log message is like this:

```
[WARN] [session.go:446] ["commit failed"] [conn=149370] ["finished txn
    ↪ "="Txn{state=invalid}"] [error="[kv:6]Error: KV error safe to
    ↪ retry tikv restarts txn: Txn(Mvcc(TxnLockNotFound{ start_ts:
    ↪ 412720515987275779, commit_ts: 412720519984971777, key: [116,
    ↪ 128, 0, 0, 0, 0, 1, 111, 16, 95, 114, 128, 0, 0, 0, 0, 0, 0, 2]
    ↪ })) [try again later]"]
```

- start_ts: The start_ts of the transaction that outputs the `TxnLockNotFound` error because its lock has been rolled back by other transactions. In the above log, `412720515987275779` is the start_ts.
- commit_ts: The commit_ts of the transaction that outputs the `TxnLockNotFound` error. In the above log, `412720519984971777` is the commit_ts.

2. View the logs of the TiKV server

   If a "TxnLockNotFound" error occurs, the TiKV log message is like this:

```
Error: KV error safe to retry restarts txn: Txn(Mvcc(TxnLockNotFound))
    ↪ [ERROR [Kv.rs:708] ["KvService::batch_raft send response fail"] [
    ↪ err=RemoteStoped]
```

Solutions:

- By checking the time interval between start_ts and commit_ts, you can confirm whether the commit time exceeds the TTL time.

  Checking the time interval using the PD control tool:

```
tiup ctl:<cluster-version> pd tso [start_ts]
tiup ctl:<cluster-version> pd tso [commit_ts]
```

- It is recommended to check whether the write performance is slow, which might cause that the efficiency of transaction commit is poor, and thus the lock is cleared.
- In the case of disabling the TiDB transaction retry, you need to catch the exception on the application side and try again.

10.14.3 Troubleshoot pessimistic lock conflicts

This section provides the solutions of common lock conflict issues in the pessimistic transaction mode.

> **Note:**
>
> Even if the pessimistic transaction mode is set, autocommit transactions still try to commit using the optimistic mode first. If a conflict occurs, the transactions will switch to the pessimistic transaction mode during automatic retry.

10.14.3.1 Read-write conflicts

The error messages and solutions are the same as Read-write conflict for optimistic lock conflict.

10.14.3.2 Pessimistic lock retry limit reached

When the transaction conflict is very serious or a write conflict occurs, the optimistic transaction will be terminated directly, and the pessimistic transaction will retry the statement with the latest data from storage until there is no write conflict.

Because TiDB's locking operation is a write operation, and the process of the operation is to read first and then write, there are two RPC requests. If a write conflict occurs in the middle of a transaction, TiDB will try again to lock the target keys, and each retry will be printed to the TiDB log. The number of retries is determined by pessimistic-txn.max-retry-count.

In the pessimistic transaction mode, if a write conflict occurs and the number of retries reaches the upper limit, an error message containing the following keywords appears in the TiDB log:

```
err="pessimistic lock retry limit reached"
```

Solutions:

- If the above error occurs frequently, it is recommended to adjust from the application side.

10.14.3.3 Lock wait timeout exceeded

In the pessimistic transaction mode, transactions wait for locks of each other. The timeout for waiting a lock is defined by the innodb_lock_wait_timeout parameter of TiDB. This is the maximum wait lock time at the SQL statement level, which is the expectation of a SQL statement Locking, but the lock has never been acquired. After this time, TiDB will not try to lock again and will return the corresponding error message to the client.

When a wait lock timeout occurs, the following error message will be returned to the client:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

Solutions:

- If the above error occurs frequently, it is recommended to adjust the application logic.

10.14.3.4 TTL manager has timed out

The transaction execution time cannot exceed the GC time limit. In addition, the TTL time of pessimistic transactions has an upper limit, whose default value is 1 hour. Therefore, a pessimistic transaction executed for more than 1 hour will fail to commit. This timeout threshold is controlled by the TiDB parameter `performance.max-txn-ttl`.

When the execution time of a pessimistic transaction exceeds the TTL time, the following error message occurs in the TiDB log:

```
TTL manager has timed out, pessimistic locks may expire, please commit or
    ↪ rollback this transaction
```

Solutions:

- First, confirm whether the application logic can be optimized. For example, large transactions may trigger TiDB's transaction size limit, which can be split into multiple small transactions.
- Also, you can adjust the related parameters properly to meet the application transaction logic.

10.14.3.5 Deadlock found when trying to get lock

Due to resource competition between two or more transactions, a deadlock occurs. If you do not handle it manually, transactions that block each other cannot be executed successfully and will wait for each other forever. To resolve dead locks, you need to manually terminate one of the transactions to resume other transaction requests.

When a pessimistic transaction has a deadlock, one of the transactions must be terminated to unlock the deadlock. The client will return the same `Error 1213` error as in MySQL, for example:

```
[err="[executor:1213]Deadlock found when trying to get lock; try restarting
    ↪ transaction"]
```

Solutions:

- If it is difficult to confirm the cause of the deadlock, for v5.1 and later versions, you are recommended to try to query the `INFORMATION_SCHEMA.DEADLOCKS` or `INFORMATION_SCHEMA.CLUSTER_DEADLOCKS` system table to get the information of deadlock waiting chain. For details, see the Deadlock errors section and the `DEADLOCKS` table document.
- If the deadlock occurs frequently, you need to adjust the transaction query logic in your application to reduce such occurrences.

10.15 Troubleshoot a TiFlash Cluster

This section describes some commonly encountered issues when using TiFlash, the reasons, and the solutions.

10.15.1 TiFlash fails to start

The issue might occur due to different reasons. It is recommended that you troubleshoot it following the steps below:

1. Check whether your system is CentOS8.

   CentOS8 does not have the `libnsl.so` system library. You can manually install it via the following command:

   ```
   shell dnf install libnsl
   ```

2. Check your system's `ulimit` parameter setting.

   ```
   shell ulimit -n 1000000
   ```

3. Use the PD Control tool to check whether there is any TiFlash instance that failed to go offline on the node (same IP and Port) and force the instance(s) to go offline. For detailed steps, refer to Scale in a TiFlash cluster.

If the above methods cannot resolve your issue, save the TiFlash log files and get support from PingCAP or the community.

10.15.2 TiFlash replica is always unavailable

This is because TiFlash is in an abnormal state caused by configuration errors or environment issues. Take the following steps to identify the faulty component:

1. Check whether PD enables the `Placement Rules` feature:

   ```
   echo 'config show replication' | /path/to/pd-ctl -u http://${pd-ip}:${
       ↪ pd-port}
   ```

   - If `true` is returned, go to the next step.
   - If `false` is returned, enable the Placement Rules feature and go to the next step.

2. Check whether the TiFlash process is working correctly by viewing `UpTime` on the TiFlash-Summary monitoring panel.

3. Check whether the TiFlash proxy status is normal through `pd-ctl`.

   ```
   echo "store" | /path/to/pd-ctl -u http://${pd-ip}:${pd-port}
   ```

   The TiFlash proxy's `store.labels` includes information such as `{"key": "engine ↪ ", "value": "tiflash"}`. You can check this information to confirm a TiFlash proxy.

4. Check whether `pd buddy` can correctly print the logs (the log path is the value of `log` in the [flash.flash_cluster] configuration item; the default log path is under the `tmp` directory configured in the TiFlash configuration file).

5. Check whether the number of configured replicas is less than or equal to the number of TiKV nodes in the cluster. If not, PD cannot replicate data to TiFlash:

```
echo 'config placement-rules show' | /path/to/pd-ctl -u http://${pd-ip
    ↪ }:${pd-port}
```

Reconfirm the value of `default: count`.

Note:

- When Placement Rules are enabled and multiple rules exist, the previously configured `max-replicas`, `location-labels`, and `isolation-level` no longer take effect. To adjust the replica policy, use the interface related to Placement Rules.
- When Placement Rules are enabled and only one default rule exists, TiDB will automatically update this default rule when `max -replicas`, `location-labels`, or `isolation-level` configurations are changed.

6. Check whether the remaining disk space of the machine (where `store` of the TiFlash node is) is sufficient. By default, when the remaining disk space is less than 20% of the `store` capacity (which is controlled by the `low-space-ratio` parameter), PD cannot schedule data to this TiFlash node.

10.15.3 Some queries return the `Region Unavailable` error

If the load pressure on TiFlash is too heavy and it causes that TiFlash data replication falls behind, some queries might return the `Region Unavailable` error.

In this case, you can balance the load pressure by adding more TiFlash nodes.

10.15.4 Data file corruption

Take the following steps to handle the data file corruption:

1. Refer to Take a TiFlash node down to take the corresponding TiFlash node down.
2. Delete the related data of the TiFlash node.
3. Redeploy the TiFlash node in the cluster.

10.15.5 TiFlash analysis is slow

If a statement contains operators or functions not supported in the MPP mode, TiDB does not select the MPP mode. Therefore, the analysis of the statement is slow. In this case, you can execute the `EXPLAIN` statement to check for operators or functions not supported in the MPP mode.

```
create table t(a datetime);
alter table t set tiflash replica 1;
insert into t values('2022-01-13');
set @@session.tidb_enforce_mpp=1;
explain select count(*) from t where subtime(a, '12:00:00') > '2022-01-01'
    ↪ group by a;
show warnings;
```

In this example, the warning message shows that TiDB does not select the MPP mode because TiDB 5.4 and earlier versions do not support the `subtime` function.

```
+---------+------+-----------------------------------------------------------------------
    ↪
> | Level   | Code | Message
    ↪                                                                  |
+---------+------+-----------------------------------------------------------------------
    ↪
| Warning | 1105 | Scalar function 'subtime'(signature: SubDatetimeAndString
    ↪ , return type: datetime) is not supported to push down to tiflash now
    ↪ . |
+---------+------+-----------------------------------------------------------------------
    ↪
```

10.15.6 Data is not replicated to TiFlash

After deploying a TiFlash node and starting replication (by performing the ALTER operation), no data is replicated to it. In this case, you can identify and address the problem by following the steps below:

1. Check whether the replication is successful by running the `ALTER table <tbl_name>` ↪ `set tiflash replica <num>` command and check the output.

    - If there is output, go to the next step.
    - If there is no output, run the `SELECT * FROM information_schema.tiflash_replica` ↪ command to check whether TiFlash replicas have been created. If not, run the `ALTER table ${tbl_name} set tiflash replica ${num}` command again, check whether other statements (for example, `add index`) have been executed, or check whether DDL executions are successful.

2. Check whether TiFlash Region replication runs correctly.

    Check whether there is any change in `progress`:

- If yes, TiFlash replication runs correctly.

- If no, TiFlash replication is abnormal. In `tidb.log`, search the log saying `Tiflash replica is not available`. Check whether `progress` of the corresponding table is updated. If not, check the `tiflash log` for further information. For example, search `lag_region_info` in `tiflash log` to find out which Region lags behind.

3. Check whether the Placement Rules function has been enabled by using pd-ctl:

```
echo 'config show replication' | /path/to/pd-ctl -u http://<pd-ip>:<pd-port>
```

   - If `true` is returned, go to the next step.
   - If `false` is returned, enable the Placement Rules feature and go to the next step.

4. Check whether the `max-replicas` configuration is correct:

   - If the value of `max-replicas` does not exceed the number of TiKV nodes in the cluster, go to the next step.
   - If the value of `max-replicas` is greater than the number of TiKV nodes in the cluster, the PD does not replicate data to the TiFlash node. To address this issue, change `max-replicas` to an integer fewer than or equal to the number of TiKV nodes in the cluster.

> **Note:**
>
> `max-replicas` is defaulted to 3. In production environments, the value is usually fewer than the number of TiKV nodes. In test environments, the value can be 1.

```
curl -X POST -d '{
    "group_id": "pd",
    "id": "default",
    "start_key": "",
    "end_key": "",
    "role": "voter",
    "count": 3,
    "location_labels": [
    "host"
    ]
}' <http://172.16.x.xxx:2379/pd/api/v1/config/rule>
```

5. Check whether TiDB has created any placement rule for tables.

   Search the logs of TiDB DDL Owner and check whether TiDB has notified PD to add placement rules. For non-partitioned tables, search `ConfigureTiFlashPDForTable`. For partitioned tables, search `ConfigureTiFlashPDForPartitions`.

- If the keyword is found, go to the next step.
- If not, collect logs of the corresponding component for troubleshooting.

6. Check whether PD has configured any placement rule for tables.

   Run the `curl http://<pd-ip>:<pd-port>/pd/api/v1/config/rules/group/tiflash` command to view all TiFlash placement rules on the current PD. If a rule with the ID being `table-<table_id>-r` is found, the PD has configured a placement rule successfully.

7. Check whether the PD schedules properly.

   Search the `pd.log` file for the `table-<table_id>-r` keyword and scheduling behaviors like `add operator`.

   - If the keyword is found, the PD schedules properly.
   - If not, the PD does not schedule properly.

10.15.7 Data replication gets stuck

If data replication on TiFlash starts normally but then all or some data fails to be replicated after a period of time, you can confirm or resolve the issue by performing the following steps:

1. Check the disk space.

   Check whether the disk space ratio is higher than the value of `low-space-ratio` (defaulted to 0.8. When the space usage of a node exceeds 80%, the PD stops migrating data to this node to avoid exhaustion of disk space).

   - If the disk usage ratio is greater than or equal to the value of `low-space-ratio`, the disk space is insufficient. To relieve the disk space, remove unnecessary files, such as `space_placeholder_file` (if necessary, set `reserve-space` to 0MB after removing the file) under the `${data}/flash/` folder.
   - If the disk usage ratio is less than the value of `low-space-ratio`, the disk space is sufficient. Go to the next step.

2. Check whether there is any `down peer` (a `down peer` might cause the replication to get stuck).

   Run the `pd-ctl region check-down-peer` command to check whether there is any `down peer`. If any, run the `pd-ctl operator add remove-peer <region-id> <tiflash-store-id>` command to remove it.

10.15.8 Data replication is slow

The causes may vary. You can address the problem by performing the following steps.

1. Increase `store limit` to accelerate replication.
2. Adjust the load on TiFlash.

   Excessively high load on TiFlash can also result in slow replication. You can check the load of TiFlash indicators on the **TiFlash-Summary** panel on Grafana:

   - `Applying snapshots Count`: `TiFlash-summary` > `raft` > `Applying snapshots Count`
   - `Snapshot Predecode Duration`: `TiFlash-summary` > `raft` > `Snapshot Predecode Duration`
   - `Snapshot Flush Duration`: `TiFlash-summary` > `raft` > `Snapshot Flush Duration`
   - `Write Stall Duration`: `TiFlash-summary` > `Storage Write Stall` > `Write Stall Duration`
   - `generate snapshot CPU`: `TiFlash-Proxy-Details` > `Thread CPU` > `Region task worker pre-handle/generate snapshot CPU`

   Based on your service priorities, adjust the load accordingly to achieve optimal performance.

10.16 Troubleshoot Write Conflicts in Optimistic Transactions

This document introduces the reason of and solutions to write conflicts in optimistic transactions.

Before TiDB v3.0.8, TiDB uses the optimistic transaction model by default. In this model, TiDB does not check conflicts during transaction execution. Instead, while the transaction is finally committed, the two-phase commit (2PC) is triggered and TiDB checks write conflicts. If a write conflict exists and the auto-retry mechanism is enabled, then TiDB retries the transaction within limited times. If the retry succeeds or has reached the upper limit on retry times, TiDB returns the result of transaction execution to the client. Therefore, if a lot of write conflicts exist in the TiDB cluster, the duration can be longer.

10.16.1 The reason of write conflicts

TiDB implements its transactions by using the Percolator transaction model. `percolator` is generally an implementation of 2PC. For the detailed 2PC process, see TiDB Optimistic Transaction Model.

After the client sends a `COMMIT` request to TiDB, TiDB starts the 2PC process:

1. TiDB chooses one key from all keys in the transaction as the primary key of the transaction.
2. TiDB sends the `prewrite` request to all the TiKV Regions involved in this commit. TiKV judges whether all keys can preview successfully.
3. TiDB receives the result that all `prewrite` requests are successful.
4. TiDB gets the `commit_ts` from PD.
5. TiDB sends the `commit` request to the TiKV Region that contains the primary key of the transaction. After TiKV receives the `commit` request, it checks the validity of the data and clears the locks left in the `prewrite` stage.
6. After the `commit` request returns successfully, TiDB returns success to the client.

The write conflict occurs in the `prewrite` stage. When the transaction finds that another transaction is writing the current key (`data.commit_ts` > `txn.start_ts`), a write conflict occurs.

10.16.2 Detect write conflicts

In the TiDB Grafana panel, check the following monitoring metrics under **KV Errors**:

- **KV Backoff OPS** indicates the count of error messages per second returned by TiKV.

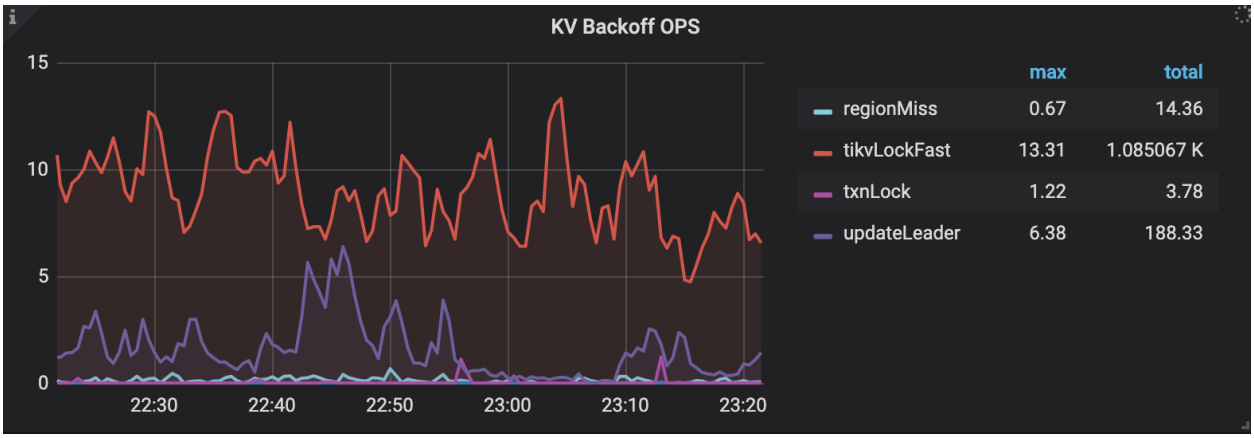


Figure 114: kv-backoff-ops

The `txnlock` metric indicates the write-write conflict. The `txnLockFast` metric indicates the read-write conflict.

- **Lock Resolve OPS** indicates the count of items related to transaction conflicts per second:



Figure 115: lock-resolve-ops

- `not_expired` indicates the TTL of the lock was not expired. The conflict transaction cannot resolve locks until the TTL is expired.
- `wait_expired` indicates that the transaction needs to wait the lock to expire.
- `expired` indicates the TTL of the lock was expired. Then the conflict transaction can resolve this lock.

- **KV Retry Duration** indicates the duration of re-sends the KV request:

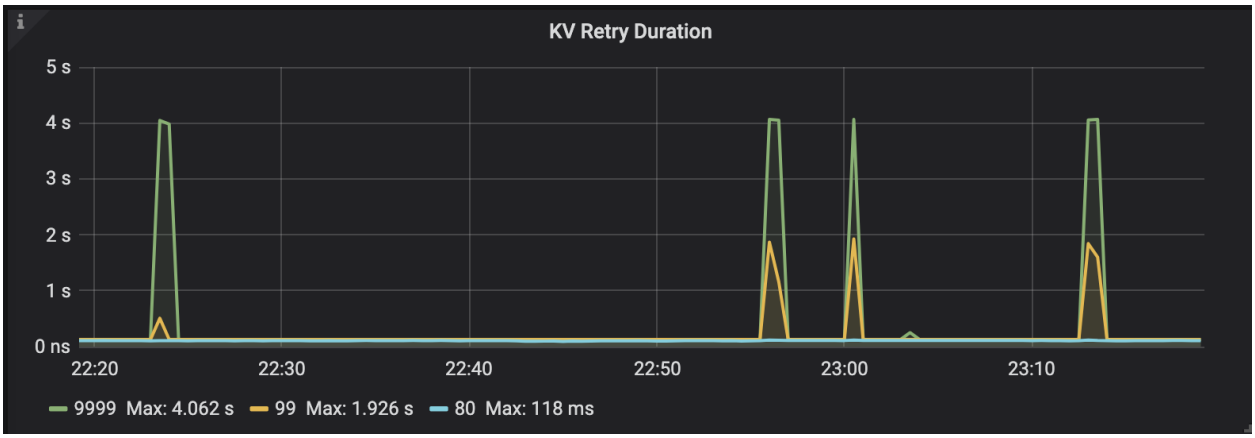


Figure 116: kv-retry-duration

You can also use `[kv:9007]Write conflict` as the keywords to search in the TiDB log. The keywords also indicate the write conflict exists in the cluster.

10.16.3 Resolve write conflicts

If many write conflicts exist in the cluster, it is recommended to find out the write conflict key and the reason, and then try to change the application logic to avoid write conflicts.

When the write conflict exists in the cluster, you can see the log similar to the following one in the TiDB log file:

```
[2020/05/12 15:17:01.568 +08:00] [WARN] [session.go:446] ["commit failed"] [
    ↪ conn=3] ["finished txn"="Txn{state=invalid}"] [error="[kv:9007]Write
    ↪ conflict, txnStartTS=416617006551793665, conflictStartTS
    ↪ =416617018650001409, conflictCommitTS=416617023093080065, key={
    ↪ tableID=47, indexID=1, indexValues={string, }} primary={tableID=47,
    ↪ indexID=1, indexValues={string, }} [try again later]"]
```

The explanation of the log above is as follows:

- `[kv:9007]Write conflict`: indicates the write-write conflict.
- `txnStartTS=416617006551793665`: indicates the `start_ts` of the current transaction. You can use the `pd-ctl` tool to convert `start_ts` to physical time.
- `conflictStartTS=416617018650001409`: indicates the `start_ts` of the write conflict transaction.
- `conflictCommitTS=416617023093080065`: indicates the `commit_ts` of the write conflict transaction.
- `key={tableID=47, indexID=1, indexValues={string, }}`: indicates the write conflict key. `tableID` indicates the ID of the write conflict table. `indexID` indicates the ID of write conflict index. If the write conflict key is a record key, the log prints `handle=x`, indicating which record(row) has a conflict. `indexValues` indicates the value of the index that has a conflict.
- `primary={tableID=47, indexID=1, indexValues={string, }}`: indicates the primary key information of the current transaction.

You can use the `pd-ctl` tool to convert the timestamp to readable time:

```
tiup ctl:<cluster-version> pd -u https://127.0.0.1:2379 tso {TIMESTAMP}
```

You can use `tableID` to find the name of the related table:

```
curl http://{TiDBIP}:10080/db-table/{tableID}
```

You can use `indexID` and the table name to find the name of the related index:

```
SELECT * FROM INFORMATION_SCHEMA.TIDB_INDEXES WHERE TABLE_SCHEMA='{db_name}
    ↪ ' AND TABLE_NAME='{table_name}' AND INDEX_ID={indexID};
```

In addition, in TiDB v3.0.8 and later versions, the pessimistic transaction becomes the default mode. The pessimistic transaction mode can avoid write conflicts during the transaction prewrite stage, so you do not need to modify the application any more. In the pessimistic transaction mode, each DML statement writes a pessimistic lock to the related keys during execution. This pessimistic lock can prevent other transactions from modifying the same keys, thus ensuring no write conflicts exist in the `prewrite` stage of the transaction 2PC.

10.17 Troubleshoot Inconsistency Between Data and Indexes

TiDB checks consistency between data and indexes when it executes transactions or the `ADMIN CHECK [TABLE|INDEX]` statement. If the check finds that a record key-value and the corresponding index key-value are inconsistent, that is, a key-value pair storing row data and the corresponding key-value pair storing its index are inconsistent (for example, more indexes or missing indexes), TiDB reports a data inconsistency error and prints the related errors in error logs.

This document describes the meanings of data inconsistency errors and provides some methods to bypass the consistency check. If a data consistency error occurs, you can get support from PingCAP or the community.

10.17.1 Error explanation

When inconsistency between data and indexes occurs, you can check TiDB error messages to know which item is inconsistent between row data and index data, or check the related error logs for further investigation.

10.17.1.1 Errors reported during transaction execution

This section lists the data inconsistency errors reported when TiDB executes transactions and explains the meanings of these errors with examples.

10.17.1.1.1 Error 8133

```
ERROR 8133 (HY000): data inconsistency in table: t, index: k2, index-
↪ count:1 != record-count:0
```

This error indicates that for the `k2` index in table `t`, the number of indexes in the table is 1 and the number of row records is 0. The number is inconsistent.

10.17.1.1.2 Error 8138

```
ERROR 8138 (HY000): writing inconsistent data in table: t, expected-
↪ values:{KindString green} != record-values:{KindString GREEN}
```

This error indicates that the transaction was attempting to write an incorrect row value. For the data to be written, the encoded row data does not match the original data before encoding.

10.17.1.1.3 Error 8139

```
ERROR 8139 (HY000): writing inconsistent data in table: t, index: i1,
↪ index-handle:4 != record-handle:3, index: tables.mutation{key:kv.Key{0
↪ x74, 0x80, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x49, 0x5f, 0x69, 0x80, 0x0, 0
↪ x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0x1, 0x68, 0x65, 0x6c, 0x6c, 0x6f, 0x0, 0x0,
```

```
↪  0x0, 0xfc, 0x1, 0x68, 0x65, 0x6c, 0x6c, 0x6f, 0x0, 0x0, 0x0, 0xfc, 0x3,
↪  0x80, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x4}, flags:0x0, value:[]uint8{0x30
↪ }, indexID:1}, record: tables.mutation{key:kv.Key{0x74, 0x80, 0x0, 0x0,
↪ 0x0, 0x0, 0x0, 0x0, 0x49, 0x5f, 0x72, 0x80, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
↪ , 0x3}, flags:0xd, value:[]uint8{0x80, 0x0, 0x2, 0x0, 0x0, 0x0, 0x1, 0x2
↪ , 0x5, 0x0, 0xa, 0x0, 0x68, 0x65, 0x6c, 0x6c, 0x6f, 0x68, 0x65, 0x6c, 0
↪ x6c, 0x6f}, indexID:0}
```

This error indicates that the handle (that is, the key of the row data) of the data to be written is inconsistent. For index `i1` in table `t`, the row to be written by the transaction has a handle of 4 in the index key-value pair and a handle of 3 in the row record key-value pair. The data of this row will not be written.

10.17.1.1.4 Error 8140

```
ERROR 8140 (HY000): writing inconsistent data in table: t, index: i2,
↪ col: c1, indexed-value:{KindString hellp} != record-value:{KindString
↪ hello}
```

This error indicates that the data in a row to be written by the transaction does not match the data in the index. For index `i2` in table `t`, a row to be written by the transaction has data `hellp` in the index key-value pair and data `hello` in the record key-value pair. The data of this row will not be written.

10.17.1.1.5 Error 8141

```
ERROR 8141 (HY000): assertion failed: key: 748000000000000405f7201330000000000000000f8
↪ , assertion: NotExist, start_ts: 430590532931813377, existing start ts:
↪ 430590532931551233, existing commit ts: 430590532931551234
```

This error indicates that the assertion failed when a transaction was being committed. Assuming that data and indexes are consistent, TiDB asserted that the key `748000000000000405f7201330000000000000000f8` did not exist. When the transaction was being committed, TiDB found the key did exist, written by the transaction with the `start ts` `430590532931551233`. TiDB will print the Multi-Version Concurrency Control (MVCC) history of this key to logs.

10.17.1.2 Errors reported in admin check

This section lists the data inconsistency errors that might occur in TiDB when you execute the `ADMIN CHECK [TABLE|INDEX]` statement, and explains the meanings of these errors with examples.

10.17.1.2.1 Error 8003

```
ERROR 8003 (HY000): table count 3 != index(idx)count 2
```

This error indicates that the table on which the `ADMIN CHECK` statement is executed has 3 row key-value pairs but only 2 index key-value pairs.

10.17.1.2.2 Error 8134

```
ERROR 8134 (HY000): data inconsistency in table: t, index: c2, col: c2
↪ , handle: "2", index-values:"KindInt64 13" != record-values:"KindInt64
↪ 12", compare err:<nil>
```

This error indicates that for index `c2` in table `t`, the value of column `c2` has the following inconsistency:

- In the index key-value pair of the row whose handle is `2`, the value of column `c2` is `13`.
- In the row record key-value pair, the value of column `c2` is `12`.

10.17.1.2.3 Error 8223

```
ERROR 8223 (HY000): data inconsistency in table: t2, index: i1, handle:
↪  {hello, hello}, index-values:"" != record-values:"handle: {hello, hello
↪ }, values: [KindString hello KindString hello]"
```

This error indicates that `index-values` are null and `record-values` are not null, meaning that there is no corresponding index for the row.

10.17.2 Solutions

If you encounter a data inconsistency error, get support from PingCAP for troubleshooting immediately instead of dealing with the error by yourself. If your application needs to skip such errors urgently, you can use the following methods to bypass the check.

10.17.2.1 Rewrite SQL

If the data inconsistency error occurs in a particular SQL statement only, you can bypass this error by rewriting the SQL statement to another equivalent form using different execution operators.

10.17.2.2 Disable error checks

For the following errors reported in transaction execution, you can bypass the corresponding checks:

- To bypass the checks of errors 8138, 8139, and 8140, configure `set @@tidb_enable_mutation_checke` ↪ `=0`.
- To bypass the checks of error 8141, configure `set @@tidb_txn_assertion_level=OFF`.

> **Note:**
>
> Disabling `tidb_enable_mutation_checker` and `tidb_txn_assertion_level` will bypass the corresponding checks of all SQL statements.

For other errors reported in transaction execution and all errors reported during the execution of the `ADMIN CHECK [TABLE|INDEX]` statement, you cannot bypass the corresponding check, because the data is already inconsistent.

10.18 Support Resources

If you encounter a problem when you use TiDB, you can reach out for support from PingCAP or the TiDB community via the following methods:

- Get support from PingCAP (requires subscription to TiDB Enterprise Edition):
    - Submit a request
- Seek help from the TiDB community:
    - The TiDB Forum
    - Slack channels: #everyone (English), #tidb-japan (Japanese)
    - Stack Overflow (questions tagged with #tidb)
- Learn TiDB's implementation and design
    - Discussions on GitHub

11 Performance Tuning

11.1 Tuning Guide

11.1.1 TiDB Performance Tuning Overview

This document introduces the basic concepts of performance tuning, such as user response time, throughput, and database time, and also provides a general process for performance tuning.

11.1.1.1 User response time and database time

11.1.1.1.1 User response time

User response time indicates how long an application takes to return the results of a request to users. As you can see from the following sequential timing diagram, the time of a typical user request contains the following:

- The network latency between the user and the application
- The processing time of the application

- The network latency during the interaction between the application and the database
- The service time of the database

The user response time is affected by various subsystems on the request chain, such as network latency and bandwidth, number and request types of concurrent users, and resource usage of server CPU and I/O. To optimize the entire system effectively, you need to first identify the bottlenecks in user response time.

To get a total user response time within a specified time range (`ΔT`), you can use the following formula:

Total user response time in `ΔT` = Average TPS (Transactions Per Second) x Average user response time x `ΔT`.

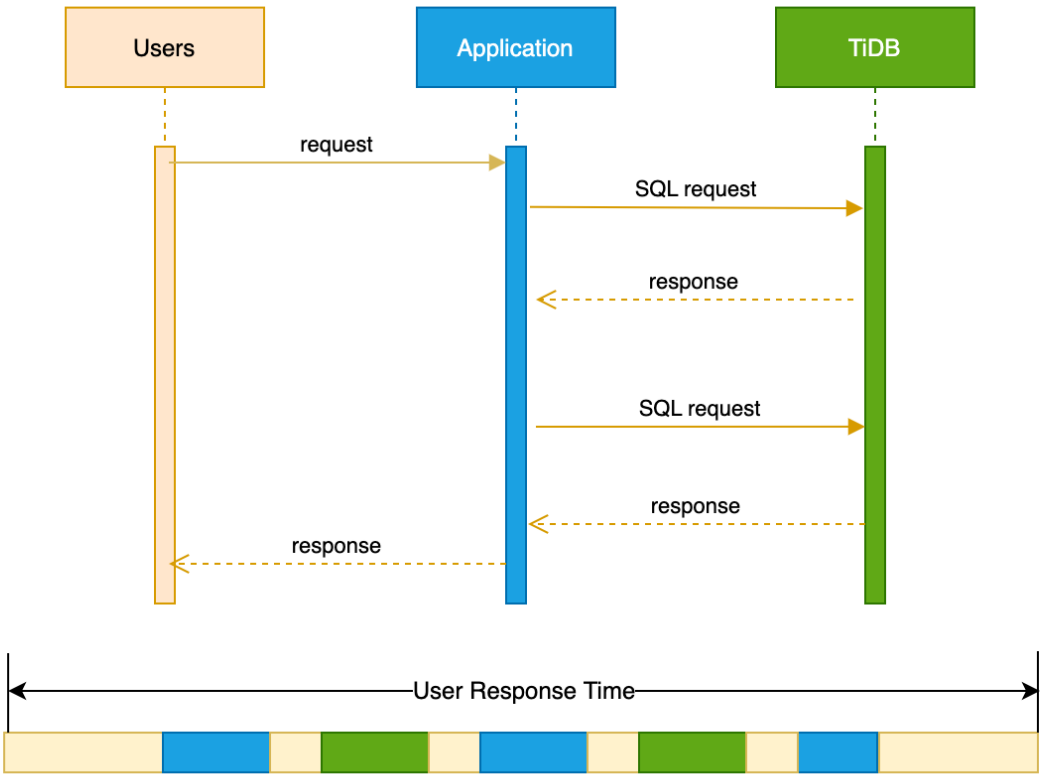


Figure 117: user_response_time

11.1.1.1.2 Database time

Database time indicates the total service time provided by a database. The database time in `ΔT` is the sum of the time that a database takes to process all application requests concurrently.

To get the database time, you can use any of the following methods:

- Method 1: Multiply the average query latency by QPS and by ΔT, that is, `DB Time in ΔT = QPS × avg latency × ΔT`
- Method 2: Multiply the average number of active sessions by ΔT, that is, `DB Time in ΔT = avg active connections × ΔT`
- Method 3: Calculate the time based on the TiDB internal Prometheus metric TiDB_server_handle_query_duration_seconds_sum, that is. `ΔT DB Time = rate(TiDB_server_handle_query_duration_seconds_sum)× ΔT`

11.1.1.2 Relationship between user response time and system throughput

User response time consists of service time, queuing time, and concurrent waiting time to complete a user request.

```
User Response time = Service time + Queuing delay + Coherency delay
```

- Service time: the time a system consumes on certain resources when processing a request, for example, the CPU time that a database consumes to complete a SQL request.
- Queuing delay: the time a system waits in a queue for service of certain resources when processing a request.
- Coherency delay: the time a system communicates and collaborates with other concurrent tasks, so that it can access shared resources when processing a request.

System throughput indicates the number of requests that can be completed by a system per second. User response time and throughput are usually inverse of each other. When the throughput increases, the system resource utilization and the queuing latency for a requested service increase accordingly. Once resource utilization exceeds a certain inflection point, the queuing latency will increase dramatically.

For example, for a database system running OLTP loads, after its CPU utilization exceeds 65%, the CPU queueing scheduling latency increases significantly. This is because concurrent requests of a system are not completely independent, which means that these requests can collaborate and compete for shared resources. For example, requests from different users might perform mutually exclusive locking operations on the same data. When the resource utilization increases, the queuing and scheduling latency increases too, which causes that the shared resources cannot be released in time and in turn prolongs the waiting time for shared resources by other tasks.

11.1.1.3 Performance tuning process

The performance tuning process consists of the following 6 steps:

1. Define a tuning objective.

2. Establish a performance baseline.
3. Identify bottlenecks in user response time.
4. Propose tuning solutions, and evaluate the benefits, risks, and costs of each solution.
5. Implement tuning solutions.
6. Evaluate tuning results.

To achieve the tuning objective of a performance tuning project, you usually need to repeat Step 2 to Step 6 multiple times.

11.1.1.3.1 Step 1. Define a tuning objective

For different types of systems, tuning objectives are different too. For example, for a financial core OLTP system, the tuning objective might be to reduce the long-tail latency of transactions; for a financial settlement system, the tuning objective might be to make better use of hardware resources and reduce the time of batch settlement tasks.

A good tuning objective should be easily quantifiable. For example:

- Good tuning objective: The p99 latency for transfer transactions needs to be less than 200 ms during peak business hours of 9 am to 10 am.
- Poor tuning objective: The system is too slow to respond so it needs to be optimized.

Defining a clear tuning objective helps guide the subsequent performance tuning steps.

11.1.1.3.2 Step 2. Establish a performance baseline

To tune performance efficiently, you need to capture the current performance data to establish a performance baseline. The performance data to be captured typically includes the following:

- Mean and long-tail values of user response time, and throughput of your application
- Database performance data such as database time, query latency, and QPS

  TiDB measures and stores performance data thoroughly in different dimensions, such as slow query logs, Top SQL, Continuous Performance Profiling, and traffic visualizer. In addition, you can perform historical backtracking and comparison of the timing metrics data stored in Prometheus.
- Resource utilization, including resources such as CPU, IO, and network
- Configuration information, such as application configurations, database configurations, and operating system configurations

11.1.1.3.3 Step 3. Identify bottlenecks in user response time

Identify or speculate on bottlenecks in user response times based on data from the performance baseline.

Applications usually do not measure and record the full chain of user requests, so you cannot effectively break down user response time from top to bottom through the application.

In contrast, databases have a complete record of performance metrics such as query latency and throughput. Based on database time, you can determine if the bottleneck in user response time is in a database.

- If the bottleneck is not in databases, you need to rely on the resource utilization collected outside databases or profile the application to identify the bottleneck outside databases. Common scenarios include insufficient resources of an application or proxy server, and insufficient usage of hardware resources caused by serial points in an application.
- If bottlenecks are in databases, you can analyze and diagnose the database performances using comprehensive tuning tools. Common scenarios include the presence of slow SQL, unreasonable usage of a database by an application, and the presence of read and write hotspots in databases.

For more information about the analysis and diagnostic methods and tools, see Performance Analysis and Tuning.

11.1.1.3.4 Step 4. Propose tuning solutions, and evaluate the benefits, risks, and costs of each solution

After identifying the bottleneck of a system through performance analysis, you can propose a tuning solution that is cost-effective, has low risks, and provides the maximum benefit based on the actual situation.

According to Amdahl's Law, the maximum gain from performance tuning depends on the percentage of the optimized part in the overall system. Therefore, you need to identify the system bottlenecks and the corresponding percentage based on the performance data, and then predict the gains after the bottleneck is resolved or optimized.

Note that even if a solution can bring the greatest potential benefits by tunning the largest bottleneck, you still need to evaluate the risks and costs of this solution. For example:

- The most straightforward tuning objective solution for a resource-overloaded system is to expand its capacity, but in practice, the expansion solution might be too costly to be adopted.
- When a slow query in a business module causes a slow response of the entire module, upgrading to a new version of the database can solve the slow query issue, but it might also affect modules that did not have this issue. Therefore, this solution might have a potentially high risk. A low-risk solution is to skip the database version upgrade and rewrite the existing slow queries for the current database version.

11.1.1.3.5 Step 5. Implement tuning solutions

Considering the benefits, risks, and costs, choose one or more tuning solutions for implementation. In the implementation process, you need to make thorough preparation for changes to the production system and record the changes in detail.

To mitigate risks and validate the benefits of a tuning solution, it is recommended that you perform validation and complete regression of changes in both test and staging environments. For example, if the selected tuning solution of a slow query is to create a new index to optimize the query access path, you need to ensure that the new index does not introduce any obvious write hotspots to the existing data insertion workload and slows down other modules.

11.1.1.3.6 Step 6. Evaluate tuning results

After applying the tuning solution, you need to evaluate the results:

- If the tuning objective is reached, the entire tuning project is completed successfully.
- If the tuning objective is not reached, you need to repeat Step 2 to Step 6 in this document until the tuning objective is reached.

After reaching your tuning objectives, you might need to further plan your system capacity to meet your business growth.

11.1.2 Performance Analysis and Tuning

This document describes a tuning approach by database time, and illustrates how to use the TiDB Performance Overview dashboard for performance analysis and tuning.

With the methods described in this document, you can analyze user response time and database time from a global and top-down perspective, to confirm whether the bottleneck in user response time is caused by database issues. If the bottleneck is in the database, you can use the database time overview and SQL latency breakdowns to identify the bottleneck and tune performance.

11.1.2.1 Performance tuning based on database time

TiDB is constantly measuring and collecting SQL processing paths and database time. Therefore, it is easy to identify database performance bottlenecks in TiDB. Based on database time metrics, you can achieve the following two goals even without data on user response time:

- Determine whether the bottleneck is in TiDB by comparing the average SQL processing latency with the idle time of a TiDB connection in a transaction.
- If the bottleneck is in TiDB, further identify the exact module in the distributed system based on database time overview, color-based performance data, key metrics, resource utilization, and top-down latency breakdowns.

11.1.2.1.1 Is TiDB the bottleneck?

- If the average idle time of TiDB connections in transactions is higher than the average SQL processing latency, the database is not to blame for the transaction latency of applications. The database time takes only a small part of the user response time, indicating that the bottleneck is outside the database.

  In this case, check the external components of the database. For example, determine whether there are sufficient hardware resources in the application server, and whether the network latency from the application to the database is excessively high.

- If the average SQL processing latency is higher than the average idle time of TiDB connections in transactions, the bottleneck in transactions is in TiDB, and the database time takes a large percentage of the user response time.

11.1.2.1.2 If the bottleneck is in TiDB, how to identify it?

The following figure shows a typical SQL process. You can see that most SQL processing paths are covered in TiDB performance metrics. The database time is broken down into different dimensions, which are colored accordingly. You can quickly understand the workload characteristics and catch the bottlenecks inside the database if any.

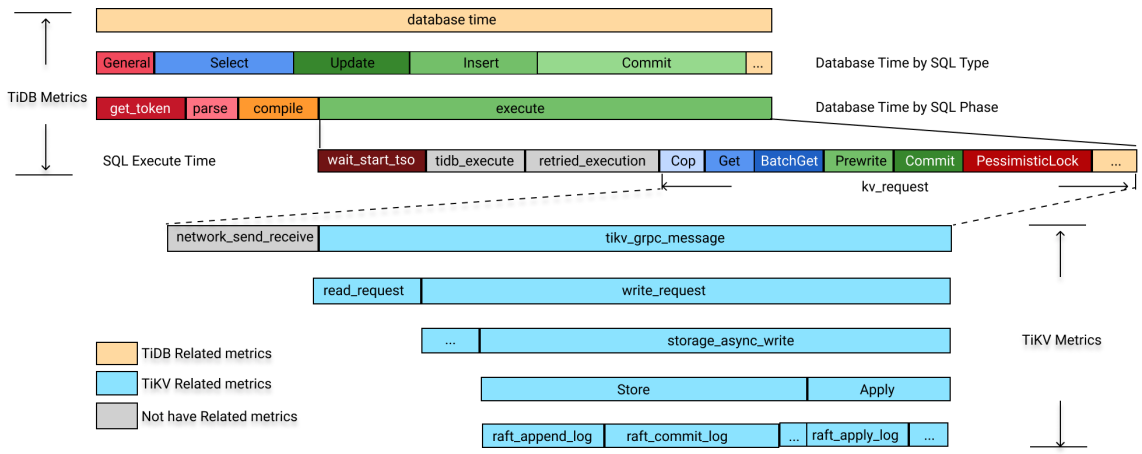


Figure 118: database time decomposition chart

Database time is the sum of all SQL processing time. A breakdown of the database time into the following three dimensions helps you quickly identify bottlenecks in TiDB:

- By SQL processing type: Determine which type of SQL statements consumes the most database time. The formula is:

```
DB Time = Select Time + Insert Time + Update Time + Delete Time +
↪ Commit Time + ...
```

- By the 4 steps of SQL processing (get_token/parse/compile/execute): Determine which step consumes the most time. The formula is:

  ```
  DB Time = Get Token Time + Parse Time + Compile Time + Execute Time
  ```

- By executor time, TSO wait time, KV request time, and execution retry time: Determine which execution step constitutes the bottleneck. The formula is:

  ```
  Execute Time ~= TiDB Executor Time + KV Request Time + PD TSO Wait Time
  ↪  + Retried execution time
  ```

11.1.2.2 Performance analysis and tuning using the Performance Overview dashboard

This section describes how to perform performance analysis and tuning based on database time using the Performance Overview dashboard in Grafana.

The Performance Overview dashboard orchestrates the metrics of TiDB, PD, and TiKV, and presents each of them in the following sections:

- Database time and SQL execution time overview: Color-coded SQL types, database time by SQL execution phase, and database time of different requests help you quickly identify database workload characteristics and performance bottlenecks.
- Key metrics and resource utilization: Contains database QPS, connection information, request command types between the applications and the database, database internal TSO and KV request OPS, and TiDB/TiKV resource usage.
- Top-down latency breakdown: Contains a comparison of query latency and connection idle time, breakdown of query latency, latency of TSO requests and KV requests in SQL execution, and breakdown of TiKV internal write latency.

11.1.2.2.1 Database time and SQL execution time overview

The database time metric is the sum of the latency that TiDB processes SQL per second, which is also the total time that TiDB concurrently processes application SQL requests per second (equal to the number of active connections).

The Performance Overview dashboard provides the following three stacked area graphs. They help you understand database workload profile and quickly identify the bottleneck causes in terms of statements, sql phase, and TiKV or PD request type during SQL execution.

- Database Time By SQL Type
- Database Time By SQL Phase
- SQL Execute Time Overview

Tune by color

The diagrams of database time breakdown and execution time overview present both expected and unexpected time consumption intuitively. Therefore, you can quickly identify performance bottleneck and learn the workload profile. Green and blue areas stand for normal time consumption and requests. If non-green or non-blue areas occupy a significant proportion in these two diagrams, the database time distribution is inappropriate.

- Database Time By SQL Type:
    - Blue: `Select` statement
    - Green: `Update`, `Insert`, `Commit` and other DML statements
    - Red: General SQL types, including `StmtPrepare`, `StmtReset`, `StmtFetch`, and `StmtClose`
- Database Time By SQL Phase: The SQL execution phase is in green and other phases are in red on general. If non-green areas are large, it means much database time is consumed in other phases than the execution phase and further cause analysis is required. A common scenario is that the compile phase shown in orange takes a large area due to unavailability of prepared plan cache.
- SQL Execute Time Overview: Green metrics stand for common KV write requests (such as `Prewrite` and `Commit`), blue metrics stand for common KV read requests (such as Cop and Get), and metrics in other colors stand for unexpected situations which you need to pay attention. For example, pessimistic lock KV requests are marked red and TSO waiting is marked dark brown. If non-blue or non-green areas are large, it means there is bottleneck during SQL execution. For example:
    - If serious lock conflicts occur, the red area will take a large proportion.
    - If excessive time is consumed in waiting TSO, the dark brown area will take a large proportion.

Example 1: TPC-C workload

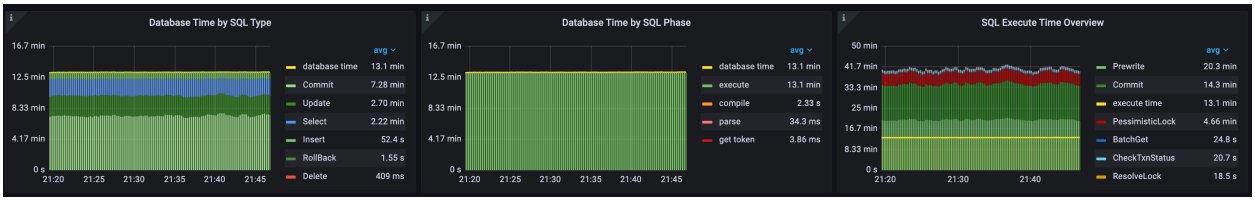


Figure 119: TPC-C

- Database Time by SQL Type: Most time-consuming statements are `commit`, `update`, `select`, and `insert` statements.

- Database Time by SQL Phase: The most time-consuming phase is SQL execution in green.
- SQL Execute Time Overview: The most time-consuming KV requests in SQL execution are `Prewrite` and `Commit` in green.

> **Note:**
>
> It is normal that the total KV request time is greater than the execute time. Because the TiDB executor may send KV requests to multiple TiKVs concurrently, causing the total KV request wait time to be greater than the execute time. In the preceding TPC-C workload, TiDB sends `Prewrite` and `Commit` requests concurrently to multiple TiKVs when a transaction is committed. Therefore, the total time for `Prewrite`, `Commit` , and `PessimisticsLock` requests in this example is obviously longer than the execute time.
>
> - The `execute` time may also be significantly greater than the total time of the KV request plus the `tso_wait` time. This means that the SQL execution time is spent mostly inside the TiDB executor. Here are two common examples:

```
> - Example 1: After TiDB executor reads a large amount of data from
    ↪ TiKV, it needs to do complex join and aggregation inside TiDB,
    ↪ which consumes a lot of time.
> - Example 2: The application experiences serious write statement lock
    ↪  conflicts. Frequent lock retries result in long `Retried
    ↪ execution time`.
```

Example 2: OLTP read-heavy workload

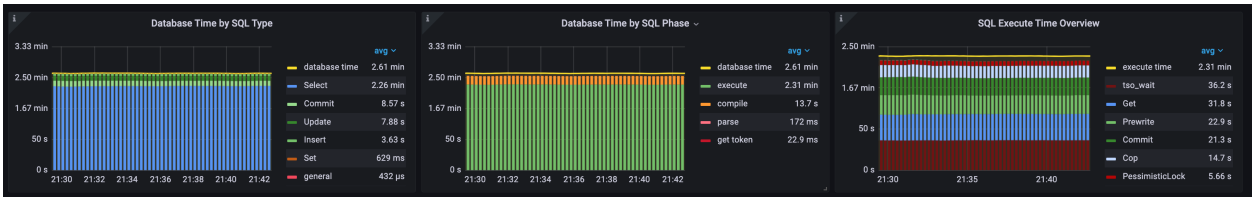


Figure 120: OLTP

- Database Time by SQL Type: Major time-consuming statements are `SELECT`, `COMMIT`, `UPDATE`, and `INSERT`, among which `SELECT` consumes most database time.
- Database Time by SQL Phase: Most time is consumed in the `execute` phase in green.

- SQL Execute Time Overview: In SQL execution phase, `pd tso_wait` in dark brown, `KV Get` in blue, and `Prewrite` and `Commit` in green are time-consuming.

Example 3: Read-only OLTP workload

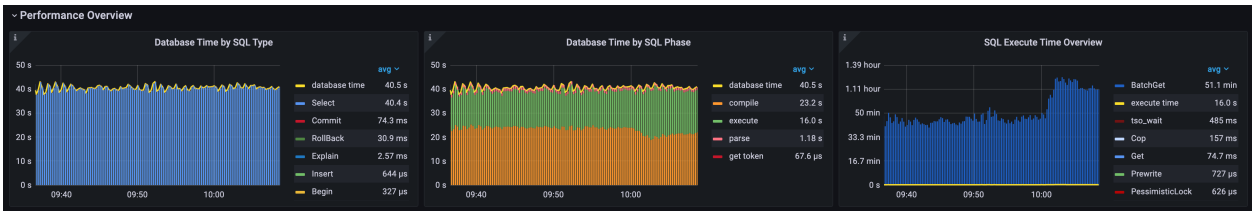


Figure 121: OLTP

- Database Time by SQL Type: Mainly are `SELECT` statements.
- Database Time by SQL Phase: Major time-consuming phases are `compile` in orange and `execute` in green. Latency in the `compile` phase is the highest, indicating that TiDB is taking too long to generate execution plans and the root cause needs to be further determined based on the subsequent performance data.
- SQL Execute Time Overview: The KV BatchGet requests in blue consume the most time during SQL execution.

> **Note:**
>
> In example 3, `SELECT` statements need to read thousands of rows concurrently from multiple TiKVs. Therefore, the total time of the `BatchGet` request is much longer than the execution time.

Example 4: Lock contention workload

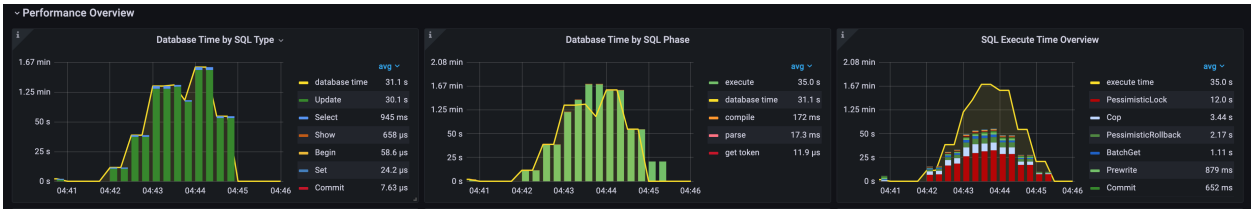


Figure 122: OLTP

- Database Time by SQL Type: Mainly are `UPDATE` statements.

- Database Time by SQL Phase: Most time is consumed in the execute phase in green.
- SQL Execute Time Overview: The KV request PessimisticLock shown in red consumes the most time during SQL execution, and the execution time is obviously longer than the total time of KV requests. This is caused by serious lock conflicts in write statements and frequent lock retries prolong `Retried execution time`. Currently, TiDB does not measure `Retried execution time`.

11.1.2.2.2 TiDB key metrics and cluster resource utilization

Query Per Second, Command Per Second, and Prepared-Plan-Cache

By checking the following three panels in Performance Overview, you can learn the application workload type, how the application interacts with TiDB, and whether the application fully utilizes TiDB prepared plan cache.

- QPS: Short for Query Per Second. It shows the count of SQL statements executed by the application.
- CPS By Type: Short for Command Per Second. Command indicates MySQL protocol-specific commands. A query statement can be sent to TiDB either by a query command or a prepared statement.
- Queries Using Plan Cache OPS: The count that the TiDB cluster hits the prepared plan cache per second. prepared plan cache only supports the `prepared statement` command. When prepared plan cache is enabled in TiDB, the following three scenarios will occur:
    - No prepared plan cache is hit: The number of plan cache hit per second is 0. The application is using the query interface, or cached plans are cleaned up by calling the StmtClose command after each StmtExecute execution.
    - All prepared plan cache is hit: The number of hits per second is equal to the number of StmtExecute commands per second.
    - Some prepared plan cache is hit: The number of hits per second is fewer than the number of StmtExecute commands per second. Prepared plan cache has known limitations, for example, it does not support subqueries, SQL statements with subqueries cannot utilize prepared plan cache.

Example 1: TPC-C workload

The TPC-C workload are mainly `UPDATE`, `SELECT`, and `INSERT` statements. The total QPS is equal to the number of StmtExecute per second and the latter is almost equal to Queries Using Plan Cache OPS. Ideally, the client caches the object of the prepared statement. In this way, the cached statement is called directly when a SQL statement is executed. All SQL executions hit the prepared plan cache, and there is no need to recompile to generate execution plans.

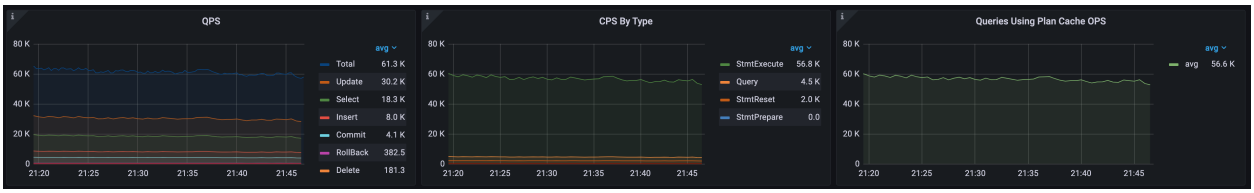


Figure 123: TPC-C

Example 2: Prepared plan cache unavailable for query commands in read-only OLTP workload

In this workload, `Commit QPS` = `Rollback QPS` = `Select QPS`. The application has enabled auto-commit concurrency, and rollback is performed every time a connection is fetched from the connection pool. As a result, these three statements are executed the same number of times.

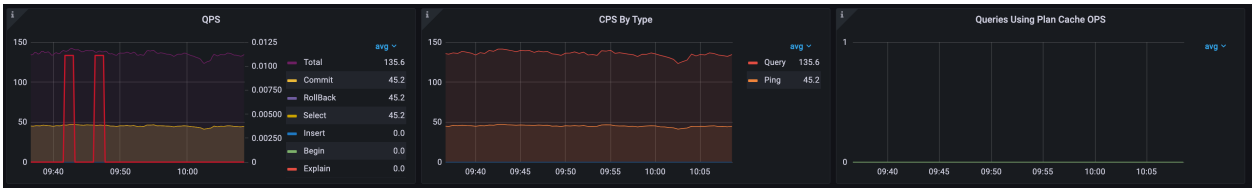


Figure 124: OLTP-Query

- The red bold line in the QPS panel stands for failed queries, and the Y-axis on the right shows the number of failed queries. A value other than 0 means the presence of failed queries.
- The total QPS is equal to the number of queries in the CPS By Type panel, the query command has been used by the application.
- The Queries Using Plan Cache OPS panel has no data, because prepared plan cache is unavailable for query command. This means that TiDB needs to parse and generate an execution plan for every query execution. As a result, the compile time is longer with increasing CPU consumption by TiDB.

Example 3: Prepared plan cache unavailable with prepared statement enabled for OLTP workload

`StmtPreare` times = `StmtExecute` times = `StmtClose` times ~= `StmtFetch` times. The application uses the prepare > execute > fetch > close loop. To prevent prepared statement object leak, many application frameworks call `close` after the `execute` phase. This creates two problems.

- A SQL execution requires four commands and four network round trips.

- Queries Using Plan Cache OPS is 0, indicating zero hit of prepared plan cache. The `StmtClose` command clears cached execution plans by default and the next `StmtPreare` command needs to generate the execution plan again.

> **Note:**
>
> Starting from TiDB v6.0.0, you can prevent the `StmtClose` command from clearing cached execution plans via the global variable (`set global tidb_ignore_prepared_cache_close_stmt=on;`). In this way, subsequent executions can hit the prepared plan cache.

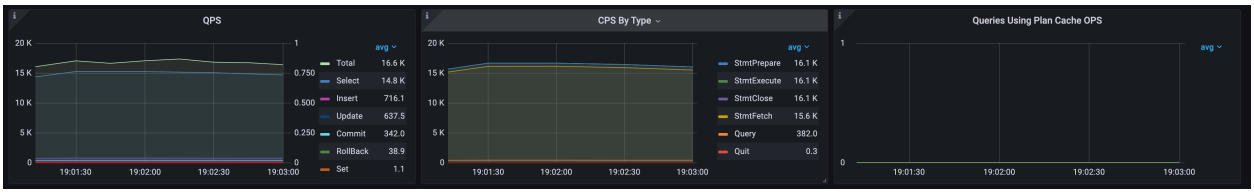


Figure 125: OLTP-Prepared

KV/TSO Request OPS and connection information

In the KV/TSO Request OPS panel, you can view the statistics of KV and TSO requests per second. Among the statistics, `kv request total` represents the sum of all requests from TiDB to TiKV. By observing the types of requests from TiDB to PD and TiKV, you can get an idea of the workload profile within the cluster.

In the Connection Count panel, you can view the total number of connections and the number of connections per TiDB. The counts help you determine whether the total number of connections is normal and the number of connections per TiDB is even. `active connections` records the number of active connections, which is equal to the database time per second.

Example 1: Busy workload



Figure 126: TPC-C

In this TPC-C workload:

- The total number of KV requests per second is 104,200. The top request types are `PessimisticsLock`, `Prewrite`, `Commit` and `BatchGet` in order of number of requests.
- The total number of connections is 810, which are evenly distributed in three TiDB instances. The number of active connections is 787.1. Therefore, 97% of the connections are active, indicating that the database is the bottleneck for this system.

Example 2: Idle workload



Figure 127: OLTP

In this workload:

- The total number of KV requests per second is 2600 and the number of TSO requests per second is 1100.
- The total number of connections is 410, which are evenly distributed in three TiDB instances. The number of active connections is only 2.5, indicating that the database system is relatively idle.

TiDB CPU, TiKV CPU, and IO usage

In the TiDB CPU and TiKV CPU/IO MBps panels, you can observe the logical CPU usage and IO throughput of TiDB and TiKV, including average, maximum, and delta (maximum CPU usage minus minimum CPU usage), based on which you can determine the overall CPU usage of TiDB and TiKV.

- Based on the `delta` value, you can determine if CPU usage in TiDB is unbalanced (usually accompanied by unbalanced application connections) and if there are read/write hot spots among the cluster.
- With an overview of TiDB and TiKV resource usage, you can quickly determine if there are resource bottlenecks in your cluster and whether TiKV or TiDB needs scale-out.

Example 1: High TiDB resource usage

In this workload, each TiDB and TiKV is configured with 8 CPUs.



Figure 128: TPC-C

- The average, maximum, and delta CPU usage of TiDB are 575%, 643%, and 136%, respectively.
- The average, maximum, and delta CPU usage of TiKV are 146%, 215%, and 118%, respectively. The average, maximum, and delta I/O throughput of TiKV are 9.06 MB/s, 19.7 MB/s, and 17.1 MB/s, respectively.

Obviously, TiDB consumes more CPU, which is near the bottleneck threshold of 8 CPUs. It is recommended that you scale out the TiDB.

Example 2: High TiKV resource usage

In the TPC-C workload below, each TiDB and TiKV is configured with 16 CPUs.

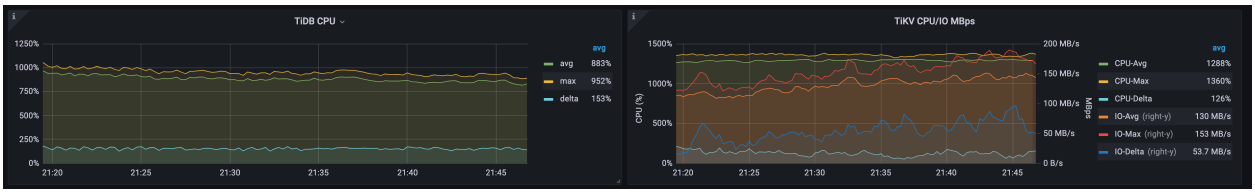


Figure 129: TPC-C

- The average, maximum, and delta CPU usage of TiDB are 883%, 962%, and 153%, respectively.
- The average, maximum, and delta CPU usage of TiKV are 1288%, 1360%, and 126%, respectively. The average, maximum, and delta I/O throughput of TiKV are 130 MB/s, 153 MB/s, and 53.7 MB/s, respectively.

Obviously, TiKV consumes more CPU, which is expected because TPC-C is a write-heavy scenario. It is recommended that you scale out the TiKV to improve performance.

11.1.2.2.3 Query latency breakdown and key latency metrics

The latency panel provides average values and 99th percentile. The average values help identify the overall bottleneck, while the 99th or 999th percentile or 999th helps determine whether there is a significant latency jitter.

Duration and Connection Idle Duration

The Duration panel contains the average and P99 latency of all statements, and the average latency of each SQL type. The Connection Idle Duration panel contains the average and the P99 connection idle duration. Connection idle duration includes the following two states:

- in-txn: The interval between processing the previous SQL and receiving the next SQL statement when the connection is within a transaction.
- not-in-txn: The interval between processing the previous SQL and receiving the next SQL statement when the connection is not within a transaction.

An applications perform transactions with the same database connction. By comparing the average query latency with the connection idle duration, you can determine if TiDB is the bottleneck for overall system, or if user response time jitter is caused by TiDB.

- If the application workload is not read-only and contains transactions, by comparing the average query latency with `avg-in-txn`, you can determine the proportion in processing transactions inside and outside the database, and identify the bottleneck in user response time.
- If the application workload is read-only or autocommit mode is on, you can compare the average query latency with `avg-not-in-txn`.

In real customer scenarios, it is not rare that the bottleneck is outside the database, for example:

- The client server configuration is too low and the CPU resources are exhausted.
- HAProxy is used as a TiDB cluster proxy, and the HAProxy CPU resource is exhausted.
- HAProxy is used as a TiDB cluster proxy, and the network bandwidth of the HAProxy server is used up under high workload.
- The network latency from the application server to the database is high. For example, the network latency is high because in public-cloud deployments the applications and the TiDB cluster are not in the same region, or the dns workload balancer and the TiDB cluster are not in the same region.
- The bottleneck is in client applications. The application server's CPU cores and Numa resources cannot be fully utilized. For example, only one JVM is used to establish thousands of JDBC connections to TiDB.

Example 1: TiDB is the bottleneck of user response time



Figure 130: TiDB is the Bottleneck

In this TPC-C workload:

- The average latency and P99 latency of all SQL statements are 477 us and 3.13 ms, respectively. The average latencies of the commit statement, insert statement, and query statement are 2.02 ms, 609 us, and 468 us, respectively.
- The average connection idle time in transactions `avg-in-txn` is 171 us.

The average query latency is significantly greater than `avg-in-txn`, which means the main bottleneck in transactions is inside the database.

Example 2: The bottleneck of user response time is not in TiDB

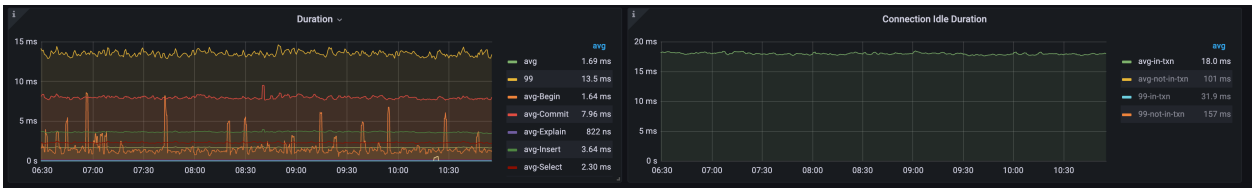


Figure 131: TiDB is the Bottleneck

In this workload, the average query latency is 1.69 ms and `avg-in-txn` is 18 ms, indicating that TiDB spends 1.69 ms on average to process a SQL statement in transactions, and then needs to wait for 18 ms to receive the next statement.

The average query latency is significantly lower than `avg-in-txn`. The bottleneck of user response time is not in TiDB. This example is in a public cloud environment, where high network latency between the application and the database results in extremely high connection idle time, because the application and the database are not in the same region.

Parse, Compile, and Execute Duration

In TiDB, there is a typical processing flow from sending query statements to returning results.

SQL processing in TiDB consists of four phases, `get token`, `parse`, `compile`, and `execute`.

- `get token`: Usually only a few microseconds and can be ignored. The token is limited only when the number of connections to a single TiDB instance reaches the token-limit limit.
- `parse`: The query statements are parsed into abstract syntax tree (AST).
- `compile`: Execution plans are compiled based on the AST from the `parse` phase and statistics. The `compile` phase contains logical optimization and physical optimization. Logical optimization optimizes query plans by rules, such as column pruning based on relational algebra. Physical optimization estimates the cost of the execution plans by statistics by a cost-based optimizer and selects a physical execution plan with the lowest cost.
- `execute`: The time consumption to execute a SQL statement. TiDB first waits for the globally unique timestamp TSO. Then the executor constructs the TiKV API request based on the Key range of the operator in the execution plan and distributes it to TiKV. `execute` time includes the TSO wait time, the KV request time, and the time spent by TiDB executor in processing data.

If an application uses the `query` or `StmtExecute` MySQL command interface only, you can use the following formula to identify the bottleneck in average latency.

```
avg Query Duration = avg Get Token + avg Parse Duration + avg Compile
    ↪ Duration + avg Execute Duration
```

Usually, the `execute` phase accounts for the most of the `query` latency. However, the `parse` and `compile` phases can also take a large part in the following cases:

- Long latency in the `parse` phase: For example, when the `query` statement is long, much CPU will be consumed to parse the SQL text.
- Long latency in the `compile` phase: If the prepared plan cache is not hit, TiDB needs to compile an execution plan for every SQL execution. The latency in the `compile` phase can be several or tens of milliseconds or even higher. If prepared plan cache is not hit, logical and physical optimization are done in the `compile` phase, which consumes a lot of CPU and memory, makes Go Runtime (TiDB is written in Go) under pressure, and affects the performance of other TiDB components. Prepared plan cache is important for efficient processing of OLTP workload in TiDB.

Example 1: Database bottleneck in the `compile` phase

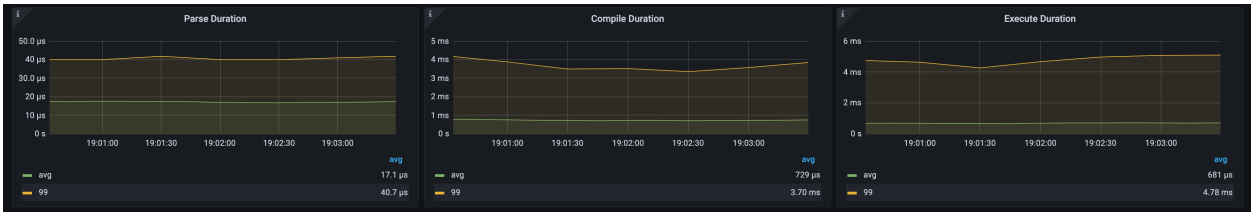


Figure 132: Compile

In the preceding figure, the average time of the `parse`, `compile`, and `execute` phases are 17.1 us, 729 us, and 681 us, respectively. The `compile` latency is high because the application uses the `query` command interface and cannot use prepared plan cache.

Example 2: Database bottleneck in the `execute` phase

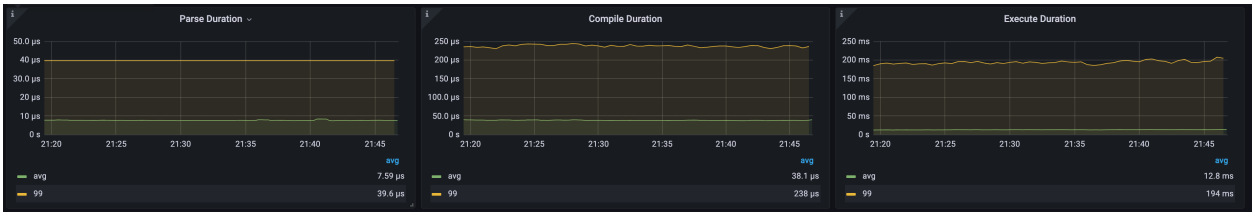


Figure 133: Execute

In this TPC-C workload, the average time of `parse`, `compile` and `execute` phases are 7.39 us, 38.1 us, and 12.8 ms, respectively. The `execute` phase is the bottleneck of the `query` latency.

KV and TSO Request Duration

TiDB interacts with PD and TiKV in the `execute` phase. As shown in the following figure, when processing SQL request, TiDB requests TSOs before entering the `parse` and `compile` phases. The PD Client does not block the caller, but returns a `TSFuture` and asynchronously sends and receives the TSO requests in the background. Once the PD client finishes handling the TSO requests, it returns `TSFuture`. The holder of the `TSFuture` needs to call the Wait method to get the final TSOs. After TiDB finishes the `parse` and `compile` phases, it enters the `execute` phase, where two situations might occur:

- If the TSO request has completed, the Wait method immediately returns an available TSO or an error
- If the TSO request has not yet completed, the Wait method is blocked until a TSO is available or an error appears (the gRPC request has been sent but no result is returned, and the network latency is high)

The TSO wait time is recorded as `TSO WAIT` and the network time of the TSO request is recorded as `TSO RPC`. After the TSO wait is complete, TiDB executor usually sends read or write requests to TiKV.

- Common KV read requests: `Get`, `BatchGet`, and `Cop`
- Common KV write requests: `PessimisticLock`, `Prewrite` and `Commit` for two-phase commits

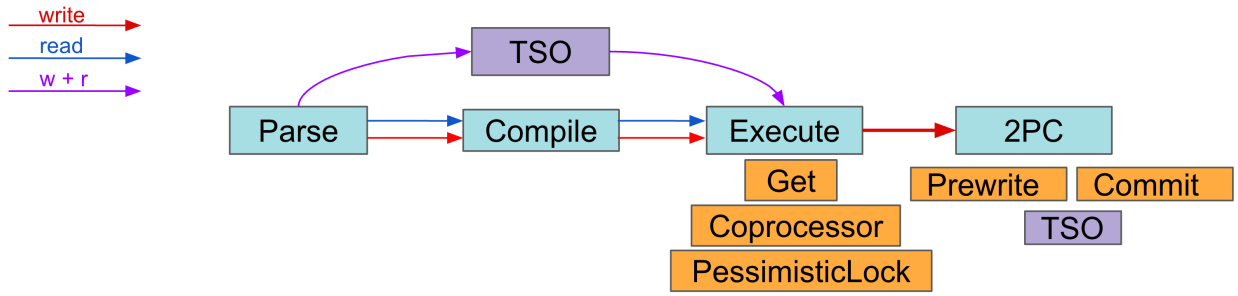


Figure 134: Execute

The indicators in this section correspond to the following three panels.

- Avg TiDB KV Request Duration: The average latency of KV requests measured by TiDB
- Avg TiKV GRPC Duration: The average latency in processing gPRC messages in TiKV
- PD TSO Wait/RPC Duration: TiDB executor TSO wait time and network latency for TSO requests (RPC)

The relationship between `Avg TiDB KV Request Duration` and `Avg TiKV GRPC Duration` is as follows:

```
Avg TiDB KV Request Duration = Avg TiKV GRPC Duration + Network latency between TiDB and TiKV + TiKV gRPC processing time + TiDB gRPC processing time and scheduling latency
```

The difference between `Avg TiDB KV Request Duration` and `Avg TiKV GRPC Duration` is closely related to the network traffic, network latency, and resource usage by TiDB and TiKV.

- In the same data center: The difference is generally less than 2 ms.
- In different availability zones in the same region: The difference is generally less than 5 ms.

Example 1: Low workload of clusters deployed on the same data center

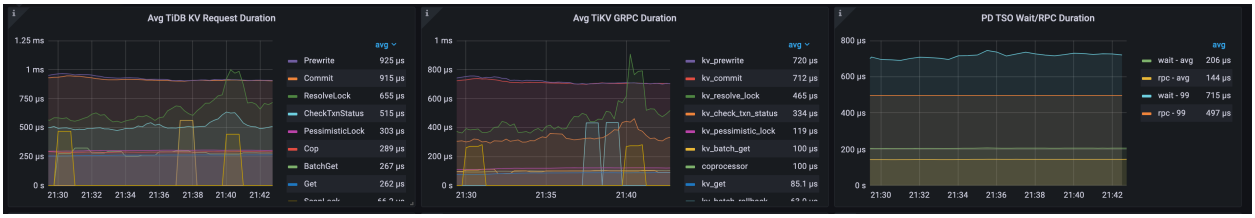


Figure 135: Same Data Center

In this workload, the average `Prewrite` latency on TiDB is 925 us, and the average `kv_prewrite` processing latency inside TiKV is 720 us. The difference is about 200 us, which is normal in the same data center. The average TSO wait latency is 206 us, and the RPC time is 144 us.

Example 2: Normal workload on public cloud clusters

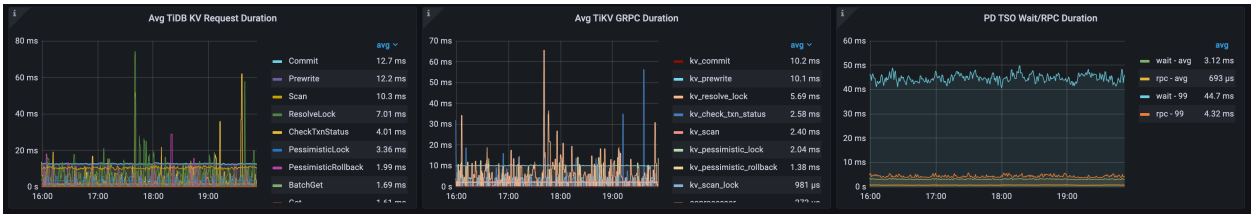


Figure 136: Cloud Env

In this example, TiDB clusters are deployed in different data centers in the same region. The average `commit` latency on TiDB is 12.7 ms, and the average `kv_commit` processing latency inside TiKV is 10.2 ms, a difference of about 2.5 ms. The average TSO wait latency is 3.12 ms, and the RPC time is 693 us.

Example 3: Resource overloaded on public cloud clusters

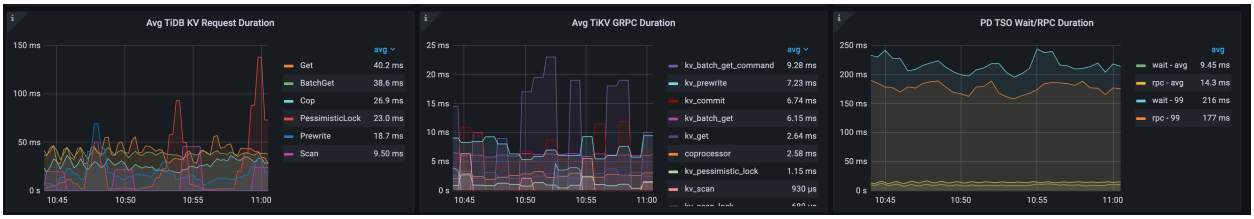


Figure 137: Cloud Env, TiDB Overloaded

In this example, the TiDB clusters are deployed in different data centers in the same region, and TiDB network and CPU resources are severely overloaded. The average `BatchGet` latency on TiDB is 38.6 ms, and the average `kv_batch_get` processing latency inside TiKV is 6.15 ms. The difference is more than 32 ms, which is much higher than the normal value. The average TSO wait latency is 9.45 ms and the RPC time is 14.3 ms.

Storage Async Write Duration, Store Duration, and Apply Duration

TiKV processes a write request in the following procedure:

- `scheduler worker` processes the write request, performs a transaction consistency check, and converts the write request into a key-value pair to be sent to the `raftstore` module.

- The TiKV consensus module `raftstore` applies the Raft consensus algorithm to make the storage layer (composed of multiple TiKVs) fault-tolerant.

  Raftstore consists of a `Store` thread and an `Apply` thread:

  - The `Store` thread processes Raft messages and new `proposals`. When a new `proposals` is received, the `Store` thread of the leader node writes to the local Raft DB and copies the message to multiple follower nodes. When this `proposals` is successfully persisted in most instances, the `proposals` is successfully committed.
  - The `Apply` thread writes the committed `proposals` to the KV DB. When the content is successfully written to the KV DB, the `Apply` thread notifies externally that the write request has completed.

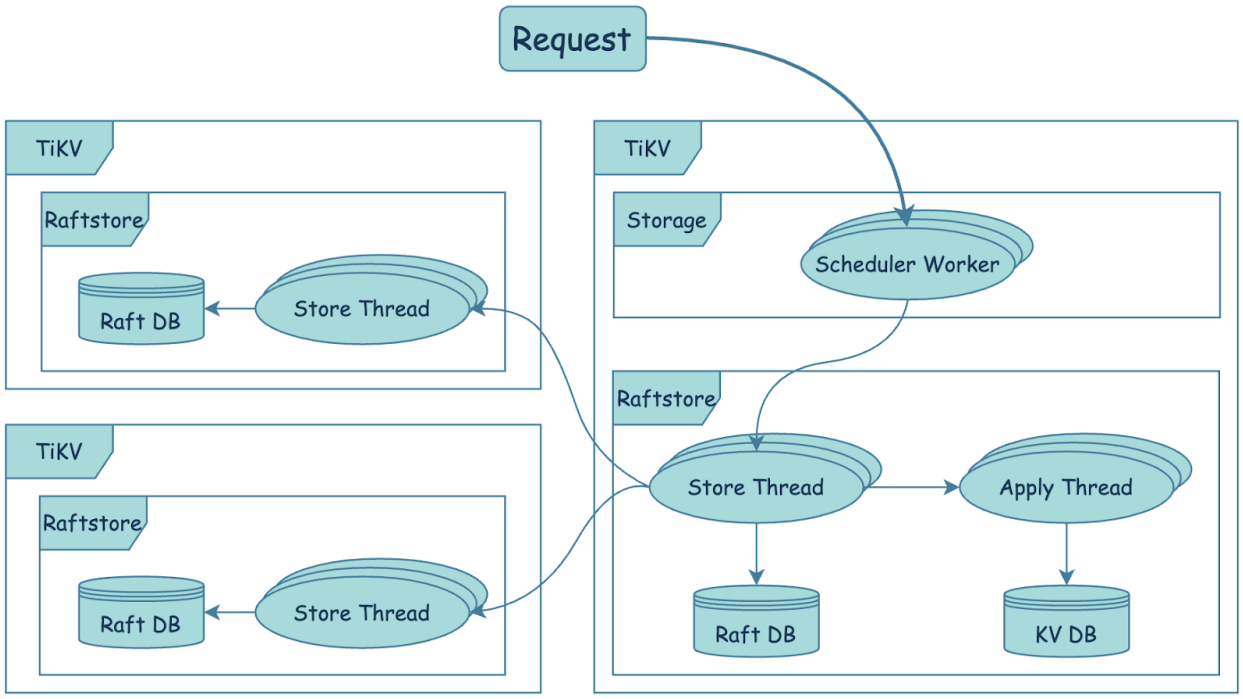


Figure 138: TiKV Write

The `Storage Async Write Duration` metric records the latency after a write request enters raftstore. The data is collected on a basis of per request.

The `Storage Async Write Duration` metric contains two parts, `Store Duration` and `Apply Duration`. You can use the following formula to determine whether the bottleneck for write requests is in the `Store` or `Apply` step.

```
avg Storage Async Write Duration = avg Store Duration + avg Apply Duration
```

Note:

`Store Duration` and `Apply Duration` are supported since v5.3.0.

Example 1: Comparison of the same OLTP workload in v5.3.0 and v5.4.0

According to the preceding formula, the QPS of a write-heavy OLTP workload in v5.4.0 is 14% higher than that in v5.3.0:

- v5.3.0: 24.4 ms ~= 17.7 ms + 6.59 ms
- v5.4.0: 21.4 ms ~= 14.0 ms + 7.33 ms

In v5.4.0, the gPRC module has been optimized to accelerate Raft log replication, which reduces `Store Duration` compared with v5.3.0.

v5.3.0:

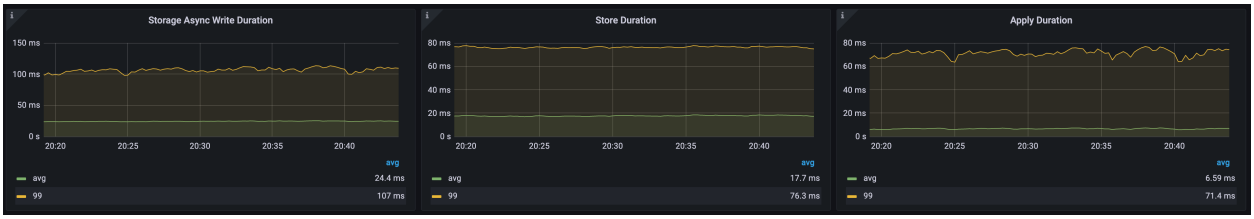


Figure 139: v5.3.0

v5.4.0:

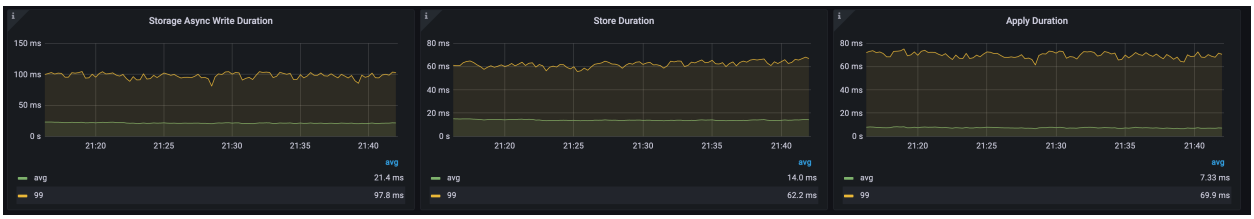


Figure 140: v5.4.0

Example 2: Store Duration is a bottleneck

Apply the preceding formula: 10.1 ms ~= 9.81 ms + 0.304 ms. The result indicates that the latency bottleneck for write requests is in `Store Duration`.



Figure 141: Store

Commit Log Duration, Append Log Duration, and Apply Log Duration

`Commit Log Duration`, `Append Log Duration`, and `Apply Log Duration` are latency metrics for key operations within raftstore. These latencies are captured at the batch operation level, with each operation combining multiple write requests. Therefore, the latencies do not directly correspond to the `Store Duration` and `Apply Duration` mentioned above.

- `Commit Log Duration` and `Append Log Duration` record time of operations performed in the `Store` thread. `Commit Log Duration` includes the time of copying Raft logs to other TiKV nodes (to ensure raft-log persistence). `Commit Log Duration` usually contains two `Append Log Duration` operations, one for the leader and the other for the follower. `Commit Log Duration` is usually significantly higher than `Append Log Duration`, because the former includes the time of copying Raft logs to other TiKV nodes through network.
- `Apply Log Duration` records the latency of `apply` Raft logs by the `Apply` thread.

Common scenarios where `Commit Log Duration` is long:

- There is a bottleneck in TiKV CPU resources and the scheduling latency is high
- `raftstore.store-pool-size` is either excessively small or large (an excessively large value might also cause performance degradation)
- The I/O latency is high, resulting in high `Append Log Duration` latency
- The network latency between TiKV nodes is high
- The number of the gRPC threads are too small, CPU usage is uneven among the GRPC threads.

Common scenarios where `Apply Log Duration` is long:

- There is a bottleneck in TiKV CPU resources and the scheduling latency is high
- `raftstore.apply-pool-size` is either excessively small or large (an excessively large value might also cause performance degradation)
- The I/O latency is high

Example 1: Comparison of the same OLTP workload in v5.3.0 and v5.4.0

The QPS of a write-heavy OLTP workload in v5.4.0 is improved by 14% compared with that in v5.3.0. The following table compares the three key latencies.

Avg Duration	v5.3.0 (ms)	v5.4.0 (ms)
Append Log Duration	0.27	0.303
Commit Log Duration	13	8.68
Apply Log Duration	0.457	0.514

In v5.4.0, the gPRC module has been optimized to accelerate Raft log replication, which reduces `Commit Log Duration` and `Store Duration` compared with v5.3.0.

v5.3.0:

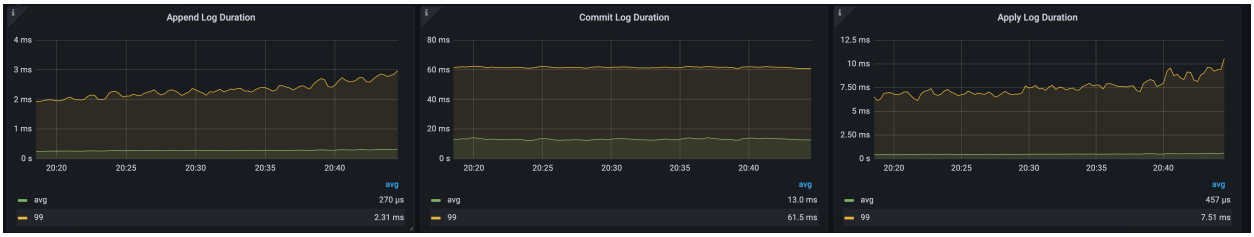


Figure 142: v5.3.0

v5.4.0:

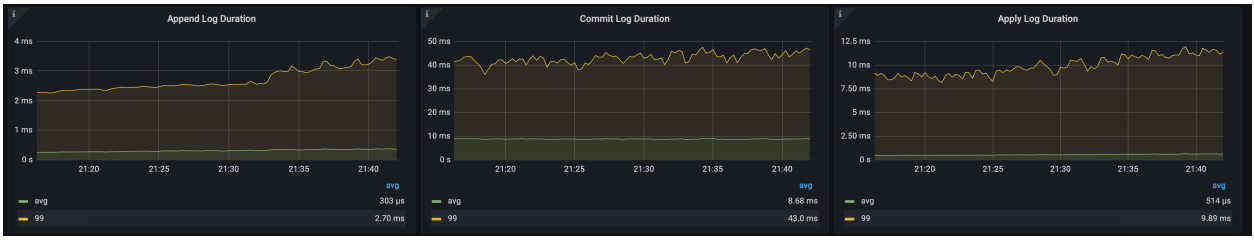


Figure 143: v5.4.0

Example 2: Commit Log Duration is a bottleneck

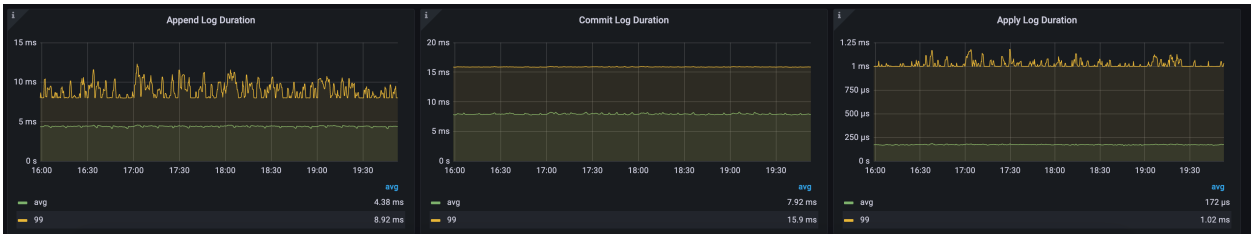


Figure 144: Store

- Average `Append Log Duration` = 4.38 ms
- Average `Commit Log Duration` = 7.92 ms
- Average `Apply Log Duration` = 172 us

For the `Store` thread, `Commit Log Duration` is obviously higher than `Apply Log Duration`. Meanwhile, `Append Log Duration` is significantly higher than `Apply Log Duration`, indicating that the `Store` thread might suffer from bottlenecks in both CPU and I/O. Possible ways to reduce `Commit Log Duration` and `Append Log Duration` are as follows:

- If TiKV CPU resources are sufficient, consider adding `Store` threads by increasing the value of `raftstore.store-pool-size`.
- If TiDB is v5.4.0 or later, consider enabling `Raft Engine` by setting `raft-engine.enable: true`. Raft Engine has a light execution path. This helps reduce I/O writes and long-tail latency of writes in some scenarios.
- If TiKV CPU resources are sufficient and TiDB is v5.3.0 or later, consider enabling `StoreWriter` by setting `raftstore.store-io-pool-size: 1`.

11.1.2.3 If my TiDB version is earlier than v6.1.0, what should I do to use the Performance Overview dashboard?

Starting from v6.1.0, Grafana has a built-in Performance Overview dashboard by default. This dashboard is compatible with TiDB v4.x and v5.x versions. If your TiDB is earlier than v6.1.0, you need to manually import `performance_overview.json`, as shown in the following figure:

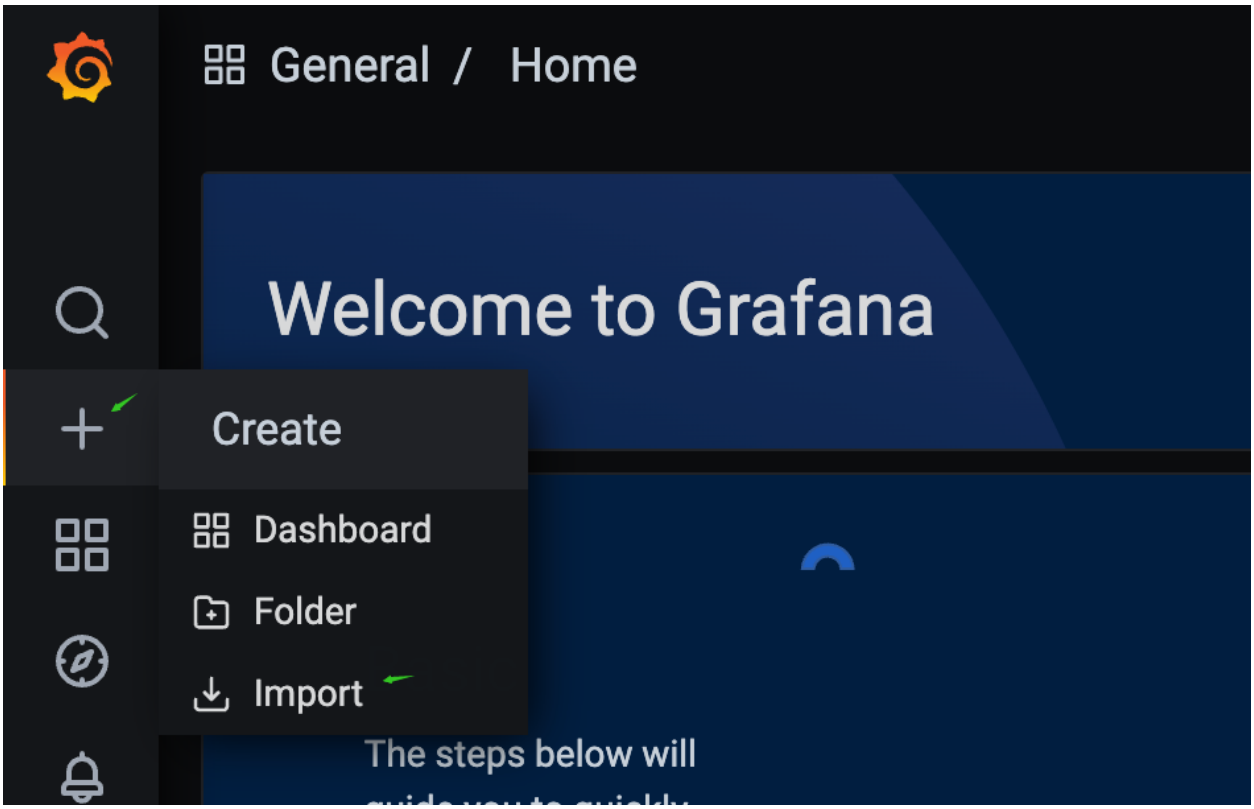


Figure 145: Store

11.1.3 Performance Tuning Practices for OLTP Scenarios

TiDB provides comprehensive performance diagnostics and analysis features, such as Top SQL and Continuous Profiling features on the TiDB Dashboard, and TiDB Performance Overview Dashboard.

This document describes how to use these features together to analyze and compare the performance of the same OLTP workload in seven different runtime scenarios, which demonstrates a performance tuning process to help you analyze and tune TiDB performance efficiently.

> **Note:**
>
> Top SQL and Continuous Profiling are not enabled by default. You need to enable them in advance.

By running the same application with different JDBC configurations in these scenarios, this document shows you how the overall system performance is affected by different

interactions between applications and databases, so that you can apply Best Practices for Developing Java Applications with TiDB for better performance.

11.1.3.1 Environment description

This document takes a core banking OLTP workload for demonstration. The configurations of the simulation environment are as follows:

- Application development language for the workload: JAVA
- SQL statements used in business: 200 statements in total, 90% of which are SELECT statements. It is a typical read-heavy OLTP workload.
- Tables used in transactions: 60 tables in total. 12 tables involve update operations, and the rest 48 tables are read-only.
- Isolation level used by the application: `read committed`.
- TiDB cluster configuration: 3 TiDB nodes and 3 TiKV nodes, with 16 CPUs allocated to each node.
- Client server configuration: 36 CPUs.

11.1.3.2 Scenario 1. Use the Query interface

11.1.3.2.1 Application configuration

The application uses the following JDBC configuration to connect to the database through the Query interface.

```
useServerPrepStmts=false
```

11.1.3.2.2 Performance analysis

TiDB Dashboard

From the Top SQL page in the TiDB Dashboard below, you can see that the non-business SQL type `SELECT @@session.tx_isolation` consumes the most resources. Although TiDB processes these types of SQL statements quickly, these types of SQL statements have the highest number of executions that result in the highest overall CPU time consumption.

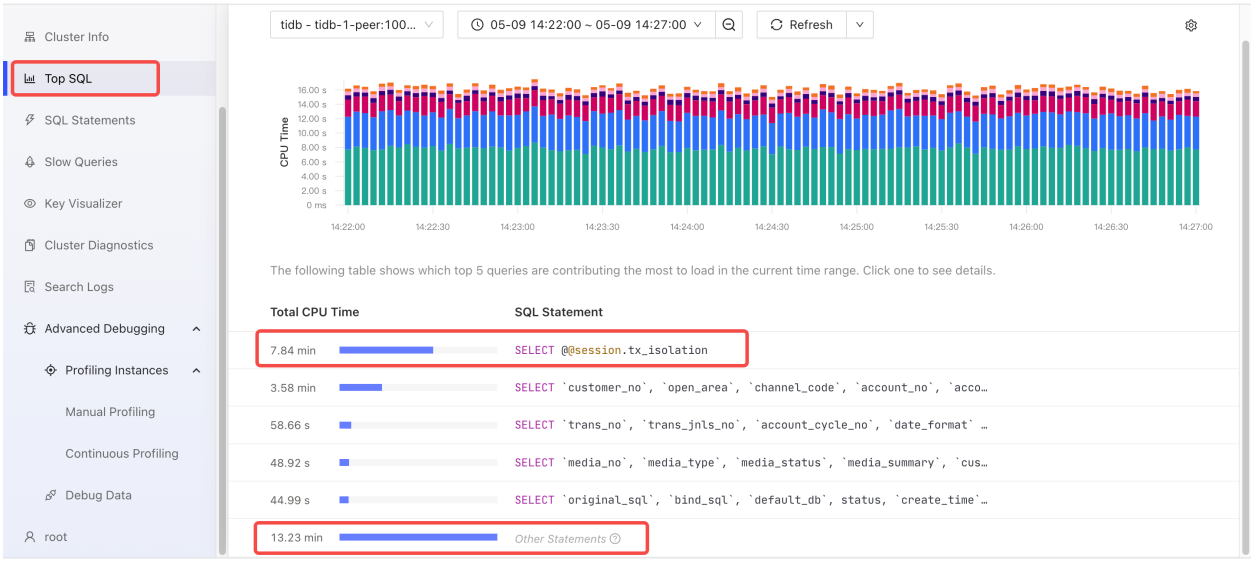


Figure 146: dashboard-for-query-interface

From the following flame chart of TiDB, you can see that the CPU consumption of functions such as `Compile` and `Optimize` is significant during the SQL execution. Because the application uses the Query interface, TiDB cannot use the execution plan cache. TiDB needs to compile and generate an execution plan for each SQL statement.

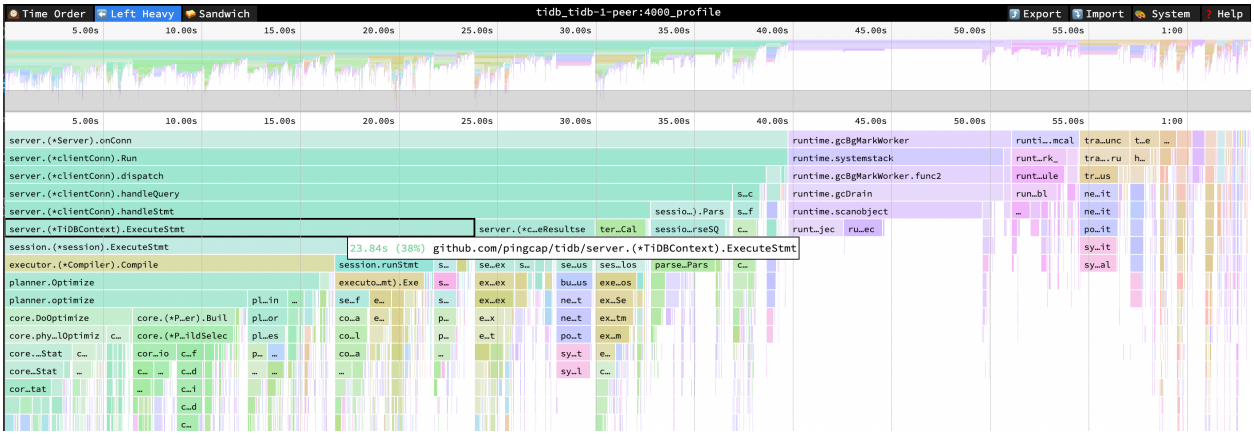


Figure 147: flame-graph-for-query-interface

- ExecuteStmt cpu = 38% cpu time = 23.84s
- Compile cpu = 27% cpu time = 17.17s
- Optimize cpu = 26% cpu time = 16.41s

Performance Overview dashboard

Check the database time overview and QPS in the following Performance Overview dashboard.

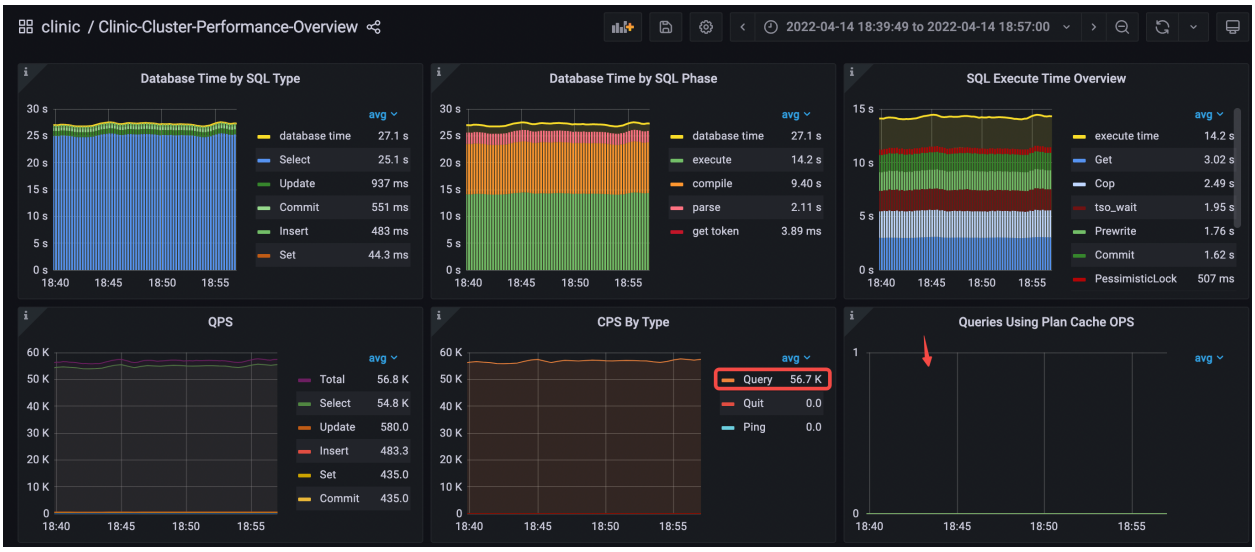


Figure 148: performance-overview-1-for-query-interface

- Database Time by SQL Type: the `Select` statement type takes most of the time.
- Database Time by SQL Phase: the `execute` and `compile` phases take most of the time.
- SQL Execute Time Overview: `Get`, `Cop`, and `tso wait` take most of the time.
- CPS By Type: only the `Query` command is used.
- Queries Using Plan Cache OPS: no data indicates that the execution plan cache is not hit.
- In the query duration, the latency of `execute` and `compile` takes the highest percentage.
- avg QPS = 56.8k

Check the resource consumption of the cluster: the average utilization of TiDB CPU is 925%, the average utilization of TiKV CPU is 201%, and the average throughput of TiKV IO is 18.7 MB/s. The resource consumption of TiDB is significantly higher.

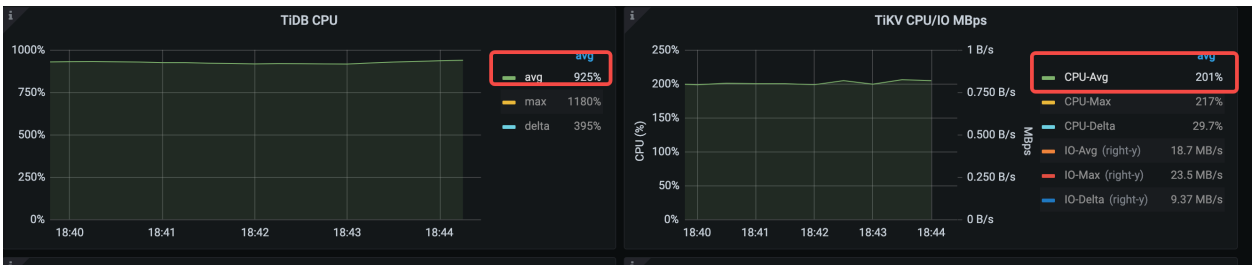


Figure 149: performance-overview-2-for-query-interface

11.1.3.2.3 Analysis conclusion

We need to eliminate these useless non-business SQL statements, which have a large number of executions and contribute to the high TiDB CPU usage.

11.1.3.3 Scenario 2. Use the maxPerformance configuration

11.1.3.3.1 Application configuration

The application adds a new parameter `useConfigs=maxPerformance` to the JDBC connection string in Scenario 1. This parameter can be used to eliminate the SQL statements sent from JDBC to the database (for example, `select @@session.transaction_read_only` ↪). The full configuration is as follows:

```
useServerPrepStmts=false&useConfigs=maxPerformance
```

11.1.3.3.2 Performance analysis

TiDB Dashboard

From the Top SQL page in the TiDB Dashboard below, you can see that `SELECT` ↪ `@@session.tx_isolation`, which consumed the most resources, has disappeared.

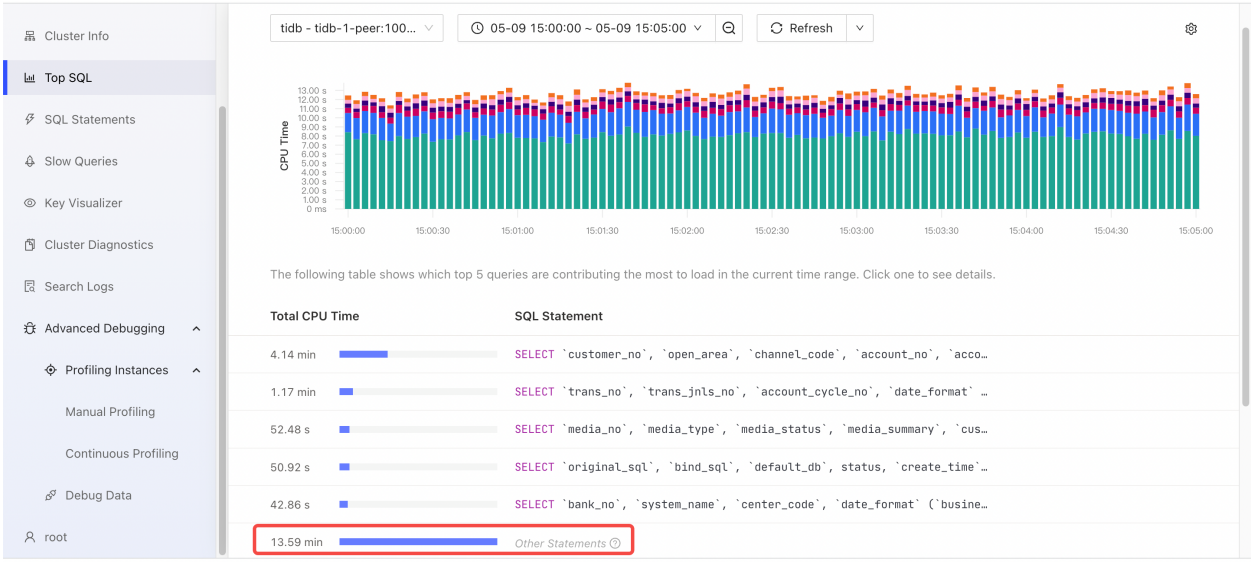


Figure 150: dashboard-for-maxPerformance

From the following flame chart of TiDB, you can see that the CPU consumption of functions such as `Compile` and `Optimize` is still significant during the SQL execution.

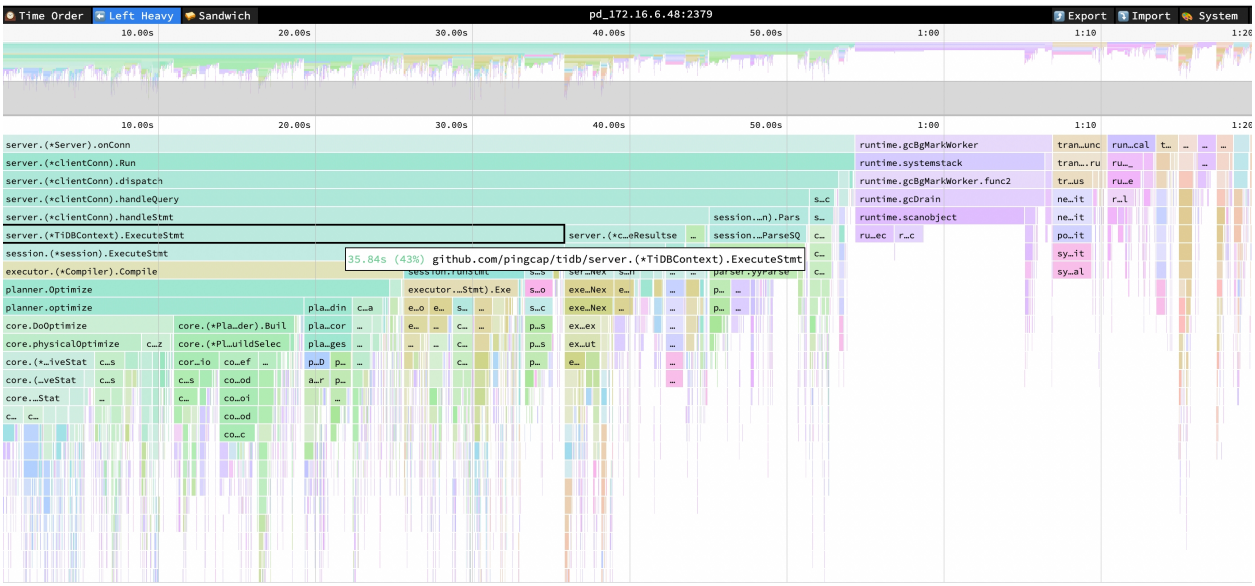


Figure 151: flame-graph-for-maxPerformance

- ExecuteStmt cpu = 43% cpu time =35.84s
- Compile cpu = 31% cpu time =25.61s
- Optimize cpu = 30% cpu time = 24.74s

Performance Overview dashboard

The data of the database time overview and QPS is as follows:

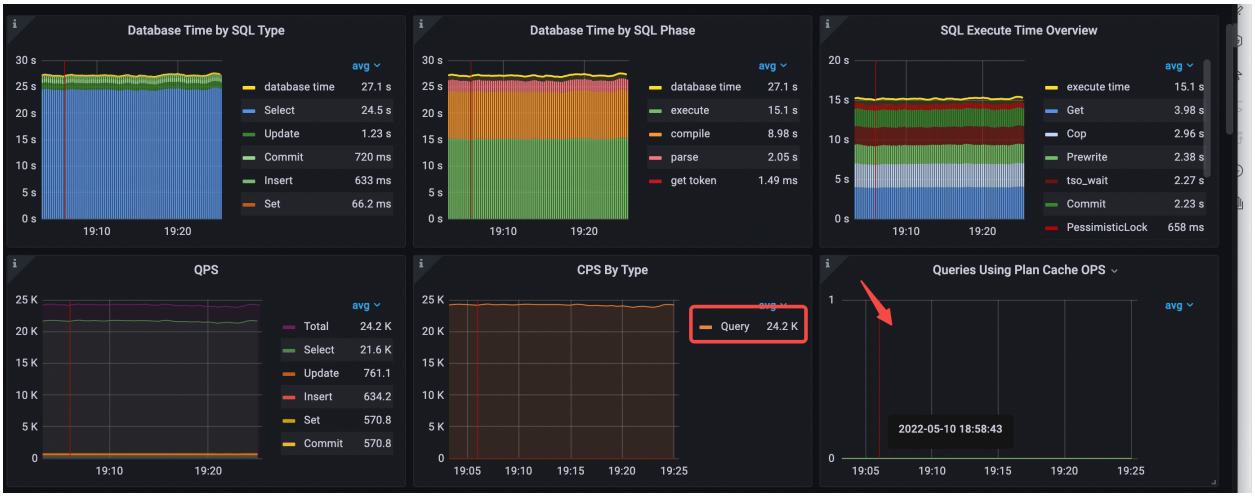


Figure 152: performance-overview-1-for-maxPerformance

- Database Time by SQL Type: the `Select` statement type takes most of the time.

- Database Time by SQL Phase: the `execute` and `compile` phases take most of the time.
- SQL Execute Time Overview: `Get`, `Cop`, `Prewrite`, and `tso wait` take most of the time.
- In the database time, the latency of `execute` and `compile` takes the highest percentage.
- CPS By Type: only the `Query` command is used.
- avg QPS = 24.2k (from 56.3k to 24.2k)
- The execution plan cache is not hit.

From Scenario 1 to Scenario 2, the average TiDB CPU utilization drops from 925% to 874%, and the average TiKV CPU utilization increases from 201% to about 250%.

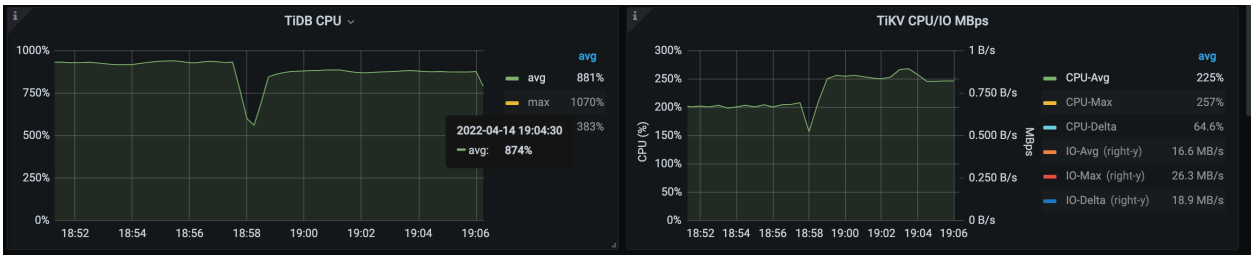


Figure 153: performance-overview-2-for-maxPerformance

The changes in key latency metrics are as follows:



Figure 154: performance-overview-3-for-maxPerformance

- avg query duration = 1.12ms (from 479 s to 1.12ms)
- avg parse duration = 84.7 s (from 37.2 s to 84.7 s)

- avg compile duration = 370 s (from 166 s to 370 s)
- avg execution duration = 626 s (from 251 s to 626 s)

11.1.3.3.3 Analysis conclusion

Compared with Scenario 1, the QPS of Scenario 2 has significantly decreased. The average query duration and average `parse`, `compile`, and `execute` durations have significantly increased. This is because SQL statements such as `select @@session.transaction_read_only` in Scenario 1, which are executed many times and have fast processing time, lower the average performance data. After Scenario 2 blocks such statements, only business-related SQL statements remain, so the average duration increases.

When the application uses the Query interface, TiDB cannot use the execution plan cache, which results in TiDB consuming high resources to compile execution plans. In this case, it is recommended that you use the Prepared Statement interface, which uses the execution plan cache of TiDB to reduce the TiDB CPU consumption caused by execution plan compiling and decrease the latency.

11.1.3.4 Scenario 3. Use the Prepared Statement interface with execution plan caching not enabled

11.1.3.4.1 Application configuration

The application uses the following connection configuration. Compared with Scenario 2, the value of the JDBC parameter `useServerPrepStmts` is modified to `true`, indicating that the Prepared Statement interface is enabled.

```
useServerPrepStmts=true&useConfigs=maxPerformance"
```

11.1.3.4.2 Performance analysis

TiDB Dashboard

From the following flame chart of TiDB, you can see that the CPU consumption of `CompileExecutePreparedStmt` and `Optimize` is still significant after the Prepared Statement interface is enabled.

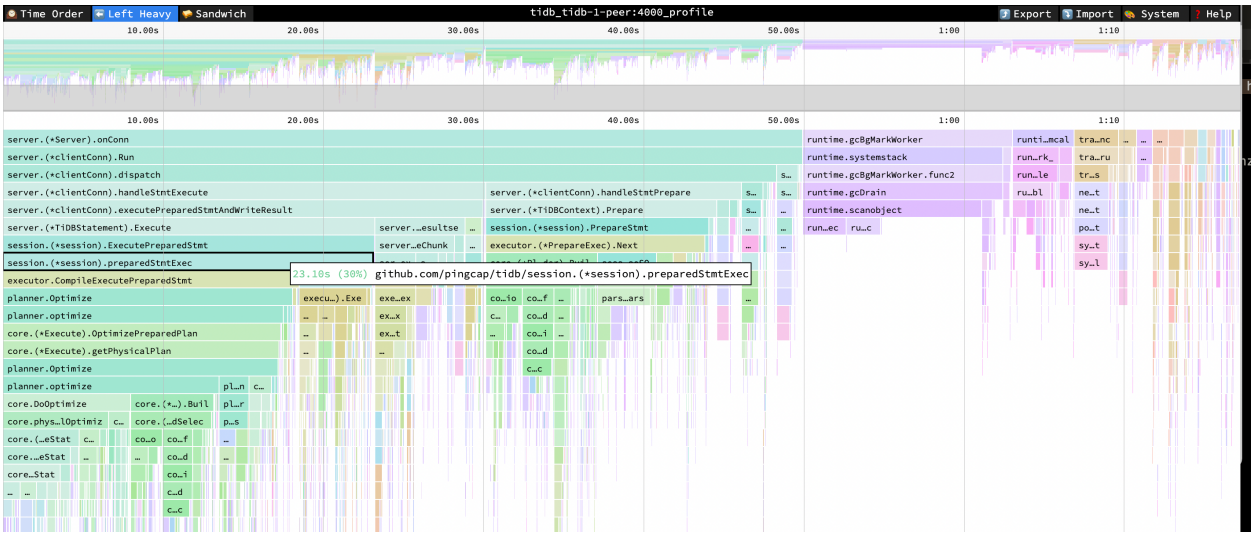


Figure 155: flame-graph-for-PrepStmts

- ExecutePreparedStmt cpu = 31% cpu time = 23.10s
- preparedStmtExec cpu = 30% cpu time = 22.92s
- CompileExecutePreparedStmt cpu = 24% cpu time = 17.83s
- Optimize cpu = 23% cpu time = 17.29s

Performance Overview dashboard

After the Prepared Statement interface is used, the data of database time overview and QPS is as follows:

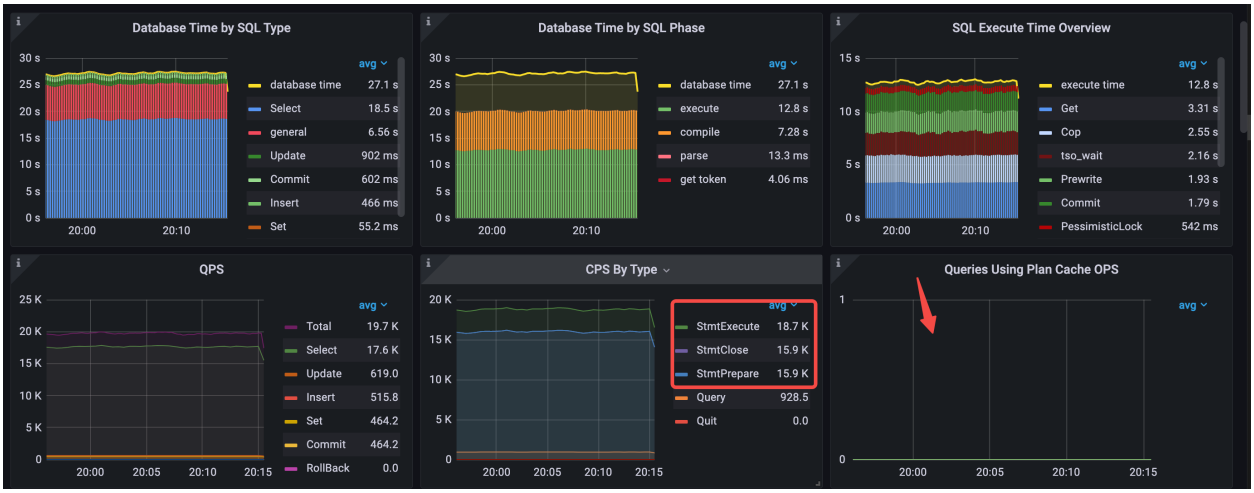


Figure 156: performance-overview-1-for-PrepStmts

The QPS drops from 24.4k to 19.7k. From the Database Time Overview, you can see that the application uses three types of Prepared commands, and the `general` statement

type (which includes the execution time of commands such as `StmtPrepare` and `StmtClose`) takes the second place in Database Time by SQL Type. This indicates that even when the Prepared Statement interface is used, the execution plan cache is not hit. The reason is that, when the `StmtClose` command is executed, TiDB clears the execution plan cache of SQL statements in the internal processing.

- Database Time by SQL Type: the `Select` statement type takes most of the time, followed by `general` statements.
- Database Time by SQL Phase: the `execute` and `compile` phases take most of the time.
- SQL Execute Time Overview: `Get`, `Cop`, `Prewrite`, and `tso wait` take most of the time.
- CPS By Type: 3 types of commands (`StmtPrepare`, `StmtExecute`, `StmtClose`) are used.
- avg QPS = 19.7k (from 24.4k to 19.7k)
- The execution plan cache is not hit.

The TiDB average CPU utilization increases from 874% to 936%.



Figure 157: performance-overview-1-for-PrepStmts

The key latency metrics are as follows:

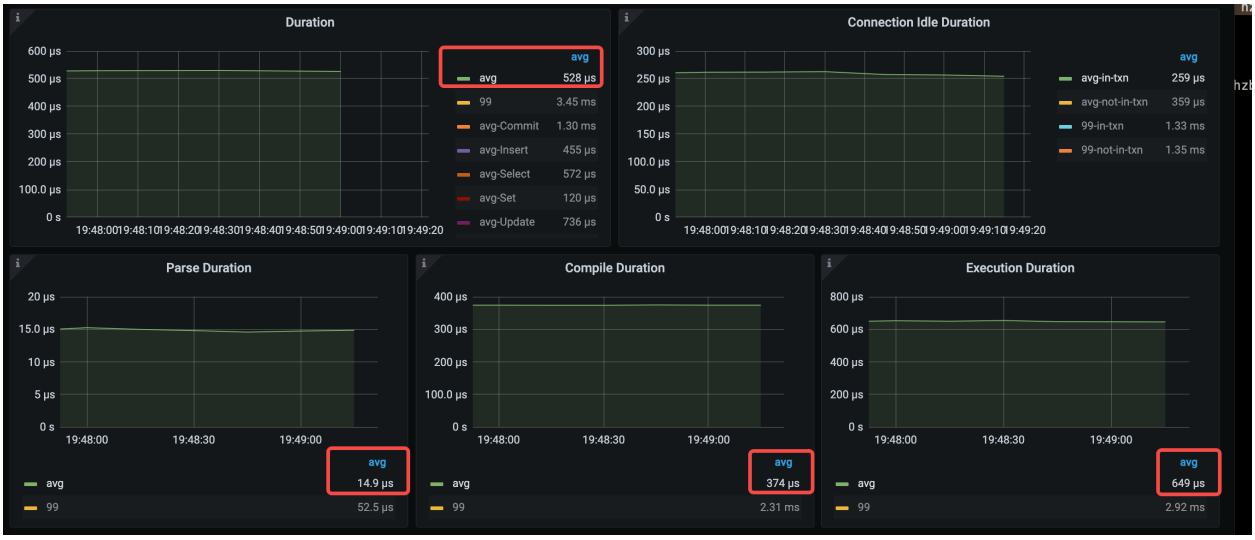


Figure 158: performance-overview-2-for-PrepStmts

- avg query duration = 528 s (from 1.12ms to 528 s)
- avg parse duration = 14.9 s (from 84.7 s to 14.9 s)
- avg compile duration = 374 s (from 370 s to 374 s)
- avg execution duration = 649 s (from 626 s to 649 s)

11.1.3.4.3 Analysis conclusion

Unlike Scenario 2, the application in Scenario 3 enables the Prepared Statement interface but still fails to hit the cache. In addition, Scenario 2 has only one CPS By Type command type (`Query`), while Scenario 3 has three more command types (`StmtPrepare`, `StmtExecute`, `StmtClose`). Compared with Scenario 2, Scenario 3 has two more network round-trip delays.

- Analysis for the decrease in QPS: From the **CPS By Type** pane, you can see that Scenario 2 has only one CPS By Type command type (`Query`), while Scenario 3 has three more command types (`StmtPrepare`, `StmtExecute`, `StmtClose`). `StmtPrepare` and `StmtClose` are non-conventional commands that are not counted by QPS, so QPS is reduced. The non-conventional commands `StmtPrepare` and `StmtClose` are counted in the `general` SQL type, so `general` time is displayed in the database overview of Scenario 3, and it accounts for more than a quarter of the database time.
- Analysis for the significant decrease in average query duration: for the `StmtPrepare` and `StmtClose` command types newly added in Scenario 3, their query duration is calculated separately in the TiDB internal processing. TiDB executes these two types of commands very quickly, so the average query duration is significantly reduced.

Although Scenario 3 uses the Prepared Statement interface, the execution plan cache is still not hit, because many application frameworks call the `StmtClose` method after

`StmtExecute` to prevent memory leaks. Starting from v6.0.0, you can set the global variable `tidb_ignore_prepared_cache_close_stmt=on;`. After that, TiDB will not clear the cached execution plans even if the application calls the `StmtClose` method, so the next SQL execution can reuse the existing execution plan and avoid compiling the execution plan repeatedly.

11.1.3.5 Scenario 4. Use the Prepared Statement interface and enable execution plan caching

11.1.3.5.1 Application configuration

The application configuration remains the same as that of Scenario 3. To resolve the issue of not hitting the cache even if the application triggers `StmtClose`, the following parameters are configured.

- Set the TiDB global variable `set global tidb_ignore_prepared_cache_close_stmt =on;` (introduced since TiDB v6.0.0, `off` by default).
- Set the TiDB configuration item `prepared-plan-cache: {enabled: true}` to enable the plan cache feature.

11.1.3.5.2 Performance analysis

TiDB Dashboard

From the flame chart of the TiDB CPU usage, you can see that `CompileExecutePreparedStmt` and `Optimize` have no significant CPU consumption. 25% of the CPU is consumed by the `Prepare` command, which contains parsing-related functions of Prepare such as `PlanBuilder` and `parseSQL`.

PreparseStmt cpu = 25% cpu time = 12.75s



Figure 159: flame-graph-for-3-commands

Performance Overview dashboard

In the Performance Overview dashboard, the most significant change is the average time of the `compile` phase, which is reduced from 8.95 seconds per second in Scenario 3 to 1.18 seconds per second. The number of queries using the execution plan cache is roughly equal to the value of `StmtExecute`. With the increase in QPS, the database time consumed by `Select` statements per second decreases, and the database time consumed by `general` statements per second type increases.

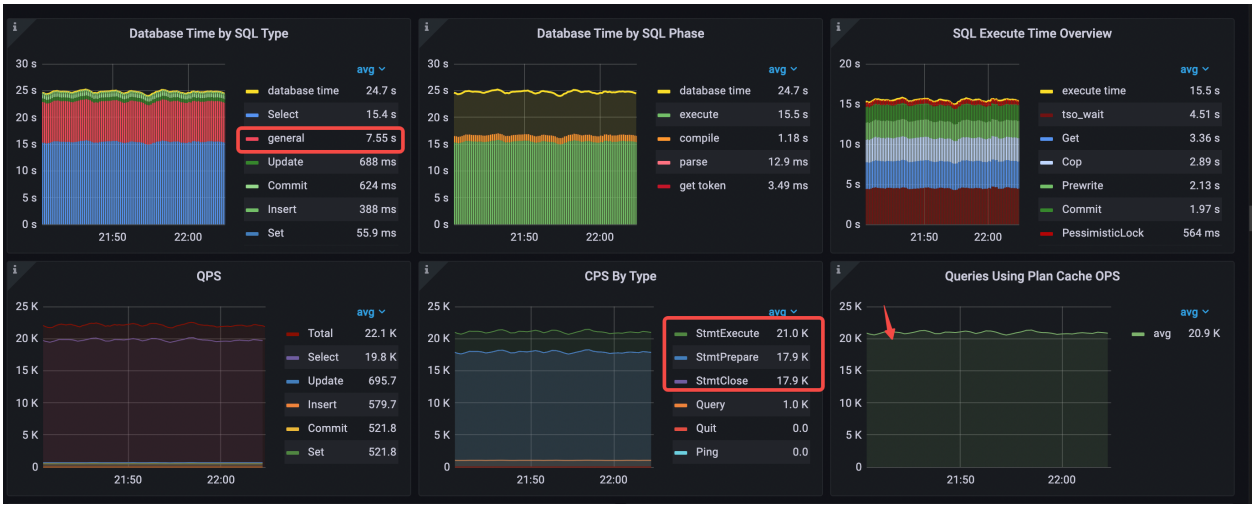


Figure 160: performance-overview-1-for-3-commands

- Database Time by SQL Type: the `Select` statement type takes the most time.
- Database Time by SQL Phase: the `execute` phase takes most of the time.
- SQL Execute Time Overview: `tso wait`, `Get`, and `Cop` take most of the time.
- Execution plan cache is hit. The value of Queries Using Plan Cache OPS roughly equals `StmtExecute` per second.
- CPS By Type: 3 types of commands (same as Scenario 3)
- Compared with scenario 3, the time consumed by `general` statements is longer because the QPS is increased.
- avg QPS = 22.1k (from 19.7k to 22.1k)

The average TiDB CPU utilization drops from 936% to 827%.

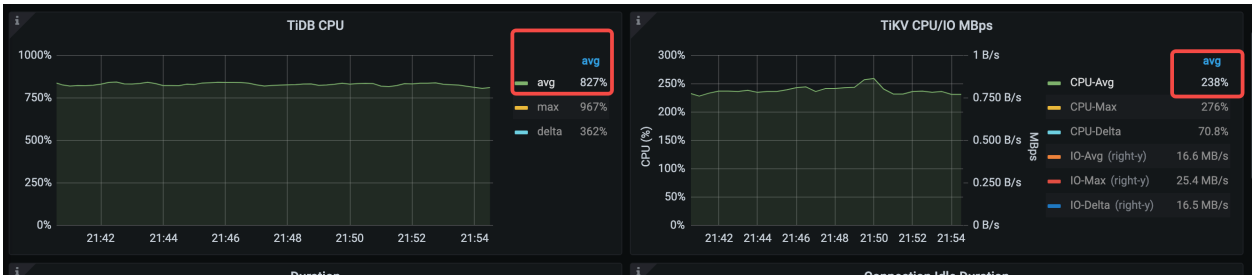


Figure 161: performance-overview-2-for-3-commands

The average `compile` time drops significantly, from 374 us to 53.3 us. Because the QPS increases, the average `execute` time increases too.



Figure 162: performance-overview-3-for-3-commands

- avg query duration = 426 s (from 528 s to 426 s)
- avg parse duration = 12.3 s (from 14.8 s to 12.3 s)
- avg compile duration = 53.3 s (from 374 s to 53.3 s)
- avg execution duration = 699 s (from 649 s to 699us)

11.1.3.5.3 Analysis conclusion

Compared with Scenario 3, Scenario 4 also uses 3 command types. The difference is that Scenario 4 hits the execution plan cache, which reduces compile duration greatly, reduces the query duration, and improves QPS.

Because the `StmtPrepare` and `StmtClose` commands consume significant database time and increase the number of interactions between the application and TiDB each time the

application executes a SQL statement. The next scenario will further tune the performance by eliminating the calls of these two commands through JDBC configurations.

11.1.3.6 Scenario 5. Cache prepared objects on the client side

11.1.3.6.1 Application configuration

Compared with Scenario 4, 3 new JDBC parameters `cachePrepStmts=true&prepStmtCacheSize=1000&prepStmtCacheSqlLimit=20480` are configured, as explained below.

- `cachePrepStmts = true`: caches Prepared Statement objects on the client side, which eliminates the calls of StmtPrepare and StmtClose.
- `prepStmtCacheSize`: the value must be greater than 0.
- `prepStmtCacheSqlLimit`: the value must be greater than the length of the SQL text.

In Scenario 5, the complete JDBC configurations are as follows.

```
useServerPrepStmts=true&cachePrepStmts=true&prepStmtCacheSize=1000&
    ↪ prepStmtCacheSqlLimit=20480&useConfigs=maxPerformance
```

11.1.3.6.2 Performance analysis

TiDB Dashboard

From the following flame chart of TiDB, you can see that the high CPU consumption of the `Prepare` command is no longer present.

- ExecutePreparedStmt cpu = 22% cpu time = 8.4s



Figure 163: flame-graph-for-1-command

Performance Overview dashboard

In the Performance Overview dashboard, the most notable changes are that the three Stmt command types in the **CPS By Type** pane drop to one type, the `general` statement type in the **Database Time by SQL Type** pane is disappeared, and the QPS in the **QPS** pane increases to 30.9k.

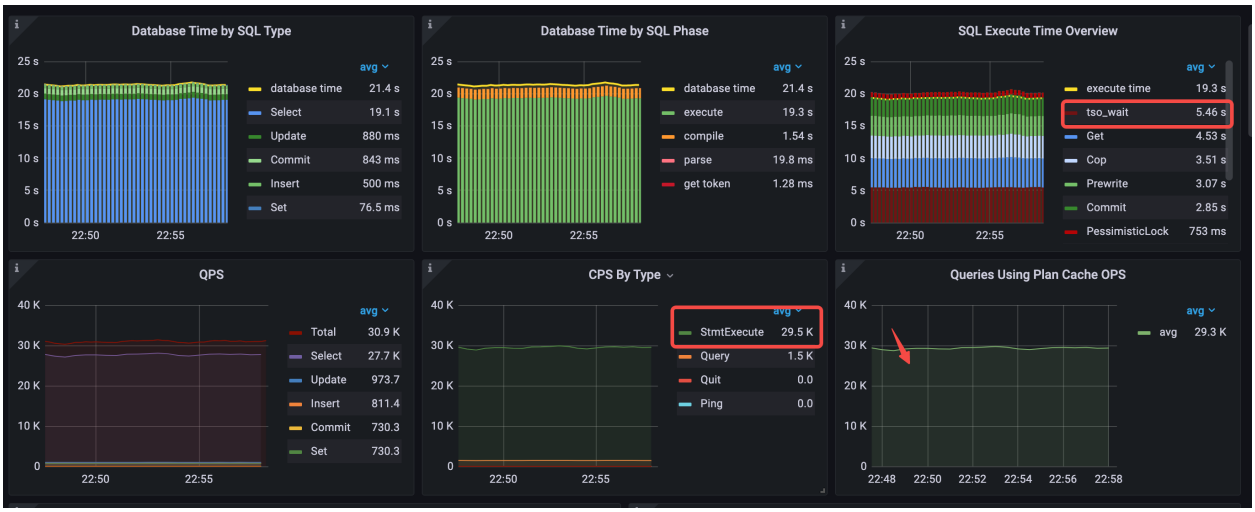


Figure 164: performance-overview-for-1-command

- Database Time by SQL Type: the `Select` statement type takes most of the time and the `general` statement type disappears.
- Database Time by SQL Phase: the `execute` phase takes most of the time.
- SQL Execute Time Overview: `tso wait`, `Get`, and `Cop` take most of the time.
- Execution plan cache is hit. The value of Queries Using Plan Cache OPS roughly equals `StmtExecute` per second.
- CPS By Type: only the `StmtExecute` command is used.
- avg QPS = 30.9k (from 22.1k to 30.9k)

The average TiDB CPU utilization drops from 827% to 577%. As the QPS increases, the average TiKV CPU utilization increases to 313%.

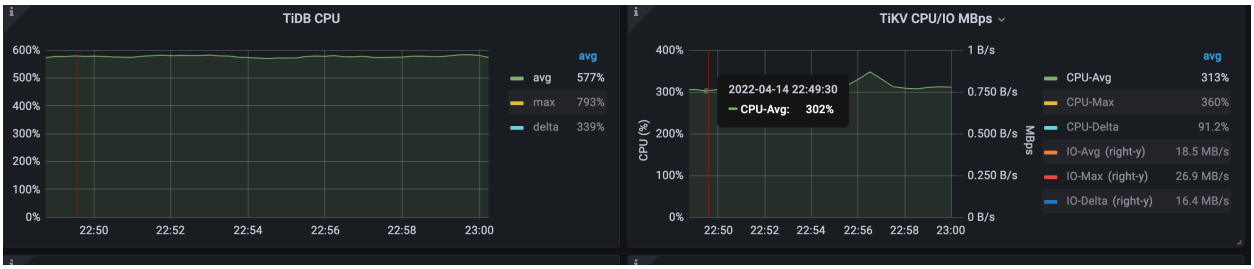


Figure 165: performance-overview-for-2-command

The key latency metrics are as follows:

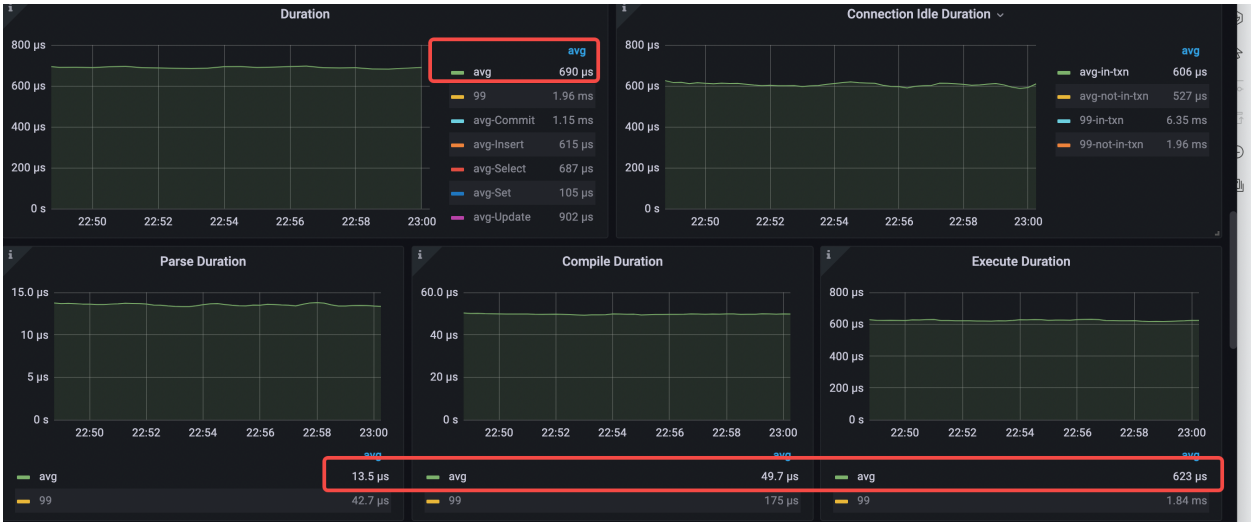


Figure 166: performance-overview-for-3-command

- avg query duration = 690 s (from 426 s to 690 s)
- avg parse duration = 13.5 s (from 12.3 s to 13.5 s)
- avg compile duration = 49.7 s (from 53.3 s to 49.7 s)
- avg execution duration = 623 s (from 699us to 623 s)
- avg pd tso wait duration = 196 s (from 224 s to 196 s)
- connection idle duration avg-in-txn = 608 s (from 250 s to 608 s)

11.1.3.6.3 Analysis conclusion

- Compared with Scenario 4, the **CPS By Type** pane in Scenario 5 has the `StmtExecute` command only, which avoids two network round trips and increases the overall system QPS.
- In the case of QPS increase, the latency decreases in terms of parse duration, compile duration, and execution duration, but the query duration increases instead. This is because TiDB processes `StmtPrepare` and `StmtClose` very quickly, and eliminating these two command types increases the average query duration.
- In Database Time by SQL Phase, `execute` takes the most time and is close to the database time. While in SQL Execute Time Overview, `tso wait` takes most of the time, and more than a quarter of `execute` time is taken to wait for TSO.
- The total `tso wait` time per second is 5.46s. The average `tso wait` time is 196 us, and the number of `tso cmd` times per second is 28k, which is very close to the QPS of 30.9k. This is because according to the implementation of the `read committed` isolation level in TiDB, every SQL statement in a transaction needs to request TSO from PD.

TiDB v6.0 provides `rc read`, which optimizes the `read committed` isolation level by reducing `tso cmd`. This feature is controlled by the global variable `set global ↪ tidb_rc_read_check_ts=on;`. When this variable is enabled, the default behavior of TiDB acts the same as the `repeatable-read` isolation level, at which only `start-ts` and `commit-ts` need to be obtained from the PD. The statements in a transaction use the `start-ts` to read data from TiKV first. If the data read from TiKV is earlier than `start-ts`, the data is returned directly. If the data read from TiKV is later than `start-ts`, the data is discarded. TiDB requests TSO from PD, and then retries the read. The `for update ts` of subsequent statements uses the latest PD TSO.

11.1.3.7 Scenario 6: Enable the `tidb_rc_read_check_ts` variable to reduce TSO requests

11.1.3.7.1 Application configuration

Compared with Scenario 5, the application configuration remains the same. The only difference is that the `set global tidb_rc_read_check_ts=on;` variable is configured to reduce TSO requests.

11.1.3.7.2 Performance analysis

Dashboard

The flame chart of the TiDB CPU does not have any significant changes.

- ExecutePreparedStmt cpu = 22% cpu time = 8.4s

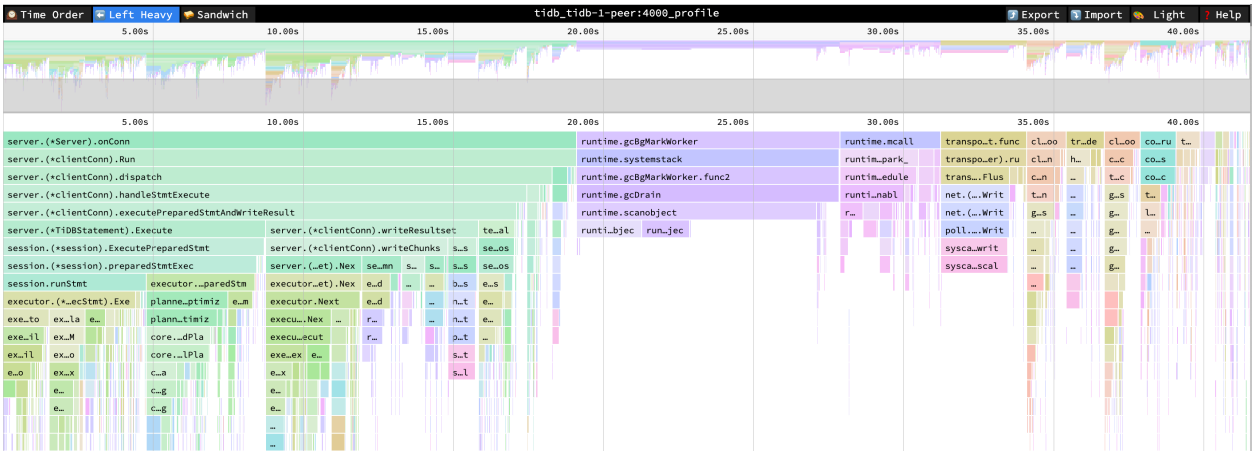


Figure 167: flame-graph-for-rc-read

Performance Overview dashboard

After using RC read, QPS increases from 30.9k to 34.9k, and the `tso wait` time consumed per second decreases from 5.46 s to 456 ms.

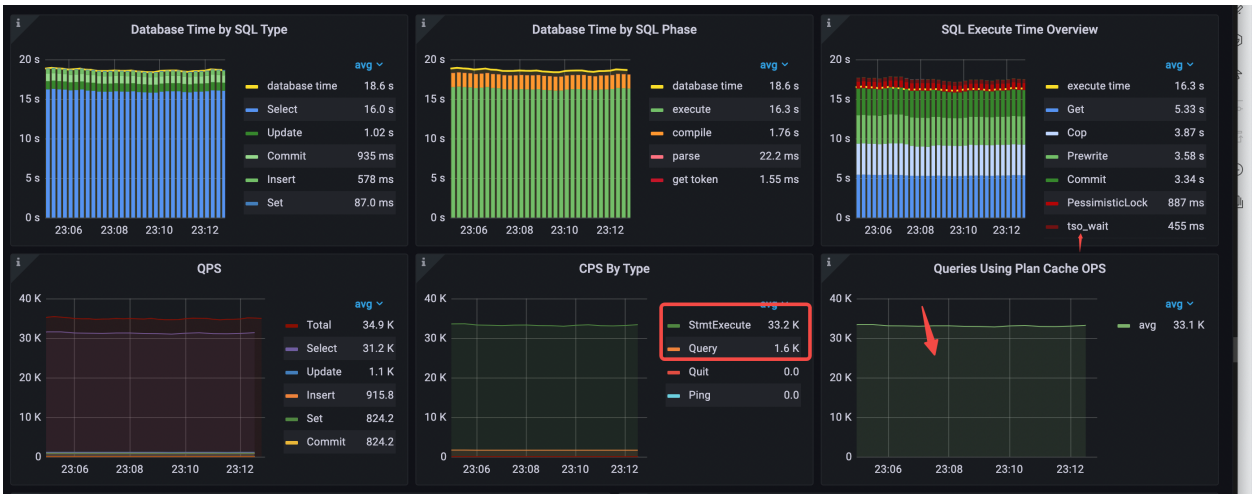


Figure 168: performance-overview-1-for-rc-read

- Database Time by SQL Type: the `Select` statement type takes most of the time.
- Database Time by SQL Phase: the `execute` phase takes most of the time.
- SQL Execute Time Overview: `Get`, `Cop`, and `Prewrite` take most of the time.
- Execution plan cache is hit. The value of Queries Using Plan Cache OPS roughly equals `StmtExecute` per second.
- CPS By Type: only the `StmtExecute` command is used.
- avg QPS = 34.9k (from 30.9k to 34.9k)

The `tso cmd` per second drops from 28.3k to 2.7k.

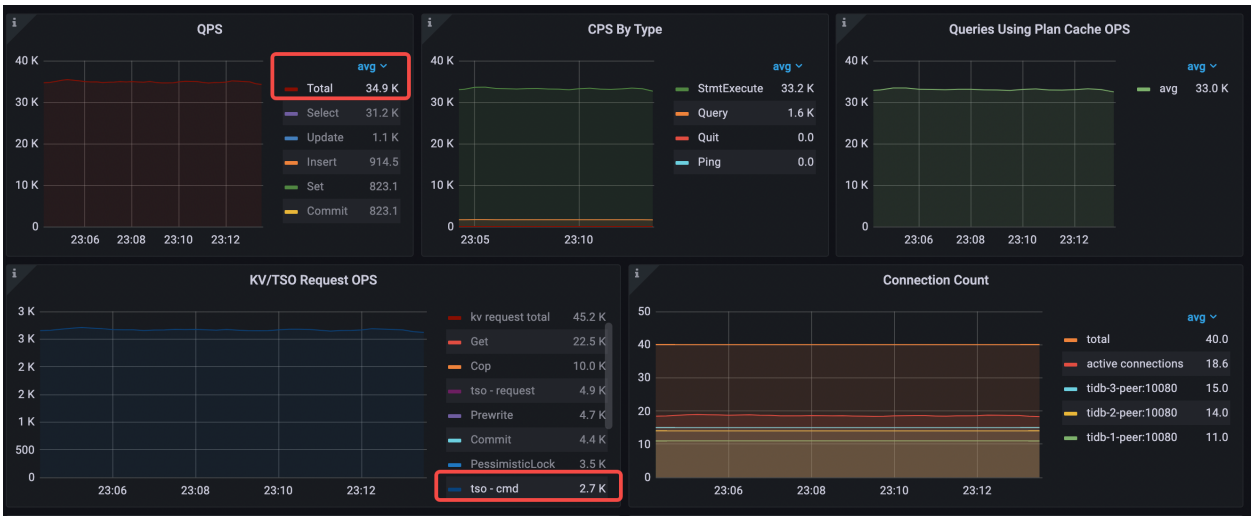


Figure 169: performance-overview-2-for-rc-read

The average TiDB CPU increases to 603% (from 577% to 603%).

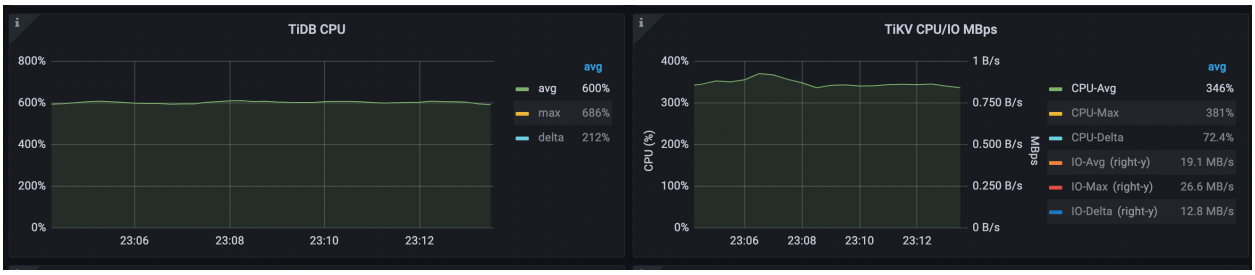


Figure 170: performance-overview-3-for-rc-read

The key latency metrics are as follows:

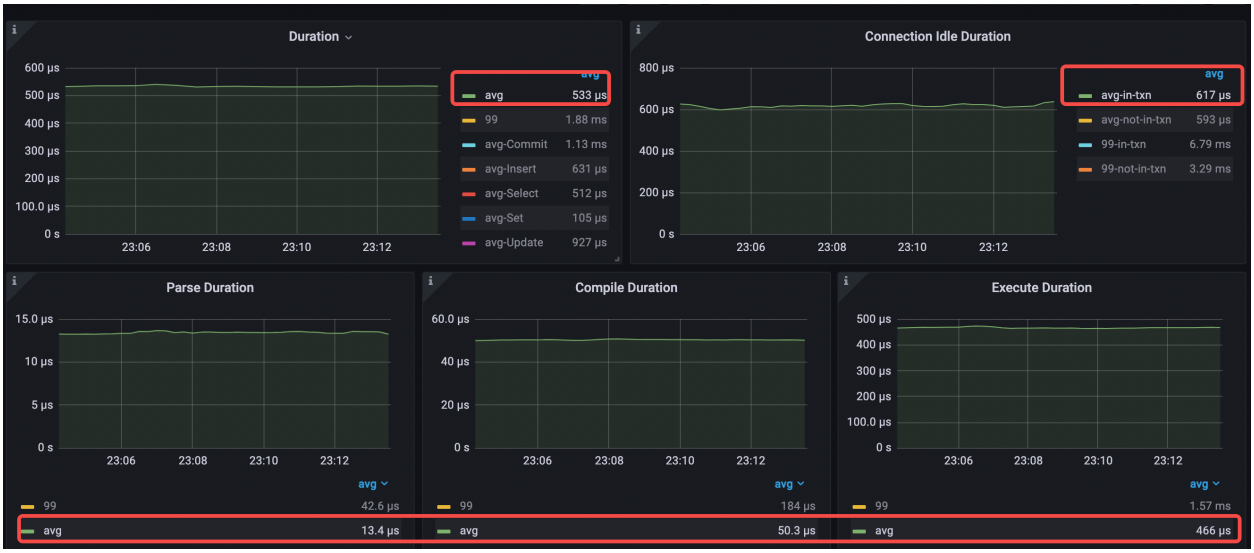


Figure 171: performance-overview-4-for-rc-read

- avg query duration = 533 s (from 690 s to 533 s)
- avg parse duration = 13.4 s (from 13.5 s to 13.4 s)
- avg compile duration = 50.3 s (from 49.7 s to 50.3 s)
- avg execution duration = 466 s (from 623 s to 466 s)
- avg pd tso wait duration = 171 s (from 196 s to 171 s)

11.1.3.7.3 Analysis conclusion

After enabling RC Read by `set global tidb_rc_read_check_ts=on;`, RC Read significantly reduces the times of `tso cmd`, thus reducing `tso wait` and average query duration, and improving QPS.

The bottlenecks of both current database time and latency are in the `execute` phase, in which the `Get` and `Cop` read requests take the highest percentage. Most of the tables in this workload are read-only or rarely modified, so you can use the small table caching feature supported since TiDB v6.0.0 to cache the data of these small tables and reduce the waiting time and resource consumption of KV read requests.

11.1.3.8 Scenario 7: Use the small table cache

11.1.3.8.1 Application configuration

Compared with Scenario 6, the application configuration remains the same. The only difference is that Scenario 7 uses SQL statements such as `alter table t1 cache;` to cache those read-only tables for the business.

11.1.3.8.2 Performance analysis

TiDB Dashboard

The flame chart of the TiDB CPU does not have any significant changes.

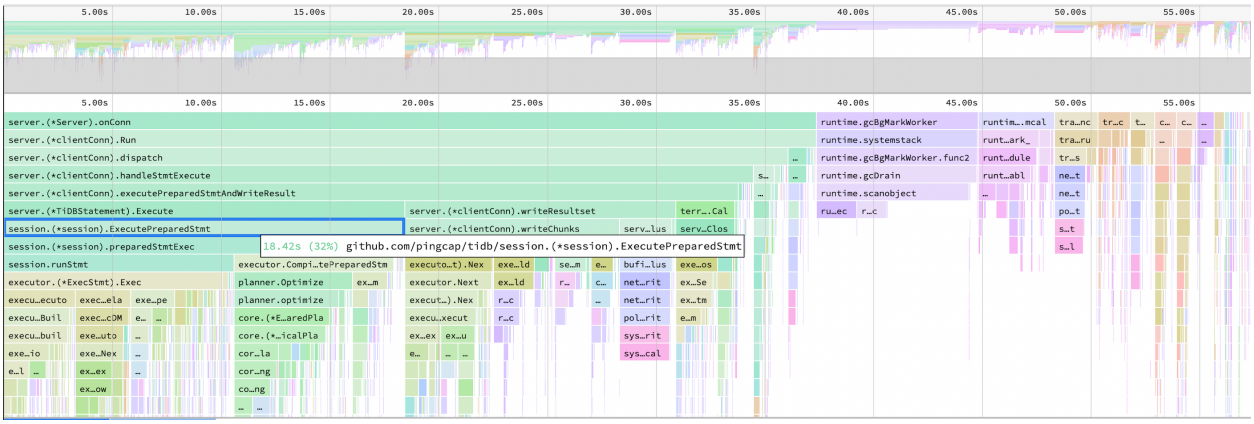


Figure 172: flame-graph-for-table-cache

Performance Overview dashboard

The QPS increases from 34.9k to 40.9k, and the KV request types take the most time in the `execute` phase change to `Prewrite` and `Commit`. The database time consumed by `Get` per second decreases from 5.33 seconds to 1.75 seconds, and the database time consumed by `Cop` per second decreases from 3.87 seconds to 1.09 seconds.

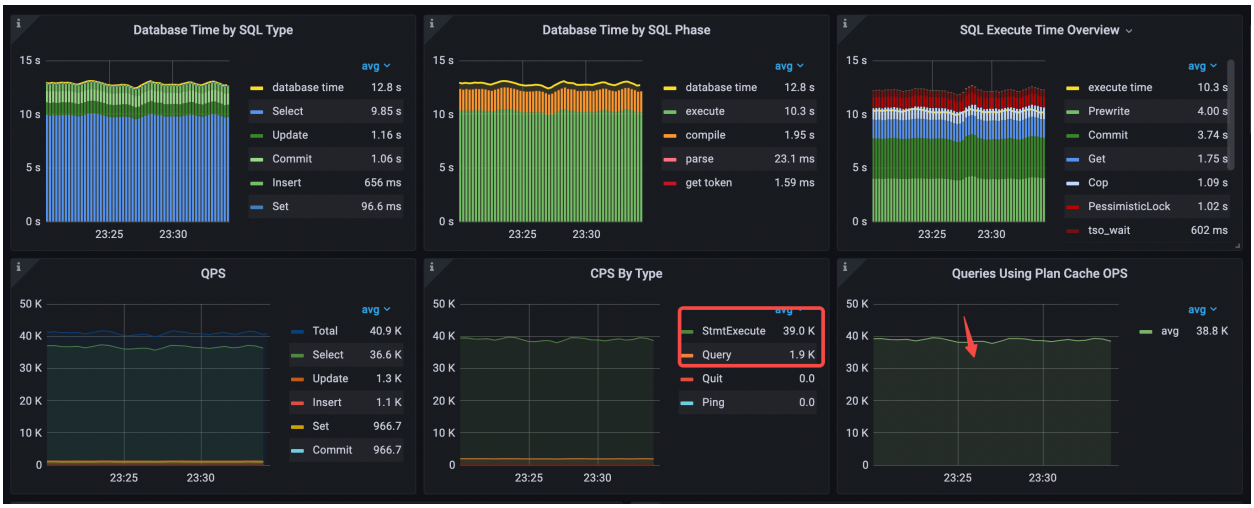


Figure 173: performance-overview-1-for-table-cache

- Database Time by SQL Type: the `Select` statement type takes most of the time.
- Database Time by SQL Phase: the `execute` and `compile` phases take most of the time.
- SQL Execute Time Overview: `Prewrite`, `Commit`, and `Get` take most of the time.
- Execution plan cache is hit. The value of Queries Using Plan Cache OPS roughly equals `StmtExecute` per second.
- CPS By Type: only the `StmtExecute` command is used.
- avg QPS = 40.9k (from 34.9k to 40.9k)

The average TiDB CPU utilization drops from 603% to 478% and the average TiKV CPU utilization drops from 346% to 256%.



Figure 174: performance-overview-2-for-table-cache

The average query latency drops from 533 us to 313 us. The average `execute` latency drops from 466 us to 250 us.

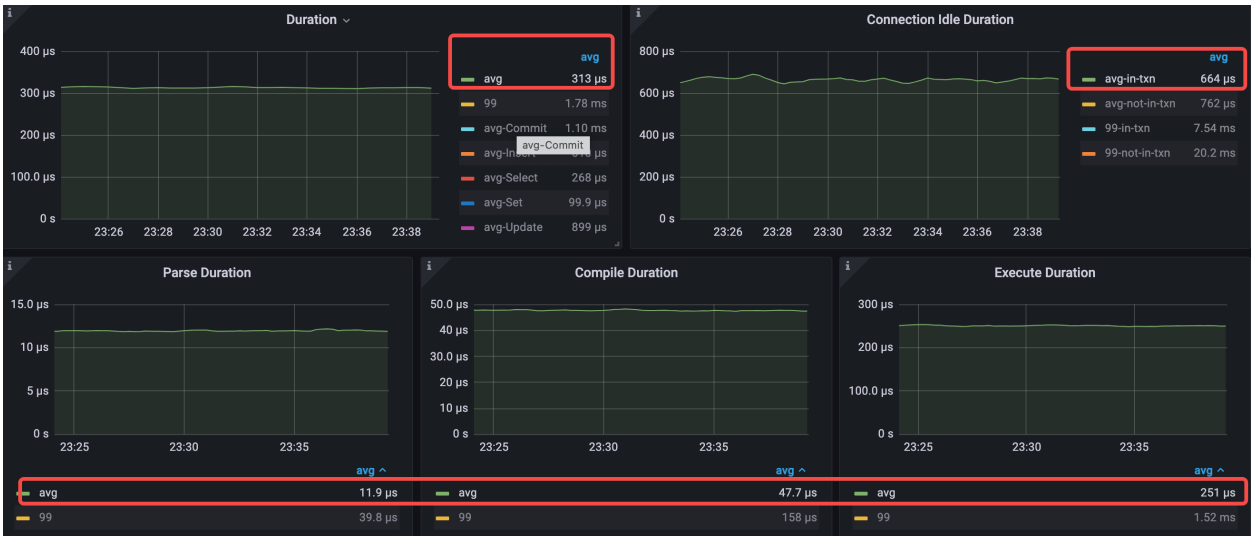


Figure 175: performance-overview-3-for-table-cache

- avg query duration = 313 s (from 533 s to 313 s)
- avg parse duration = 11.9 s (from 13.4 s to 11.9 s)
- avg compile duration = 47.7 s (from 50.3 s to 47.7 s)
- avg execution duration = 251 s (from 466 s to 251 s)

11.1.3.8.3 Analysis conclusion

After caching all read-only tables, the `Execute Duration` drops significantly because all read-only tables are cached in TiDB and there is no need to query data in TiKV for those tables, so the query duration drops and the QPS increases.

This is an optimistic result because data of read-only tables in actual business might be too large for TiDB to cache them all. Another limitation is that although the small table caching feature supports write operations, the write operation requires a default wait of 3 seconds to ensure that the cache of all TiDB nodes is invalidated first, which might not be feasible to applications with strict latency requirements.

11.1.3.9 Summary

The following table lists the performance of seven different scenarios.

Metrics	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6	Scenario 7	Comparing Scenario 5 with Scenario 2 (%)	Comparing Scenario 7 with Scenario 3 (%)
query duration	479 s	1120 s	528 s	426 s	690 s	533 s	313 s	-38%	-41%
QPS	56.3k	24.2k	19.7k	22.1k	30.9k	34.9k	40.9k	+28%	+108%

In these scenarios, Scenario 2 is a common scenario where applications use the Query interface, and Scenario 5 is an ideal scenario where applications use the Prepared Statement interface.

- Comparing Scenario 5 with Scenario 2, you can see that by using best practices for Java application development and caching Prepared Statement objects on the client side, each SQL statement requires only one command and database interaction to hit the execution plan cache, which results in a 38% drop in query latency and a 28% increase in QPS, while the average TiDB CPU utilization drops from 936% to 577%.
- Comparing Scenario 7 with Scenario 3, you can see that with the latest TiDB optimization features such as RC Read and small table cache on top of Scenario 5, latency is reduced by 41% and QPS is increased by 108%, while the average TiDB CPU utilization drops from 936% to 478%.

By comparing the performance of each scenario, we can draw the following conclusions:

- The execution plan cache of TiDB plays a critical role in the OLTP performance tuning. The RC Read and small table cache features introduced from v6.0.0 also play an important role in the further performance tuning of this workload.
- TiDB is compatible with different commands of the MySQL protocol. When using the Prepared Statement interface and setting the following JDBC connection parameters, the application can achieve its best performance:

```
useServerPrepStmts=true&cachePrepStmts=true&prepStmtCacheSize=1000&
    ↪ prepStmtCacheSqlLimit=20480&useConfigs=maxPerformance
```

- It is recommended that you use TiDB Dashboard (for example, the Top SQL feature and Continuous Profiling feature) and Performance Overview dashboard for performance analysis and tuning.

- With the Top SQL feature, you can visually monitor and explore the CPU consumption of each SQL statement in your database during execution to troubleshoot database performance issues.
- With Continuous Profiling, you can continuously collect performance data from each instance of TiDB, TiKV, and PD. When applications use different interfaces to interact with TiDB, the difference in the CPU consumption of TiDB is huge.
- With Performance Overview Dashboard, you can get an overview of database time and SQL execution time breakdown information. You can analyze and diagnose performance based on database time to determine whether the performance bottleneck of the entire system is in TiDB or not. If the bottleneck is in TiDB, you can use the database time and latency breakdowns, along with load profile and resource usage, to identify performance bottlenecks within TiDB and tune the performance accordingly.

With a combination usage of these features, you can analyze and tune performance for real-world applications efficiently.

11.2 Configuration Tuning

11.2.1 Tune Operating System Performance

This document introduces how to tune each subsystem of CentOS 7.

Note:

- The default configuration of the CentOS 7 operating system is suitable for most services running under moderate workloads. Adjusting the performance of a particular subsystem might negatively affects other subsystems. Therefore, before tuning the system, back up all the user data and configuration information.
- Fully test all the changes in the test environment before applying them to the production environment.

11.2.1.1 Performance analysis methods

System tuning must be based on the results of system performance analysis. This section lists common methods for performance analysis.

11.2.1.1.1 In 60 seconds

Linux Performance Analysis in 60,000 Milliseconds is published by the author Brendan Gregg and the Netflix Performance Engineering team. All tools used can be obtained

from the official release of Linux. You can analyze outputs of the following list items to troubleshoot most common performance issues.

- `uptime`
- `dmesg | tail`
- `vmstat 1`
- `mpstat -P ALL 1`
- `pidstat 1`
- `iostat -xz 1`
- `free -m`
- `sar -n DEV 1`
- `sar -n TCP,ETCP 1`
- `top`

For detailed usage, see the corresponding `man` instructions.

11.2.1.1.2 perf

perf is an important performance analysis tool provided by the Linux kernel, which covers hardware level (CPU/PMU, performance monitoring unit) features and software features (software counters, trace points). For detailed usage, see perf Examples.

11.2.1.1.3 BCC/bpftrace

Starting from CentOS 7.6, the Linux kernel has supported Berkeley Packet Filter (BPF). Therefore, you can choose proper tools to conduct an in-depth analysis based on the results in In 60 seconds. Compared with perf/ftrace, BPF provides programmability and smaller performance overhead. Compared with kprobe, BPF provides higher security and is more suitable for the production environments. For detailed usage of the BCC toolkit, see BPF Compiler Collection (BCC).

11.2.1.2 Performance tuning

This section introduces performance tuning based on the classified kernel subsystems.

11.2.1.2.1 CPU—frequency scaling

cpufreq is a module that dynamically adjusts the CPU frequency. It supports five modes. To ensure service performance, select the performance mode and fix the CPU frequency at the highest supported operating frequency without dynamic adjustment. The command for this operation is `cpupower frequency-set --governor performance`.

11.2.1.2.2 CPU—interrupt affinity

- Automatic balance can be implemented through the `irqbalance` service.
- Manual balance:
    - Identify the devices that need to balance interrupts. Starting from CentOS 7.5, the system automatically configures the best interrupt affinity for certain devices and their drivers, such as devices that use the `be2iscsi` driver and NVMe settings. You can no longer manually configure interrupt affinity for such devices.
    - For other devices, check the chip manual to see whether these devices support distributing interrupts.
        - If they do not, all interrupts of these devices are routed to the same CPU and cannot be modified.
        - If they do, calculate the `smp_affinity` mask and set the corresponding configuration file. For details, see the kernel document.

11.2.1.2.3 NUMA CPU binding

To avoid accessing memory across Non-Uniform Memory Access (NUMA) nodes as much as possible, you can bind a thread/process to certain CPU cores by setting the CPU affinity of the thread. For ordinary programs, you can use the `numactl` command for the CPU binding. For detailed usage, see the Linux manual pages. For network interface card (NIC) interrupts, see tune network.

11.2.1.2.4 Memory—transparent huge page (THP)

It is **NOT** recommended to use THP for database applications, because databases often have sparse rather than continuous memory access patterns. If high-level memory fragmentation is serious, a higher latency will occur when THP pages are allocated. If the direct compaction is enabled for THP, the CPU usage will surge. Therefore, it is recommended to disable THP.

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

11.2.1.2.5 Memory—virtual memory parameters

- `dirty_ratio` percentage ratio. When the total amount of dirty page caches reach this percentage ratio of the total system memory, the system starts to use the `pdflush` operation to write the dirty page caches to disk. The default value of `dirty_ratio` is 20% and usually does not need adjustment. For high-performance SSDs such as NVMe devices, lowering this value helps improve the efficiency of memory reclamation.

- `dirty_background_ratio` percentage ratio. When the total amount of dirty page caches reach this percentage ratio of the total system memory, the system starts to write the dirty page caches to the disk in the background. The default value of `dirty_background_ratio` is 10% and usually does not need adjustment. For high-performance SSDs such as NVMe devices, setting a lower value helps improve the efficiency of memory reclamation.

11.2.1.2.6 Storage and file system

The core I/O stack link is long, including the file system layer, the block device layer, and the driver layer.

I/O scheduler

The I/O scheduler determines when and how long I/O operations run on the storage device. It is also called I/O elevator. For SSD devices, it is recommended to set the I/O scheduling policy to `noop`.

```
echo noop > /sys/block/${SSD_DEV_NAME}/queue/scheduler
```

Formatting parameters—block size

Blocks are the working units of the file system. The block size determines how much data can be stored in a single block, and thus determines the minimum amount of data to be written or read each time.

The default block size is suitable for most scenarios. However, if the block size (or the size of multiple blocks) is the same or slightly larger than the amount of data normally read or written each time, the file system performs better and the data storage efficiency is higher. Small files still use the entire block. Files can be distributed among multiple blocks, but this will increase runtime overhead.

When using the `mkfs` command to format a device, specify the block size as a part of the file system options. The parameters that specify the block size vary with the file system. For details, see the corresponding `mkfs` manual pages, such as using `man mkfs.ext4`.

`mount` parameters

If the `noatime` option is enabled in the `mount` command, the update of metadata is disabled when files are read. If the `nodiratime` behavior is enabled, the update of metadata is disabled when the directory is read.

11.2.1.2.7 Network tuning

The network subsystem consists of many different parts with sensitive connections. The CentOS 7 network subsystem is designed to provide the best performance for most workloads and automatically optimizes the performance of these workloads. Therefore, usually you do not need to manually adjust network performance.

Network issues are usually caused by issues of hardware or related devices. So before tuning the protocol stack, rule out hardware issues.

Although the network stack is largely self-optimizing, the following aspects in the network packet processing might become the bottleneck and affect performance:

- NIC hardware cache: To correctly observe the packet loss at the hardware level, use the `ethtool -S ${NIC_DEV_NAME}` command to observe the `drops` field. When packet loss occurs, it might be that the processing speed of the hard/soft interrupts cannot catch up with the receiving speed of NIC. If the received buffer size is less than the upper limit, you can also try to increase the RX buffer to avoid packet loss. The query command is: `ethtool -g ${NIC_DEV_NAME}`, and the modification command is `ethtool -G ${NIC_DEV_NAME}`.

- Hardware interrupts: If the NIC supports the Receive-Side Scaling (RSS, also called multi-NIC receiving) feature, observe the `/proc/interrupts` NIC interrupts. If the interrupts are uneven, see CPU—frequency scaling, CPU—interrupt affinity, and NUMA CPU binding. If the NIC does not support RSS or the number of RSS is much smaller than the number of physical CPU cores, configure Receive Packet Steering (RPS, which can be regarded as the software implementation of RSS), and the RPS extension Receive Flow Steering (RFS). For detailed configuration, see the kernel document.

- Software interrupts: Observe the monitoring of `/proc/net/softnet_stat`. If the values of the other columns except the third column are increasing, properly adjust the value of `net.core.netdev_budget` or `net.core.dev_weight` for `softirq` to get more CPU time. In addition, you also need to check the CPU usage to determine which tasks are frequently using the CPU and whether they can be optimized.

- Receive queue of application sockets: Monitor the `Resv-q` column of `ss -nmp`. If the queue is full, consider increasing the size of the application socket cache or use the automatic cache adjustment method. In addition, consider whether you can optimize the architecture of the application layer and reduce the interval between reading sockets.

- Ethernet flow control: If the NIC and switch support the flow control feature, you can use this feature to leave some time for the kernel to process the data in the NIC queue, to avoid the issue of NIC buffer overflow.

- Interrupts coalescing: Too frequent hardware interrupts reduces system performance, and too late hardware interrupts causes packet loss. Newer NICs support the interrupt coalescing feature and allow the driver to automatically adjust the number of hardware interrupts. You can execute `ethtool -c ${NIC_DEV_NAME}` to check and `ethtool -C ${NIC_DEV_NAME}` to enable this feature. The adaptive mode allows the NIC to automatically adjust the interrupt coalescing. In this mode, the driver checks the traffic mode and kernel receiving mode, and evaluates the coalescing settings in real time to prevent packet loss. NICs of different brands have different features and default configurations. For details, see the NIC manuals.

- Adapter queue: Before processing the protocol stack, the kernel uses this queue to buffer the data received by the NIC, and each CPU has its own backlog queue. The

maximum number of packets that can be cached in this queue is `netdev_max_backlog` ↪ . Observe the second column of `/proc/net/softnet_stat`. When the second column of a row continues to increase, it means that the CPU [row-1] queue is full and the data packet is lost. To resolve this problem, continue to double the `net.core.` ↪ `netdev_max_backlog` value.

- Send queue: The length value of a send queue determines the number of packets that can be queued before sending. The default value is `1000`, which is sufficient for 10 Gbps. But if you have observed the value of TX errors from the output of `ip -s link`, you can try to double it: `ip link set dev ${NIC_DEV_NAME} txqueuelen 2000`.
- Driver: NIC drivers usually provide tuning parameters. See the device hardware manual and its driver documentation.

11.2.2 TiDB Memory Control

Currently, TiDB can track the memory quota of a single SQL query and take actions to prevent OOM (out of memory) or troubleshoot OOM when the memory usage exceeds a specific threshold value. The system variable `tidb_mem_oom_action` specifies the action to take when a query reaches the memory limit:

- A value of `LOG` means that queries will continue to execute when the `tidb_mem_quota_query` ↪ limit is reached, but TiDB will print an entry to the log.
- A value of `CANCEL` means TiDB stops executing the SQL query immediately after the `tidb_mem_quota_query` limit is reached, and returns an error to the client. The error information clearly shows the memory usage of each physical execution operator that consumes memory in the SQL execution process.

11.2.2.1 Configure the memory quota of a query

The system variable `tidb_mem_quota_query` sets the limit for a query in bytes. Some usage examples:

```
-- Set the threshold value of memory quota for a single SQL query to 8GB:
SET tidb_mem_quota_query = 8 << 30;
```

```
-- Set the threshold value of memory quota for a single SQL query to 8MB:
SET tidb_mem_quota_query = 8 << 20;
```

```
-- Set the threshold value of memory quota for a single SQL query to 8KB:
SET tidb_mem_quota_query = 8 << 10;
```

11.2.2.2 Configure the memory usage threshold of a tidb-server instance

In the TiDB configuration file, you can set the memory usage threshold of a tidb-server instance by configuring server-memory-quota.

The following example sets the total memory usage of a tidb-server instance to 32 GB:

```
[performance]
server-memory-quota = 34359738368
```

In this configuration, when the memory usage of a tidb-server instance reaches 32 GB, the instance starts to kill running SQL statements randomly until the memory usage drops below 32 GB. SQL operations that are forced to terminate return an `Out Of Global Memory Limit!` error message to the client.

Warning:

- `server-memory-quota` is still an experimental feature. It is **NOT** recommended that you use it in a production environment.
- The default value of `server-memory-quota` is `0`, which means no memory limit.

11.2.2.3 Trigger the alarm of excessive memory usage

In the default configuration, a tidb-server instance prints an alarm log and records associated status files when the machine memory usage reaches 80% of its total memory. You can set the memory usage ratio threshold by configuring `memory-usage-alarm-ratio`. For detailed alarm rules, refer to the description of memory-usage-alarm-ratio.

Note that after the alarm is triggered once, it will be triggered again only if the memory usage rate has been below the threshold for more than ten seconds and reaches the threshold again. In addition, to avoid storing excessive status files generated by alarms, currently, TiDB only retains the status files generated during the recent five alarms.

The following example constructs a memory-intensive SQL statement that triggers the alarm:

1. Set `memory-usage-alarm-ratio` to `0.8`:

   ```
   [performance]
   memory-usage-alarm-ratio = 0.8
   ```

2. Execute `CREATE TABLE t(a int);` and insert 1000 rows of data.

3. Execute `select * from t t1 join t t2 join t t3 order by t1.a`. This SQL statement outputs one billion records, which consumes a large amount of memory and therefore triggers the alarm.

4. Check the `tidb.log` file which records the total system memory, current system memory usage, memory usage of the tidb-server instance, and the directory of status files.

```
[2020/11/30 15:25:17.252 +08:00] [WARN] [memory_usage_alarm.go:141] ["
    ↪ tidb-server has the risk of OOM. Running SQLs and heap profile
    ↪ will be recorded in record path"] ["is server-memory-quota set"=
    ↪ false] ["system memory total"=33682427904] ["system memory usage
    ↪ "=27142864896] ["tidb-server memory usage"=22417922896] [memory-
    ↪ usage-alarm-ratio=0.8] ["record path"="/tmp/1000_tidb/
    ↪ MC4wLjAuMDo0MDAwLzAuMC4wLjA6MTAwODA=/tmp-storage/record"]
```

   The fields of the example log file above are described as follows:

   - `is server-memory-quota set` indicates whether `server-memory-quota` is set.
   - `system memory total` indicates the total memory of the current system.
   - `system memory usage` indicates the current system memory usage.
   - `tidb-server memory usage` indicates the memory usage of the tidb-server instance.
   - `memory-usage-alarm-ratio` indicates the value of `memory-usage-alarm-ratio`.
   - `record path` indicates the directory of status files.

5. You can see a set of files in the directory of status files (In the above example, the directory is `/tmp/1000_tidb/MC4wLjAuMDo0MDAwLzAuMC4wLjA6MTAwODA=/tmp-storage ↪ /record`), including `goroutinue`, `heap`, and `running_sql`. These three files are suffixed with the time when status files are logged. They respectively record goroutine stack information, the usage status of heap memory, and the running SQL information when the alarm is triggered. For the format of log content in `running_sql`, refer to `expensive-queries`.

11.2.2.4 Other memory control behaviors of tidb-server

11.2.2.4.1 Flow control

- TiDB supports dynamic memory control for the operator that reads data. By default, this operator uses the maximum number of threads that `tidb_distsql_scan_concurrency ↪` allows to read data. When the memory usage of a single SQL execution exceeds `tidb_mem_quota_query` each time, the operator that reads data stops one thread.
- This flow control behavior is controlled by the system variable `tidb_enable_rate_limit_action ↪` . This variable is enabled by default, which might make the memory usage not under the control of `tidb_mem_quota_query` in some cases. Therefore, it is recommended to set the value of `tidb_enable_rate_limit_action` to `OFF`.

- When the flow control behavior is triggered, TiDB outputs a log containing the keywords `memory exceeds quota, destroy one token now`.

11.2.2.4.2 Disk spill

TiDB supports disk spill for execution operators. When the memory usage of a SQL execution exceeds the memory quota, tidb-server can spill the intermediate data of execution operators to the disk to relieve memory pressure. Operators supporting disk spill include Sort, MergeJoin, HashJoin, and HashAgg.

- The disk spill behavior is jointly controlled by `tidb_mem_quota_query`, `oom-use-tmp-storage`, `tmp-storage-path`, and `tmp-storage-quota` parameters.
- When the disk spill is triggered, TiDB outputs a log containing the keywords `memory exceeds quota, spill to disk now` or `memory exceeds quota, set aggregate mode to spill-mode`.
- Disk spill for the Sort, MergeJoin, and HashJoin operator is introduced in v4.0.0; disk spill for the HashAgg operator is introduced in v5.2.0.
- When the SQL executions containing Sort, MergeJoin, or HashJoin cause OOM, TiDB triggers disk spill by default. When SQL executions containing HashAgg cause OOM, TiDB does not trigger disk spill by default. You can configure the system variable `tidb_executor_concurrency = 1` to trigger disk spill for HashAgg.

> **Note:**
>
> The disk spill for HashAgg does not support SQL executions containing the `DISTINCT` aggregate function. When a SQL execution containing a `DISTINCT` aggregate function uses too much memory, the disk spill does not apply.

The following example uses a memory-consuming SQL statement to demonstrate the disk spill feature for HashAgg:

1. Configure the memory quota of a SQL statement to 1GB (1 GB by default):

```
SET tidb_mem_quota_query = 1 << 30;
```

2. Create a single table `CREATE TABLE t(a int);` and insert 256 rows of different data.

3. Execute the following SQL statement:

```
[tidb]> explain analyze select /*+ HASH_AGG() */ count(*) from t t1
    ↪ join t t2 join t t3 group by t1.a, t2.a, t3.a;
```

Because executing this SQL statement occupies too much memory, the following "Out of Memory Quota" error message is returned:

```
ERROR 1105 (HY000): Out Of Memory Quota![conn_id=3]
```

4. Configure the system variable `tidb_executor_concurrency` to 1. With this configuration, when out of memory, HashAgg automatically tries to trigger disk spill.

```
SET tidb_executor_concurrency = 1;
```

5. Execute the same SQL statement. You can find that this time, the statement is successfully executed and no error message is returned. From the following detailed execution plan, you can see that HashAgg has used 600 MB of hard disk space.

```
[tidb]> explain analyze select /*+ HASH_AGG() */ count(*) from t t1
    ↪ join t t2 join t t3 group by t1.a, t2.a, t3.a;
```

```
+--
    ↪ ------------------------------+------------+---------+-----------+------
    ↪
| id                            | estRows    | actRows | task      | access
    ↪ object | execution info
    ↪
    ↪ | operator info                                          |
    ↪ memory   | disk    |
+--
    ↪ ------------------------------+------------+---------+-----------+------
    ↪
| HashAgg_11                    | 204.80     | 16777216 | root    |
    ↪              | time:1m37.4s, loops:16385
    ↪
    ↪ | group by:test.t.a, test.t.a, test.t.a, funcs:count(1)->Column
    ↪ #7 | 1.13 GB | 600.0 MB |
|  ─HashJoin_12                 | 16777216.00 | 16777216 | root  |
    ↪              | time:21.5s, loops:16385, build_hash_table:{total
    ↪ :267.2µs, fetch:228.9µs, build:38.2µs}, probe:{concurrency:1,
    ↪ total:35s, max:35s, probe:35s, fetch:962.2µs} | CARTESIAN inner
    ↪ join                        | 8.23 KB  | 4 KB     |
|    ─TableReader_21(Build)     | 256.00     | 256      | root      |
    ↪              | time:87.2µs, loops:2, cop_task: {num: 1, max: 150
    ↪ µs, proc_keys: 0, rpc_num: 1, rpc_time: 145.1µs,
    ↪ copr_cache_hit_ratio: 0.00}                    | data:
    ↪ TableFullScan_20                                     | 885 Bytes |
    ↪ N/A    |
|      ─TableFullScan_20        | 256.00     | 256      | cop[tikv] |
    ↪ table:t3   | tikv_task:{time:23.2µs, loops:256}
```

```
    ↪
    ↪ | keep order:false, stats:pseudo                          | N/A
    ↪     | N/A     |
|    ─HashJoin_14(Probe)       | 65536.00   | 65536   | root     |
    ↪             | time:728.1µs, loops:65, build_hash_table:{total
    ↪ :307.5µs, fetch:277.6µs, build:29.9µs}, probe:{concurrency:1,
    ↪ total:34.3s, max:34.3s, probe:34.3s, fetch:278µs} | CARTESIAN
    ↪ inner join                    | 8.23 KB  | 4 KB    |
|      ─TableReader_19(Build)  | 256.00     | 256     | root     |
    ↪             | time:126.2µs, loops:2, cop_task: {num: 1, max:
    ↪ 308.4µs, proc_keys: 0, rpc_num: 1, rpc_time: 295.3µs,
    ↪ copr_cache_hit_ratio: 0.00}               | data:TableFullScan_18
    ↪                                     | 885 Bytes | N/A    |
|        ─TableFullScan_18     | 256.00     | 256     | cop[tikv] |
    ↪ table:t2   | tikv_task:{time:79.2µs, loops:256}
    ↪
    ↪ | keep order:false, stats:pseudo                          | N/A
    ↪     | N/A     |
|      ─TableReader_17(Probe)  | 256.00     | 256     | root     |
    ↪             | time:211.1µs, loops:2, cop_task: {num: 1, max:
    ↪ 295.5µs, proc_keys: 0, rpc_num: 1, rpc_time: 279.7µs,
    ↪ copr_cache_hit_ratio: 0.00}               | data:TableFullScan_16
    ↪                                     | 885 Bytes | N/A    |
|        ─TableFullScan_16     | 256.00     | 256     | cop[tikv] |
    ↪ table:t1   | tikv_task:{time:71.4µs, loops:256}
    ↪
    ↪ | keep order:false, stats:pseudo                          | N/A
    ↪     | N/A     |
+--
    ↪ ------------------------------+------------+----------+-----------+------
    ↪
9 rows in set (1 min 37.428 sec)
```

11.2.2.5 Others

11.2.2.5.1 Mitigate OOM issues by configuring `GOMEMLIMIT`

GO 1.19 introduces an environment variable GOMEMLIMIT to set the memory limit that triggers GC.

For v6.1.3 <= TiDB < v6.5.0, you can mitigate a typical category of OOM issues by manually setting `GOMEMLIMIT`. The typical category of OOM issues is: before OOM occurs, the estimated memory in use on Grafana occupies only half of the entire memory (TiDB-Runtime > Memory Usage > estimate-inuse), as shown in the following figure:

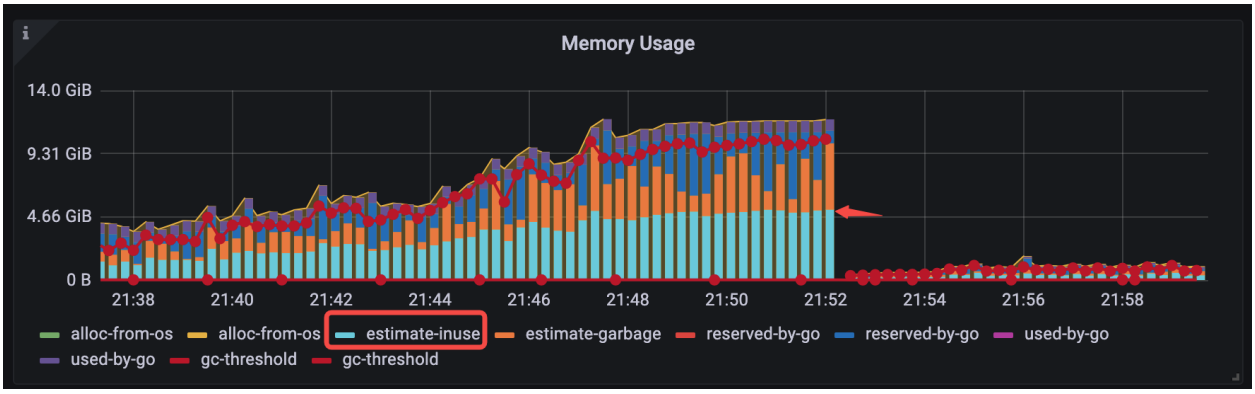


Figure 176: normal OOM case example

To verify the performance of `GOMEMLIMIT`, a test is performed to compare the specific memory usage with and without `GOMEMLIMIT` configuration.

- In TiDB v6.1.2, the TiDB server encounters OOM (system memory: about 48 GiB) after the simulated workload runs for several minutes:



Figure 177: v6.1.2 workload oom

- In TiDB v6.1.3, `GOMEMLIMIT` is set to 40000 MiB. It is found that the simulated workload runs stably for a long time, OOM does not occur in the TiDB server, and the maximum memory usage of the process is stable at around 40.8 GiB:

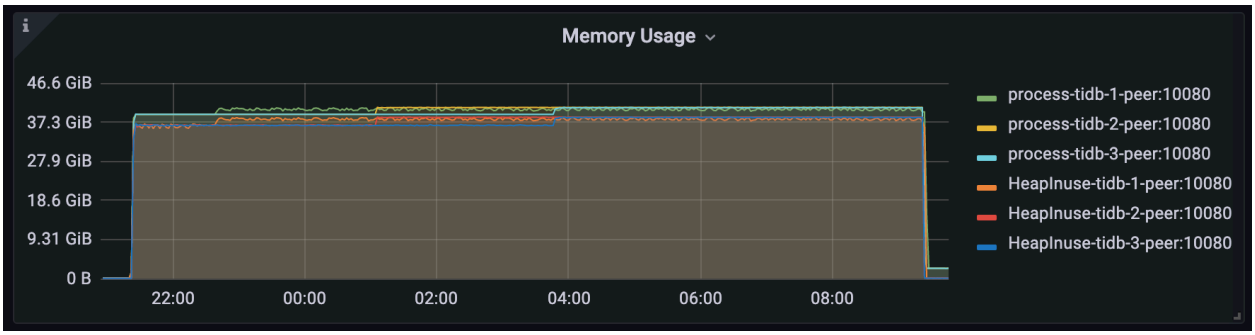


Figure 178: v6.1.3 workload no oom with GOMEMLIMIT

11.2.3 Tune TiKV Thread Pool Performance

This document introduces TiKV internal thread pools and how to tune their performance.

11.2.3.1 Thread pool introduction

The TiKV thread pool is mainly composed of gRPC, Scheduler, UnifyReadPool, Raftstore, StoreWriter, Apply, RocksDB, and some scheduled tasks and detection components that do not consume much CPU. This document mainly introduces a few CPU-intensive thread pools that affect the performance of read and write requests.

- The gRPC thread pool: it handles all network requests and forwards requests of different task types to different thread pools.
- The Scheduler thread pool: it detects write transaction conflicts, converts requests like the two-phase commit, pessimistic locking, and transaction rollbacks into key-value pair arrays, and then sends them to the Raftstore thread for Raft log replication.
- The Raftstore thread pool:
    - It processes all Raft messages and the proposal to add a new log.
    - It writes Raft logs to the disk. If the value of `store-io-pool-size` is `0`, the Raftstore thread writes the logs to the disk; if the value is not `0`, the Raftstore thread sends the logs to the StoreWriter thread.
    - When Raft logs in the majority of replicas are consistent, the Raftstore thread sends the logs to the Apply thread.
- The StoreWriter thread pool: it writes all Raft logs to the disk and returns the result to the Raftstore thread.
- The Apply thread pool: it receives the submitted log sent from the Raftstore thread pool, parses it as a key-value request, then writes it to RocksDB, calls the callback

function to notify the gRPC thread pool that the write request is complete, and returns the result to the client.

- The RocksDB thread pool: it is a thread pool for RocksDB to compact and flush tasks. For RocksDB's architecture and `Compact` operation, refer to RocksDB: A Persistent Key-Value Store for Flash and RAM Storage.
- The UnifyReadPool thread pool: it is a combination of the Coprocessor thread pool and Storage Read Pool. All read requests such as kv get, kv batch get, raw kv get, and coprocessor are executed in this thread pool.

11.2.3.2 TiKV read-only requests

TiKV's read requests are divided into the following types:

- Simple queries that specify a certain row or several rows, running in the Storage Read Pool.
- Complex aggregate calculation and range queries, running in the Coprocessor Read Pool.

Starting from TiKV v5.0, all read requests use the unified thread pool for queries by default. If your TiKV cluster is upgraded from TiKV v4.0 and the `use-unified-pool` configuration of `readpool.storage` was set to `false` before the upgrade, all read requests continue using different thread pools after the upgrade. In this scenario, to make all read requests use the unified thread pool for queries, you can set the value of `readpool.storage` ↪ `.use-unified-pool` to `true`.

11.2.3.3 Performance tuning for TiKV thread pools

- The gRPC thread pool.

  The default size (configured by `server.grpc-concurrency`) of the gRPC thread pool is `5`. This thread pool has almost no computing overhead and is mainly responsible for network I/O and deserialization requests, so generally you do not need to adjust the default configuration.

  - If the machine deployed with TiKV has a small number (less than or equal to 8) of CPU cores, consider setting the `server.grpc-concurrency` configuration item to `2`.
  - If the machine deployed with TiKV has very high configuration, TiKV undertakes a large number of read and write requests, and the value of `gRPC poll CPU` that monitors Thread CPU on Grafana exceeds 80% of `server.grpc-concurrency` ↪ , then consider increasing the value of `server.grpc-concurrency` to keep the thread pool usage rate below 80% (that is, the metric on Grafana is lower than `80% * server.grpc-concurrency`).

- The Scheduler thread pool.

  When TiKV detects that the number of machine CPU cores is larger than or equal to 16, the default size (configured by `storage.scheduler-worker-pool-size`) of the Scheduler thread pool is `8`; when TiKV detects that the number of machine CPU cores is smaller than 16, the default size is `4`.

  This thread pool is mainly used to convert complex transaction requests into simple key-value read and write requests. However, **the Scheduler thread pool itself does not perform any write operation**.

  - If it detects a transaction conflict, then this thread pool returns the conflict result to the client in advance.
  - If no conflict is detected, then this thread pool merges the key-value requests that perform write operations into a Raft log and sends it to the Raftstore thread for Raft log replication.

  Generally speaking, to avoid excessive thread switching, it is best to ensure that the utilization rate of the Scheduler thread pool is between 50% and 75%. If the thread pool size is `8`, then it is recommended to keep `TiKV-Details.Thread CPU.scheduler worker CPU` on Grafana between 400% and 600%.

- The Raftstore thread pool.

  The Raftstore thread pool is the most complex thread pool in TiKV. The default size (configured by `raftstore.store-pool-size`) of this thread pool is `2`. For the StoreWriter thread pool, the default size (configured by `raftstore.store-io-pool-size`) is `0`.

  - When the size of the StoreWriter thread pool is 0, all write requests are written into RocksDB in the way of `fsync` by the Raftstore thread. In this case, it is recommended to tune the performance as follows:
    - Keep the overall CPU usage of the Raftstore thread below 60%. When the number of Raftstore threads is 2, keep the **TiKV-Details**, **Thread CPU**, **Raft store CPU** on Grafana below 120%. Due to I/O requests, the CPU usage of Raftstore threads in theory is always lower than 100%.
    - Do not increase the size of the Raftstore thread pool to improve write performance without careful consideration, because this might increase the disk burden and degrade performance.
  - When the size of the StoreWriter thread pool is not 0, all write requests are written into RocksDB in the way of `fsync` by the StoreWriter thread. In this case, it is recommended to tune the performance as follows:
    - Enable the StoreWriter thread pool ONLY when the overall CPU resources are sufficient. When the StoreWriter thread pool is enabled, keep the CPU usage of the StoreWriter thread and the Raftstore thread below 80%.

Compared with the case that the write requests are processed by the Raftstore thread, in theory, when the write requests are processed by the StoreWriter thread, write latency and the tail latency of data read are significantly reduced. However, as the write speed grows faster, the number of Raft logs increases accordingly. This can cause the CPU overhead of the Raftstore threads, the Apply threads, and the gRPC threads to increase. In this case, insufficient CPU resources might offset the tuning effect, and as a result, the write speed might become slower than before. Therefore, if the CPU resources are not sufficient, it is not recommended to enable the StoreWriter thread. Because the Raftstore thread sends most of the I/O requests to the StoreWriter thread, you need to keep the CPU usage of the Raftstore thread below 80%.

- In most cases, set the size of the StoreWriter thread pool to 1 or 2. This is because the size of the StoreWriter thread pool affects the number of Raft logs, so the value of the thread pool size should not be too large. If the CPU usage is higher than 80%, consider increasing the thread pool size.
- Pay attention to the impact of increasing Raft logs on the CPU overhead of other thread pools. If necessary, you need to increase the number of Raftstore threads, Apply threads, and gRPC threads accordingly.

- The UnifyReadPool thread pool.

  The UnifyReadPool is responsible for handling all read requests. The default size (configured by `readpool.unified.max-thread-count`) is 80% of the number of the machine's CPU cores. For example, if the machine CPU has 16 cores, the default thread pool size is 12. It is recommended to adjust the CPU usage rate according to the application workloads and keep it between 60% and 90% of the thread pool size.

  If the peak value of the `TiKV-Details.Thread CPU.Unified read pool CPU` on Grafana does not exceed 800%, then it is recommended to set `readpool.unified` ↪ `.max-thread-count` to `10`. Too many threads can cause more frequent thread switching, and take up resources of other thread pools.

- The RocksDB thread pool.

  The RocksDB thread pool is a thread pool for RocksDB to compact and flush tasks. Usually, you do not need to configure it.

  - If the machine has a small number of CPU cores, set both `rocksdb.max-` ↪ `background-jobs` and `raftdb.max-background-jobs` to `4`.
  - If you encounter write stall, go to Write Stall Reason in **RocksDB-kv** on Grafana and check on the metrics that are not `0`.
    - If it is caused by reasons related to pending compaction bytes, set `rocksdb` ↪ `.max-sub-compactions` to `2` or `3`. This configuration item indicates the number of sub-threads allowed for a single compaction job. Its default value is `3` in TiKV 4.0 and `1` in TiKV 3.0.

- If the reason is related to memtable count, it is recommended to increase the `max-write-buffer-number` of all columns (`5` by default).
- If the reason is related to the level0 file limit, it is recommended to increase values of the following parameters to `64` or a larger number:

```
rocksdb.defaultcf.level0-slowdown-writes-trigger
rocksdb.writecf.level0-slowdown-writes-trigger
rocksdb.lockcf.level0-slowdown-writes-trigger
rocksdb.defaultcf.level0-stop-writes-trigger
rocksdb.writecf.level0-stop-writes-trigger
rocksdb.lockcf.level0-stop-writes-trigger
```

11.2.4 Tune TiKV Memory Parameter Performance

This document describes how to tune the TiKV parameters for optimal performance. You can find the default configuration file in etc/config-template.toml. To modify the configuration, you can use TiUP or modify TiKV dynamically for a limited set of configuration items. For the complete configuration, see TiKV configuration file.

TiKV uses RocksDB for persistent storage at the bottom level of the TiKV architecture. Therefore, many of the performance parameters are related to RocksDB. TiKV uses two RocksDB instances: the default RocksDB instance stores KV data, the Raft RocksDB instance (RaftDB) stores Raft logs.

TiKV implements `Column Families` (CF) from RocksDB.

- The default RocksDB instance stores KV data in the `default`, `write` and `lock` CFs.
  - The `default` CF stores the actual data. The corresponding parameters are in `[rocksdb.defaultcf]`.
  - The `write` CF stores the version information in Multi-Version Concurrency Control (MVCC) and index-related data. The corresponding parameters are in `[rocksdb.writecf]`.
  - The `lock` CF stores the lock information. The system uses the default parameters.
- The Raft RocksDB (RaftDB) instance stores Raft logs.
  - The `default` CF stores the Raft log. The corresponding parameters are in `[raftdb.defaultcf]`.

After TiKV 3.0, by default, all CFs share one block cache instance. You can configure the size of the cache by setting the `capacity` parameter under `[storage.block-cache]`. The bigger the block cache, the more hot data can be cached, and the easier to read data, in the meantime, the more system memory is occupied. To use a separate block cache instance for each CF, set `shared=false` under `[storage.block-cache]`, and configure individual

block cache size for each CF. For example, you can configure the size of `write` CF by setting the `block-cache-size` parameter under `[rocksdb.writecf]`.

Before TiKV 3.0, shared block cache is not supported, and you need to configure block cache for each CF individually.

Each CF also has a separate `write buffer`. You can configure the size by setting the `write-buffer-size` parameter.

11.2.4.1 Parameter specification

```
### Log level: trace, debug, warn, error, info, off.
log-level = "info"

[server]
### Set listening address
### addr = "127.0.0.1:20160"

### Size of thread pool for gRPC
### grpc-concurrency = 4
### The number of gRPC connections between each TiKV instance
### grpc-raft-conn-num = 10

### Most read requests from TiDB are sent to the coprocessor of TiKV. This
   ↪ parameter is used to set the number of threads
### of the coprocessor. If many read requests exist, add the number of
   ↪ threads and keep the number within that of the
### system CPU cores. For example, for a 32-core machine deployed with TiKV,
   ↪  you can even set this parameter to 30 in
### repeatable read scenarios. If this parameter is not set, TiKV
   ↪ automatically sets it to CPU cores * 0.8.
### end-point-concurrency = 8

### Tag the TiKV instances to schedule replicas.
### labels = {zone = "cn-east-1", host = "118", disk = "ssd"}

[storage]
### The data directory
### data-dir = "/tmp/tikv/store"

### In most cases, you can use the default value. When importing data, it is
   ↪  recommended to set the parameter to 1024000.
### scheduler-concurrency = 102400
### This parameter controls the number of write threads. When write
   ↪ operations occur frequently, set this parameter value
### higher. Run `top -H -p tikv-pid` and if the threads named `sched-worker-
```

```
   ↪ pool` are busy, set the value of parameter
### `scheduler-worker-pool-size` higher and increase the number of write
   ↪ threads.
### scheduler-worker-pool-size = 4

[storage.block-cache]
#### Whether to create a shared block cache for all RocksDB column families.
#### ## Block cache is used by RocksDB to cache uncompressed blocks. Big
   ↪ block cache can speed up read.
#### It is recommended to turn on shared block cache. Since only the total
   ↪ cache size need to be
#### set, it is easier to configure. In most cases, it should be able to
   ↪ auto-balance cache usage
#### between column families with standard LRU algorithm.
#### ## The rest of config in the storage.block-cache session is effective
   ↪ only when shared block cache
#### is on.
### shared = true

#### Size of the shared block cache. Normally it should be tuned to 30%-50%
   ↪ of system's total memory.
#### When the config is not set, it is decided by the sum of the following
   ↪ fields or their default
#### value:
####  * rocksdb.defaultcf.block-cache-size or 25% of system's total memory
####  * rocksdb.writecf.block-cache-size or 15% of system's total memory
####  * rocksdb.lockcf.block-cache-size or 2% of system's total memory
####  * raftdb.defaultcf.block-cache-size or 2% of system's total memory
#### ## To deploy multiple TiKV nodes on a single physical machine,
   ↪ configure this parameter explicitly.
#### Otherwise, the OOM problem might occur in TiKV.
### capacity = "1GB"

[pd]
### PD address
### endpoints = ["127.0.0.1:2379","127.0.0.2:2379","127.0.0.3:2379"]

[metric]
### The interval of pushing metrics to Prometheus Pushgateway
interval = "15s"
### Prometheus Pushgateway address
address = ""
job = "tikv"

[raftstore]
```

```
### Raft RocksDB directory. The default value is Raft subdirectory of [
    ↪ storage.data-dir].
### If there are multiple disks on the machine, store the data of Raft
    ↪ RocksDB on different disks to improve TiKV performance.
### raftdb-path = "/tmp/tikv/store/raft"

### When the data size change in a Region is larger than the threshold value
    ↪ , TiKV checks whether this Region needs split.
### To reduce the costs of scanning data in the checking process, set the
    ↪ value to 32 MB during the data import process. In the normal
    ↪ operation status, set it to the default value.
region-split-check-diff = "32MB"

[coprocessor]
#### If the size of a Region with the range of [a,e) is larger than the
    ↪ value of `region_max_size`, TiKV trys to split the Region to several
    ↪ Regions, for example, the Regions with the ranges of [a,b), [b,c), [c
    ↪ ,d), and [d,e).
#### After the Region split, the size of the split Regions is equal to the
    ↪ value of `region_split_size` (or slightly larger than the value of `
    ↪ region_split_size`).
### region-max-size = "144MB"
### region-split-size = "96MB"

[rocksdb]
### The maximum number of threads of RocksDB background tasks. The
    ↪ background tasks include compaction and flush.
### For detailed information why RocksDB needs to implement compaction, see
    ↪ RocksDB-related materials. When write
### traffic (like the importing data size) is big, it is recommended to
    ↪ enable more threads. But set the number of the enabled
### threads smaller than that of CPU cores. For example, when importing data
    ↪ , for a machine with a 32-core CPU,
### set the value to 28.
### max-background-jobs = 8

### The maximum number of file handles RocksDB can open
### max-open-files = 40960

### The file size limit of RocksDB MANIFEST. For more details, see https://
    ↪ github.com/facebook/rocksdb/wiki/MANIFEST
max-manifest-file-size = "20MB"

### The directory of RocksDB write-ahead logs. If there are two disks on the
    ↪  machine, store the RocksDB data and WAL logs
```

```
### on different disks to improve TiKV performance.
### wal-dir = "/tmp/tikv/store"

### Use the following two parameters to deal with RocksDB archiving WAL.
### For more details, see https://github.com/facebook/rocksdb/wiki/How-to-
    ↪ persist-in-memory-RocksDB-database%3F
### wal-ttl-seconds = 0
### wal-size-limit = 0

### In most cases, set the maximum total size of RocksDB WAL logs to the
    ↪ default value.
### max-total-wal-size = "4GB"

### Use this parameter to enable or disable the statistics of RocksDB.
### enable-statistics = true

### Use this parameter to enable the readahead feature during RocksDB
    ↪ compaction. If you are using mechanical disks, it is recommended to
    ↪ set the value to 2MB at least.
### compaction-readahead-size = "2MB"

[rocksdb.defaultcf]
### The data block size. RocksDB compresses data based on the unit of block.
### Similar to page in other databases, block is the smallest unit cached in
    ↪  block-cache.
block-size = "64KB"

### The compaction mode of each layer of RocksDB data. The optional values
    ↪ include no, snappy, zlib,
### bzip2, lz4, lz4hc, and zstd.
### "no:no:lz4:lz4:lz4:zstd:zstd" indicates there is no compaction of level0
    ↪  and level1; lz4 compaction algorithm is used
### from level2 to level4; zstd compaction algorithm is used from level5 to
    ↪ level6.
### "no" means no compaction. "lz4" is a compaction algorithm with moderate
    ↪ speed and compaction ratio. The
### compaction ratio of zlib is high. It is friendly to the storage space,
    ↪ but its compaction speed is slow. This
### compaction occupies many CPU resources. Different machines deploy
    ↪ compaction modes according to CPU and I/O resources.
### For example, if you use the compaction mode of "no:no:lz4:lz4:lz4:zstd:
    ↪ zstd" and find much I/O pressure of the
### system (run the iostat command to find %util lasts 100%, or run the top
    ↪ command to find many iowaits) when writing
### (importing) a lot of data while the CPU resources are adequate, you can
```

```
    ↪ compress level0 and level1 and exchange CPU
### resources for I/O resources. If you use the compaction mode of "no:no:
    ↪ lz4:lz4:lz4:zstd:zstd" and you find the I/O
### pressure of the system is not big when writing a lot of data, but CPU
    ↪ resources are inadequate. Then run the top
### command and choose the -H option. If you find a lot of bg threads (
    ↪ namely the compaction thread of RocksDB) are
### running, you can exchange I/O resources for CPU resources and change the
    ↪  compaction mode to "no:no:no:lz4:lz4:zstd:zstd".
### In a word, it aims at making full use of the existing resources of the
    ↪ system and improving TiKV performance
### in terms of the current resources.
compression-per-level = ["no", "no", "lz4", "lz4", "lz4", "zstd", "zstd"]

### The RocksDB memtable size
write-buffer-size = "128MB"

### The maximum number of the memtables. The data written into RocksDB is
    ↪ first recorded in the WAL log, and then inserted
### into memtables. When the memtable reaches the size limit of `write-
    ↪ buffer-size`, it turns into read only and generates
### a new memtable receiving new write operations. The flush threads of
    ↪ RocksDB will flush the read only memtable to the
### disks to become an sst file of level0. `max-background-flushes` controls
    ↪  the maximum number of flush threads. When the
### flush threads are busy, resulting in the number of the memtables waiting
    ↪  to be flushed to the disks reaching the limit
### of `max-write-buffer-number`, RocksDB stalls the new operation.
### "Stall" is a flow control mechanism of RocksDB. When importing data, you
    ↪  can set the `max-write-buffer-number` value
### higher, like 10.
max-write-buffer-number = 5

### When the number of sst files of level0 reaches the limit of `level0-
    ↪ slowdown-writes-trigger`, RocksDB
### tries to slow down the write operation, because too many sst files of
    ↪ level0 can cause higher read pressure of
### RocksDB. `level0-slowdown-writes-trigger` and `level0-stop-writes-
    ↪ trigger` are for the flow control of RocksDB.
### When the number of sst files of level0 reaches 4 (the default value),
    ↪ the sst files of level0 and the sst files
### of level1 which overlap those of level0 implement compaction to relieve
    ↪ the read pressure.
level0-slowdown-writes-trigger = 20
```

```
### When the number of sst files of level0 reaches the limit of `level0-stop
    ↪ -writes-trigger`, RocksDB stalls the new
### write operation.
level0-stop-writes-trigger = 36

### When the level1 data size reaches the limit value of `max-bytes-for-
    ↪ level-base`, the sst files of level1
### and their overlap sst files of level2 implement compaction. The golden
    ↪ rule: the first reference principle
### of setting `max-bytes-for-level-base` is guaranteeing that the `max-
    ↪ bytes-for-level-base` value is roughly equal to the
### data volume of level0. Thus unnecessary compaction is reduced. For
    ↪ example, if the compaction mode is
### "no:no:lz4:lz4:lz4:lz4:lz4", the `max-bytes-for-level-base` value is
    ↪ write-buffer-size * 4, because there is no
### compaction of level0 and level1 and the trigger condition of compaction
    ↪ for level0 is that the number of the
### sst files reaches 4 (the default value). When both level0 and level1
    ↪ adopt compaction, it is necessary to analyze
### RocksDB logs to know the size of an sst file compressed from an mentable
    ↪ . For example, if the file size is 32MB,
### the proposed value of `max-bytes-for-level-base` is 32MB * 4 = 128MB.
max-bytes-for-level-base = "512MB"

### The sst file size. The sst file size of level0 is influenced by the
    ↪ compaction algorithm of `write-buffer-size`
### and level0. `target-file-size-base` is used to control the size of a
    ↪ single sst file of level1-level6.
target-file-size-base = "32MB"

[rocksdb.writecf]
### Set it the same as `rocksdb.defaultcf.compression-per-level`.
compression-per-level = ["no", "no", "lz4", "lz4", "lz4", "zstd", "zstd"]

### Set it the same as `rocksdb.defaultcf.write-buffer-size`.
write-buffer-size = "128MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1

### Set it the same as `rocksdb.defaultcf.max-bytes-for-level-base`.
max-bytes-for-level-base = "512MB"
target-file-size-base = "32MB"

[raftdb]
### The maximum number of the file handles RaftDB can open
```

```
### max-open-files = 40960

### Configure this parameter to enable or disable the RaftDB statistics
    ↪ information.
### enable-statistics = true

### Enable the readahead feature in RaftDB compaction. If you are using
    ↪ mechanical disks, it is recommended to set
### this value to 2MB at least.
### compaction-readahead-size = "2MB"

[raftdb.defaultcf]
### Set it the same as `rocksdb.defaultcf.compression-per-level`.
compression-per-level = ["no", "no", "lz4", "lz4", "lz4", "zstd", "zstd"]

### Set it the same as `rocksdb.defaultcf.write-buffer-size`.
write-buffer-size = "128MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1

### Set it the same as `rocksdb.defaultcf.max-bytes-for-level-base`.
max-bytes-for-level-base = "512MB"
target-file-size-base = "32MB"
```

11.2.4.2 TiKV memory usage

Besides `block cache` and `write buffer` which occupy the system memory, the system memory is occupied in the following scenarios:

- Some of the memory is reserved as the system's page cache.
- When TiKV processes large queries such as `select * from ...`, it reads data, generates the corresponding data structure in the memory, and returns this structure to TiDB. During this process, TiKV occupies some of the memory.

11.2.4.3 Recommended configuration of TiKV

- In production environments, it is not recommended to deploy TiKV on the machine whose CPU cores are less than 8 or the memory is less than 32GB.
- If you demand a high write throughput, it is recommended to use a disk with good throughput capacity.
- If you demand a very low read-write latency, it is recommended to use SSD with high IOPS.

11.2.5 Follower Read

When a read hotspot appears in a Region, the Region leader can become a read bottleneck for the entire system. In this situation, enabling the Follower Read feature can significantly reduce the load of the leader, and improve the throughput of the whole system by balancing the load among multiple followers. This document introduces the use and implementation mechanism of Follower Read.

11.2.5.1 Overview

The Follower Read feature refers to using any follower replica of a Region to serve a read request under the premise of strongly consistent reads. This feature improves the throughput of the TiDB cluster and reduces the load of the leader. It contains a series of load balancing mechanisms that offload TiKV read loads from the leader replica to the follower replica in a Region. TiKV's Follower Read implementation provides users with strongly consistent reads.

> **Note:**
>
> To achieve strongly consistent reads, the follower node currently needs to request the current execution progress from the leader node (that is `ReadIndex`), which causes an additional network request overhead. Therefore, the main benefits of Follower Read are to isolate read requests from write requests in the cluster and to increase overall read throughput.

11.2.5.2 Usage

To enable TiDB's Follower Read feature, modify the value of the `tidb_replica_read` variable to `follower` or `leader-and-follower`:

```
set [session | global] tidb_replica_read = '<target value>';
```

Scope: SESSION | GLOBAL

Default: leader

This variable is used to set the expected data read mode.

- When the value of `tidb_replica_read` is set to `leader` or an empty string, TiDB maintains its original behavior and sends all read operations to the leader replica to perform.
- When the value of `tidb_replica_read` is set to `follower`, TiDB selects a follower replica of the Region to perform all read operations.
- When the value of `tidb_replica_read` is set to `leader-and-follower`, TiDB can select any replicas to perform read operations.

11.2.5.3 Implementation mechanism

Before the Follower Read feature was introduced, TiDB applied the strong leader principle and submitted all read and write requests to the leader node of a Region to handle. Although TiKV can distribute Regions evenly on multiple physical nodes, for each Region, only the leader can provide external services. The other followers can do nothing to handle read requests but receive the data replicated from the leader at all times and prepare for voting to elect a leader in case of a failover.

To allow data reading in the follower node without violating linearizability or affecting Snapshot Isolation in TiDB, the follower node needs to use `ReadIndex` of the Raft protocol to ensure that the read request can read the latest data that has been committed on the leader. At the TiDB level, the Follower Read feature simply needs to send the read request of a Region to a follower replica based on the load balancing policy.

11.2.5.3.1 Strongly consistent reads

When the follower node processes a read request, it first uses `ReadIndex` of the Raft protocol to interact with the leader of the Region, to obtain the latest commit index of the current Raft group. After the latest commit index of the leader is applied locally to the follower, the processing of a read request starts.

11.2.5.3.2 Follower replica selection strategy

Because the Follower Read feature does not affect TiDB's Snapshot Isolation transaction isolation level, TiDB adopts the round-robin strategy to select the follower replica. Currently, for the coprocessor requests, the granularity of the Follower Read load balancing policy is at the connection level. For a TiDB client connected to a specific Region, the selected follower is fixed, and is switched only when it fails or the scheduling policy is adjusted.

However, for the non-coprocessor requests, such as a point query, the granularity of the Follower Read load balancing policy is at the transaction level. For a TiDB transaction on a specific Region, the selected follower is fixed, and is switched only when it fails or the scheduling policy is adjusted.

11.2.6 Tune Region Performance

This document introduces how to tune Region performance by adjusting the Region size and how to use bucket to optimize concurrent queries when the Region size is large.

11.2.6.1 Overview

TiKV automatically shards bottom-layered data. Data is split into multiple Regions based on the key ranges. When the size of a Region exceeds a threshold, TiKV splits it into two or more Regions.

In scenarios involving large datasets, if the Region size is relatively small, TiKV might have too many Regions, which causes more resource consumption and performance regression. Since v6.1.0, TiDB supports customizing Region size. The default size of a Region is 96 MiB. To reduce the number of Regions, you can adjust Regions to a larger size.

To reduce the performance overhead of many Regions, you can also enable Hibernate Region or `Region Merge`.

11.2.6.2 Use `region-split-size` to adjust Region size

> **Note:**
>
> The recommended range for the Region size is [48 MiB, 256 MiB]. Commonly used sizes include 96 MiB, 128 MiB, and 256 MiB. It is NOT recommended to set the Region size beyond 1 GiB. Avoid setting the size to more than 10 GiB. An excessively large Region size might result in the following side effects:
>
> - Performance jitters
> - Decreased query performance, especially for queries that deal with a large range of data
> - Slower Region scheduling

To adjust the Region size, you can use the `coprocessor.region-split-size` configuration item. When TiFlash is used, the Region size should not exceed 256 MiB.

When the Dumpling tool is used, the Region size should not exceed 1 GiB. In this case, you need to reduce the concurrency after increasing the Region size; otherwise, TiDB might run out of memory.

11.2.6.3 Use bucket to increase concurrency

> **Warning:**
>
> Currently, this is an experimental feature introduced in TiDB v6.1.0. It is not recommended that you use it in production environments.

After Regions are set to a larger size, if you want to further improve the query concurrency, you can set `coprocessor.enable-region-bucket` to `true` to increase the query concurrency. When you use this configuration, Regions are divided into buckets. Buckets are smaller ranges within a Region and are used as the unit of concurrent query to improve

the scan concurrency. You can control the bucket size using `coprocessor.region-bucket-size`. The default value is `96MiB`.

11.2.7 Tune TiFlash Performance

This document introduces how to tune the performance of TiFlash, including planning machine resources and tuning TiDB parameters.

11.2.7.1 Plan resources

If you want to save machine resources and have no requirement on isolation, you can use the method that combines the deployment of both TiKV and TiFlash. It is recommended that you save enough resources for TiKV and TiFlash respectively, and do not share disks.

11.2.7.2 Tune TiDB parameters

1. For the TiDB node dedicated to OLAP/TiFlash, it is recommended that you increase the value of the `tidb_distsql_scan_concurrency` configuration item for this node to `80`:

```
set @@tidb_distsql_scan_concurrency = 80;
```

2. Enable the super batch feature:

   You can use the `tidb_allow_batch_cop` variable to set whether to merge Region requests when reading from TiFlash.

   When the number of Regions involved in the query is relatively large, try to set this variable to `1` (effective for coprocessor requests with `aggregation` operators that are pushed down to TiFlash), or set this variable to `2` (effective for all coprocessor requests that are pushed down to TiFlash).

```
set @@tidb_allow_batch_cop = 1;
```

3. Enable the optimization of pushing down aggregate functions before TiDB operators such as `JOIN` or `UNION`:

   You can use the `tidb_opt_agg_push_down` variable to control the optimizer to execute this optimization. When the aggregate operations are quite slow in the query, try to set this variable to `1`.

```
set @@tidb_opt_agg_push_down = 1;
```

4. Enable the optimization of pushing down aggregate functions with `Distinct` before TiDB operators such as `JOIN` or `UNION`:

   You can use the `tidb_opt_distinct_agg_push_down` variable to control the optimizer to execute this optimization. When the aggregate operations with `Distinct` are quite slow in the query, try to set this variable to `1`.

```
set @@tidb_opt_distinct_agg_push_down = 1;
```

11.2.8 Coprocessor Cache

Starting from v4.0, the TiDB instance supports caching the results of the calculation that is pushed down to TiKV (the Coprocessor Cache feature), which can accelerate the calculation process in some scenarios.

11.2.8.1 Configuration

You can configure Coprocessor Cache via the `tikv-client.copr-cache` configuration items in the TiDB configuration file. For details about how to enable and configure Coprocessor Cache, see TiDB Configuration File.

11.2.8.2 Feature description

- When a SQL statement is executed on a single TiDB instance for the first time, the execution result is not cached.
- Calculation results are cached in the memory of TiDB. If the TiDB instance is restarted, the cache becomes invalid.
- The cache is not shared among TiDB instances.
- Only push-down calculation result is cached. Even if cache is hit, TiDB still need to perform subsequent calculation.
- The cache is in the unit of Region. Writing data to a Region causes the Region cache to be invalid. For this reason, the Coprocessor Cache feature mainly takes effect on the data that rarely changes.
- When push-down calculation requests are the same, the cache is hit. Usually in the following scenarios, the push-down calculation requests are the same or partially the same:
    - The SQL statements are the same. For example, the same SQL statement is executed repeatedly.

      In this scenario, all the push-down calculation requests are consistent, and all requests can use the push-down calculation cache.
    - The SQL statements contain a changing condition, and the other parts are consistent. The changing condition is the primary key of the table or the partition.

      In this scenario, some of the push-down calculation requests are the same with some previous requests, and these calculation requests can use the cached (previous) push-down calculation result.
    - The SQL statements contain multiple changing conditions and the other parts are consistent. The changing conditions exactly match a compound index column.

In this scenario, some of the push-down calculation requests are the same with some previous requests, and these calculation requests can use the cached (previous) push-down calculation result.

- This feature is transparent to users. Enabling or disabling this feature does not affect the calculation result and only affects the SQL execution time.

11.2.8.3 Check the cache effect

You can check the cache effect of Coprocessor by executing `EXPLAIN ANALYZE` or viewing the Grafana monitoring panel.

11.2.8.3.1 Use `EXPLAIN ANALYZE`

You can view the cache hit rate in Operators for accessing tables by using the `EXPLAIN ANALYZE` statement. See the following example:

```
EXPLAIN ANALYZE SELECT * FROM t USE INDEX(a);
+--
   ↪ ----------------------------+----------+---------+----------+---------------
   ↪
| id                          | estRows  | actRows | task    | access object
   ↪       | execution info
   ↪
   ↪ | operator info              | memory             | disk |
+--
   ↪ ----------------------------+----------+---------+----------+---------------
   ↪
| IndexLookUp_6               | 262400.00 | 262400 | root   |
   ↪                      | time:620.513742ms, loops:258, cop_task: {num:
   ↪ 4, max: 5.530817ms, min: 1.51829ms, avg: 2.70883ms, p95: 5.530817ms,
   ↪ max_proc_keys: 2480, p95_proc_keys: 2480, tot_proc: 1ms, tot_wait: 1
   ↪ ms, rpc_num: 4, rpc_time: 10.816328ms, copr_cache_hit_rate: 0.75} | |
   ↪  6.685169219970703 MB | N/A |
|  ─IndexFullScan_4(Build)   | 262400.00 | 262400 | cop[tikv] | table:t,
   ↪ index:a(a, c) | proc max:93ms, min:1ms, p80:93ms, p95:93ms, iters
   ↪ :275, tasks:4
   ↪
   ↪ | keep order:false, stats:pseudo | 1.7549400329589844 MB | N/A |
|  ─TableRowIDScan_5(Probe)  | 262400.00 | 0     | cop[tikv] | table:t
   ↪           | time:0ns, loops:0
   ↪
   ↪ | keep order:false, stats:pseudo | N/A            | N/A |
+--
   ↪ ----------------------------+----------+---------+----------+---------------
   ↪
```

```
3 rows in set (0.62 sec)
```

The column `execution info` of the execution result gives the `copr_cache_hit_ratio` information, which indicates the hit rate of the Coprocessor Cache. The `0.75` in the above example means that the hit rate is about 75%.

11.2.8.3.2 View the Grafana monitoring panel

In Grafana, you can see the **copr-cache** panel in the `distsql` subsystem under the `tidb` namespace. This panel monitors the number of hits, misses, and cache discards of the Coprocessor Cache in the entire cluster.

11.2.9 Garbage Collection (GC)

11.2.9.1 GC Overview

TiDB uses MVCC to control transaction concurrency. When you update the data, the original data is not deleted immediately but is kept together with the new data, with a timestamp to distinguish the version. The goal of Garbage Collection (GC) is to clear the obsolete data.

11.2.9.1.1 GC process

Each TiDB cluster contains a TiDB instance that is selected as the GC leader, which controls the GC process.

GC runs periodically on TiDB. For each GC, TiDB firstly calculates a timestamp called "safe point". Then, TiDB clears the obsolete data under the premise that all the snapshots after the safe point retain the integrity of the data. Specifically, there are three steps involved in each GC process:

1. Resolve Locks. During this step, TiDB scans locks before the safe point on all Regions and clears these locks.
2. Delete Ranges. During this step, the obsolete data of the entire range generated from the `DROP TABLE`/`DROP INDEX` operation is quickly cleared.
3. Do GC. During this step, each TiKV node scans data on it and deletes unneeded old versions of each key.

In the default configuration, GC is triggered every 10 minutes. Each GC retains data of the recent 10 minutes, which means that the GC life time is 10 minutes by default (safe point = the current time - GC life time). If one round of GC has been running for too long, before this round of GC is completed, the next round of GC will not start even if it is time to trigger the next GC. In addition, for long-duration transactions to run properly after exceeding the GC life time, the safe point does not exceed the start time (start_ts) of the ongoing transactions.

11.2.9.1.2 Implementation details

Resolve Locks

The TiDB transaction model is implemented based on Google's Percolator. It's mainly a two-phase commit protocol with some practical optimizations. When the first phase is finished, all the related keys are locked. Among these locks, one is the primary lock and the others are secondary locks which contain a pointer to the primary lock; in the second phase, the key with the primary lock gets a write record and its lock is removed. The write record indicates the write or delete operation in the history or the transactional rollback record of this key. The type of write record that replaces the primary lock indicates whether the corresponding transaction is committed successfully. Then all the secondary locks are replaced successively. If, for some reason such as failure, these secondary locks are retained and not replaced, you can still find the primary key based on the information in the secondary locks and determines whether the entire transaction is committed based on whether the primary key is committed. However, if the primary key information is cleared by GC and this transaction has uncommitted secondary locks, you will never learn whether these locks can be committed. As a result, data integrity cannot be guaranteed.

The Resolve Locks step clears the locks before the safe point. This means that if the primary key of a lock is committed, this lock needs to be committed; otherwise, it needs to be rolled back. If the primary key is still locked (not committed or rolled back), this transaction is seen as timing out and rolled back.

The Resolve Locks step is implemented in either of the following two ways, which can be configured using the system variable `tidb_gc_scan_lock_mode`:

> **Warning:**
>
> Currently, `PHYSICAL` (Green GC) is an experimental feature. It is not recommended that you use it in the production environment.

- `LEGACY` (default): The GC leader sends requests to all Regions to scan obsolete locks, checks the primary key statuses of scanned locks, and sends requests to commit or roll back the corresponding transaction.
- `PHYSICAL`: TiDB bypasses the Raft layer and directly scans data on each TiKV node.

Delete Ranges

A great amount of data with consecutive keys is removed during operations such as `DROP TABLE/INDEX`. Removing each key and performing GC later for them can result in low execution efficiency on storage reclaiming. In such scenarios, TiDB actually does not delete each key. Instead, it only records the range to be removed and the timestamp of the deletion. Then the Delete Ranges step performs a fast physical deletion on the ranges whose timestamp is before the safe point.

Do GC

The Do GC step clears the outdated versions for all keys. To guarantee that all timestamps after the safe point have consistent snapshots, this step deletes the data committed before the safe point, but retains the last write for each key before the safe point as long as it is not a deletion.

In this step, TiDB only needs to send the safe point to PD, and then the whole round of GC is completed. TiKV automatically detects the change of safe point and performs GC for all Region leaders on the current node. At the same time, the GC leader can continue to trigger the next round of GC.

> **Note:**
>
> Starting with TiDB 5.0, the Do GC step will always use the `DISTRIBUTED` gc mode. This replaces the earlier `CENTRAL` gc mode, which was implemented by TiDB servers sending GC requests to each Region.

11.2.9.2 Garbage Collection Configuration

Garbage collection is configured via the following system variables:

- `tidb_gc_enable`
- `tidb_gc_run_interval`
- `tidb_gc_life_time`
- `tidb_gc_concurrency`
- `tidb_gc_scan_lock_mode`
- `tidb_gc_max_wait_time`

11.2.9.2.1 GC I/O limit

TiKV supports the GC I/O limit. You can configure `gc.max-write-bytes-per-sec` to limit writes of a GC worker per second, and thus to reduce the impact on normal requests.

`0` indicates disabling this feature.

You can dynamically modify this configuration using tikv-ctl:

```
tikv-ctl --host=ip:port modify-tikv-config -n gc.max-write-bytes-per-sec -v
    ↪ 10MB
```

11.2.9.2.2 Changes in TiDB 5.0

In previous releases of TiDB, garbage collection was configured via the `mysql.tidb` system table. While changes to this table continue to be supported, it is recommended to

use the system variables provided. This helps ensure that any changes to configuration can be validated, and prevent unexpected behavior (#20655).

The `CENTRAL` garbage collection mode is no longer supported. The `DISTRIBUTED` GC mode (which has been the default since TiDB 3.0) will automatically be used in its place. This mode is more efficient, since TiDB no longer needs to send requests to each TiKV region to trigger garbage collection.

For information on changes in previous releases, refer to earlier versions of this document using the *TIDB version selector* in the left hand menu.

11.2.9.2.3 Changes in TiDB 6.1.0

Before TiDB v6.1.0, the transaction in TiDB does not affect the GC safe point. Since v6.1.0, TiDB considers the startTS of the transaction when calculating the GC safe point, to resolve the problem that the data to be accessed has been cleared. If the transaction is too long, the safe point will be blocked for a long time, which affects the application performance.

In TiDB v6.1.0, the system variable `tidb_gc_max_wait_time` is introduced to control the maximum time that active transactions block the GC safe point. After the value is exceeded, the GC safe point is forwarded forcefully.

GC in Compaction Filter

Based on the `DISTRIBUTED` GC mode, the mechanism of GC in Compaction Filter uses the compaction process of RocksDB, instead of a separate GC worker thread, to run GC. This new GC mechanism helps to avoid extra disk read caused by GC. Also, after clearing the obsolete data, it avoids a large number of left tombstone marks which degrade the sequential scan performance.

The following example shows how to enable the mechanism in the TiKV configuration file:

```
[gc]
enable-compaction-filter = true
```

You can also enable this GC mechanism by modifying the configuration dynamically. See the following example:

```
show config where type = 'tikv' and name like '%enable-compaction-filter%';
```

```
+------+------------------+-----------------------------+-------+
| Type | Instance         | Name                        | Value |
+------+------------------+-----------------------------+-------+
| tikv | 172.16.5.37:20163 | gc.enable-compaction-filter | false |
| tikv | 172.16.5.36:20163 | gc.enable-compaction-filter | false |
| tikv | 172.16.5.35:20163 | gc.enable-compaction-filter | false |
+------+------------------+-----------------------------+-------+
```

```
set config tikv gc.enable-compaction-filter = true;
show config where type = 'tikv' and name like '%enable-compaction-filter%';
```

```
+------+------------------+-----------------------------+-------+
| Type | Instance         | Name                        | Value |
+------+------------------+-----------------------------+-------+
| tikv | 172.16.5.37:20163 | gc.enable-compaction-filter | true |
| tikv | 172.16.5.36:20163 | gc.enable-compaction-filter | true |
| tikv | 172.16.5.35:20163 | gc.enable-compaction-filter | true |
+------+------------------+-----------------------------+-------+
```

11.3 SQL Tuning

11.3.1 SQL Tuning Overview

SQL is a declarative language. That is, an SQL statement describes *what the final result should look like* and not a set of steps to execute in sequence. TiDB will optimize the execution, and is semantically permitted to execute parts of the query in any order provided that it correctly returns the final result as described.

A useful comparison to SQL optimization, is to describe what happens when you use GPS navigation. From your provided address, *2955 Campus Drive San Mateo CA 94403*, the GPS software plans the most time-efficient way to route you. It may make use of various statistics such as previous trips, meta data such as speed limits, and in modern cases, a live feed of traffic information. Several of these analogies translate to TiDB.

This section introduces several concepts about query execution:

- Understanding the Query Execution Plan introduces how to use the `EXPLAIN` statement to understand how TiDB has decided to execute a statement.
- SQL Optimization Process introduces what optimizations TiDB is capable of using to improve query execution performance.
- Control Execution Plans introduces ways to control the generation of the execution plan. This can be useful in cases where the execution plan decided by TiDB is suboptimal.

11.3.2 Understanding the Query Execution Plan

11.3.2.1 TiDB Query Execution Plan Overview

> **Note:**
>
> When you use the MySQL client to connect to TiDB, to read the output result in a clearer way without line wrapping, you can use the `pager less -S` command. Then, after the `EXPLAIN` result is output, you can press the right arrow → button on your keyboard to horizontally scroll through the output.

SQL is a declarative language. It describes what the results of a query should look like, **not the methodology** to actually retrieve those results. TiDB considers all the possible ways in which a query could be executed, including using what order to join tables and whether any potential indexes can be used. The process of *considering query execution plans* is known as SQL optimization.

The `EXPLAIN` statement shows the selected execution plan for a given statement. That is, after considering hundreds or thousands of ways in which the query could be executed, TiDB believes that this *plan* will consume the least resources and execute in the shortest amount of time:

```
CREATE TABLE t (id INT NOT NULL PRIMARY KEY auto_increment, a INT NOT NULL,
    ↪  pad1 VARCHAR(255), INDEX(a));
INSERT INTO t VALUES (1, 1, 'aaa'),(2,2, 'bbb');
EXPLAIN SELECT * FROM t WHERE a = 1;
```

```
Query OK, 0 rows affected (0.96 sec)

Query OK, 2 rows affected (0.02 sec)
Records: 2 Duplicates: 0 Warnings: 0

+--
    ↪ ----------------------------+---------+----------+-------------------+------
    ↪
| id                          | estRows | task     | access object     |
    ↪ operator info                              |
+--
    ↪ ----------------------------+---------+----------+-------------------+------
    ↪
| IndexLookUp_10              | 10.00   | root     |                   |
    ↪                                          |
|  ─IndexRangeScan_8(Build)   | 10.00   | cop[tikv] | table:t, index:a(a) |
    ↪ range:[1,1], keep order:false, stats:pseudo |
|  ─TableRowIDScan_9(Probe)   | 10.00   | cop[tikv] | table:t          | keep
    ↪ order:false, stats:pseudo          |
```

```
+--
    ↪ ----------------------------+---------+-----------+--------------------+------
    ↪
3 rows in set (0.00 sec)
```

`EXPLAIN` does not execute the actual query. `EXPLAIN ANALYZE` can be used to execute the query and show `EXPLAIN` information. This can be useful in diagnosing cases where the execution plan selected is suboptimal. For additional examples of using `EXPLAIN`, see the following documents:

- Indexes
- Joins
- Subqueries
- Aggregation
- Views
- Partitions

11.3.2.1.1 Understand EXPLAIN output

The following describes the output of the `EXPLAIN` statement above:

- `id` describes the name of an operator, or sub-task that is required to execute the SQL statement. See Operator overview for additional details.
- `estRows` shows an estimate of the number of rows TiDB expects to process. This number might be based on dictionary information, such as when the access method is based on a primary or unique key, or it could be based on statistics such as a CMSketch or histogram.
- `task` shows where an operator is performing the work. See Task overview for additional details.
- `access object` shows the table, partition and index that is being accessed. The parts of the index are also shown, as in the case above that the column `a` from the index was used. This can be useful in cases where you have composite indexes.
- `operator info` shows additional details about the access. See Operator info overview for additional details.

Operator overview

An operator is a particular step that is executed as part of returning query results. The operators that perform table scans (of the disk or the TiKV Block Cache) are listed as follows:

- **TableFullScan**: Full table scan
- **TableRangeScan**: Table scans with the specified range
- **TableRowIDScan**: Scans the table data based on the RowID. Usually follows an index read operation to retrieve the matching data rows.

- **IndexFullScan**: Similar to a "full table scan", except that an index is scanned, rather than the table data.
- **IndexRangeScan**: Index scans with the specified range.

TiDB aggregates the data or calculation results scanned from TiKV/TiFlash. The data aggregation operators can be divided into the following categories:

- **TableReader**: Aggregates the data obtained by the underlying operators like `TableFullScan` or `TableRangeScan` in TiKV.
- **IndexReader**: Aggregates the data obtained by the underlying operators like `IndexFullScan` or `IndexRangeScan` in TiKV.
- **IndexLookUp**: First aggregates the RowID (in TiKV) scanned by the `Build` side. Then at the `Probe` side, accurately reads the data from TiKV based on these RowIDs. At the `Build` side, there are operators like `IndexFullScan` or `IndexRangeScan`; at the `Probe` side, there is the `TableRowIDScan` operator.
- **IndexMerge**: Similar to `IndexLookUp`. `IndexMerge` can be seen as an extension of `IndexLookupReader`. `IndexMerge` supports reading multiple indexes at the same time. There are many `Build`s and one `Probe`. The execution process of `IndexMerge` the same as that of `IndexLookUp`.

While the structure appears as a tree, executing the query does not strictly require the child nodes to be completed before the parent nodes. TiDB supports intra-query parallelism, so a more accurate way to describe the execution is that the child nodes *flow into* their parent nodes. Parent, child and sibling operators *might* potentially be executing parts of the query in parallel.

In the previous example, the `└─IndexRangeScan_8(Build)` operator finds the internal `RowID` for rows that match the `a(a)` index. The `└─TableRowIDScan_9(Probe)` operator then retrieves these rows from the table.

Range query

In the `WHERE/HAVING/ON` conditions, the TiDB optimizer analyzes the result returned by the primary key query or the index key query. For example, these conditions might include comparison operators of the numeric and date type, such as `>`, `<`, `=`, `>=`, `<=`, and the character type such as `LIKE`.

Note:

- In order to use an index, the condition must be *sargable*. For example, the condition `YEAR(date_column)< 1992` cannot use an index, but `date_column < '1992-01-01` can.

- It is recommended to compare data of the same type and character set and collation. Mixing types may require additional `cast` operations, or prevent indexes from being used.
- You can also use `AND` (intersection) and `OR` (union) to combine the range query conditions of one column. For a multi-dimensional composite index, you can use conditions in multiple columns. For example, regarding the composite index `(a, b, c)`:
  - When `a` is an equivalent query, continue to figure out the query range of `b`; when `b` is also an equivalent query, continue to figure out the query range of `c`.
  - Otherwise, if `a` is a non-equivalent query, you can only figure out the range of `a`.

Task overview

Currently, calculation tasks of TiDB can be divided into two categories: cop tasks and root tasks. A `cop[tikv]` task indicates that the operator is performed inside the TiKV coprocessor. A `root` task indicates that it will be completed inside of TiDB.

One of the goals of SQL optimization is to push the calculation down to TiKV as much as possible. The Coprocessor in TiKV supports most of the built-in SQL functions (including the aggregate functions and the scalar functions), SQL `LIMIT` operations, index scans, and table scans.

Operator info overview

The `operator info` can show useful information such as which conditions were able to be pushed down:

- `range: [1,1]` shows that the predicate from the where clause of the query (`a = 1`) was pushed right down to TiKV (the task is of `cop[tikv]`).
- `keep order:false` shows that the semantics of this query did not require TiKV to return the results in order. If the query were to be modified to require an order (such as `SELECT * FROM t WHERE a = 1 ORDER BY id`), then this condition would be `keep order:true`.
- `stats:pseudo` shows that the estimates shown in `estRows` might not be accurate. TiDB periodically updates statistics as part of a background operation. A manual update can also be performed by running `ANALYZE TABLE t`.

Different operators output different information after the `EXPLAIN` statement is executed. You can use optimizer hints to control the behavior of the optimizer, and thereby controlling the selection of the physical operators. For example, `/*+ HASH_JOIN(t1, t2)*/` means that the optimizer uses the `Hash Join` algorithm. For more details, see Optimizer Hints.

11.3.2.2 `EXPLAIN` Walkthrough

Because SQL is a declarative language, you cannot automatically tell whether a query is executed efficiently. You must first use the `EXPLAIN` statement to learn the current execution plan.

The following statement from the bikeshare example database counts how many trips were taken on July 1, 2017:

```
EXPLAIN SELECT count(*) FROM trips WHERE start_date BETWEEN '2017-07-01
    ↪ 00:00:00' AND '2017-07-01 23:59:59';
```

```
+--
    ↪ ----------------------------+----------+----------+---------------+-----------
    ↪
| id                          | estRows | task     | access object | operator
    ↪ info
    ↪
    ↪ |
+--
    ↪ ----------------------------+----------+----------+---------------+-----------
    ↪
| StreamAgg_20                | 1.00    | root     |               | funcs:count(
    ↪ Column#13)->Column#11
    ↪
    ↪ |
| └─TableReader_21            | 1.00    | root     |               | data:
    ↪ StreamAgg_9
    ↪
    ↪ |
|   └─StreamAgg_9             | 1.00    | cop[tikv] |              | funcs:count
    ↪ (1)->Column#13
    ↪
    ↪ |
|     └─Selection_19          | 250.00  | cop[tikv] |              | ge(
    ↪ bikeshare.trips.start_date, 2017-07-01 00:00:00.000000), le(bikeshare
    ↪ .trips.start_date, 2017-07-01 23:59:59.000000) |
|       └─TableFullScan_18    | 10000.00 | cop[tikv] | table:trips | keep
    ↪ order:false, stats:pseudo
    ↪
    ↪ |
+--
    ↪ ----------------------------+----------+----------+---------------+-----------
    ↪
5 rows in set (0.00 sec)
```

From the child operator `└─TableFullScan_18` back, you can see its execution process as follows, which is currently suboptimal:

1. The coprocessor (TiKV) reads the entire `trips` table as a `TableFullScan` operation. It then passes the rows that it reads to the `Selection_19` operator, which is still within TiKV.
2. The `WHERE start_date BETWEEN ..` predicate is then filtered in the `Selection_19` operator. Approximately `250` rows are estimated to meet this selection. Note that this number is estimated according to the statistics and the operator's logic. The `└─TableFullScan_18` operator shows `stats:pseudo`, which means that the table does not have the actual statistical information. After running `ANALYZE TABLE trips` to collect statistical information, the statistics are expected to be more accurate.
3. The rows that meet the selection criteria then have a `count` function applied to them. This is also completed inside the `StreamAgg_9` operator, which is still inside TiKV (`cop[tikv]`). The TiKV coprocessor can execute a number of MySQL built-in functions, `count` being one of them.
4. The results from `StreamAgg_9` are then sent to the `TableReader_21` operator which is now inside the TiDB server (the task of `root`). The `estRows` column value for this operator is `1`, which means that the operator will receive one row from each of the TiKV Regions to be accessed. For more information about these requests, see `EXPLAIN ANALYZE`.
5. The `StreamAgg_20` operator then applies a `count` function to each of the rows from the `└─TableReader_21` operator, which you can see from `SHOW TABLE REGIONS` and will be about 56 rows. Because this is the root operator, it then returns results to the client.

> **Note:**
>
> For a general view of the Regions that a table contains, execute `SHOW TABLE REGIONS`.

11.3.2.2.1 Assess the current performance

`EXPLAIN` only returns the query execution plan but does not execute the query. To get the actual execution time, you can either execute the query or use `EXPLAIN ANALYZE`:

```
EXPLAIN ANALYZE SELECT count(*) FROM trips WHERE start_date BETWEEN '
    ↪ 2017-07-01 00:00:00' AND '2017-07-01 23:59:59';
```

```
+--
    ↪ ----------------------------+----------+----------+-----------+---------------+-
    ↪
```

```
| id                          | estRows | actRows | task     | access object |
    ↪ execution info
    ↪
    ↪ | operator info
    ↪
    ↪ | memory   | disk |
+--
    ↪ ----------------------------+---------+---------+----------+---------------+-
    ↪
| StreamAgg_20                | 1.00    | 1       | root     |               |
    ↪ time:1.031417203s, loops:2
    ↪
    ↪ | funcs:count(Column#13)->Column#11
    ↪
    ↪ | 632 Bytes | N/A |
|  ─TableReader_21            | 1.00    | 56      | root     |               |
    ↪ time:1.031408123s, loops:2, cop_task: {num: 56, max: 782.147269ms,
    ↪ min: 5.759953ms, avg: 252.005927ms, p95: 609.294603ms, max_proc_keys:
    ↪  910371, p95_proc_keys: 704775, tot_proc: 11.524s, tot_wait: 580ms,
    ↪ rpc_num: 56, rpc_time: 14.111932641s} | data:StreamAgg_9
    ↪                                                              | 328 Bytes |
    ↪ N/A |
|    ─StreamAgg_9             | 1.00    | 56      | cop[tikv] |               |
    ↪ proc max:640ms, min:8ms, p80:276ms, p95:480ms, iters:18695, tasks:56
    ↪
    ↪ | funcs:count(1)->Column#13
    ↪
    ↪ | N/A      | N/A |
|      ─Selection_19          | 250.00  | 11409   | cop[tikv] |               |
    ↪ proc max:640ms, min:8ms, p80:276ms, p95:476ms, iters:18695, tasks:56
    ↪
    ↪ | ge(bikeshare.trips.start_date, 2017-07-01 00:00:00.000000), le(
    ↪ bikeshare.trips.start_date, 2017-07-01 23:59:59.000000) | N/A | N/A |
|       ─TableFullScan_18     | 10000.00 | 19117643 | cop[tikv] | table:trips
    ↪  | proc max:612ms, min:8ms, p80:248ms, p95:460ms, iters:18695, tasks
    ↪ :56
    ↪
    ↪ | keep order:false, stats:pseudo
    ↪
    ↪ | N/A      | N/A |
+--
    ↪ ----------------------------+---------+---------+----------+---------------+-
    ↪
5 rows in set (1.03 sec)
```

The example query above takes `1.03` seconds to execute, which is an ideal performance.

From the result of `EXPLAIN ANALYZE` above, `actRows` indicates that some of the estimates (`estRows`) are inaccurate (expecting 10 thousand rows but finding 19 million rows), which is already indicated in the `operator info` (`stats:pseudo`) of `-TableFullScan_18`. If you run `ANALYZE TABLE` first and then `EXPLAIN ANALYZE` again, you can see that the estimates are much closer:

```
ANALYZE TABLE trips;
EXPLAIN ANALYZE SELECT count(*) FROM trips WHERE start_date BETWEEN '
    ↪ 2017-07-01 00:00:00' AND '2017-07-01 23:59:59';
```

```
Query OK, 0 rows affected (10.22 sec)

+--
    ↪ ---------------------------+-------------+---------+-----------+-------------
    ↪
| id                          | estRows    | actRows | task      | access object
    ↪ | execution info
    ↪
    ↪ | operator info
    ↪
    ↪ | memory   | disk |
+--
    ↪ ---------------------------+-------------+---------+-----------+-------------
    ↪
| StreamAgg_20                | 1.00       | 1       | root      |              |
    ↪ time:926.393612ms, loops:2
    ↪
    ↪ | funcs:count(Column#13)->Column#11
    ↪
    ↪ | 632 Bytes | N/A |
|  -TableReader_21            | 1.00       | 56      | root      |              |
    ↪  time:926.384792ms, loops:2, cop_task: {num: 56, max: 850.94424ms,
    ↪ min: 6.042079ms, avg: 234.987725ms, p95: 495.474806ms, max_proc_keys:
    ↪  910371, p95_proc_keys: 704775, tot_proc: 10.656s, tot_wait: 904ms,
    ↪ rpc_num: 56, rpc_time: 13.158911952s} | data:StreamAgg_9
    ↪                                                          | 328 Bytes |
    ↪  N/A |
|    -StreamAgg_9             | 1.00       | 56      | cop[tikv] |              |
    ↪  proc max:592ms, min:4ms, p80:244ms, p95:480ms, iters:18695, tasks:56
    ↪
    ↪
    ↪ | funcs:count(1)->Column#13
    ↪
    ↪ | N/A      | N/A |
```

```
|     └─Selection_19        | 432.89    | 11409   | cop[tikv] |           |
    ↪  proc max:592ms, min:4ms, p80:244ms, p95:480ms, iters:18695, tasks:56
    ↪
    ↪
    ↪ | ge(bikeshare.trips.start_date, 2017-07-01 00:00:00.000000), le(
    ↪ bikeshare.trips.start_date, 2017-07-01 23:59:59.000000) | N/A | N/A |
|       └─TableFullScan_18  | 19117643.00 | 19117643 | cop[tikv] | table:
    ↪ trips | proc max:564ms, min:4ms, p80:228ms, p95:456ms, iters:18695,
    ↪ tasks:56
    ↪
    ↪ | keep order:false
    ↪
    ↪ | N/A      | N/A |
+--
    ↪ ---------------------------+-------------+---------+-----------+--------------
    ↪
5 rows in set (0.93 sec)
```

After `ANALYZE TABLE` is executed, you can see that the estimated rows for the `└─ ↪ TableFullScan_18` operator is accurate and the estimate for `└─Selection_19` is now also much closer. In the two cases above, although the execution plan (the set of operators TiDB uses to execute this query) has not changed, quite frequently sub-optimal plans are caused by outdated statistics.

In addition to `ANALYZE TABLE`, TiDB automatically regenerates statistics as a background operation after the threshold of `tidb_auto_analyze_ratio` is reached. You can see how close TiDB is to this threshold (how healthy TiDB considers the statistics to be) by executing the `SHOW STATS_HEALTHY` statement:

```
SHOW STATS_HEALTHY;
```

```
+----------+------------+----------------+---------+
| Db_name  | Table_name | Partition_name | Healthy |
+----------+------------+----------------+---------+
| bikeshare | trips   |                |     100 |
+----------+------------+----------------+---------+
1 row in set (0.00 sec)
```

11.3.2.2.2 Identify optimizations

The current execution plan is efficient in the following aspects:

- Most of the work is handled inside the TiKV coprocessor. Only 56 rows need to be sent across the network back to TiDB for processing. Each of these rows is short and contains only the count that matches the selection.

- Aggregating the count of rows both in TiDB (`StreamAgg_20`) and in TiKV (`StreamAgg_9`) uses the stream aggregation, which is very efficient in its memory usage.

The biggest issue with the current execution plan is that the predicate `start_date BETWEEN '2017-07-01 00:00:00' AND '2017-07-01 23:59:59'` does not apply immediately. All rows are read first with a `TableFullScan` operator, and then a selection is applied afterwards. You can find out the cause from the output of `SHOW CREATE TABLE trips`:

```
SHOW CREATE TABLE trips\G
```

```
*************************** 1. row ***************************
       Table: trips
Create Table: CREATE TABLE `trips` (
  `trip_id` bigint(20) NOT NULL AUTO_INCREMENT,
  `duration` int(11) NOT NULL,
  `start_date` datetime DEFAULT NULL,
  `end_date` datetime DEFAULT NULL,
  `start_station_number` int(11) DEFAULT NULL,
  `start_station` varchar(255) DEFAULT NULL,
  `end_station_number` int(11) DEFAULT NULL,
  `end_station` varchar(255) DEFAULT NULL,
  `bike_number` varchar(255) DEFAULT NULL,
  `member_type` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`trip_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin AUTO_INCREMENT
    ↪ =20477318
1 row in set (0.00 sec)
```

There is **NO** index on `start_date`. You would need an index in order to push this predicate into an index reader operator. Add an index as follows:

```
ALTER TABLE trips ADD INDEX (start_date);
```

```
Query OK, 0 rows affected (2 min 10.23 sec)
```

Note:

You can monitor the progress of DDL jobs using the `ADMIN SHOW DDL JOBS` command. The defaults in TiDB are carefully chosen so that adding an index does not impact production workloads too much. For testing environments, consider increasing the `tidb_ddl_reorg_batch_size` and `tidb_ddl_reorg_worker_cnt` values. On a reference system, a batch size of `10240` and worker count of `32` can achieve a 10x performance improvement over the defaults.

After adding an index, you can then repeat the query in `EXPLAIN`. In the following output, you can see that a new execution plan is chosen, and the `TableFullScan` and `Selection` operators have been eliminated:

```
EXPLAIN SELECT count(*) FROM trips WHERE start_date BETWEEN '2017-07-01
    ↪ 00:00:00' AND '2017-07-01 23:59:59';
```

```
+--
    ↪ -------------------------+---------+-----------+-------------------------
    ↪
| id                        | estRows | task    | access object
    ↪                         | operator info
    ↪                                                |
+--
    ↪ -------------------------+---------+-----------+-------------------------
    ↪
| StreamAgg_17              | 1.00    | root      |
    ↪                                               | funcs:count(Column#13)->Column
    ↪ #11                           |
|  ─IndexReader_18          | 1.00    | root      |
    ↪                                               | index:StreamAgg_9
    ↪                                                 |
|    ─StreamAgg_9           | 1.00    | cop[tikv] |
    ↪                                                   | funcs:count(1)->Column#13
    ↪                                     |
|      ─IndexRangeScan_16   | 8471.88 | cop[tikv] | table:trips, index:
    ↪ start_date(start_date) | range:[2017-07-01 00:00:00,2017-07-01
    ↪ 23:59:59], keep order:false |
+--
    ↪ -------------------------+---------+-----------+-------------------------
    ↪
4 rows in set (0.00 sec)
```

To compare the actual execution time, you can again use `EXPLAIN ANALYZE`:

```
EXPLAIN ANALYZE SELECT count(*) FROM trips WHERE start_date BETWEEN '
    ↪ 2017-07-01 00:00:00' AND '2017-07-01 23:59:59';
```

```
+--
    ↪ -------------------------+---------+---------+-----------+-----------------
    ↪
| id                        | estRows | actRows | task    | access object
    ↪                         | execution info
    ↪
    ↪ | operator info                                                | memory
    ↪   | disk |
```

```
+--
   ↪ -------------------------+---------+---------+----------+-------------------
   ↪
| StreamAgg_17              | 1.00    | 1       | root     |
   ↪                                      | time:4.516728ms, loops:2
   ↪
   ↪ | funcs:count(Column#13)->Column#11                          | 372
   ↪ Bytes | N/A |
|  ─IndexReader_18          | 1.00    | 1       | root     |
   ↪                                      | time:4.514278ms, loops:2,
   ↪ cop_task: {num: 1, max:4.462288ms, proc_keys: 11409, rpc_num: 1,
   ↪ rpc_time: 4.457148ms} | index:StreamAgg_9                          |
   ↪ 238 Bytes | N/A |
|    ─StreamAgg_9           | 1.00    | 1       | cop[tikv] |
   ↪                                    | time:4ms, loops:12
   ↪
   ↪ | funcs:count(1)->Column#13                                  | N/A
   ↪      | N/A |
|      ─IndexRangeScan_16   | 8471.88 | 11409 | cop[tikv] | table:trips,
   ↪ index:start_date(start_date) | time:4ms, loops:12
   ↪
   ↪ | range:[2017-07-01 00:00:00,2017-07-01 23:59:59], keep order:false |
   ↪  N/A | N/A |
+--
   ↪ -------------------------+---------+---------+----------+-------------------
   ↪
4 rows in set (0.00 sec)
```

From the result above, the query time has reduced from 1.03 seconds to 0.0 seconds.

> **Note:**
>
> Another optimization that applies here is the coprocessor cache. If you are unable to add indexes, consider enabling the coprocessor cache. When it is enabled, as long as the Region has not been modified since the operator is last executed, TiKV will return the value from the cache. This will also help reduce much of the cost of the expensive `TableFullScan` and `Selection` operators.

11.3.2.3 Explain Statements That Use Indexes

TiDB supports several operators which make use of indexes to speed up query execution:

- IndexLookup
- IndexReader
- Point_Get and Batch_Point_Get
- IndexFullScan

The examples in this document are based on the following sample data:

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY auto_increment,
 intkey INT NOT NULL,
 pad1 VARBINARY(1024),
 INDEX (intkey)
);

INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM
   ↪ dual;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM t1
   ↪ a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM t1
   ↪ a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM t1
   ↪ a JOIN t1 b JOIN t1 c LIMIT 10000;
```

11.3.2.3.1 IndexLookup

TiDB uses the `IndexLookup` operator when retrieving data from a secondary index. In this case, the following queries will all use the `IndexLookup` operator on the `intkey` index:

```
EXPLAIN SELECT * FROM t1 WHERE intkey = 123;
EXPLAIN SELECT * FROM t1 WHERE intkey < 10;
EXPLAIN SELECT * FROM t1 WHERE intkey BETWEEN 300 AND 310;
EXPLAIN SELECT * FROM t1 WHERE intkey IN (123,29,98);
EXPLAIN SELECT * FROM t1 WHERE intkey >= 99 AND intkey <= 103;
```

```
+--
   ↪ ----------------------------+---------+-----------+---------------------------
   ↪
| id                          | estRows | task    | access object
   ↪                | operator info                 |
+--
   ↪ ----------------------------+---------+-----------+---------------------------
   ↪
| IndexLookUp_10              | 1.00    | root      |
   ↪                                |                                |
| ─IndexRangeScan_8(Build)  | 1.00    | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | range:[123,123], keep order:false |
```

```
|  └─TableRowIDScan_9(Probe)  | 1.00   | cop[tikv] | table:t1
   ↪                  | keep order:false             |
+--
   ↪ ---------------------------+--------+----------+-------------------------
   ↪
3 rows in set (0.00 sec)

+--
   ↪ ---------------------------+--------+----------+-------------------------
   ↪
| id                          | estRows | task    | access object
   ↪              | operator info                 |
+--
   ↪ ---------------------------+--------+----------+-------------------------
   ↪
| IndexLookUp_10              | 3.60   | root     |
   ↪                             |                                |
|  └─IndexRangeScan_8(Build)  | 3.60   | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | range:[-inf,10), keep order:false |
|  └─TableRowIDScan_9(Probe)  | 3.60   | cop[tikv] | table:t1
   ↪                  | keep order:false             |
+--
   ↪ ---------------------------+--------+----------+-------------------------
   ↪
3 rows in set (0.00 sec)

+--
   ↪ ---------------------------+--------+----------+-------------------------
   ↪
| id                          | estRows | task    | access object
   ↪              | operator info                 |
+--
   ↪ ---------------------------+--------+----------+-------------------------
   ↪
| IndexLookUp_10              | 5.67   | root     |
   ↪                             |                                |
|  └─IndexRangeScan_8(Build)  | 5.67   | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | range:[300,310], keep order:false |
|  └─TableRowIDScan_9(Probe)  | 5.67   | cop[tikv] | table:t1
   ↪                  | keep order:false             |
+--
   ↪ ---------------------------+--------+----------+-------------------------
   ↪
3 rows in set (0.00 sec)
```

```
+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
| id                          | estRows | task     | access object
   ↪              | operator info                                  |
+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
| IndexLookUp_10              | 4.00    | root     |
   ↪                               |
   ↪                                                  |
|  ─IndexRangeScan_8(Build)  | 4.00    | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | range:[29,29], [98,98], [123,123], keep order:false |
|  ─TableRowIDScan_9(Probe)  | 4.00    | cop[tikv] | table:t1
   ↪                   | keep order:false                               |
+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
3 rows in set (0.00 sec)

+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
| id                          | estRows | task     | access object
   ↪              | operator info                  |
+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
| IndexLookUp_10              | 6.00    | root     |
   ↪                               |                                |
|  ─IndexRangeScan_8(Build)  | 6.00    | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | range:[99,103], keep order:false |
|  ─TableRowIDScan_9(Probe)  | 6.00    | cop[tikv] | table:t1
   ↪                   | keep order:false               |
+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
3 rows in set (0.00 sec)
```

The `IndexLookup` operator has two child nodes:

- The `─IndexRangeScan_8(Build)` operator performs a range scan on the `intkey` index and retrieves the values of the internal `RowID` (for this table, the primary key).
- The `─TableRowIDScan_9(Probe)` operator then retrieves the full row from the table data.

Because an `IndexLookup` task requires two steps, the SQL Optimizer might choose the `TableFullScan` operator based on statistics in scenarios where a large number of rows match. In the following example, a large number of rows match the condition of `intkey > 100`, and a `TableFullScan` is chosen:

```
EXPLAIN SELECT * FROM t1 WHERE intkey > 100;
```

```
+--
   ↪ ---------------------+---------+----------+--------------+-----------------
   ↪
| id                    | estRows | task     | access object | operator info
   ↪      |
+--
   ↪ ---------------------+---------+----------+--------------+-----------------
   ↪
| TableReader_7         | 898.50 | root     |               | data:Selection_6
   ↪     |
|  ─Selection_6         | 898.50 | cop[tikv] |              | gt(test.t1.intkey
   ↪ , 100) |
|    ─TableFullScan_5   | 1010.00 | cop[tikv] | table:t1 | keep order:false
   ↪     |
+--
   ↪ ---------------------+---------+----------+--------------+-----------------
   ↪
3 rows in set (0.00 sec)
```

The `IndexLookup` operator can also be used to efficiently optimize `LIMIT` on an indexed column:

```
EXPLAIN SELECT * FROM t1 ORDER BY intkey DESC LIMIT 10;
```

```
+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
| id                           | estRows | task     | access object
   ↪              | operator info                    |
+--
   ↪ ----------------------------+---------+----------+-------------------------
   ↪
| IndexLookUp_21               | 10.00   | root     |
   ↪                              | limit embedded(offset:0, count:10) |
|  ─Limit_20(Build)            | 10.00   | cop[tikv] |
   ↪                               | offset:0, count:10             |
|    ─IndexFullScan_18         | 10.00   | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | keep order:true, desc    |
```

```
|  └─TableRowIDScan_19(Probe)  | 10.00  | cop[tikv] | table:t1
   ↪                  | keep order:false, stats:pseudo |
+--
   ↪ ----------------------------+--------+----------+-----------------------
   ↪
4 rows in set (0.00 sec)
```

In the above example, the last 10 rows are read from the index `intkey`. These `RowID` values are then retrieved from the table data.

11.3.2.3.2 IndexReader

TiDB supports the *covering index optimization*. If all rows can be retrieved from an index, TiDB will skip the second step that is usually required in an `IndexLookup`. Consider the following two examples:

```
EXPLAIN SELECT * FROM t1 WHERE intkey = 123;
EXPLAIN SELECT id FROM t1 WHERE intkey = 123;
```

```
+--
   ↪ ----------------------------+--------+----------+-----------------------
   ↪
| id                          | estRows | task    | access object
   ↪              | operator info                  |
+--
   ↪ ----------------------------+--------+----------+-----------------------
   ↪
| IndexLookUp_10              | 1.00    | root     |
   ↪                             |                                |
|  └─IndexRangeScan_8(Build)  | 1.00    | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | range:[123,123], keep order:false |
|  └─TableRowIDScan_9(Probe)  | 1.00    | cop[tikv] | table:t1
   ↪                  | keep order:false               |
+--
   ↪ ----------------------------+--------+----------+-----------------------
   ↪
3 rows in set (0.00 sec)

+--
   ↪ ----------------------+--------+----------+-----------------------------+
   ↪
| id                     | estRows | task    | access object               |
   ↪ operator info                 |
+--
   ↪ ----------------------+--------+----------+-----------------------------+
   ↪
```

```
| Projection_4          | 1.00   | root     |                                  |
   ↪ test.t1.id                    |
|  ─IndexReader_6       | 1.00   | root     |                                  |
   ↪ index:IndexRangeScan_5        |
|    ─IndexRangeScan_5  | 1.00   | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | range:[123,123], keep order:false |
+--
   ↪ -----------------------+---------+-----------+-------------------------------+
   ↪
3 rows in set (0.00 sec)
```

Because `id` is also the internal `RowID`, it is stored in the `intkey` index. After using the `intkey` index as part of `─IndexRangeScan_5`, the value of the `RowID` can be returned directly.

11.3.2.3.3 Point_Get and Batch_Point_Get

TiDB uses the `Point_Get` or `Batch_Point_Get` operator when retrieving data directly from a primary key or unique key. These operators are more efficient than `IndexLookup`. For example:

```
EXPLAIN SELECT * FROM t1 WHERE id = 1234;
EXPLAIN SELECT * FROM t1 WHERE id IN (1234,123);

ALTER TABLE t1 ADD unique_key INT;
UPDATE t1 SET unique_key = id;
ALTER TABLE t1 ADD UNIQUE KEY (unique_key);

EXPLAIN SELECT * FROM t1 WHERE unique_key = 1234;
EXPLAIN SELECT * FROM t1 WHERE unique_key IN (1234, 123);
```

```
+-------------+---------+------+---------------+---------------+
| id          | estRows | task | access object | operator info |
+-------------+---------+------+---------------+---------------+
| Point_Get_1 | 1.00 | root | table:t1   | handle:1234 |
+-------------+---------+------+---------------+---------------+
1 row in set (0.00 sec)

+--
   ↪ -----------------+---------+------+---------------+-----------------------------
   ↪
| id              | estRows | task | access object | operator info
   ↪                          |
+--
   ↪ -----------------+---------+------+---------------+-----------------------------
   ↪
```

```
| Batch_Point_Get_1 | 2.00 | root | table:t1   | handle:[1234 123], keep
    ↪ order:false, desc:false |
+--
    ↪ ----------------+---------+------+--------------+-----------------------------
    ↪
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.27 sec)

Query OK, 1010 rows affected (0.06 sec)
Rows matched: 1010 Changed: 1010 Warnings: 0

Query OK, 0 rows affected (0.37 sec)

+--
    ↪ -----------+---------+------+-----------------------------------------+---------
    ↪
| id          | estRows | task | access object                           | operator
    ↪ info |
+--
    ↪ -----------+---------+------+-----------------------------------------+---------
    ↪
| Point_Get_1 | 1.00 | root | table:t1, index:unique_key(unique_key) |
    ↪       |
+--
    ↪ -----------+---------+------+-----------------------------------------+---------
    ↪
1 row in set (0.00 sec)

+--
    ↪ ----------------+---------+------+-------------------------------------------+----
    ↪
| id                | estRows | task | access object                             |
    ↪ operator info              |
+--
    ↪ ----------------+---------+------+-------------------------------------------+----
    ↪
| Batch_Point_Get_1 | 2.00 | root | table:t1, index:unique_key(unique_key)
    ↪ | keep order:false, desc:false |
+--
    ↪ ----------------+---------+------+-------------------------------------------+----
    ↪
1 row in set (0.00 sec)
```

11.3.2.3.4 IndexFullScan

Because indexes are ordered, the `IndexFullScan` operator can be used to optimize common queries such as the `MIN` or `MAX` values for an indexed value:

```
EXPLAIN SELECT MIN(intkey) FROM t1;
EXPLAIN SELECT MAX(intkey) FROM t1;
```

```
+--
   ↪ ---------------------------+---------+----------+-------------------------
   ↪
| id                          | estRows | task     | access object
   ↪ | operator info                    |
+--
   ↪ ---------------------------+---------+----------+-------------------------
   ↪
| StreamAgg_12                | 1.00    | root     |
   ↪ | funcs:min(test.t1.intkey)->Column#4 |
|  ─Limit_16                  | 1.00    | root     |
   ↪                               | offset:0, count:1                |
|    ─IndexReader_29          | 1.00    | root     |
   ↪                               | index:Limit_28                   |
|      ─Limit_28              | 1.00    | cop[tikv] |
   ↪                              | offset:0, count:1                |
|        ─IndexFullScan_27    | 1.00    | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | keep order:true           |
+--
   ↪ ---------------------------+---------+----------+-------------------------
   ↪
5 rows in set (0.00 sec)

+--
   ↪ ---------------------------+---------+----------+-------------------------
   ↪
| id                          | estRows | task     | access object
   ↪ | operator info                    |
+--
   ↪ ---------------------------+---------+----------+-------------------------
   ↪
| StreamAgg_12                | 1.00    | root     |
   ↪ | funcs:max(test.t1.intkey)->Column#4 |
|  ─Limit_16                  | 1.00    | root     |
   ↪                               | offset:0, count:1                |
|    ─IndexReader_29          | 1.00    | root     |
   ↪                               | index:Limit_28                   |
|      ─Limit_28              | 1.00    | cop[tikv] |
```

```
    ↪                              | offset:0, count:1                |
|       └─IndexFullScan_27   | 1.00    | cop[tikv] | table:t1, index:intkey(
    ↪ intkey) | keep order:true, desc     |
+--
    ↪ ----------------------------+---------+----------+-------------------------
    ↪
5 rows in set (0.00 sec)
```

In the above statements, an `IndexFullScan` task is performed on each TiKV Region. Despite the name `FullScan`, only the first row needs to be read (`└─Limit_28`). Each TiKV Region returns its `MIN` or `MAX` value to TiDB, which then performs Stream Aggregation to filter for a single row. Stream Aggregation with the aggregation function `MAX` or `MIN` also ensures that `NULL` is returned if the table is empty.

By contrast, executing the `MIN` function on an unindexed value will result in `TableFullScan`. The query will require all rows to be scanned in TiKV, but a `TopN` calculation is performed to ensure each TiKV Region only returns one row to TiDB. Although `TopN` prevents excessive rows from being transferred between TiKV and TiDB, this statement is still considered far less efficient than the above example where `MIN` is able to make use of an index.

```
EXPLAIN SELECT MIN(pad1) FROM t1;
```

```
+--
    ↪ ----------------------------+---------+----------+--------------+----------
    ↪
| id                          | estRows | task    | access object | operator
    ↪ info                |
+--
    ↪ ----------------------------+---------+----------+--------------+----------
    ↪
| StreamAgg_13                | 1.00    | root     |               | funcs:min(
    ↪ test.t1.pad1)->Column#4 |
| └─TopN_14                   | 1.00    | root     |               | test.t1.
    ↪ pad1, offset:0, count:1 |
|   └─TableReader_23          | 1.00    | root     |               | data:
    ↪ TopN_22                  |
|     └─TopN_22               | 1.00    | cop[tikv] |              | test.t1.
    ↪ pad1, offset:0, count:1 |
|       └─Selection_21        | 1008.99 | cop[tikv] |              | not(isnull(
    ↪ test.t1.pad1))     |
|         └─TableFullScan_20  | 1010.00 | cop[tikv] | table:t1 | keep order:
    ↪ false               |
+--
    ↪ ----------------------------+---------+----------+--------------+----------
    ↪
```

```
6 rows in set (0.00 sec)
```

The following statements will use the `IndexFullScan` operator to scan every row in the index:

```
EXPLAIN SELECT SUM(intkey) FROM t1;
EXPLAIN SELECT AVG(intkey) FROM t1;
```

```
+--
   ↪ -------------------------+---------+-----------+-------------------------------
   ↪
| id                      | estRows | task    | access object              |
   ↪ operator info                   |
+--
   ↪ -------------------------+---------+-----------+-------------------------------
   ↪
| StreamAgg_20            | 1.00    | root     |                            |
   ↪ funcs:sum(Column#6)->Column#4  |
| ─IndexReader_21         | 1.00    | root     |                            |
   ↪  index:StreamAgg_8              |
|   ─StreamAgg_8          | 1.00    | cop[tikv] |                           |
   ↪  funcs:sum(test.t1.intkey)->Column#6 |
|     ─IndexFullScan_19   | 1010.00 | cop[tikv] | table:t1, index:intkey(
   ↪ intkey) | keep order:false        |
+--
   ↪ -------------------------+---------+-----------+-------------------------------
   ↪
4 rows in set (0.00 sec)

+--
   ↪ -------------------------+---------+-----------+-------------------------------
   ↪
| id                      | estRows | task    | access object              |
   ↪ operator info                                                      |
+--
   ↪ -------------------------+---------+-----------+-------------------------------
   ↪
| StreamAgg_20            | 1.00    | root     |                            |
   ↪ funcs:avg(Column#7, Column#8)->Column#4                            |
| ─IndexReader_21         | 1.00    | root     |                            |
   ↪  index:StreamAgg_8                                                 |
|   ─StreamAgg_8          | 1.00    | cop[tikv] |                           |
   ↪  funcs:count(test.t1.intkey)->Column#7, funcs:sum(test.t1.intkey)->
   ↪ Column#8 |
|     ─IndexFullScan_19   | 1010.00 | cop[tikv] | table:t1, index:intkey(
```

```
    ↪ intkey) | keep order:false                                                  |
+--
    ↪ --------------------------+---------+-----------+-------------------------------
    ↪
4 rows in set (0.00 sec)
```

In the above examples, `IndexFullScan` is more efficient than `TableFullScan` because the width of the value in the `(intkey + RowID)` index is less than the width of the full row.

The following statement does not support using an `IndexFullScan` operator because additional columns are required from the table:

```
EXPLAIN SELECT AVG(intkey), ANY_VALUE(pad1) FROM t1;
```

```
+--
    ↪ ---------------------------+---------+-----------+---------------+-------------
    ↪
| id                        | estRows | task    | access object | operator
    ↪ info
    ↪
    ↪ |
+--
    ↪ ---------------------------+---------+-----------+---------------+-------------
    ↪
| Projection_4              | 1.00    | root     |               | Column#4,
    ↪ any_value(test.t1.pad1)->Column#5
    ↪                                                                  |
| ─StreamAgg_16             | 1.00    | root     |               | funcs:avg(
    ↪ Column#10, Column#11)->Column#4, funcs:firstrow(Column#12)->test.t1.
    ↪ pad1                      |
|   ─TableReader_17         | 1.00    | root     |               | data:
    ↪ StreamAgg_8
    ↪
    ↪ |
|     ─StreamAgg_8          | 1.00    | cop[tikv] |              | funcs:count(
    ↪ test.t1.intkey)->Column#10, funcs:sum(test.t1.intkey)->Column#11,
    ↪ funcs:firstrow(test.t1.pad1)->Column#12 |
|       ─TableFullScan_15   | 1010.00 | cop[tikv] | table:t1 | keep order:
    ↪ false
    ↪
    ↪ |
+--
    ↪ ---------------------------+---------+-----------+---------------+-------------
    ↪
5 rows in set (0.00 sec)
```

11.3.2.4 Explain Statements That Use Joins

In TiDB, the SQL Optimizer needs to decide in which order tables should be joined and what is the most efficient join algorithm for a particular SQL statement. The examples in this document are based on the following sample data:

```
CREATE TABLE t1 (id BIGINT NOT NULL PRIMARY KEY auto_increment, pad1 BLOB,
   ↪ pad2 BLOB, pad3 BLOB, int_col INT NOT NULL DEFAULT 0);
CREATE TABLE t2 (id BIGINT NOT NULL PRIMARY KEY auto_increment, t1_id
   ↪ BIGINT NOT NULL, pad1 BLOB, pad2 BLOB, pad3 BLOB, INDEX(t1_id));
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM dual;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
   ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
```

```
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
UPDATE t1 SET int_col = 1 WHERE pad1 = (SELECT pad1 FROM t1 ORDER BY RAND()
    ↪  LIMIT 1);
SELECT SLEEP(1);
ANALYZE TABLE t1, t2;
```

11.3.2.4.1 Index Join

If the number of estimated rows that need to be joined is small (typically less than 10000 rows), it is preferable to use the index join method. This method of join works similar to the primary method of join used in MySQL. In the following example, the operator `└─TableReader_28(Build)` first reads the table `t1`. For each row that matches, TiDB will probe the table `t2`:

```
EXPLAIN SELECT /*+ INL_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON t1.id =
    ↪ t2.t1_id;
```

```
+--
    ↪ -------------------------------+-----------+-----------+-----------------------
    ↪
| id                            | estRows  | task      | access object
    ↪            | operator info
    ↪                                                                 |
+--
    ↪ -------------------------------+-----------+-----------+-----------------------
    ↪
| IndexJoin_10                  | 180000.00 | root      |
    ↪                             | inner join, inner:IndexLookUp_9, outer key
    ↪ :test.t1.id, inner key:test.t2.t1_id |
|  ├─TableReader_28(Build)      | 142020.00 | root      |
    ↪                             | data:TableFullScan_27
    ↪                                                       |
|  │ └─TableFullScan_27         | 142020.00 | cop[tikv] | table:t1
    ↪              | keep order:false
    ↪                                                     |
|  └─IndexLookUp_9(Probe)       | 1.27      | root      |
    ↪                             |
    ↪
    ↪ |
|    └─IndexRangeScan_7(Build)  | 1.27      | cop[tikv] | table:t2, index:
    ↪ t1_id(t1_id) | range: decided by [eq(test.t2.t1_id, test.t1.id)],
    ↪ keep order:false |
```

```
|   └─TableRowIDScan_8(Probe)  | 1.27     | cop[tikv] | table:t2
   ↪                 | keep order:false
   ↪                                                    |
+--
   ↪ ------------------------------+----------+----------+----------------------
   ↪
6 rows in set (0.00 sec)
```

Index join is efficient in memory usage, but might be slower to execute than other join methods when a large number of probe operations are required. Consider also the following query:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id=t2.t1_id WHERE t1.pad1 = 'value'
   ↪ and t2.pad1='value';
```

In an inner join operation, TiDB implements join reordering and might access either `t1` or `t2` first. Assume that TiDB selects `t1` as the first table to apply the `build` step, and then TiDB is able to filter on the predicate `t1.col = 'value'` before probing the table `t2`. The filter for the predicate `t2.col='value'` will be applied on each probe of table `t2`, which might be less efficient than other join methods.

Index join is effective if the build side is small and the probe side is pre-indexed and large. Consider the following query where an index join performs worse than a hash join and is not chosen by the SQL Optimizer:

```
-- DROP previously added index
ALTER TABLE t2 DROP INDEX t1_id;

EXPLAIN ANALYZE SELECT /*+ INL_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
   ↪ t1.id = t2.t1_id WHERE t1.int_col = 1;
EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
   ↪ t1.id = t2.t1_id WHERE t1.int_col = 1;
EXPLAIN ANALYZE SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.t1_id WHERE t1
   ↪ .int_col = 1;
```

```
Query OK, 0 rows affected (0.29 sec)

+--
   ↪ -------------------------+---------+--------+----------+---------------+---
   ↪
| id                        | estRows | actRows | task    | access object |
   ↪ execution info
   ↪
   ↪ | operator info
   ↪                                                          | memory
   ↪             | disk |
```

```
+--
   ↪ -------------------------+---------+--------+----------+---------------+---
   ↪
| IndexJoin_13              | 90000.00 | 20000 | root     |               | time
   ↪ :613.19955ms, loops:21, inner:{total:42.494047ms, concurrency:5, task
   ↪ :12, construct:33.149671ms, fetch:9.322956ms, build:8.66µs}, probe
   ↪ :32.435355ms
   ↪
   ↪ | inner join, inner:TableReader_9, outer key:test.t2.t1_id, inner key
   ↪ :test.t1.id | 269.63341903686523 MB | N/A |
| ─TableReader_19(Build)  | 90000.00 | 90000 | root     |               | time
   ↪ :586.613252ms, loops:95, cop_task: {num: 3, max: 205.893949ms, min:
   ↪ 185.051354ms, avg: 194.878702ms, p95: 205.893949ms, max_proc_keys:
   ↪ 31715, p95_proc_keys: 31715, tot_proc: 332ms, tot_wait: 4ms, rpc_num:
   ↪  4, rpc_time: 584.907774ms, copr_cache_hit_ratio: 0.00}, backoff{
   ↪ regionMiss: 2ms} | data:TableFullScan_18           | 182.624906539917
   ↪ MB | N/A |
|   ─TableFullScan_18     | 90000.00 | 90000 | cop[tikv] | table:t2  | time
   ↪ :0ns, loops:0, tikv_task:{proc max:72ms, min:64ms, p80:72ms, p95:72ms
   ↪ , iters:102, tasks:3}
   ↪
   ↪ | keep order:false
   ↪                                                    | N/A
   ↪                 | N/A |
| ─TableReader_9(Probe)   | 0.00    | 5      | root     |               | time
   ↪ :8.432051ms, loops:14, cop_task: {num: 14, max: 629.805µs, min:
   ↪ 226.129µs, avg: 420.979µs, p95: 629.805µs, max_proc_keys: 4,
   ↪ p95_proc_keys: 4, rpc_num: 15, rpc_time: 5.953229ms,
   ↪ copr_cache_hit_ratio: 0.00}
   ↪                                              | data:Selection_8
   ↪                                                      | N/A
   ↪                 | N/A |
|   ─Selection_8          | 0.00    | 5      | cop[tikv] |               | time
   ↪ :0ns, loops:0, tikv_task:{proc max:0s, min:0s, p80:0s, p95:0s, iters
   ↪ :14, tasks:14}
   ↪
   ↪ | eq(test.t1.int_col, 1)
   ↪                                                 | N/A                | N
   ↪ /A |
|     ─TableRangeScan_7   | 1.00    | 25     | cop[tikv] | table:t1  | time
   ↪ :0ns, loops:0, tikv_task:{proc max:0s, min:0s, p80:0s, p95:0s, iters
   ↪ :14, tasks:14}
   ↪
   ↪ | range: decided by [test.t2.t1_id], keep order:false
   ↪                   | N/A                | N/A |
```

```
+--
   ↪ -------------------------+---------+--------+----------+---------------+---
   ↪
6 rows in set (0.61 sec)

+--
   ↪ -------------------------+---------+--------+----------+---------------+--
   ↪
| id                        | estRows | actRows | task    | access object |
   ↪ execution info
   ↪
   ↪ | operator info                                | memory            |
   ↪ disk   |
+--
   ↪ -------------------------+---------+--------+----------+---------------+--
   ↪
| HashJoin_19               | 90000.00 | 20000 | root    |               | time
   ↪ :406.098528ms, loops:22, build_hash_table:{total:148.574644ms, fetch
   ↪ :146.843636ms, build:1.731008ms}, probe:{concurrency:5, total
   ↪ :2.026547436s, max:406.039309ms, probe:205.337813ms, fetch
   ↪ :1.821209623s}            | inner join, equal:[eq(test.t1.id, test.t2.
   ↪ t1_id)] | 30.00731658935547 MB | 0 Bytes |
|  ─TableReader_22(Build)   | 71.01   | 10000  | root    |               |
   ↪ time:147.072725ms, loops:12, cop_task: {num: 3, max: 145.847743ms,
   ↪ min: 50.932527ms, avg: 113.009029ms, p95: 145.847743ms, max_proc_keys
   ↪ : 31724, p95_proc_keys: 31724, tot_proc: 284ms, rpc_num: 3, rpc_time:
   ↪  338.950488ms, copr_cache_hit_ratio: 0.00} | data:Selection_21 |
   ↪ 29.679713249206543 MB | N/A |
|    ─Selection_21          | 71.01   | 10000  | cop[tikv] |               |
   ↪ time:0ns, loops:0, tikv_task:{proc max:132ms, min:48ms, p80:132ms,
   ↪ p95:132ms, iters:83, tasks:3}
   ↪
   ↪ | eq(test.t1.int_col, 1)                       | N/A                | N
   ↪ /A    |
|      ─TableFullScan_20    | 71010.00 | 71010 | cop[tikv] | table:t1  |
   ↪ time:0ns, loops:0, tikv_task:{proc max:128ms, min:48ms, p80:128ms,
   ↪ p95:128ms, iters:83, tasks:3}
   ↪
   ↪ | keep order:false                             | N/A                | N
   ↪ /A    |
|  ─TableReader_24(Probe)   | 90000.00 | 90000 | root    |               |
   ↪ time:365.918504ms, loops:91, cop_task: {num: 3, max: 398.62145ms, min
   ↪ : 338.460345ms, avg: 358.732721ms, p95: 398.62145ms, max_proc_keys:
   ↪ 31715, p95_proc_keys: 31715, tot_proc: 536ms, rpc_num: 3, rpc_time:
   ↪ 1.076128895s, copr_cache_hit_ratio: 0.00} | data:TableFullScan_23 |
```

```
    ↪ 182.62489891052246 MB | N/A |
|    └─TableFullScan_23      | 90000.00 | 90000 | cop[tikv] | table:t2  |
    ↪ time:0ns, loops:0, tikv_task:{proc max:100ms, min:40ms, p80:100ms,
    ↪ p95:100ms, iters:102, tasks:3}
    ↪
    ↪ | keep order:false                              | N/A                | N
    ↪ /A    |
+--
    ↪ ---------------------------+---------+--------+----------+---------------+--
    ↪
6 rows in set (0.41 sec)

+--
    ↪ ---------------------------+---------+--------+----------+---------------+--
    ↪
| id                          | estRows | actRows | task    | access object |
    ↪ execution info
    ↪
    ↪ | operator info                                 | memory             |
    ↪ disk   |
+--
    ↪ ---------------------------+---------+--------+----------+---------------+--
    ↪
| HashJoin_20                 | 90000.00 | 20000 | root     |               | time
    ↪ :441.897092ms, loops:21, build_hash_table:{total:138.600864ms, fetch
    ↪ :136.353899ms, build:2.246965ms}, probe:{concurrency:5, total
    ↪ :2.207403854s, max:441.850032ms, probe:148.01937ms, fetch:2.059384484
    ↪ s}              | inner join, equal:[eq(test.t1.id, test.t2.t1_id)] |
    ↪ 30.00731658935547 MB | 0 Bytes |
|  ├─TableReader_25(Build)   | 71.01   | 10000  | root     |               |
    ↪ time:138.081807ms, loops:12, cop_task: {num: 3, max: 134.702901ms,
    ↪ min: 53.356202ms, avg: 93.372186ms, p95: 134.702901ms, max_proc_keys:
    ↪  31724, p95_proc_keys: 31724, tot_proc: 236ms, rpc_num: 3, rpc_time:
    ↪ 280.017658ms, copr_cache_hit_ratio: 0.00} | data:Selection_24 |
    ↪ 29.680171966552734 MB | N/A |
|  │  └─Selection_24          | 71.01   | 10000  | cop[tikv] |               |
    ↪ time:0ns, loops:0, tikv_task:{proc max:80ms, min:52ms, p80:80ms, p95
    ↪ :80ms, iters:83, tasks:3}
    ↪
    ↪ | eq(test.t1.int_col, 1)                        | N/A                | N
    ↪ /A    |
|  │    └─TableFullScan_23    | 71010.00 | 71010 | cop[tikv] | table:t1  |
    ↪ time:0ns, loops:0, tikv_task:{proc max:80ms, min:52ms, p80:80ms, p95
    ↪ :80ms, iters:83, tasks:3}
    ↪
```

```
    ↪ | keep order:false                                | N/A                | N
    ↪ /A    |
|  ─TableReader_22(Probe)   | 90000.00 | 90000 | root    |                    |
    ↪ time:413.560548ms, loops:91, cop_task: {num: 3, max: 432.938474ms,
    ↪ min: 231.263355ms, avg: 365.710741ms, p95: 432.938474ms,
    ↪ max_proc_keys: 31715, p95_proc_keys: 31715, tot_proc: 488ms, rpc_num:
    ↪  3, rpc_time: 1.097021983s, copr_cache_hit_ratio: 0.00} | data:
    ↪ TableFullScan_21 | 182.62489891052246 MB | N/A |
|    ─TableFullScan_21      | 90000.00 | 90000 | cop[tikv] | table:t2  |
    ↪ time:0ns, loops:0, tikv_task:{proc max:80ms, min:80ms, p80:80ms, p95
    ↪ :80ms, iters:102, tasks:3}
    ↪
    ↪ | keep order:false                                | N/A                | N
    ↪ /A    |
+--
    ↪ ----------------------------+----------+---------+-----------+---------------+--
    ↪
6 rows in set (0.44 sec)
```

In the above example, the index join operation is missing an index on `t1.int_col`. Once this index is added, the performance of the operation improves from `0.61 sec` to `0.14 sec`, as the following result shows:

```
-- Re-add index
ALTER TABLE t2 ADD INDEX (t1_id);

EXPLAIN ANALYZE SELECT /*+ INL_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
    ↪ t1.id = t2.t1_id WHERE t1.int_col = 1;
EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
    ↪ t1.id = t2.t1_id WHERE t1.int_col = 1;
EXPLAIN ANALYZE SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.t1_id WHERE t1
    ↪ .int_col = 1;
```

```
Query OK, 0 rows affected (3.65 sec)

+--
    ↪ ------------------------------+----------+---------+-----------+--------------
    ↪
| id                            | estRows | actRows | task    | access object
    ↪             | execution info
    ↪
    ↪ | operator info
    ↪                                                                | memory
    ↪              | disk |
+--
```

```
   ↪ ------------------------------+---------+--------+----------+---------------
   ↪
| IndexJoin_11                  | 90000.00 | 0     | root     |
   ↪                              | time:136.876686ms, loops:1, inner:{total
   ↪ :114.948158ms, concurrency:5, task:7, construct:5.329114ms, fetch
   ↪ :109.610054ms, build:2.38µs}, probe:1.699799ms
   ↪
   ↪ | inner join, inner:IndexLookUp_10, outer key:test.t1.id, inner key:
   ↪ test.t2.t1_id | 29.864535331726074 MB | N/A |
|  ─TableReader_32(Build)      | 10000.00 | 10000 | root    |
   ↪                              | time:95.755212ms, loops:12, cop_task: {num
   ↪ : 3, max: 95.652443ms, min: 30.758712ms, avg: 57.545129ms, p95:
   ↪ 95.652443ms, max_proc_keys: 31724, p95_proc_keys: 31724, tot_proc:
   ↪ 124ms, rpc_num: 3, rpc_time: 172.528417ms, copr_cache_hit_ratio:
   ↪ 0.00} | data:Selection_31                                          |
   ↪ 29.679298400878906 MB | N/A |
|    ─Selection_31             | 10000.00 | 10000 | cop[tikv] |
   ↪                             | time:0ns, loops:0, tikv_task:{proc max:44ms,
   ↪  min:28ms, p80:44ms, p95:44ms, iters:84, tasks:3}
   ↪
   ↪ | eq(test.t1.int_col, 1)
   ↪                                                  | N/A               |
   ↪ N/A |
|      ─TableFullScan_30       | 71010.00 | 71010 | cop[tikv] | table:t1
   ↪                | time:0ns, loops:0, tikv_task:{proc max:44ms, min:28ms
   ↪ , p80:44ms, p95:44ms, iters:84, tasks:3}
   ↪
   ↪ | keep order:false
   ↪                                                        | N/A
   ↪                 | N/A |
|  ─IndexLookUp_10(Probe)      | 9.00    | 0      | root     |
   ↪                              | time:103.93801ms, loops:7
   ↪
   ↪ |
   ↪
   ↪ | 2.1787109375 KB    | N/A |
|    ─IndexRangeScan_8(Build)  | 9.00    | 0      | cop[tikv] | table:t2,
   ↪ index:t1_id(t1_id) | time:0s, loops:0, cop_task: {num: 7, max:
   ↪ 23.969244ms, min: 12.003682ms, avg: 14.659066ms, p95: 23.969244ms,
   ↪ tot_proc: 100ms, rpc_num: 7, rpc_time: 102.435966ms,
   ↪ copr_cache_hit_ratio: 0.00}, tikv_task:{proc max:24ms, min:12ms, p80
   ↪ :16ms, p95:24ms, iters:7, tasks:7} | range: decided by [eq(test.t2.
   ↪ t1_id, test.t1.id)], keep order:false | N/A | N/A |
|    ─TableRowIDScan_9(Probe)  | 9.00    | 0      | cop[tikv] | table:t2
   ↪                | time:0ns, loops:0
```

```
    ↪
    ↪ | keep order:false
    ↪                                                  | N/A
    ↪                 | N/A |
+--
    ↪ ------------------------------+---------+--------+----------+---------------
    ↪
7 rows in set (0.14 sec)

+--
    ↪ ----------------------------+---------+--------+----------+----------------+--
    ↪
| id                          | estRows | actRows | task    | access object |
    ↪ execution info
    ↪
    ↪ | operator info                                | memory             |
    ↪ disk   |
+--
    ↪ ----------------------------+---------+--------+----------+----------------+--
    ↪
| HashJoin_31                 | 90000.00 | 0     | root     |                | time
    ↪ :402.263795ms, loops:1, build_hash_table:{total:128.467151ms, fetch
    ↪ :126.871282ms, build:1.595869ms}, probe:{concurrency:5, total
    ↪ :2.010969815s, max:402.212295ms, probe:8.924769ms, fetch:2.002045046s
    ↪ }              | inner join, equal:[eq(test.t1.id, test.t2.t1_id)] |
    ↪ 29.689788818359375 MB | 0 Bytes |
|  ─TableReader_34(Build)   | 10000.00 | 10000 | root    |                |
    ↪ time:126.765972ms, loops:11, cop_task: {num: 3, max: 126.721293ms,
    ↪ min: 54.375481ms, avg: 84.518849ms, p95: 126.721293ms, max_proc_keys:
    ↪  31724, p95_proc_keys: 31724, tot_proc: 208ms, rpc_num: 3, rpc_time:
    ↪ 253.478218ms, copr_cache_hit_ratio: 0.00} | data:Selection_33 |
    ↪ 29.679292678833008 MB | N/A |
|    ─Selection_33          | 10000.00 | 10000 | cop[tikv] |                |
    ↪ time:0ns, loops:0, tikv_task:{proc max:72ms, min:56ms, p80:72ms, p95
    ↪ :72ms, iters:84, tasks:3}
    ↪
    ↪ | eq(test.t1.int_col, 1)                       | N/A                | N
    ↪ /A    |
|      ─TableFullScan_32    | 71010.00 | 71010 | cop[tikv] | table:t1  |
    ↪ time:0ns, loops:0, tikv_task:{proc max:72ms, min:56ms, p80:72ms, p95
    ↪ :72ms, iters:84, tasks:3}
    ↪
    ↪ | keep order:false                             | N/A                | N
    ↪ /A    |
|  ─TableReader_36(Probe)   | 90000.00 | 90000 | root    |                |
```

```
    ↪ time:400.447175ms, loops:90, cop_task: {num: 3, max: 400.838264ms,
    ↪ min: 309.474053ms, avg: 341.01943ms, p95: 400.838264ms, max_proc_keys
    ↪ : 31719, p95_proc_keys: 31719, tot_proc: 528ms, rpc_num: 3, rpc_time:
    ↪  1.02298055s, copr_cache_hit_ratio: 0.00} | data:TableFullScan_35 |
    ↪ 182.62786674499512 MB | N/A |
|    └─TableFullScan_35      | 90000.00 | 90000 | cop[tikv] | table:t2  |
    ↪ time:0ns, loops:0, tikv_task:{proc max:108ms, min:72ms, p80:108ms,
    ↪ p95:108ms, iters:102, tasks:3}
    ↪
    ↪ | keep order:false                          | N/A               | N
    ↪ /A    |
+--
    ↪ ---------------------------+---------+--------+----------+---------------+--
    ↪
6 rows in set (0.40 sec)

+--
    ↪ ---------------------------+---------+--------+----------+---------------+--
    ↪
| id                          | estRows | actRows | task    | access object |
    ↪ execution info
    ↪
    ↪ | operator info                             | memory            |
    ↪ disk    |
+--
    ↪ ---------------------------+---------+--------+----------+---------------+--
    ↪
| HashJoin_32                 | 90000.00 | 0     | root     |               | time
    ↪ :356.282882ms, loops:1, build_hash_table:{total:154.187155ms, fetch
    ↪ :151.259305ms, build:2.92785ms}, probe:{concurrency:5, total
    ↪ :1.781087041s, max:356.238312ms, probe:7.406146ms, fetch:1.773680895s
    ↪ }                  | inner join, equal:[eq(test.t1.id, test.t2.t1_id)] |
    ↪  29.689788818359375 MB | 0 Bytes |
|  ├─TableReader_41(Build)   | 10000.00 | 10000 | root    |               |
    ↪ time:151.190175ms, loops:11, cop_task: {num: 3, max: 151.055697ms,
    ↪ min: 56.214348ms, avg: 96.70463ms, p95: 151.055697ms, max_proc_keys:
    ↪ 31724, p95_proc_keys: 31724, tot_proc: 240ms, rpc_num: 3, rpc_time:
    ↪ 290.019942ms, copr_cache_hit_ratio: 0.00} | data:Selection_40 |
    ↪ 29.679292678833008 MB | N/A |
|  │  └─Selection_40          | 10000.00 | 10000 | cop[tikv] |               |
    ↪ time:0ns, loops:0, tikv_task:{proc max:80ms, min:56ms, p80:80ms, p95
    ↪ :80ms, iters:84, tasks:3}
    ↪
    ↪ | eq(test.t1.int_col, 1)                    | N/A               | N
    ↪ /A    |
```

```
|     └─TableFullScan_39    | 71010.00 | 71010 | cop[tikv] | table:t1  |
   ↪ time:0ns, loops:0, tikv_task:{proc max:80ms, min:56ms, p80:80ms, p95
   ↪ :80ms, iters:84, tasks:3}
   ↪
   ↪ | keep order:false                          | N/A                | N
   ↪ /A    |
|  └─TableReader_43(Probe)   | 90000.00 | 90000 | root    |             |
   ↪ time:354.68523ms, loops:90, cop_task: {num: 3, max: 354.313475ms, min
   ↪ : 328.460762ms, avg: 345.530558ms, p95: 354.313475ms, max_proc_keys:
   ↪ 31719, p95_proc_keys: 31719, tot_proc: 508ms, rpc_num: 3, rpc_time:
   ↪ 1.036492374s, copr_cache_hit_ratio: 0.00} | data:TableFullScan_42 |
   ↪ 182.62786102294922 MB | N/A |
|    └─TableFullScan_42      | 90000.00 | 90000 | cop[tikv] | table:t2  |
   ↪ time:0ns, loops:0, tikv_task:{proc max:84ms, min:64ms, p80:84ms, p95
   ↪ :84ms, iters:102, tasks:3}
   ↪
   ↪ | keep order:false                          | N/A                | N
   ↪ /A    |
+--
   ↪ ---------------------------+----------+---------+-----------+---------------+--
   ↪
6 rows in set (0.36 sec)
```

Note:

In the above example, the SQL Optimizer selects the hash join plan which performs worse than the index join. Query optimization is an NP-complete problem, and less-than-optimal plans might be chosen. If this is a frequent query, it is recommended to use SQL Plan Management to bind a hint to a query, which can be easier to manage than inserting hints into queries that your application sends to TiDB.

Variations of Index Join

An index join operation using the hint `INL_JOIN` creates a hash table of the intermediate results before joining on the outer table. TiDB also supports creating a hash table on the outer table using the hint `INL_HASH_JOIN`. Each of these variations of index join is automatically selected by the SQL Optimizer.

Configuration

Index join performance is influenced by the following system variables:

- tidb_index_join_batch_size (default value: 25000) - the batch size of index lookup join operations.
- tidb_index_lookup_join_concurrency (default value: 4) - the number of concurrent index lookup tasks.

11.3.2.4.2 Hash Join

In a hash join operation, TiDB reads and caches the data on the Build side of the join in a hash table, and then reads the data on the Probe side of the join, probing the hash table to access required rows. Hash joins require more memory to execute than index joins but execute much faster when there are a lot of rows that need to be joined. The hash join operator is multi-threaded in TiDB and executes in parallel.

An example of hash join is as follows:

```
EXPLAIN SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id = t2.id;
```

```
+--
   ↪ ---------------------------+----------+----------+---------------+-----------
   ↪
| id                          | estRows  | task     | access object | operator
   ↪ info                                |
+--
   ↪ ---------------------------+----------+----------+---------------+-----------
   ↪
| HashJoin_27                 | 142020.00 | root    |               | inner join,
   ↪ equal:[eq(test.t1.id, test.t2.id)] |
| ├─TableReader_29(Build)     | 142020.00 | root    |               | data:
   ↪ TableFullScan_28                    |
| │ └─TableFullScan_28        | 142020.00 | cop[tikv] | table:t1 | keep order:
   ↪ false                               |
| └─TableReader_31(Probe)     | 180000.00 | root    |               | data:
   ↪ TableFullScan_30                    |
|   └─TableFullScan_30        | 180000.00 | cop[tikv] | table:t2 | keep order:
   ↪ false                               |
+--
   ↪ ---------------------------+----------+----------+---------------+-----------
   ↪
5 rows in set (0.00 sec)
```

For the execution process of HashJoin_27, TiDB performs the following operations in order:

1. Cache the data of the Build side in memory.
2. Construct a Hash Table on the Build side based on the cached data.
3. Read the data at the Probe side.

4. Use the data of the `Probe` side to probe the Hash Table.
5. Return qualified data to the user.

The `operator info` column in the `EXPLAIN` result table also records other information about `HashJoin_27`, including whether the query is Inner Join or Outer Join, and what are the conditions of Join. In the above example, the query is an Inner Join, where the Join condition `equal:[eq(test.t1.id, test.t2.id)]` partly corresponds with the query condition `WHERE t1.id = t2.id`. The operator info of the other Join operators in the following examples is similar to this one.

Runtime Statistics

If `tidb_mem_quota_query` (default value: 1GB) is exceeded, TiDB will attempt to use temporary storage on condition that the `oom-use-tmp-storage` value is `true` (default). This means that the `Build` operator used as part of the hash join might be created on disk. Runtime statistics such as memory usage are visible in the `execution info` of the `EXPLAIN ANALYZE` result table. The following example shows the output of `EXPLAIN ANALYZE` with a 1GB (default) `tidb_mem_quota_query` quota, and a 500MB quota. At 500MB, disk is used for temporary storage:

```
EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id =
    ↪  t2.id;
SET tidb_mem_quota_query=500 * 1024 * 1024;
EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id =
    ↪  t2.id;
```

```
+--
    ↪ ---------------------------+----------+---------+----------+---------------+--
    ↪
| id                          | estRows  | actRows | task     | access object |
    ↪ execution info
    ↪
    ↪ | operator info                                | memory             |
    ↪ disk    |
+--
    ↪ ---------------------------+----------+---------+----------+---------------+--
    ↪
| HashJoin_27                 | 142020.00 | 71010 | root     |               | time
    ↪ :647.508572ms, loops:72, build_hash_table:{total:579.254415ms, fetch
    ↪ :566.91012ms, build:12.344295ms}, probe:{concurrency:5, total
    ↪ :3.23315006s, max:647.520113ms, probe:330.884716ms, fetch:2.902265344
    ↪ s}                | inner join, equal:[eq(test.t1.id, test.t2.id)] |
    ↪ 209.61642456054688 MB | 0 Bytes |
|  ─TableReader_29(Build)  | 142020.00 | 71010 | root     |               |
    ↪ time:567.088247ms, loops:72, cop_task: {num: 2, max: 569.809411ms,
    ↪ min: 369.67451ms, avg: 469.74196ms, p95: 569.809411ms, max_proc_keys:
```

```
   ↪  39245, p95_proc_keys: 39245, tot_proc: 400ms, rpc_num: 2, rpc_time:
   ↪ 939.447231ms, copr_cache_hit_ratio: 0.00} | data:TableFullScan_28 |
   ↪ 210.2100534439087 MB | N/A |
|    └─TableFullScan_28     | 142020.00 | 71010 | cop[tikv] | table:t1  |
   ↪ proc max:64ms, min:48ms, p80:64ms, p95:64ms, iters:79, tasks:2
   ↪
   ↪ | keep order:false                          | N/A                | N/A
   ↪     |
|  └─TableReader_31(Probe)  | 180000.00 | 90000 | root    |             |
   ↪ time:337.233636ms, loops:91, cop_task: {num: 3, max: 569.790741ms,
   ↪ min: 332.758911ms, avg: 421.543165ms, p95: 569.790741ms,
   ↪ max_proc_keys: 31719, p95_proc_keys: 31719, tot_proc: 500ms, rpc_num:
   ↪  3, rpc_time: 1.264570696s, copr_cache_hit_ratio: 0.00} | data:
   ↪ TableFullScan_30 | 267.1126985549927 MB | N/A |
|    └─TableFullScan_30     | 180000.00 | 90000 | cop[tikv] | table:t2  |
   ↪ proc max:84ms, min:72ms, p80:84ms, p95:84ms, iters:102, tasks:3
   ↪
   ↪ | keep order:false                          | N/A                | N/A
   ↪     |
+--
   ↪ --------------------------+----------+--------+----------+---------------+--
   ↪
5 rows in set (0.65 sec)

Query OK, 0 rows affected (0.00 sec)

+--
   ↪ --------------------------+----------+--------+----------+---------------+--
   ↪
| id                        | estRows  | actRows | task    | access object |
   ↪ execution info
   ↪
   ↪ | operator info                             | memory             |
   ↪ disk              |
+--
   ↪ --------------------------+----------+--------+----------+---------------+--
   ↪
| HashJoin_27               | 142020.00 | 71010 | root    |               | time
   ↪ :963.983353ms, loops:72, build_hash_table:{total:775.961447ms, fetch
   ↪ :503.789677ms, build:272.17177ms}, probe:{concurrency:5, total
   ↪ :4.805454793s, max:963.973133ms, probe:922.156835ms, fetch
   ↪ :3.883297958s}           | inner join, equal:[eq(test.t1.id, test.t2.
   ↪ id)] | 93.53974533081055 MB | 210.7459259033203 MB |
|  ├─TableReader_29(Build)  | 142020.00 | 71010 | root    |             |
   ↪ time:504.062018ms, loops:72, cop_task: {num: 2, max: 509.276857ms,
```

```
   ↪ min: 402.66386ms, avg: 455.970358ms, p95: 509.276857ms, max_proc_keys
   ↪ : 39245, p95_proc_keys: 39245, tot_proc: 384ms, rpc_num: 2, rpc_time:
   ↪  911.893237ms, copr_cache_hit_ratio: 0.00} | data:TableFullScan_28 |
   ↪ 210.20934200286865 MB | N/A |
|    TableFullScan_28     | 142020.00 | 71010 | cop[tikv] | table:t1  |
   ↪ proc max:88ms, min:72ms, p80:88ms, p95:88ms, iters:79, tasks:2
   ↪
   ↪ | keep order:false                          | N/A                | N/A
   ↪                 |
|  TableReader_31(Probe)  | 180000.00 | 90000 | root    |             |
   ↪ time:363.058382ms, loops:91, cop_task: {num: 3, max: 412.659191ms,
   ↪ min: 358.489688ms, avg: 391.463008ms, p95: 412.659191ms,
   ↪ max_proc_keys: 31719, p95_proc_keys: 31719, tot_proc: 484ms, rpc_num:
   ↪  3, rpc_time: 1.174326746s, copr_cache_hit_ratio: 0.00} | data:
   ↪ TableFullScan_30 | 267.11340618133545 MB | N/A |
|    TableFullScan_30     | 180000.00 | 90000 | cop[tikv] | table:t2  |
   ↪ proc max:92ms, min:64ms, p80:92ms, p95:92ms, iters:102, tasks:3
   ↪
   ↪ | keep order:false                          | N/A                | N/A
   ↪                 |
+--
   ↪ -------------------------+----------+---------+----------+---------------+--
   ↪
5 rows in set (0.98 sec)
```

Configuration

Hash join performance is influenced by the following system variables:

- `tidb_mem_quota_query` (default value: 1GB) - if the memory quota for a query is exceeded, TiDB will attempt to spill the `Build` operator of a hash join to disk to save memory.
- `tidb_hash_join_concurrency` (default value: `5`) - the number of concurrent hash join tasks.

11.3.2.4.3 Merge Join

Merge join is a special sort of join that applies when both sides of the join are read in sorted order. It can be described as similar to an *efficient zipper merge*: as data is read on both the `Build` and the `Probe` sides of the join, the join operation works like a streaming operation. Merge joins require far less memory than hash join but do not execute in parallel.

The following is an example:

```
EXPLAIN SELECT /*+ MERGE_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id = t2.id;
```

```
+--
   ↪ -------------------------+----------+----------+---------------+-----------
   ↪
| id                        | estRows  | task     | access object | operator
   ↪ info                                |
+--
   ↪ -------------------------+----------+----------+---------------+-----------
   ↪
| MergeJoin_7               | 142020.00 | root    |               | inner join,
   ↪ left key:test.t1.id, right key:test.t2.id |
|  ─TableReader_12(Build)  | 180000.00 | root    |               | data:
   ↪ TableFullScan_11                          |
|    ─TableFullScan_11     | 180000.00 | cop[tikv] | table:t2 | keep order:
   ↪ true                                      |
|  ─TableReader_10(Probe)  | 142020.00 | root    |               | data:
   ↪ TableFullScan_9                           |
|    ─TableFullScan_9      | 142020.00 | cop[tikv] | table:t1 | keep order:
   ↪ true                                      |
+--
   ↪ -------------------------+----------+----------+---------------+-----------
   ↪
5 rows in set (0.00 sec)
```

For the execution process of the merge join operator, TiDB performs the following operations:

1. Read all the data of a Join Group from the `Build` side into the memory.
2. Read the data of the `Probe` side.
3. Compare whether each row of data on the `Probe` side matches a complete Join Group on the `Build` side. Apart from equivalent conditions, there are non-equivalent conditions. Here "match" mainly refers to checking whether non-equivalent conditions are met. Join Group refers to the data with the same value among all Join Keys.

11.3.2.5 Explain Statements in the MPP Mode

TiDB supports using the MPP mode to execute queries. In the MPP mode, the TiDB optimizer generates execution plans for MPP. Note that the MPP mode is only available for tables that have replicas on TiFlash.

The examples in this document are based on the following sample data:

```
CREATE TABLE t1 (id int, value int);
INSERT INTO t1 values(1,2),(2,3),(1,3);
ALTER TABLE t1 set tiflash replica 1;
ANALYZE TABLE t1;
```

```
SET tidb_allow_mpp = 1;
```

11.3.2.5.1 MPP query fragments and MPP tasks

In the MPP mode, a query is logically sliced into multiple query fragments. Take the following statement as an example:

```
EXPLAIN SELECT COUNT(*) FROM t1 GROUP BY id;
```

This query is divided into two fragments in the MPP mode. One for the first-stage aggregation and the other for the second-stage aggregation, also the final aggregation. When this query is executed, each query fragment is instantiated into one or more MPP tasks.

11.3.2.5.2 Exchange operators

`ExchangeReceiver` and `ExchangeSenderare` two exchange operators specific for MPP execution plans. The `ExchangeReceiver` operator reads data from downstream query fragments and the `ExchangeSender` operator sends data from downstream query fragments to upstream query fragments. In the MPP mode, the root operator of each MPP query fragment is `ExchangeSender`, meaning that query fragments are delimited by the `ExchangeSender` operator.

The following is a simple MPP execution plan:

```
EXPLAIN SELECT COUNT(*) FROM t1 GROUP BY id;
```

```
+--
   ↪ ----------------------------------+---------+------------------+--------------
   ↪
| id                                | estRows | task              | access object |
   ↪  operator info                                  |
+--
   ↪ ----------------------------------+---------+------------------+--------------
   ↪
| TableReader_31                    | 2.00    | root              |               |
   ↪ data:ExchangeSender_30                          |
|  ─ExchangeSender_30               | 2.00    | batchCop[tiflash] |               |
   ↪ ExchangeType: PassThrough                       |
|    ─Projection_26                 | 2.00    | batchCop[tiflash] |               |
   ↪ Column#4                                        |
|      ─HashAgg_27                  | 2.00    | batchCop[tiflash] |               |
   ↪ group by:test.t1.id, funcs:sum(Column#7)->Column#4 |
|        ─ExchangeReceiver_29       | 2.00    | batchCop[tiflash] |               |
   ↪                                                 |
|          ─ExchangeSender_28       | 2.00    | batchCop[tiflash] |               |
   ↪ ExchangeType: HashPartition, Hash Cols: test.t1.id |
```

```
|           └─HashAgg_9            | 2.00   | batchCop[tiflash] |           |
   ↪ group by:test.t1.id, funcs:count(1)->Column#7 |
|             └─TableFullScan_25   | 3.00   | batchCop[tiflash] | table:t1  |
   ↪ keep order:false                             |
+--
   ↪ ---------------------------------+---------+-------------------+-------------
   ↪
```

The above execution plan contains two query fragments:

- The first is `[TableFullScan_25, HashAgg_9, ExchangeSender_28]`, which is mainly responsible for the first-stage aggregation.
- The second is `[ExchangeReceiver_29, HashAgg_27, Projection_26, ExchangeSender_30 ↪ ]`, which is mainly responsible for the second-stage aggregation.

The `operator info` column of the `ExchangeSender` operator shows the exchange type information. Currently, there are three exchange types. See the following:

- HashPartition: The `ExchangeSender` operator firstly partitions data according to the Hash values and then distributes data to the `ExchangeReceiver` operator of upstream MPP tasks. This exchange type is often used for Hash Aggregation and Shuffle Hash Join algorithms.
- Broadcast: The `ExchangeSender` operator distributes data to upstream MPP tasks through broadcast. This exchange type is often used for Broadcast Join.
- PassThrough: The `ExchangeSender` operator sends data to the only upstream MPP task, which is different from the Broadcast type. This exchange type is often used when returning data to TiDB.

In the example execution plan, the exchange type of the operator `ExchangeSender_28` is HashPartition, meaning that it performs the Hash Aggregation algorithm. The exchange type of the operator `ExchangeSender_30` is PassThrough, meaning that it is used to return data to TiDB.

MPP is also often applied to join operations. The MPP mode in TiDB supports the following two join algorithms:

- Shuffle Hash Join: Shuffle the data input from the join operation using the HashPartition exchange type. Then, upstream MPP tasks join data within the same partition.
- Broadcast Join: Broadcast data of the small table in the join operation to each node, after which each node joins the data separately.

The following is a typical execution plan for Shuffle Hash Join:

```
SET tidb_broadcast_join_threshold_count=0;
SET tidb_broadcast_join_threshold_size=0;
EXPLAIN SELECT COUNT(*) FROM t1 a JOIN t1 b ON a.id = b.id;
```

```
+--
   ↪ ------------------------------------+--------+-------------+---------------+
   ↪
| id                                | estRows | task      | access object |
   ↪ operator info                                |
+--
   ↪ ------------------------------------+--------+-------------+---------------+
   ↪
| StreamAgg_14                      | 1.00    | root        |               |
   ↪ funcs:count(1)->Column#7                     |
|  TableReader_48                    | 9.00    | root        |               |
   ↪ data:ExchangeSender_47                       |
|    ExchangeSender_47               | 9.00    | cop[tiflash] |               |
   ↪ ExchangeType: PassThrough                    |
|      HashJoin_44                   | 9.00    | cop[tiflash] |               |
   ↪ inner join, equal:[eq(test.t1.id, test.t1.id)] |
|        ExchangeReceiver_19(Build) | 6.00    | cop[tiflash] |               |
   ↪                                              |
|          ExchangeSender_18         | 6.00    | cop[tiflash] |               |
   ↪ ExchangeType: HashPartition, Hash Cols: test.t1.id |
|            Selection_17            | 6.00    | cop[tiflash] |               |
   ↪ not(isnull(test.t1.id))                      |
|              TableFullScan_16      | 6.00    | cop[tiflash] | table:a       |
   ↪ keep order:false                             |
|        ExchangeReceiver_23(Probe) | 6.00    | cop[tiflash] |               |
   ↪                                              |
|          ExchangeSender_22         | 6.00    | cop[tiflash] |               |
   ↪ ExchangeType: HashPartition, Hash Cols: test.t1.id |
|            Selection_21            | 6.00    | cop[tiflash] |               |
   ↪ not(isnull(test.t1.id))                      |
|              TableFullScan_20      | 6.00    | cop[tiflash] | table:b       |
   ↪ keep order:false                             |
+--
   ↪ ------------------------------------+--------+-------------+---------------+
   ↪
12 rows in set (0.00 sec)
```

In the above execution plan:

- The query fragment `[TableFullScan_20, Selection_21, ExchangeSender_22]`

reads data from table b and shuffles data to upstream MPP tasks.

- The query fragment `[TableFullScan_16, Selection_17, ExchangeSender_18]` reads data from table a and shuffles data to upstream MPP tasks.
- The query fragment `[ExchangeReceiver_19, ExchangeReceiver_23, HashJoin_44 ↪ , ExchangeSender_47]` joins all data and returns it to TiDB.

A typical execution plan for Broadcast Join is as follows:

```
EXPLAIN SELECT COUNT(*) FROM t1 a JOIN t1 b ON a.id = b.id;
```

```
+--
   ↪ ------------------------------------+--------+-------------+--------------+
   ↪
| id                                  | estRows | task      | access object |
   ↪ operator info                           |
+--
   ↪ ------------------------------------+--------+-------------+--------------+
   ↪
| StreamAgg_15                        | 1.00    | root        |              |
   ↪ funcs:count(1)->Column#7                  |
|  ─TableReader_47                     | 9.00    | root        |              |
   ↪ data:ExchangeSender_46                    |
|    ─ExchangeSender_46                | 9.00    | cop[tiflash] |              |
   ↪ ExchangeType: PassThrough                 |
|      ─HashJoin_43                    | 9.00    | cop[tiflash] |              |
   ↪ inner join, equal:[eq(test.t1.id, test.t1.id)] |
|        ─ExchangeReceiver_20(Build) | 6.00    | cop[tiflash] |              |
   ↪                                           |
|          ─ExchangeSender_19          | 6.00    | cop[tiflash] |              |
   ↪ ExchangeType: Broadcast                   |
|            ─Selection_18             | 6.00    | cop[tiflash] |              |
   ↪ not(isnull(test.t1.id))                   |
|              ─TableFullScan_17       | 6.00    | cop[tiflash] | table:a      |
   ↪ keep order:false                          |
|        ─Selection_22(Probe)          | 6.00    | cop[tiflash] |              |
   ↪ not(isnull(test.t1.id))                   |
|          ─TableFullScan_21           | 6.00    | cop[tiflash] | table:b      |
   ↪ keep order:false                          |
+--
   ↪ ------------------------------------+--------+-------------+--------------+
   ↪
```

In the above execution plan:

- The query fragment `[TableFullScan_17, Selection_18, ExchangeSender_19]` reads data from the small table (table a) and broadcasts the data to each node that

contains data from the large table (table b).

- The query fragment `[TableFullScan_21, Selection_22, ExchangeReceiver_20, HashJoin_43, ExchangeSender_46]` joins all data and returns it to TiDB.

11.3.2.5.3 `EXPLAIN ANALYZE` statements in the MPP mode

The `EXPLAIN ANALYZE` statement is similar to `EXPLAIN`, but it also outputs some runtime information.

The following is the output of a simple `EXPLAIN ANALYZE` example:

```
EXPLAIN ANALYZE SELECT COUNT(*) FROM t1 GROUP BY id;
```

```
+--
   ↪ ----------------------------------+---------+---------+-------------------+-----
   ↪
| id                                | estRows | actRows | task          | access
   ↪  object | execution info
   ↪                                                                          |
   ↪ operator info                                                 | memory |
   ↪ disk |
+--
   ↪ ----------------------------------+---------+---------+-------------------+-----
   ↪
| TableReader_31                    | 4.00    | 2       | root              |
   ↪              | time:44.5ms, loops:2, cop_task: {num: 1, max: 0s,
   ↪ proc_keys: 0, copr_cache_hit_ratio: 0.00} | data:ExchangeSender_30
   ↪                       | N/A    | N/A |
|  ─ExchangeSender_30               | 4.00    | 2       | batchCop[tiflash] |
   ↪           | tiflash_task:{time:16.5ms, loops:1, threads:1}
   ↪                                     | ExchangeType: PassThrough, tasks:
   ↪ [2, 3, 4]             | N/A    | N/A |
|    ─Projection_26                 | 4.00    | 2       | batchCop[tiflash] |
   ↪           | tiflash_task:{time:16.5ms, loops:1, threads:1}
   ↪                                     | Column#4
   ↪                                                   | N/A    | N/A |
|      ─HashAgg_27                  | 4.00    | 2       | batchCop[tiflash] |
   ↪           | tiflash_task:{time:16.5ms, loops:1, threads:1}
   ↪                                     | group by:test.t1.id, funcs:sum(
   ↪ Column#7)->Column#4     | N/A    | N/A |
|        ─ExchangeReceiver_29       | 4.00    | 2       | batchCop[tiflash] |
   ↪           | tiflash_task:{time:14.5ms, loops:1, threads:20}
   ↪                                     |
   ↪                                                   | N/A    | N/A
   ↪ |
```

```
|         └─ExchangeSender_28     | 4.00   | 0      | batchCop[tiflash] |
    ↪           | tiflash_task:{time:9.49ms, loops:0, threads:0}
    ↪                                     | ExchangeType: HashPartition, Hash
    ↪ Cols: test.t1.id, tasks: [1] | N/A | N/A |
|           └─HashAgg_9           | 4.00   | 0      | batchCop[tiflash] |
    ↪           | tiflash_task:{time:9.49ms, loops:0, threads:0}
    ↪                                     | group by:test.t1.id, funcs:count
    ↪ (1)->Column#7          | N/A    | N/A |
|             └─TableFullScan_25  | 6.00   | 0      | batchCop[tiflash] |
    ↪ table:t1  | tiflash_task:{time:9.49ms, loops:0, threads:0}
    ↪                                     | keep order:false
    ↪                                         | N/A    | N/A |
+--
    ↪ ---------------------------------+---------+---------+-------------------+-----
    ↪
```

Compared to the output of `EXPLAIN`, the `operator info` column of the operator `ExchangeSender` also shows `tasks`, which records the id of the MPP task that the query fragment instantiates into. In addition, each MPP operator has a `threads` field in the `execution info` column, which records the concurrency of operations when TiDB executes this operator. If the cluster consists of multiple nodes, this concurrency is the result of adding up the concurrency of all nodes.

11.3.2.6 Explain Statements That Use Subqueries

TiDB performs several optimizations to improve the performance of subqueries. This document describes some of these optimizations for common subqueries and how to interpret the output of `EXPLAIN`.

The examples in this document are based on the following sample data:

```
CREATE TABLE t1 (id BIGINT NOT NULL PRIMARY KEY auto_increment, pad1 BLOB,
    ↪ pad2 BLOB, pad3 BLOB, int_col INT NOT NULL DEFAULT 0);
CREATE TABLE t2 (id BIGINT NOT NULL PRIMARY KEY auto_increment, t1_id
    ↪ BIGINT NOT NULL, pad1 BLOB, pad2 BLOB, pad3 BLOB, INDEX(t1_id));
CREATE TABLE t3 (
 id INT NOT NULL PRIMARY KEY auto_increment,
 t1_id INT NOT NULL,
 UNIQUE (t1_id)
);

INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM dual;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
```

```
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
UPDATE t1 SET int_col = 1 WHERE pad1 = (SELECT pad1 FROM t1 ORDER BY RAND()
    ↪  LIMIT 1);
INSERT INTO t3 SELECT NULL, id FROM t1 WHERE id < 1000;

SELECT SLEEP(1);
ANALYZE TABLE t1, t2, t3;
```

11.3.2.6.1 Inner join (non-unique subquery)

In the following example, the `IN` subquery searches for a list of IDs from the table `t2` ↪ . For semantic correctness, TiDB needs to guarantee that the column `t1_id` is unique. Using `EXPLAIN`, you can see the execution plan used to remove duplicates and perform an `INNER JOIN` operation:

```
EXPLAIN SELECT * FROM t1 WHERE id IN (SELECT t1_id FROM t2);
```

```
+--
   ↪ -------------------------------+---------+----------+----------------------
   ↪
| id                            | estRows | task     | access object
   ↪            | operator info
   ↪
   ↪ |
+--
   ↪ -------------------------------+---------+----------+----------------------
   ↪
| IndexJoin_14                  | 5.00    | root     |
   ↪                             | inner join, inner:IndexLookUp_13, outer
   ↪ key:test.t2.t1_id, inner key:test.t1.id, equal cond:eq(test.t2.t1_id,
   ↪  test.t1.id) |
|  ─StreamAgg_49(Build)         | 5.00    | root     |
   ↪                             | group by:test.t2.t1_id, funcs:firstrow(
   ↪ test.t2.t1_id)->test.t2.t1_id                                          |
|    ─IndexReader_50            | 5.00    | root     |
   ↪                             | index:StreamAgg_39
   ↪
   ↪ |
|      ─StreamAgg_39            | 5.00    | cop[tikv] |
   ↪                            | group by:test.t2.t1_id,
   ↪
   ↪ |
|        ─IndexFullScan_31      | 50000.00 | cop[tikv] | table:t2, index:
   ↪ t1_id(t1_id) | keep order:true
   ↪
   ↪ |
|  ─IndexLookUp_13(Probe)       | 1.00    | root     |
   ↪                             |
   ↪
   ↪ |
|    ─IndexRangeScan_11(Build) | 1.00    | cop[tikv] | table:t1, index:
   ↪ PRIMARY(id) | range: decided by [eq(test.t1.id, test.t2.t1_id)], keep
   ↪  order:false                                    |
|    ─TableRowIDScan_12(Probe) | 1.00    | cop[tikv] | table:t1
   ↪                | keep order:false
```

```
    ↪
    ↪ |
+--
    ↪ ------------------------------+---------+----------+----------------------
    ↪
8 rows in set (0.00 sec)
```

From the query results above, you can see that TiDB uses the index join operation `IndexJoin_14` to join and transform the subquery. In the execution plan, the execution process is as follows:

1. The index scanning operator `IndexFullScan_31` at the TiKV side reads the values of the `t2.t1_id` column.
2. Some tasks of the `StreamAgg_39` operator deduplicate the values of `t1_id` in TiKV.
3. Some tasks of the `StreamAgg_49(Build)` operator deduplicate the values of `t1_id` in TiDB. The deduplication is performed by the aggregate function `firstrow(test.t2.t1_id)`.
4. The operation results are joined with the primary key of the `t1` table. The join condition is `eq(test.t1.id, test.t2.t1_id)`.

11.3.2.6.2 Inner join (unique subquery)

In the previous example, aggregation is required to ensure that the values of `t1_id` are unique before joining against the table `t1`. But in the following example, `t3.t1_id` is already guaranteed unique because of a `UNIQUE` constraint:

```
EXPLAIN SELECT * FROM t1 WHERE id IN (SELECT t1_id FROM t3);
```

```
+--
    ↪ ------------------------------+---------+----------+----------------------
    ↪
| id                            | estRows | task    | access object
    ↪           | operator info
    ↪
    ↪ |
+--
    ↪ ------------------------------+---------+----------+----------------------
    ↪
| IndexJoin_17                  | 1978.13 | root    |
    ↪                           | inner join, inner:IndexLookUp_16, outer key
    ↪ :test.t3.t1_id, inner key:test.t1.id, equal cond:eq(test.t3.t1_id,
    ↪ test.t1.id) |
|  ─TableReader_44(Build)       | 1978.00 | root    |
    ↪                           | data:TableFullScan_43
    ↪
    ↪ |
```

```
|    ─TableFullScan_43          | 1978.00 | cop[tikv] | table:t3
   ↪               | keep order:false
   ↪
   ↪ |
|  ─IndexLookUp_16(Probe)       | 1.00    | root      |
   ↪                            |
   ↪
   ↪ |
|    ─IndexRangeScan_14(Build) | 1.00    | cop[tikv] | table:t1, index:
   ↪ PRIMARY(id) | range: decided by [eq(test.t1.id, test.t3.t1_id)], keep
   ↪  order:false                                    |
|    ─TableRowIDScan_15(Probe) | 1.00    | cop[tikv] | table:t1
   ↪                | keep order:false
   ↪
   ↪ |
+--
   ↪ ------------------------------+---------+-----------+----------------------
   ↪
6 rows in set (0.01 sec)
```

Semantically because `t3.t1_id` is guaranteed unique, it can be executed directly as an `INNER JOIN`.

11.3.2.6.3 Semi join (correlated subquery)

In the previous two examples, TiDB is able to perform an `INNER JOIN` operation after the data inside the subquery is made unique (via `StreamAgg`) or guaranteed unique. Both joins are performed using an Index Join.

In this example, TiDB chooses a different execution plan:

```
EXPLAIN SELECT * FROM t1 WHERE id IN (SELECT t1_id FROM t2 WHERE t1_id !=
   ↪ t1.int_col);
```

```
+--
   ↪ -------------------------+----------+----------+-------------------------
   ↪
| id                        | estRows  | task     | access object           |
   ↪  operator info
   ↪
   ↪ |
+--
   ↪ -------------------------+----------+----------+-------------------------
   ↪
| MergeJoin_9               | 45446.40 | root     |                         |
   ↪  semi join, left key:test.t1.id, right key:test.t2.t1_id, other cond:
   ↪ ne(test.t2.t1_id, test.t1.int_col) |
```

```
|  ─IndexReader_24(Build)  | 180000.00 | root   |
   ↪ | index:IndexFullScan_23
   ↪                                                                    |
|    ─IndexFullScan_23     | 180000.00 | cop[tikv] | table:t2, index:t1_id(
   ↪ t1_id) | keep order:true
   ↪                                                                   |
|  ─TableReader_22(Probe)  | 56808.00 | root    |
   ↪ | data:Selection_21
   ↪
   ↪ |
|    ─Selection_21         | 56808.00 | cop[tikv] |
   ↪ | ne(test.t1.id, test.t1.int_col)
   ↪                                                                |
|      ─TableFullScan_20   | 71010.00 | cop[tikv] | table:t1
   ↪ | keep order:true
   ↪
   ↪ |
+--
   ↪ --------------------------+----------+----------+-------------------------
   ↪
6 rows in set (0.00 sec)
```

From the result above, you can see that TiDB uses a `Semi Join` algorithm. Semi-join differs from inner join: semi-join only permits the first value on the right key (`t2.t1_id`), which means that the duplicates are eliminated as a part of the join operator task. The join algorithm is also Merge Join, which is like an efficient zipper-merge as the operator reads data from both the left and the right side in sorted order.

The original statement is considered a *correlated subquery*, because the subquery refers to a column (`t1.int_col`) that exists outside of the subquery. However, the output of `EXPLAIN` shows the execution plan after the subquery decorrelation optimization has been applied. The condition `t1_id != t1.int_col` is rewritten to `t1.id != t1.int_col`. TiDB can perform this in `─Selection_21` as it is reading data from the table `t1`, so this decorrelation and rewriting make the execution a lot more efficient.

11.3.2.6.4 Anti semi join (`NOT IN` subquery)

In the following example, the query semantically returns all rows from the table `t3` *unless* `t3.t1_id` is in the subquery:

```
EXPLAIN SELECT * FROM t3 WHERE t1_id NOT IN (SELECT id FROM t1 WHERE
   ↪ int_col < 100);
```

```
+--
   ↪ --------------------------+---------+----------+--------------+------------
   ↪
```

```
| id                          | estRows | task    | access object | operator
    ↪ info                                                          |
+--
    ↪ --------------------------+---------+----------+---------------+-------------
    ↪
| IndexMergeJoin_20           | 1598.40 | root    |               | anti semi join
    ↪ , inner:TableReader_15, outer key:test.t3.t1_id, inner key:test.t1.id
    ↪  |
|  ─TableReader_28(Build)     | 1998.00 | root    |               | data:
    ↪ TableFullScan_27                                                    |
|    ─TableFullScan_27        | 1998.00 | cop[tikv] | table:t3    | keep order:
    ↪ false                                                               |
|  ─TableReader_15(Probe)     | 1.00    | root    |               | data:
    ↪ Selection_14                                                        |
|    ─Selection_14            | 1.00    | cop[tikv] |               | lt(test.t1.
    ↪ int_col, 100)                                                       |
|      ─TableRangeScan_13     | 1.00    | cop[tikv] | table:t1    | range:
    ↪ decided by [test.t3.t1_id], keep order:true                         |
+--
    ↪ --------------------------+---------+----------+---------------+-------------
    ↪
6 rows in set (0.00 sec)
```

This query starts by reading the table `t3` and then probes the table `t1` based on the `PRIMARY KEY`. The join type is an *anti semi join*; anti because this example is for the non-existence of the value (`NOT IN`) and semi-join because only the first row needs to match before the join is rejected.

11.3.2.7 Explain Statements Using Aggregation

When aggregating data, the SQL Optimizer will select either a Hash Aggregation or Stream Aggregation operator. To improve query efficiency, aggregation is performed at both the coprocessor and TiDB layers. Consider the following example:

```
CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY auto_increment, pad1 BLOB,
    ↪ pad2 BLOB, pad3 BLOB);
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM dual;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
```

```
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
SELECT SLEEP(1);
ANALYZE TABLE t1;
```

From the output of SHOW TABLE REGIONS, you can see that this table is split into multiple Regions:

```
SHOW TABLE t1 REGIONS;
```

```
+--
    ↪ ---------+--------------+--------------+-----------+-----------------+-------+--
    ↪
| REGION_ID | START_KEY | END_KEY    | LEADER_ID | LEADER_STORE_ID | PEERS |
    ↪  SCATTERING | WRITTEN_BYTES | READ_BYTES | APPROXIMATE_SIZE(MB) |
    ↪ APPROXIMATE_KEYS |
+--
    ↪ ---------+--------------+--------------+-----------+-----------------+-------+--
    ↪
|       64 | t_64_      | t_64_r_31766 |     65 |             1 | 65    |
    ↪         0 |         1325 | 102033520 |               98 |
    ↪ 52797 |
|       66 | t_64_r_31766 | t_64_r_63531 |   67 |             1 | 67    |
    ↪         0 |         1325 |  72522521 |              104 |
    ↪ 78495 |
```

```
|       68 | t_64_r_63531 | t_64_r_95296 |   69 |             1 | 69   |
    ↪         0 |         1325 |         0 |                104 |
    ↪ 95433 |
|        2 | t_64_r_95296 |              |        3 |             1 | 3    |
    ↪         0 |         1501 |         0 |                 81 |
    ↪ 63211 |
+--
    ↪ ---------+--------------+--------------+----------+---------------+------+--
    ↪
4 rows in set (0.00 sec)
```

Using `EXPLAIN` with the following aggregation statement, you can see that `StreamAgg_8` is first performed on each Region inside TiKV. Each TiKV Region will then send one row back to TiDB, which aggregates the data from each Region in `StreamAgg_16`:

```
EXPLAIN SELECT COUNT(*) FROM t1;
```

```
+--
    ↪ -------------------------+----------+----------+--------------+-----------
    ↪
| id                        | estRows  | task     | access object | operator
    ↪ info             |
+--
    ↪ -------------------------+----------+----------+--------------+-----------
    ↪
| StreamAgg_16              | 1.00     | root     |               | funcs:count(
    ↪ Column#7)->Column#5 |
|  ─TableReader_17          | 1.00     | root     |               | data:
    ↪ StreamAgg_8             |
|    ─StreamAgg_8           | 1.00     | cop[tikv] |              | funcs:count
    ↪ (1)->Column#7    |
|      ─TableFullScan_15    | 242020.00 | cop[tikv] | table:t1 | keep order:
    ↪ false            |
+--
    ↪ -------------------------+----------+----------+--------------+-----------
    ↪
4 rows in set (0.00 sec)
```

This is easiest to observe in `EXPLAIN ANALYZE`, where the `actRows` matches the number of Regions from `SHOW TABLE REGIONS` because a `TableFullScan` is being used and there are no secondary indexes:

```
EXPLAIN ANALYZE SELECT COUNT(*) FROM t1;
```

```
+--
   ↪ ------------------------+----------+--------+----------+---------------+---
   ↪
| id                        | estRows  | actRows | task    | access object |
   ↪ execution info
   ↪
   ↪ | operator info               | memory   | disk |
+--
   ↪ ------------------------+----------+--------+----------+---------------+---
   ↪
| StreamAgg_16              | 1.00     | 1       | root     |               | time
   ↪ :12.609575ms, loops:2
   ↪
   ↪ | funcs:count(Column#7)->Column#5 | 372 Bytes | N/A |
|  ─TableReader_17          | 1.00     | 4       | root     |               | time
   ↪ :12.605155ms, loops:2, cop_task: {num: 4, max: 12.538245ms, min:
   ↪ 9.256838ms, avg: 10.895114ms, p95: 12.538245ms, max_proc_keys: 31765,
   ↪  p95_proc_keys: 31765, tot_proc: 48ms, rpc_num: 4, rpc_time:
   ↪ 43.530707ms, copr_cache_hit_ratio: 0.00} | data:StreamAgg_8 | 293
   ↪ Bytes | N/A |
|    ─StreamAgg_8           | 1.00     | 4       | cop[tikv] |               | proc
   ↪  max:12ms, min:12ms, p80:12ms, p95:12ms, iters:122, tasks:4
   ↪
   ↪ | funcs:count(1)->Column#7    | N/A      | N/A |
|      ─TableFullScan_15    | 242020.00 | 121010 | cop[tikv] | table:t1 |
   ↪ proc max:12ms, min:12ms, p80:12ms, p95:12ms, iters:122, tasks:4
   ↪
   ↪ | keep order:false            | N/A      | N/A |
+--
   ↪ ------------------------+----------+--------+----------+---------------+---
   ↪
4 rows in set (0.01 sec)
```

11.3.2.7.1 Hash Aggregation

The Hash Aggregation algorithm uses a hash table to store intermediate results while performing aggregation. It executes in parallel using multiple threads but consumes more memory than Stream Aggregation.

The following is an example of the `HashAgg` operator:

```
EXPLAIN SELECT /*+ HASH_AGG() */ count(*) FROM t1;
```

```
+--
   ↪ ------------------------+----------+----------+-------------+---------------
```

```
    ↪
| id                    | estRows  | task     | access object | operator
    ↪ info             |
+--
    ↪ ----------------------+----------+----------+---------------+-----------
    ↪
| HashAgg_9             | 1.00     | root     |               | funcs:count(
    ↪ Column#6)->Column#5 |
|  ─TableReader_10      | 1.00     | root     |               | data:
    ↪ HashAgg_5              |
|    ─HashAgg_5         | 1.00     | cop[tikv] |              | funcs:count
    ↪ (1)->Column#6    |
|      ─TableFullScan_8 | 242020.00 | cop[tikv] | table:t1 | keep order:
    ↪ false            |
+--
    ↪ ----------------------+----------+----------+---------------+-----------
    ↪
4 rows in set (0.00 sec)
```

The `operator info` shows that the hashing function used to aggregate the data is `funcs:count(1)->Column#6`.

11.3.2.7.2 Stream Aggregation

The Stream Aggregation algorithm usually consumes less memory than Hash Aggregation. However, this operator requires that data is sent ordered so that it can *stream* and apply the aggregation on values as they arrive.

Consider the following example:

```
CREATE TABLE t2 (id INT NOT NULL PRIMARY KEY, col1 INT NOT NULL);
INSERT INTO t2 VALUES (1, 9),(2, 3),(3,1),(4,8),(6,3);
EXPLAIN SELECT /*+ STREAM_AGG() */ col1, count(*) FROM t2 GROUP BY col1;
```

```
Query OK, 0 rows affected (0.11 sec)

Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0

+--
    ↪ ---------------------------+----------+----------+---------------+-----------
    ↪
| id                          | estRows | task     | access object | operator
    ↪ info
    ↪ |
```

```
+--
    ↪ --------------------------+---------+----------+--------------+-----------
    ↪
| Projection_4              | 8000.00 | root     |              | test.t2.col1
    ↪ , Column#3                                                                 |
|  └─StreamAgg_8            | 8000.00 | root     |              | group by:
    ↪ test.t2.col1, funcs:count(1)->Column#3, funcs:firstrow(test.t2.col1)
    ↪ ->test.t2.col1 |
|    └─Sort_13              | 10000.00 | root    |              | test.t2.
    ↪ col1
    ↪                                                                          |
|      └─TableReader_12     | 10000.00 | root    |              | data:
    ↪ TableFullScan_11
    ↪                                                                            |
|        └─TableFullScan_11 | 10000.00 | cop[tikv] | table:t2 | keep order:
    ↪ false, stats:pseudo
    ↪ |
+--
    ↪ --------------------------+---------+----------+--------------+-----------
    ↪
5 rows in set (0.00 sec)
```

In this example, the `└─Sort_13` operator can be eliminated by adding an index on `col1` ↪ . Once the index is added, the data can be read in order and the `└─Sort_13` operator is eliminated:

```
ALTER TABLE t2 ADD INDEX (col1);
EXPLAIN SELECT /*+ STREAM_AGG() */ col1, count(*) FROM t2 GROUP BY col1;
```

```
Query OK, 0 rows affected (0.28 sec)

+--
    ↪ --------------------------+---------+----------+-------------------------+
    ↪
| id                        | estRows | task     | access object           |
    ↪ operator info
    ↪
    ↪ |
+--
    ↪ --------------------------+---------+----------+-------------------------+
    ↪
| Projection_4              | 4.00    | root     |                         |
    ↪ test.t2.col1, Column#3
    ↪                                                                        |
|  └─StreamAgg_14           | 4.00    | root     |                           |
```

```
    ↪ group by:test.t2.col1, funcs:count(Column#4)->Column#3, funcs:
    ↪ firstrow(test.t2.col1)->test.t2.col1 |
|    ─IndexReader_15        | 4.00    | root     |                          |
    ↪ index:StreamAgg_8
    ↪                                                                          |
|      ─StreamAgg_8         | 4.00    | cop[tikv] |                          |
    ↪ group by:test.t2.col1, funcs:count(1)->Column#4
    ↪                                              |
|        ─IndexFullScan_13  | 5.00    | cop[tikv] | table:t2, index:col1(col1
    ↪ ) | keep order:true, stats:pseudo
    ↪                                                        |
+--
    ↪ ---------------------------+---------+----------+-------------------------+
    ↪
5 rows in set (0.00 sec)
```

11.3.2.8 EXPLAIN Statements Using Views

`EXPLAIN` displays the tables and indexes that a view references, not the name of the view itself. This is because views are only virtual tables and do not store any data themselves. The definition of the view and the rest of the statement are merged together during SQL optimization.

From the bikeshare example database, you can see that the following two queries are executed in a similar manner:

```
ALTER TABLE trips ADD INDEX (duration);
CREATE OR REPLACE VIEW long_trips AS SELECT * FROM trips WHERE duration >
    ↪ 3600;
EXPLAIN SELECT * FROM long_trips;
EXPLAIN SELECT * FROM trips WHERE duration > 3600;
```

```
Query OK, 0 rows affected (2 min 10.11 sec)

Query OK, 0 rows affected (0.13 sec)

+--
    ↪ ----------------------------+-----------+----------+-----------------------
    ↪
| id                           | estRows   | task     | access object
    ↪                      | operator info                    |
+--
    ↪ ----------------------------+-----------+----------+-----------------------
    ↪
| IndexLookUp_12               | 6372547.67 | root    |
    ↪                                    |                                  |
```

```
|  ─IndexRangeScan_10(Build)  | 6372547.67 | cop[tikv] | table:trips, index:
   ↪ duration(duration) | range:(3600,+inf], keep order:false |
|  ─TableRowIDScan_11(Probe)  | 6372547.67 | cop[tikv] | table:trips
   ↪                    | keep order:false                |
+--
   ↪ ----------------------------+-----------+----------+----------------------
   ↪
3 rows in set (0.00 sec)

+--
   ↪ ----------------------------+----------+----------+----------------------
   ↪
| id                           | estRows  | task     | access object
   ↪                     | operator info                   |
+--
   ↪ ----------------------------+----------+----------+----------------------
   ↪
| IndexLookUp_10               | 833219.37 | root    |
   ↪                                    |                                  |
|  ─IndexRangeScan_8(Build)  | 833219.37 | cop[tikv] | table:trips, index:
   ↪ duration(duration) | range:(3600,+inf], keep order:false |
|  ─TableRowIDScan_9(Probe)  | 833219.37 | cop[tikv] | table:trips
   ↪                    | keep order:false                |
+--
   ↪ ----------------------------+----------+----------+----------------------
   ↪
3 rows in set (0.00 sec)
```

Similarly, predicates from the view are pushed down to the base table:

```
EXPLAIN SELECT * FROM long_trips WHERE bike_number = 'W00950';
EXPLAIN SELECT * FROM trips WHERE bike_number = 'W00950';
```

```
+--
   ↪ ----------------------------+---------+----------+----------------------
   ↪
| id                           | estRows | task    | access object
   ↪                     | operator info                          |
+--
   ↪ ----------------------------+---------+----------+----------------------
   ↪
| IndexLookUp_14               | 3.33    | root     |
   ↪                                   |
   ↪                                          |
|  ─IndexRangeScan_11(Build)  | 3333.33 | cop[tikv] | table:trips, index:
```

```
    ↪ duration(duration) | range:(3600,+inf], keep order:false, stats:
    ↪ pseudo |
|  ─Selection_13(Probe)       | 3.33    | cop[tikv] |
    ↪                                   | eq(bikeshare.trips.bike_number, "
    ↪ W00950")   |
|    ─TableRowIDScan_12       | 3333.33 | cop[tikv] | table:trips
    ↪                     | keep order:false, stats:pseudo              |
+--
    ↪ ------------------------------+---------+-----------+-------------------------
    ↪
4 rows in set (0.00 sec)

+--
    ↪ ----------------------+------------+----------+--------------+-------------
    ↪
| id                    | estRows    | task     | access object | operator
    ↪ info                          |
+--
    ↪ ----------------------+------------+----------+--------------+-------------
    ↪
| TableReader_7         | 43.00      | root     |               | data:
    ↪ Selection_6                     |
|  ─Selection_6         | 43.00      | cop[tikv] |              | eq(bikeshare.
    ↪ trips.bike_number, "W00950") |
|    ─TableFullScan_5   | 19117643.00 | cop[tikv] | table:trips | keep order
    ↪ :false                  |
+--
    ↪ ----------------------+------------+----------+--------------+-------------
    ↪
3 rows in set (0.00 sec)
```

In the first statement above, you can see that the index is used to satisfy the view definition, and then the `bike_number = 'W00950'` is applied when TiDB reads the table row. In the second statement, there are no indexes to satisfy the statement, and a `TableFullScan` is used.

TiDB makes use of indexes that satisfy both the view definition and the statement itself. Consider the following composite index:

```
ALTER TABLE trips ADD INDEX (bike_number, duration);
EXPLAIN SELECT * FROM long_trips WHERE bike_number = 'W00950';
EXPLAIN SELECT * FROM trips WHERE bike_number = 'W00950';
```

```
Query OK, 0 rows affected (2 min 31.20 sec)
```

```
+--
   ↪ ----------------------------+---------+----------+-----------------------
   ↪
| id                          | estRows | task     | access object
   ↪                                     | operator info
   ↪                                     |
+--
   ↪ ----------------------------+---------+----------+-----------------------
   ↪
| IndexLookUp_13              | 63725.48 | root    |
   ↪                                                  |
   ↪                                                  |
|  ─IndexRangeScan_11(Build) | 63725.48 | cop[tikv] | table:trips, index:
   ↪ bike_number(bike_number, duration) | range:("W00950" 3600,"W00950" +
   ↪ inf], keep order:false |
|  ─TableRowIDScan_12(Probe) | 63725.48 | cop[tikv] | table:trips
   ↪                                     | keep order:false
   ↪                                  |
+--
   ↪ ----------------------------+---------+----------+-----------------------
   ↪
3 rows in set (0.00 sec)

+--
   ↪ ----------------------------+---------+----------+-----------------------
   ↪
| id                          | estRows | task     | access object
   ↪                                     | operator info
   ↪                                       |
+--
   ↪ ----------------------------+---------+----------+-----------------------
   ↪
| IndexLookUp_10              | 19117.64 | root    |
   ↪                                                  |
   ↪                                                |
|  ─IndexRangeScan_8(Build)  | 19117.64 | cop[tikv] | table:trips, index:
   ↪ bike_number(bike_number, duration) | range:["W00950","W00950"], keep
   ↪ order:false |
|  ─TableRowIDScan_9(Probe)  | 19117.64 | cop[tikv] | table:trips
   ↪                                     | keep order:false
   ↪                                  |
+--
   ↪ ----------------------------+---------+----------+-----------------------
   ↪
3 rows in set (0.00 sec)
```

In the first statement, TiDB is able to use both parts of the composite index (`bike_number, duration`). In the second statement, only the first part which is `bike_number` of the index `(bike_number, duration)` is used.

11.3.2.9 Explain Statements Using Partitions

The `EXPLAIN` statement displays the partitions that TiDB needs to access in order to execute a query. Because of partition pruning, the displayed partitions are often only a subset of the overall partitions. This document describes some of the optimizations for common partitioned tables, and how to interpret the output of `EXPLAIN`.

The sample data used in this document:

```
CREATE TABLE t1 (
 id BIGINT NOT NULL auto_increment,
 d date NOT NULL,
 pad1 BLOB,
 pad2 BLOB,
 pad3 BLOB,
 PRIMARY KEY (id,d)
) PARTITION BY RANGE (YEAR(d)) (
 PARTITION p2016 VALUES LESS THAN (2017),
 PARTITION p2017 VALUES LESS THAN (2018),
 PARTITION p2018 VALUES LESS THAN (2019),
 PARTITION p2019 VALUES LESS THAN (2020),
 PARTITION pmax VALUES LESS THAN MAXVALUE
);

INSERT INTO t1 (d, pad1, pad2, pad3) VALUES
 ('2016-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2016-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2016-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2017-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2017-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2017-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2018-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2018-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2018-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2019-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2019-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2019-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2020-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2020-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
 ('2020-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024));
```

```
INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;

SELECT SLEEP(1);
ANALYZE TABLE t1;
```

The following example shows a statement against the newly created partitioned table:

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE d = '2017-06-01';
```

```
+--
    ↪ ---------------------------+---------+-----------+-------------------------+-
    ↪
| id                          | estRows | task    | access object           |
    ↪ operator info                          |
+--
    ↪ ---------------------------+---------+-----------+-------------------------+-
    ↪
| StreamAgg_21                | 1.00    | root      |                         |
    ↪ funcs:count(Column#8)->Column#6      |
|  ─TableReader_22            | 1.00    | root      |                          |
    ↪ data:StreamAgg_10                      |
|    ─StreamAgg_10            | 1.00    | cop[tikv] |                          |
    ↪ funcs:count(1)->Column#8               |
|      ─Selection_20          | 8.87    | cop[tikv] |                          | eq
    ↪ (test.t1.d, 2017-06-01 00:00:00.000000) |
|        ─TableFullScan_19    | 8870.00 | cop[tikv] | table:t1, partition:
    ↪ p2017 | keep order:false               |
+--
    ↪ ---------------------------+---------+-----------+-------------------------+-
    ↪
5 rows in set (0.01 sec)
```

Starting from the inner-most (`─TableFullScan_19`) operator and working back towards the root operator (`StreamAgg_21`):

- TiDB successfully identified that only one partition (`p2017`) needed to be accessed. This is noted under `access object`.

- The partition itself was scanned in the operator `-TableFullScan_19` and then `-Selection_20` was applied to filter for rows that have a start date of `2017-06-01 00:00:00.000000`.
- The rows that match `-Selection_20` are then stream aggregated in the coprocessor, which natively understands the `count` function.
- Each coprocessor request then sends back one row to `-TableReader_22` inside TiDB, which is then stream aggregated under `StreamAgg_21` and one row is returned to the client.

In the following example, partition pruning does not eliminate any partitions:

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE YEAR(d) = 2017;
```

```
+--
   ↪ ---------------------------------+----------+-----------+--------------------
   ↪
| id                                | estRows | task     | access object
   ↪          | operator info                 |
+--
   ↪ ---------------------------------+----------+-----------+--------------------
   ↪
| HashAgg_20                        | 1.00    | root     |
   ↪                           | funcs:count(Column#7)->Column#6 |
|  -PartitionUnion_21               | 5.00    | root     |
   ↪                           |                                 |
|    -StreamAgg_36                  | 1.00    | root     |
   ↪                           | funcs:count(Column#9)->Column#7 |
|      -TableReader_37              | 1.00    | root     |
   ↪                           | data:StreamAgg_25               |
|        -StreamAgg_25              | 1.00    | cop[tikv] |
   ↪                          | funcs:count(1)->Column#9        |
|          -Selection_35            | 6000.00 | cop[tikv] |
   ↪                         | eq(year(test.t1.d), 2017)       |
|            -TableFullScan_34      | 7500.00 | cop[tikv] | table:t1,
   ↪ partition:p2016 | keep order:false          |
|    -StreamAgg_55                  | 1.00    | root     |
   ↪                           | funcs:count(Column#11)->Column#7 |
|      -TableReader_56              | 1.00    | root     |
   ↪                           | data:StreamAgg_44               |
|        -StreamAgg_44              | 1.00    | cop[tikv] |
   ↪                          | funcs:count(1)->Column#11       |
|          -Selection_54            | 14192.00 | cop[tikv] |
   ↪                         | eq(year(test.t1.d), 2017)       |
|            -TableFullScan_53      | 17740.00 | cop[tikv] | table:t1,
   ↪ partition:p2017 | keep order:false          |
```

```
|    ─StreamAgg_74                 | 1.00    | root     |
   ↪                          | funcs:count(Column#13)->Column#7 |
|      ─TableReader_75             | 1.00    | root     |
   ↪                          | data:StreamAgg_63            |
|        ─StreamAgg_63             | 1.00    | cop[tikv] |
   ↪                         | funcs:count(1)->Column#13    |
|          ─Selection_73           | 3977.60 | cop[tikv] |
   ↪                        | eq(year(test.t1.d), 2017)    |
|            ─TableFullScan_72     | 4972.00 | cop[tikv] | table:t1,
   ↪ partition:p2018 | keep order:false       |
|    ─StreamAgg_93                 | 1.00    | root     |
   ↪                          | funcs:count(Column#15)->Column#7 |
|      ─TableReader_94             | 1.00    | root     |
   ↪                          | data:StreamAgg_82            |
|        ─StreamAgg_82             | 1.00    | cop[tikv] |
   ↪                         | funcs:count(1)->Column#15    |
|          ─Selection_92           | 20361.60 | cop[tikv] |
   ↪                       | eq(year(test.t1.d), 2017)    |
|            ─TableFullScan_91     | 25452.00 | cop[tikv] | table:t1,
   ↪ partition:p2019 | keep order:false       |
|    ─StreamAgg_112                | 1.00    | root     |
   ↪                          | funcs:count(Column#17)->Column#7 |
|      ─TableReader_113            | 1.00    | root     |
   ↪                          | data:StreamAgg_101           |
|        ─StreamAgg_101            | 1.00    | cop[tikv] |
   ↪                         | funcs:count(1)->Column#17    |
|          ─Selection_111          | 8892.80 | cop[tikv] |
   ↪                        | eq(year(test.t1.d), 2017)    |
|            ─TableFullScan_110    | 11116.00 | cop[tikv] | table:t1,
   ↪ partition:pmax | keep order:false       |
+--
   ↪ ---------------------------------+----------+-----------+----------------------
   ↪
27 rows in set (0.00 sec)
```

From the output above:

- TiDB believes that it needs to access all of the partitions `(p2016..pMax)`. This is because the predicate `YEAR(d)= 2017` is considered non-sargable. This issue is not specific to TiDB.
- As each partition is scanned, a `Selection` operator filters out rows that do not match the year of 2017.
- A stream aggregation on each partition is performed to count the number of rows that match.
- The operator `─PartitionUnion_21` unions the results from accessing each partition.

11.3.2.10 Explain Statements Using Index Merge

`IndexMerge` is a method introduced in TiDB v4.0 to access tables. Using this method, the TiDB optimizer can use multiple indexes per table and merge the results returned by each index. In some scenarios, this method makes the query more efficient by avoiding full table scans.

```
mysql> EXPLAIN SELECT * from t where a = 1 or b = 1;
+--
   ↪ ----------------------+---------+----------+--------------+----------------
   ↪
| id                    | estRows | task     | access object | operator info
   ↪                  |
+--
   ↪ ----------------------+---------+----------+--------------+----------------
   ↪
| TableReader_7         | 8000.00 | root     |               | data:Selection_6
   ↪                 |
|  ─Selection_6         | 8000.00 | cop[tikv] |              | or(eq(test.t.a,
   ↪ 1), eq(test.t.b, 1)) |
|    ─TableFullScan_5   | 10000.00 | cop[tikv] | table:t   | keep order:false
   ↪ , stats:pseudo  |
+--
   ↪ ----------------------+---------+----------+--------------+----------------
   ↪
mysql> set @@tidb_enable_index_merge = 1;
mysql> explain select * from t use index(idx_a, idx_b) where a > 1 or b >
   ↪ 1;
+--
   ↪ ----------------------------+--------+----------+-----------------------+-
   ↪
| id                          | estRows | task    | access object         |
   ↪ operator info                                |
+--
   ↪ ----------------------------+--------+----------+-----------------------+-
   ↪
| IndexMerge_16               | 6666.67 | root    |                       |
   ↪                                            |
|  ─IndexRangeScan_13(Build)  | 3333.33 | cop[tikv] | table:t, index:idx_a(a
   ↪ ) | range:(1,+inf], keep order:false, stats:pseudo |
|  ─IndexRangeScan_14(Build)  | 3333.33 | cop[tikv] | table:t, index:idx_b(b
   ↪ ) | range:(1,+inf], keep order:false, stats:pseudo |
|  ─TableRowIDScan_15(Probe)  | 6666.67 | cop[tikv] | table:t           |
   ↪ keep order:false, stats:pseudo           |
+--
   ↪ ----------------------------+--------+----------+-----------------------+-
```

```
↪
```

In the above query, the filter condition is a `WHERE` clause that uses `OR` as the connector. Without `IndexMerge`, you can use only one index per table. `a = 1` cannot be pushed down to the index `a`; neither can `b = 1` be pushed down to the index `b`. The full table scan is inefficient when a huge volume of data exists in `t`. To handle such a scenario, `IndexMerge` is introduced in TiDB to access tables.

`IndexMerge` allows the optimizer to use multiple indexes per table, and merge the results returned by each index to generate the execution plan of the latter `IndexMerge` in the figure above. Here the `IndexMerge_16` operator has three child nodes, among which `IndexRangeScan_13` and `IndexRangeScan_14` get all the `RowID`s that meet the conditions based on the result of range scan, and then the `TableRowIDScan_15` operator accurately reads all the data that meets the conditions according to these `RowID`s.

For the scan operation that is performed on a specific range of data, such as `IndexRangeScan`/`TableRangeScan`, the `operator info` column in the result has additional information about the scan range compared with other scan operations like `IndexFullScan`/`TableFullScan`. In the above example, the `range:(1,+inf]` in the `IndexRangeScan_13` operator indicates that the operator scans the data from 1 to positive infinity.

Note:

- The Index Merge feature is enabled by default from v5.4.0. That is, `tidb_enable_index_merge` is `ON`.
- You can use the SQL hint `USE_INDEX_MERGE` to force the optimizer to apply Index Merge, regardless of the setting of `tidb_enable_index_merge` ↪ . To enable Index Merge when the filtering conditions contain expressions that cannot be pushed down, you must use the SQL hint `USE_INDEX_MERGE`.
- Index Merge supports only disjunctive normal form (expressions connected by `or`) and does not support conjunctive normal form (expressions connected by `and`).
- Index Merge is not supported in tempoaray tables for now.

11.3.3 SQL Optimization Process

11.3.3.1 SQL Optimization Process

In TiDB, the process from inputting a query to getting the execution result according to the final execution plan is illustrated as follows:

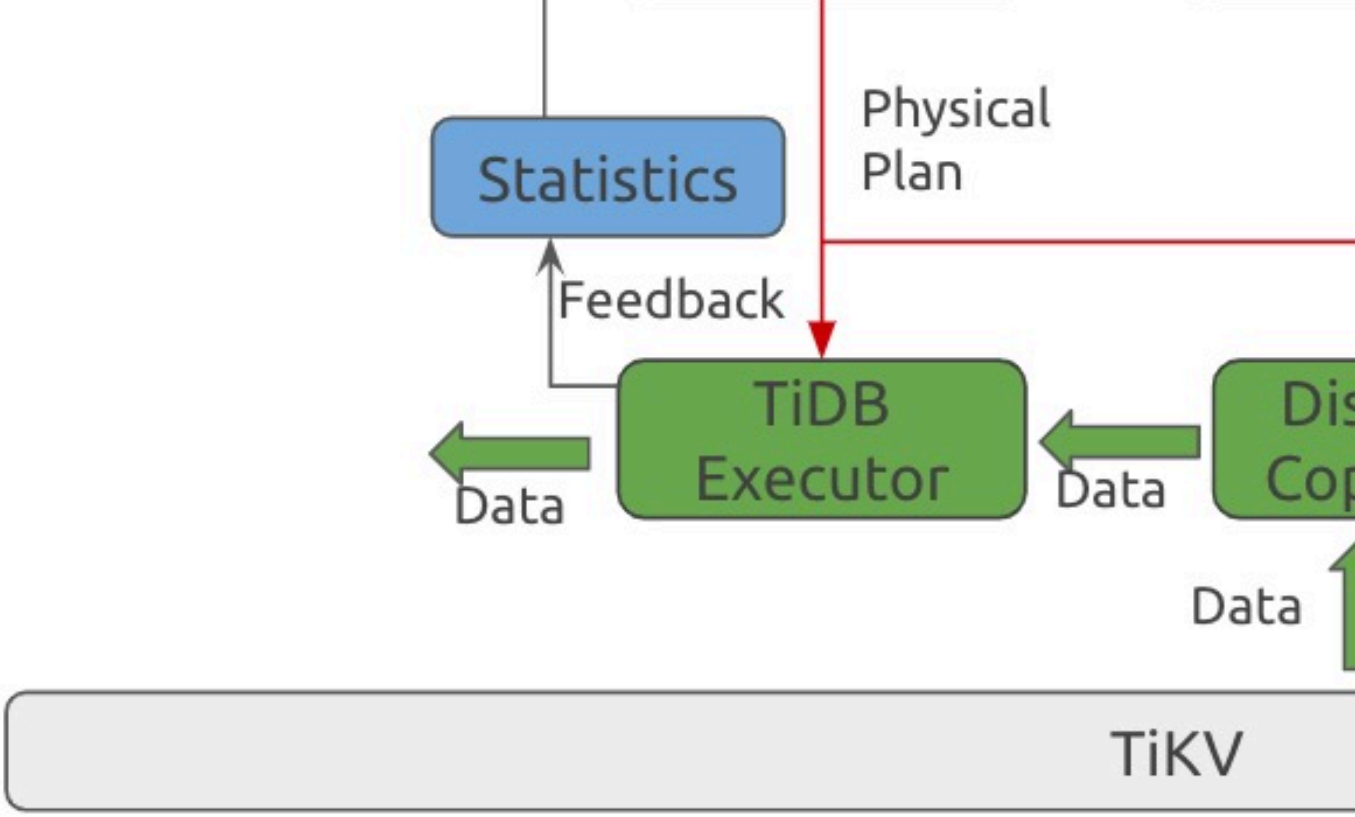


Figure 179: SQL Optimization Process

After parsing the original query text by `parser` and some simple validity checks, TiDB first makes some logically equivalent changes to the query. For detailed changes, see SQL Logical Optimization.

Through these equivalent changes, this query becomes easier to handle in the logical execution plan. After the equivalent change is done, TiDB obtains a query plan structure equivalent to the original query, and then obtains a final execution plan based on the data distribution and the specific execution cost of an operator. For details, see SQL Physical Optimization.

At the same time, when TiDB executes the `PREPARE` statement, you can choose to enable caching to reduce the cost of generating the execution plan in TiDB. For details, see Execution Plan Cache.

11.3.3.2 Logic Optimization

11.3.3.2.1 SQL Logical Optimization

This chapter explains some key logic rewrites to help you understand how TiDB generates the final query plan. For example, when you execute the `select * from t where t.a in (select t1.a from t1 where t1.b=t.b)` query in TiDB, you will find that the `IN` sub-query `t.a in (select t1.a from t1 where t1.b=t.b)` does not exist because TiDB has made some rewrites here.

This chapter introduces the following key rewrites:

- Subquery Related Optimizations
- Column Pruning
- Decorrelation of Correlated Subquery
- Eliminate Max/Min
- Predicates Push Down
- Partition Pruning
- TopN and Limit Operator Push Down
- Join Reorder

11.3.3.2.2 Subquery Related Optimizations

This article mainly introduces subquery related optimizations.

Subqueries usually appear in the following situations:

- `NOT IN (SELECT ... FROM ...)`
- `NOT EXISTS (SELECT ... FROM ...)`
- `IN (SELECT ... FROM ..)`
- `EXISTS (SELECT ... FROM ...)`
- `... >/>=/</<=/=/!= (SELECT ... FROM ...)`

Sometimes a subquery contains non-subquery columns, such as `select * from t where t.a in (select * from t2 where t.b=t2.b)`. The `t.b` column in the subquery does not belong to the subquery, it is introduced from the outside of the subquery. This kind of subquery is usually called a "correlated subquery", and the externally introduced column is called a "correlated column". For optimizations about correlated subquery, see Decorrelation of correlated subquery. This article focuses on subqueries that do not involve correlated columns.

By default, subqueries use `semi join` mentioned in Understanding TiDB Execution Plan as the execution method. For some special subqueries, TiDB do some logical rewrite to get better performance.

`... < ALL (SELECT ... FROM ...)` or `... > ANY (SELECT ... FROM ...)`

In this case, `ALL` and `ANY` can be replaced by `MAX` and `MIN`. When the table is empty, the result of `MAX(EXPR)` and `MIN(EXPR)` is NULL. It works the same when the result of `EXPR` contains `NULL`. Whether the result of `EXPR` contains `NULL` may affect the final result of the expression, so the complete rewrite is given in the following form:

- `t.id < all (select s.id from s)` is rewritten as `t.id < min(s.id)and if(sum(s.id is null)!= 0, null, true)`
- `t.id < any (select s.id from s)` is rewritten as `t.id < max(s.id)or if(sum(s.id is null)!= 0, null, false)`

`... != ANY (SELECT ... FROM ...)`

In this case, if all the values from the subquery are distinct, it is enough to compare the query with them. If the number of different values in the subquery is more than one, then there must be inequality. Therefore, such subqueries can be rewritten as follows:

- `select * from t where t.id != any (select s.id from s)` is rewritten as `select t.* from t, (select s.id, count(distinct s.id)as cnt_distinct from s)where (t.id != s.id or cnt_distinct > 1)`

`... = ALL (SELECT ... FROM ...)`

In this case, when the number of different values in the subquery is more than one, then the result of this expression must be false. Therefore, such subquery is rewritten into the following form in TiDB:

- `select * from t where t.id = all (select s.id from s)` is rewritten as `select t.* from t, (select s.id, count(distinct s.id)as cnt_distinct from s )where (t.id = s.id and cnt_distinct <= 1)`

`... IN (SELECT ... FROM ...)`

In this case, the subquery of `IN` is rewritten into `SELECT ... FROM ... GROUP ...`, and then rewritten into the normal form of `JOIN`.

For example, `select * from t1 where t1.a in (select t2.a from t2)` is rewritten as `select t1.* from t1, (select distinct(a)a from t2)t2 where t1.a = t2.a`. The form of `DISTINCT` attribute here can be eliminated automatically if `t2.a` has the `UNIQUE` attribute.

```
explain select * from t1 where t1.a in (select t2.a from t2);
```

```
+----------------------------+---------+----------+-----------------------+----------+
| id                         | estRows | task     | access object         | operator info                                                
```

```
|  ─TableReader_11(Probe)   | 1.00   | root     |                       | data
   ↪ :TableRangeScan_10                                                  |
|    ─TableRangeScan_10     | 1.00   | cop[tikv] | table:t1              |
   ↪ range: decided by [test.t2.a], keep order:false, stats:pseudo     |
+--
   ↪ ---------------------------+---------+----------+-----------------------+----
   ↪
```

This rewrite gets better performance when the `IN` subquery is relatively small and the external query is relatively large, because without rewriting, using `index join` with t2 as the driving table is impossible. However, the disadvantage is that when the aggregation cannot be automatically eliminated during the rewrite and the `t2` table is relatively large, this rewrite affects the performance of the query. Currently, the variable tidb_opt_insubq_to_join_and_agg is used to control this optimization. When this optimization is not suitable, you can manually disable it.

`EXISTS` subquery and `... >/>=/</<=/=/!= (SELECT ... FROM ...)`

At present, for a subquery in such scenarios, if the subquery is not a correlated subquery, TiDB evaluates it in advance in the optimization stage, and directly replaces it with a result set. As shown in the figure below, the `EXISTS` subquery is evaluated to `TRUE` in the optimization stage in advance, so it does not show in the final execution result.

```
create table t1(a int);
create table t2(a int);
insert into t2 values(1);
explain select * from t1 where exists (select * from t2);
```

```
+--
   ↪ ---------------------+----------+----------+--------------+-----------------
   ↪
| id                    | estRows | task     | access object | operator info
   ↪            |
+--
   ↪ ---------------------+----------+----------+--------------+-----------------
   ↪
| TableReader_12        | 10000.00 | root     |               | data:
   ↪ TableFullScan_11       |
|  ─TableFullScan_11    | 10000.00 | cop[tikv] | table:t1 | keep order:false,
   ↪  stats:pseudo |
+--
   ↪ ---------------------+----------+----------+--------------+-----------------
   ↪
```

11.3.3.2.3 Column Pruning

The basic idea of column pruning is that for columns not used in the operator, the optimizer does not need to retain them during optimization. Removing these columns reduces the use of I/O resources and facilitates the subsequent optimization. The following is an example of column repetition:

Suppose there are four columns (a, b, c, and d) in table t. You can execute the following statement:

```
select a from t where b> 5
```

In this query, only column a and column b are used, and column c and column d are redundant. Regarding the query plan of this statement, the `Selection` operator uses column b. Then the `DataSource` operator uses columns a and column b. Columns c and column d can be pruned because the `DataSource` operator does not read them.

Therefore, when TiDB performs a top-down scanning during the logic optimization phase, redundant columns are pruned to reduce waste of resources. This scanning process is called "Column Pruning", corresponding to the `columnPruner` rule. If you want to disable this rule, refer to The Blocklist of Optimization Rules and Expression Pushdown.

11.3.3.2.4 Decorrelation of Correlated Subquery

Subquery related optimizations describes how TiDB handles subqueries when there are no correlated columns. Because decorrelation of correlated subquery is complex, this article introduces some simple scenarios and the scope to which the optimization rule applies.

Introduction

Take `select * from t1 where t1.a < (select sum(t2.a)from t2 where t2.b = t1.b)` as an example. The subquery `t1.a < (select sum(t2.a)from t2 where t2.b = t1.b)` here refers to the correlated column in the query condition `t2.b=t1.b`, this condition happens to be an equivalent condition, so the query can be rewritten as `select t1.* from t1, (select b, sum(a)sum_a from t2 group by b)t2 where t1.b = t2.b and t1.a < t2.sum_a;`. In this way, a correlated subquery is rewritten into `JOIN`.

The reason why TiDB needs to do this rewriting is that the correlated subquery is bound to its external query result every time the subquery is executed. In the above example, if `t1.a` has 10 million values, this subquery would repeat 10 million times, because the condition `t2.b=t1.b` varies with the value of `t1.a`. When the correlation is lifted somehow, this subquery would execute only once.

Restrictions

The disadvantage of this rewriting is that when the correlation is not lifted, the optimizer can use the index on the correlated column. That is, although this subquery may repeat many times, the index can be used to filter data each time. After using the rewriting rule, the position of the correlated column usually changes. Although the subquery is only executed once, the single execution time would be longer than that without decorrelation.

Therefore, when there are few external values, do not perform decorrelation, because it may bring better execution performance. At present, this optimization can be disabled by setting `subquery decorrelation` optimization rules in blocklist of optimization rules and expression pushdown.

Example

```
create table t1(a int, b int);
create table t2(a int, b int, index idx(b));
explain select * from t1 where t1.a < (select sum(t2.a) from t2 where t2.b
    ↪ = t1.b);
```

```
+--
    ↪ ---------------------------------+---------+----------+--------------+--------
    ↪
| id                              | estRows | task     | access object |
    ↪ operator info
    ↪                                                                    |
+--
    ↪ ---------------------------------+---------+----------+--------------+--------
    ↪
| HashJoin_11                     | 9990.00 | root     |              | inner
    ↪ join, equal:[eq(test.t1.b, test.t2.b)], other cond:lt(cast(test.t1.a)
    ↪ , Column#7) |
|  ─HashAgg_23(Build)             | 7992.00 | root     |              | group by
    ↪ :test.t2.b, funcs:sum(Column#8)->Column#7, funcs:firstrow(test.t2.b)
    ↪ ->test.t2.b |
|    ─TableReader_24              | 7992.00 | root     |              | data:
    ↪ HashAgg_16
    ↪                                                                     |
|      ─HashAgg_16                | 7992.00 | cop[tikv] |             | group by
    ↪ :test.t2.b, funcs:sum(test.t2.a)->Column#8                          |
|        ─Selection_22            | 9990.00 | cop[tikv] |             | not(
    ↪ isnull(test.t2.b))
    ↪                                                       |
|          ─TableFullScan_21      | 10000.00 | cop[tikv] | table:t2 | keep
    ↪ order:false, stats:pseudo
    ↪                                                     |
|  ─TableReader_15(Probe)         | 9990.00 | root     |              | data:
    ↪ Selection_14
    ↪                                                                  |
|    ─Selection_14                | 9990.00 | cop[tikv] |             | not(
    ↪ isnull(test.t1.b))
    ↪                                                       |
|      ─TableFullScan_13          | 10000.00 | cop[tikv] | table:t1 | keep
    ↪ order:false, stats:pseudo
```

```
    ↪                                                  |
+--
    ↪ --------------------------------+---------+----------+--------------+-------
    ↪
```

The above is an example where the optimization takes effect. `HashJoin_11` is a normal `inner join`.

Then, turn off the subquery decorrelation rules:

```
insert into mysql.opt_rule_blacklist values("decorrelate");
admin reload opt_rule_blacklist;
explain select * from t1 where t1.a < (select sum(t2.a) from t2 where t2.b
    ↪ = t1.b);
```

```
+--
    ↪ ------------------------------------+---------+----------+-----------------
    ↪
| id                                  | estRows | task     | access object
    ↪       | operator info
    ↪                                                        |
+--
    ↪ ------------------------------------+---------+----------+-----------------
    ↪
| Projection_10                       | 10000.00 | root    |
    ↪                     | test.t1.a, test.t1.b
    ↪                                                |
|  ─Apply_12                           | 10000.00 | root    |
    ↪                     | CARTESIAN inner join, other cond:lt(cast(test.
    ↪ t1.a), Column#7)     |
|    ─TableReader_14(Build)            | 10000.00 | root    |
    ↪                     | data:TableFullScan_13
    ↪                                               |
|      ─TableFullScan_13               | 10000.00 | cop[tikv] | table:t1
    ↪         | keep order:false, stats:pseudo
    ↪                                        |
|    ─MaxOneRow_15(Probe)              | 1.00    | root     |
    ↪                     |
    ↪                                                                  |
|      ─HashAgg_27                     | 1.00    | root     |
    ↪                     | funcs:sum(Column#10)->Column#7
    ↪                                        |
|        ─IndexLookUp_28               | 1.00    | root     |
    ↪                     |
    ↪                                                                  |
```

```
|         └─IndexRangeScan_25(Build) | 10.00   | cop[tikv] | table:t2, index
    ↪ :idx(b) | range: decided by [eq(test.t2.b, test.t1.b)], keep order:
    ↪ false, stats:pseudo |
|         └─HashAgg_17(Probe)         | 1.00    | cop[tikv] |
    ↪                     | funcs:sum(test.t2.a)->Column#10
    ↪                                   |
|           └─TableRowIDScan_26       | 10.00   | cop[tikv] | table:t2
    ↪           | keep order:false, stats:pseudo
    ↪                                    |
+--
    ↪ ------------------------------------+---------+----------+-----------------
    ↪
```

After disabling the subquery decorrelation rule, you can see `range: decided by [eq(` ↪ `test.t2.b, test.t1.b)]` in `operator info` of `IndexRangeScan_25(Build)`. It means that the decorrelation of correlated subquery is not performed and TiDB uses the index range query.

11.3.3.2.5 Eliminate Max/Min

When a SQL statement contains `max/min` functions, the query optimizer tries to convert the `max/min` aggregate functions to the TopN operator by applying the `max/min` optimization rule. In this way, TiDB can perform the query more efficiently through indexes.

This optimization rule is divided into the following two types according to the number of `max/min` functions in the `select` statement:

- The statement with only one `max/min` function
- The statement with multiple `max/min` functions

One `max/min` function

When a SQL statement meets the following conditions, this rule is applied:

- The statement contains only one aggregate function, which is `max` or `min`.
- The aggregate function has no related `group by` clause.

For example:

```
select max(a) from t
```

The optimization rule rewrites the statement as follows:

```
select max(a) from (select a from t where a is not null order by a desc
    ↪ limit 1) t
```

When column `a` has an index, or when column `a` is the prefix of some composite index, with the help of index, the new SQL statement can find the maximum or minimum value by scanning only one row of data. This optimization avoids full table scan.

The example statement has the following execution plan:

```
mysql> explain select max(a) from t;
+--
   ↪ ---------------------------+---------+----------+-----------------------+---
   ↪
| id                          | estRows | task     | access object         |
   ↪ operator info                    |
+--
   ↪ ---------------------------+---------+----------+-----------------------+---
   ↪
| StreamAgg_13                | 1.00    | root     |                       |
   ↪ funcs:max(test.t.a)->Column#4   |
| └─Limit_17                  | 1.00    | root     |                       |
   ↪ offset:0, count:1                |
|   └─IndexReader_27          | 1.00    | root     |                       |
   ↪ index:Limit_26                   |
|     └─Limit_26              | 1.00    | cop[tikv] |                       |
   ↪ offset:0, count:1                |
|       └─IndexFullScan_25    | 1.00    | cop[tikv] | table:t, index:idx_a(a) |
   ↪  keep order:true, desc, stats:pseudo |
+--
   ↪ ---------------------------+---------+----------+-----------------------+---
   ↪
5 rows in set (0.00 sec)
```

Multiple `max`/`min` functions

When a SQL statement meets the following conditions, this rule is applied:

- The statement contains multiple aggregate functions, which are all `max` or `min` functions.
- None of the aggregate functions has a related `group by` clause.
- The columns in each `max`/`min` function has indexes to preserve the order.

For example:

```
select max(a) - min(a) from t
```

The optimization rule first checks whether column `a` has an index to preserve its order. If yes, the SQL statement is rewritten as the Cartesian product of two subqueries:

```
select max_a - min_a
from
    (select max(a) as max_a from t) t1,
    (select min(a) as min_a from t) t2
```

Through the rewrite, the optimizer can apply the rule for statements with only one `max` ↪ `/min` function to the two subqueries respectively. The statement is then rewritten as follows:

```
select max_a - min_a
from
    (select max(a) as max_a from (select a from t where a is not null order
       ↪  by a desc limit 1) t) t1,
    (select min(a) as min_a from (select a from t where a is not null order
       ↪  by a asc limit 1) t) t2
```

Similarly, if column `a` has an index to preserve its order, the optimized execution only scans two rows of data instead of the whole table. However, if column `a` does not have an index to preserve its order, this rule results in two full table scans, but the execution only needs one full table scan if it is not rewritten. Therefore, in such cases, this rule is not applied.

The final execution plan is as follows:

```
mysql> explain select max(a)-min(a) from t;
+--
   ↪ ----------------------------------+---------+-----------+-------------------
   ↪
| id                                  | estRows | task      | access object       |
   ↪  operator info                        |
+--
   ↪ ----------------------------------+---------+-----------+-------------------
   ↪
| Projection_17                       | 1.00    | root      |                     |
   ↪  minus(Column#4, Column#5)->Column#6 |
|  ─HashJoin_18                       | 1.00    | root      |
   ↪ | CARTESIAN inner join                |
|    ─StreamAgg_45(Build)             | 1.00    | root      |
   ↪ | funcs:min(test.t.a)->Column#5  |
|      ─Limit_49                      | 1.00    | root      |
   ↪ | offset:0, count:1                   |
|        ─IndexReader_59              | 1.00    | root      |
   ↪ | index:Limit_58                      |
|          ─Limit_58                  | 1.00    | cop[tikv] |
   ↪ | offset:0, count:1                   |
```

```
|          └─IndexFullScan_57    | 1.00   | cop[tikv] | table:t, index:
    ↪ idx_a(a) | keep order:true, stats:pseudo |
|   └─StreamAgg_24(Probe)        | 1.00   | root      |
    ↪ | funcs:max(test.t.a)->Column#4  |
|     └─Limit_28                 | 1.00   | root      |
    ↪ | offset:0, count:1              |
|       └─IndexReader_38         | 1.00   | root      |
    ↪ | index:Limit_37                 |
|         └─Limit_37             | 1.00   | cop[tikv] |
    ↪ | offset:0, count:1              |
|           └─IndexFullScan_36   | 1.00   | cop[tikv] | table:t, index:
    ↪ idx_a(a) | keep order:true, desc, stats:pseudo |
+--
    ↪ ---------------------------------+---------+----------+---------------------
    ↪
12 rows in set (0.01 sec)
```

11.3.3.2.6 Predicates Push Down (PPD)

This document introduces one of the TiDB's logic optimization rules—Predicate Push Down (PPD). It aims to help you understand the predicate push down and know its applicable and inapplicable scenarios.

PPD pushes down selection operators to data source as close as possible to complete data filtering as early as possible, which significantly reduces the cost of data transmission or computation.

Examples

The following cases describe the optimization of PPD. Case 1, 2, and 3 are scenarios where PPD is applicable, and Case 4, 5, and 6 are scenarios where PPD is not applicable.

Case 1: push predicates to storage layer

```
create table t(id int primary key, a int);
explain select * from t where a < 1;
+--
    ↪ ----------------------+----------+-----------+---------------+----------------
    ↪
| id                    | estRows | task    | access object | operator info
    ↪            |
+--
    ↪ ----------------------+----------+-----------+---------------+----------------
    ↪
| TableReader_7         | 3323.33 | root    |               | data:Selection_6
    ↪           |
|  └─Selection_6         | 3323.33 | cop[tikv] |             | lt(test.t.a, 1)
    ↪            |
```

```
|   └─TableFullScan_5   | 10000.00 | cop[tikv] | table:t  | keep order:false
    ↪ , stats:pseudo |
+--
    ↪ ----------------------+----------+-----------+--------------+----------------
    ↪
3 rows in set (0.00 sec)
```

In this query, pushing down the predicate `a < 1` to the TiKV layer to filter the data can reduce the overhead of network transmission.

Case 2: push predicates to storage layer

```
create table t(id int primary key, a int not null);
explain select * from t where a < substring('123', 1, 1);
+--
    ↪ ----------------------+----------+-----------+--------------+----------------
    ↪
| id                    | estRows | task    | access object | operator info
    ↪            |
+--
    ↪ ----------------------+----------+-----------+--------------+----------------
    ↪
| TableReader_7         | 3323.33 | root    |               | data:Selection_6
    ↪           |
|  └─Selection_6        | 3323.33 | cop[tikv] |             | lt(test.t.a, 1)
    ↪             |
|    └─TableFullScan_5  | 10000.00 | cop[tikv] | table:t  | keep order:false
    ↪ , stats:pseudo |
+--
    ↪ ----------------------+----------+-----------+--------------+----------------
    ↪
```

This query has the same execution plan as the query in case 1, because the input parameters of the `substring` of the predicate `a < substring('123', 1, 1)` are constants, so they can be calculated in advance. Then the predicate is simplified to the equivalent predicate `a < 1`. After that, TiDB can push `a < 1` down to TiKV.

Case 3: push predicates below join operator

```
create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t join s on t.a = s.a where t.a < 1;
+--
    ↪ --------------------------+----------+-----------+--------------+-----------
    ↪
| id                        | estRows | task    | access object | operator
    ↪ info                       |
```

```
+--
   ↪ ---------------------------+---------+----------+--------------+-----------
   ↪
| HashJoin_8                | 4154.17 | root     |              | inner join,
   ↪ equal:[eq(test.t.a, test.s.a)] |
| ─TableReader_15(Build)    | 3323.33 | root     |              | data:
   ↪ Selection_14                     |
|   ─Selection_14           | 3323.33 | cop[tikv] |             | lt(test.s.a
   ↪ , 1)                             |
|     ─TableFullScan_13     | 10000.00 | cop[tikv] | table:s    | keep order:
   ↪ false, stats:pseudo              |
| ─TableReader_12(Probe)    | 3323.33 | root     |              | data:
   ↪ Selection_11                     |
|   ─Selection_11           | 3323.33 | cop[tikv] |             | lt(test.t.a
   ↪ , 1)                             |
|     ─TableFullScan_10     | 10000.00 | cop[tikv] | table:t    | keep order:
   ↪ false, stats:pseudo              |
+--
   ↪ ---------------------------+---------+----------+--------------+-----------
   ↪
7 rows in set (0.00 sec)
```

In this query, the predicate `t.a < 1` is pushed below join to filter in advance, which can reduce the calculation overhead of join.

In addition, This SQL statement has an inner join executed, and the `ON` condition is `t.a = s.a`. The predicate `s.a <1` can be derived from `t.a < 1` and pushed down to `s` table below the join operator. Filtering the `s` table can further reduce the calculation overhead of join.

Case 4: predicates that are not supported by storage layers cannot be pushed down

```
create table t(id int primary key, a int not null);
desc select * from t where substring('123', a, 1) = '1';
+--
   ↪ ----------------------+---------+----------+-------------+-----------------
   ↪
| id                    | estRows | task     | access object | operator info
   ↪                     |
+--
   ↪ ----------------------+---------+----------+-------------+-----------------
   ↪
| Selection_7           | 2.00    | root     |             | eq(substring("123
   ↪ ", test.t.a, 1), "1") |
| ─TableReader_6        | 2.00    | root     |             | data:
   ↪ TableFullScan_5               |
```

```
|   └─TableFullScan_5   | 2.00   | cop[tikv] | table:t    | keep order:false,
    ↪  stats:pseudo    |
+--
    ↪ ----------------------+---------+-----------+--------------+-----------------
    ↪
```

In this query, there is a predicate `substring('123', a, 1)= '1'`.

From the `explain` results, we can see that the predicate is not pushed down to TiKV for calculation. This is because the TiKV coprocessor does not support the built-in function `substring`.

Case 5: predicates of inner tables on the outer join can't be pushed down

```
create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t left join s on t.a = s.a where s.a is null;
+--
    ↪ ----------------------------+----------+-----------+--------------+----------
    ↪
| id                          | estRows | task    | access object | operator
    ↪ info                           |
+--
    ↪ ----------------------------+----------+-----------+--------------+----------
    ↪
| Selection_7                 | 10000.00 | root    |               | isnull(test
    ↪ .s.a)                          |
| └─HashJoin_8                | 12500.00 | root    |               | left outer
    ↪  join, equal:[eq(test.t.a, test.s.a)] |
|   ├─TableReader_13(Build)   | 10000.00 | root    |               | data:
    ↪ TableFullScan_12                 |
|   │ └─TableFullScan_12      | 10000.00 | cop[tikv] | table:s   | keep order:
    ↪ false, stats:pseudo            |
|   └─TableReader_11(Probe)   | 10000.00 | root    |               | data:
    ↪ TableFullScan_10                 |
|     └─TableFullScan_10      | 10000.00 | cop[tikv] | table:t   | keep order:
    ↪ false, stats:pseudo            |
+--
    ↪ ----------------------------+----------+-----------+--------------+----------
    ↪
6 rows in set (0.00 sec)
```

In this query, there is a predicate `s.a is null` on the inner table `s`.

From the `explain` results, we can see that the predicate is not pushed below join operator. This is because the outer join fills the inner table with `NULL` values when the `on` condition isn't satisfied, and the predicate `s.a is null` is used to filter the results after the join. If

it is pushed down to the inner table below join, the execution plan is not equivalent to the original one.

Case 6: the predicates which contain user variables cannot be pushed down

```
create table t(id int primary key, a char);
set @a = 1;
explain select * from t where a < @a;
+--
   ↪ ----------------------+---------+----------+--------------+----------------
   ↪
| id                    | estRows | task     | access object | operator info
   ↪            |
+--
   ↪ ----------------------+---------+----------+--------------+----------------
   ↪
| Selection_5           | 8000.00 | root     |               | lt(test.t.a,
   ↪ getvar("a"))  |
| └─TableReader_7       | 10000.00 | root    |               | data:
   ↪ TableFullScan_6       |
|   └─TableFullScan_6   | 10000.00 | cop[tikv] | table:t   | keep order:false
   ↪ , stats:pseudo |
+--
   ↪ ----------------------+---------+----------+--------------+----------------
   ↪
3 rows in set (0.00 sec)
```

In this query, there is a predicate `a < @a` on table `t`. The `@a` of the predicate is a user variable.

As can be seen from `explain` results, the predicate is not like case 2, which is simplified to `a < 1` and pushed down to TiKV. This is because the value of the user variable `@a` may change during the computation, and TiKV is not aware of the changes. So TiDB does not replace `@a` with `1`, and does not push down it to TiKV.

An example to help you understand is as follows:

```
create table t(id int primary key, a int);
insert into t values(1, 1), (2,2);
set @a = 1;
select id, a, @a:=@a+1 from t where a = @a;
+----+------+----------+
| id | a    | @a:=@a+1 |
+----+------+----------+
|  1 |    1 | 2        |
|  2 |    2 | 3        |
+----+------+----------+
2 rows in set (0.00 sec)
```

As you can see from this query, the value of `@a` will change during the query. So if you replace `a = @a` with `a = 1` and push it down to TiKV, it's not an equivalent execution plan.

11.3.3.2.7 Partition Pruning

Partition pruning is a performance optimization that applies to partitioned tables. It analyzes the filter conditions in query statements, and eliminates (*prunes*) partitions from consideration when they do not contain any data that will be required. By eliminating the non-required partitions, TiDB is able to reduce the amount of data that needs to be accessed and potentially significantly improving query execution times.

The following is an example:

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY,
 pad VARCHAR(100)
)
PARTITION BY RANGE COLUMNS(id) (
 PARTITION p0 VALUES LESS THAN (100),
 PARTITION p1 VALUES LESS THAN (200),
 PARTITION p2 VALUES LESS THAN (MAXVALUE)
);

INSERT INTO t1 VALUES (1, 'test1'),(101, 'test2'), (201, 'test3');
EXPLAIN SELECT * FROM t1 WHERE id BETWEEN 80 AND 120;
```

```
+--
   ↪ -------------------------+---------+-----------+-----------------------+------
   ↪
| id                      | estRows | task    | access object       |
   ↪ operator info                              |
+--
   ↪ -------------------------+---------+-----------+-----------------------+------
   ↪
| PartitionUnion_8        | 80.00  | root     |                     |
   ↪                                            |
|  ─TableReader_10        | 40.00  | root     |                     | data:
   ↪ TableRangeScan_9                     |
|    ─TableRangeScan_9    | 40.00  | cop[tikv] | table:t1, partition:p0 |
   ↪ range:[80,120], keep order:false, stats:pseudo |
|  ─TableReader_12        | 40.00  | root     |                     | data:
   ↪ TableRangeScan_11                    |
|    ─TableRangeScan_11   | 40.00  | cop[tikv] | table:t1, partition:p1 |
   ↪ range:[80,120], keep order:false, stats:pseudo |
+--
   ↪ -------------------------+---------+-----------+-----------------------+------
```

```
    ↪
5 rows in set (0.00 sec)
```

Usage scenarios of partition pruning

The usage scenarios of partition pruning are different for the two types of partitioned tables: Range partitioned tables and Hash partitioned tables.

Use partition pruning in Hash partitioned tables

This section describes the applicable and inapplicable usage scenarios of partition pruning in Hash partitioned tables.

Applicable scenario in Hash partitioned tables

Partition pruning applies only to the query condition of equality comparison in Hash partitioned tables.

```
create table t (x int) partition by hash(x) partitions 4;
explain select * from t where x = 1;
```

```
+--
    ↪ ----------------------+----------+-----------+-----------------------+--------
    ↪
| id                    | estRows | task    | access object       | operator
    ↪ info               |
+--
    ↪ ----------------------+----------+-----------+-----------------------+--------
    ↪
| TableReader_8         | 10.00   | root    |                       | data:
    ↪ Selection_7           |
|  ─Selection_7         | 10.00   | cop[tikv] |                     | eq(test.t
    ↪ .x, 1)             |
|    ─TableFullScan_6   | 10000.00 | cop[tikv] | table:t, partition:p1 |
    ↪ keep order:false, stats:pseudo |
+--
    ↪ ----------------------+----------+-----------+-----------------------+--------
    ↪
```

In the SQL statement above, it can be known from the condition `x = 1` that all results fall in one partition. The value `1` can be confirmed to be in the `p1` partition after passing through the Hash partition. Therefore, only the `p1` partition needs to be scanned, and there is no need to access the `p2`, `p3`, and `p4` partitions that will not have matching results. From the execution plan, only one `TableFullScan` operator appears and the `p1` partition is specified in `access object`, so it can be confirmed that `partition pruning` takes effect.

Inapplicable scenarios in Hash partitioned tables

This section describes two inapplicable usage scenarios of partition pruning in Hash partitioned tables.

Scenario one

If you cannot confirm the condition that the query result falls in only one partition (such as `in`, `between`, `>`, `<`, `>=`, `<=`), you cannot use the partition pruning optimization. For example:

```
create table t (x int) partition by hash(x) partitions 4;
explain select * from t where x > 2;
```

```
+--
   ↪ ---------------------------+---------+----------+---------------------+----
   ↪
| id                          | estRows | task     | access object       |
   ↪ operator info               |
+--
   ↪ ---------------------------+---------+----------+---------------------+----
   ↪
| Union_10                    | 13333.33 | root    |                     |
   ↪                             |
|  ─TableReader_13            | 3333.33 | root     |                     | data
   ↪ :Selection_12             |
|    ─Selection_12            | 3333.33 | cop[tikv] |                    | gt(
   ↪ test.t.x, 2)              |
|      ─TableFullScan_11      | 10000.00 | cop[tikv] | table:t, partition:p0
   ↪ | keep order:false, stats:pseudo |
|  ─TableReader_16            | 3333.33 | root     |                     | data
   ↪ :Selection_15             |
|    ─Selection_15            | 3333.33 | cop[tikv] |                    | gt(
   ↪ test.t.x, 2)              |
|      ─TableFullScan_14      | 10000.00 | cop[tikv] | table:t, partition:p1
   ↪ | keep order:false, stats:pseudo |
|  ─TableReader_19            | 3333.33 | root     |                     | data
   ↪ :Selection_18             |
|    ─Selection_18            | 3333.33 | cop[tikv] |                    | gt(
   ↪ test.t.x, 2)              |
|      ─TableFullScan_17      | 10000.00 | cop[tikv] | table:t, partition:p2
   ↪ | keep order:false, stats:pseudo |
|  ─TableReader_22            | 3333.33 | root     |                     | data
   ↪ :Selection_21             |
|    ─Selection_21            | 3333.33 | cop[tikv] |                    | gt(
   ↪ test.t.x, 2)              |
|      ─TableFullScan_20      | 10000.00 | cop[tikv] | table:t, partition:p3
   ↪ | keep order:false, stats:pseudo |
+--
   ↪ ---------------------------+---------+----------+---------------------+----
   ↪
```

In this case, partition pruning is inapplicable because the corresponding Hash partition cannot be confirmed by the `x > 2` condition.

Scenario two

Because the rule optimization of partition pruning is performed during the generation phase of the query plan, partition pruning is not suitable for scenarios where the filter conditions can be obtained only during the execution phase. For example:

```
create table t (x int) partition by hash(x) partitions 4;
explain select * from t2 where x = (select * from t1 where t2.x = t1.x and
    ↪ t2.x < 2);
```

```
+--
    ↪ ------------------------------------+---------+----------+------------------
    ↪
| id                                | estRows | task     | access object
    ↪       | operator info                                |
+--
    ↪ ------------------------------------+---------+----------+------------------
    ↪
| Projection_13                     | 9990.00 | root     |
    ↪                       | test.t2.x                                    |
|  ─Apply_15                        | 9990.00 | root     |
    ↪                       | inner join, equal:[eq(test.t2.x, test.t1.x)] |
|    ─TableReader_18(Build)         | 9990.00 | root     |
    ↪                       | data:Selection_17                            |
|      ─Selection_17                | 9990.00 | cop[tikv] |
    ↪                     | not(isnull(test.t2.x))                       |
|        ─TableFullScan_16          | 10000.00 | cop[tikv] | table:t2
    ↪          | keep order:false, stats:pseudo               |
|    ─Selection_19(Probe)           | 0.80     | root     |
    ↪                       | not(isnull(test.t1.x))                       |
|      ─MaxOneRow_20                | 1.00     | root     |
    ↪                       |                                              |
|        ─Union_21                  | 2.00     | root     |
    ↪                       |                                              |
|          ─TableReader_24          | 2.00     | root     |
    ↪                       | data:Selection_23                            |
|            ─Selection_23          | 2.00     | cop[tikv] |
    ↪                     | eq(test.t2.x, test.t1.x), lt(test.t2.x, 2)   |
|              ─TableFullScan_22    | 2500.00 | cop[tikv] | table:t1,
    ↪ partition:p0 | keep order:false, stats:pseudo    |
|          ─TableReader_27          | 2.00     | root     |
    ↪                       | data:Selection_26                            |
|            ─Selection_26          | 2.00     | cop[tikv] |
    ↪                     | eq(test.t2.x, test.t1.x), lt(test.t2.x, 2)   |
```

```
|            └─TableFullScan_25    | 2500.00 | cop[tikv] | table:t1,
    ↪ partition:p1 | keep order:false, stats:pseudo    |
+--
    ↪ ---------------------------------+---------+----------+-------------------
    ↪
```

Each time this query reads a row from `t2`, it will query on the `t1` partitioned table. Theoretically, the filter condition of `t1.x = val` is met at this time, but in fact, partition pruning takes effect only in the generation phase of the query plan, not the execution phase.

Use partition pruning in Range partitioned tables

This section describes the applicable and inapplicable usage scenarios of partition pruning in Range partitioned tables.

Applicable scenarios in Range partitioned tables

This section describes three applicable usage scenarios of partition pruning in Range partitioned tables.

Scenario one

Partition pruning applies to the query condition of equality comparison in Range partitioned tables. For example:

```
create table t (x int) partition by range (x) (
    partition p0 values less than (5),
    partition p1 values less than (10),
    partition p2 values less than (15)
    );
explain select * from t where x = 3;
```

```
+--
    ↪ ----------------------+---------+----------+----------------------+--------
    ↪
| id                    | estRows | task     | access object       | operator
    ↪ info               |
+--
    ↪ ----------------------+---------+----------+----------------------+--------
    ↪
| TableReader_8         | 10.00   | root     |                      | data:
    ↪ Selection_7           |
|  └─Selection_7        | 10.00   | cop[tikv] |                      | eq(test.t
    ↪ .x, 3)             |
|    └─TableFullScan_6  | 10000.00 | cop[tikv] | table:t, partition:p0 |
    ↪ keep order:false, stats:pseudo |
+--
    ↪ ----------------------+---------+----------+----------------------+--------
    ↪
```

Partition pruning also applies to the equality comparison that uses the `in` query condition. For example:

```
create table t (x int) partition by range (x) (
    partition p0 values less than (5),
    partition p1 values less than (10),
    partition p2 values less than (15)
    );
explain select * from t where x in(1,13);
```

```
+--
    ↪ --------------------------+---------+----------+---------------------+-----
    ↪
| id                        | estRows | task     | access object       |
    ↪ operator info               |
+--
    ↪ --------------------------+---------+----------+---------------------+-----
    ↪
| Union_8                   | 40.00   | root     |                     |
    ↪                             |
|  ─TableReader_11          | 20.00   | root     |                     | data:
    ↪ Selection_10            |
|    ─Selection_10          | 20.00   | cop[tikv] |                    | in(
    ↪ test.t.x, 1, 13)          |
|      ─TableFullScan_9     | 10000.00 | cop[tikv] | table:t, partition:p0 |
    ↪  keep order:false, stats:pseudo |
|  ─TableReader_14          | 20.00   | root     |                     | data:
    ↪ Selection_13            |
|    ─Selection_13          | 20.00   | cop[tikv] |                    | in(
    ↪ test.t.x, 1, 13)          |
|      ─TableFullScan_12    | 10000.00 | cop[tikv] | table:t, partition:p2 |
    ↪  keep order:false, stats:pseudo |
+--
    ↪ --------------------------+---------+----------+---------------------+-----
    ↪
```

In the SQL statement above, it can be known from the `x in(1,13)` condition that all results fall in a few partitions. After analysis, it is found that all records of `x = 1` are in the `p0` partition, and all records of `x = 13` are in the `p2` partition, so only `p0` and `p2` partitions need to be accessed.

Scenario two

Partition pruning applies to the query condition of interval comparison, such as `between`, `>`, `<`, `=`, `>=`, `<=`. For example:

```
create table t (x int) partition by range (x) (
```

```
    partition p0 values less than (5),
    partition p1 values less than (10),
    partition p2 values less than (15)
    );
explain select * from t where x between 7 and 14;
```

```
+--
    ↪ -------------------------+---------+----------+--------------------+-----
    ↪
| id                        | estRows | task     | access object      |
    ↪ operator info                 |
+--
    ↪ -------------------------+---------+----------+--------------------+-----
    ↪
| Union_8                   | 500.00  | root     |                    |
    ↪                                 |
|  ─TableReader_11          | 250.00  | root     |                    | data:
    ↪ Selection_10              |
|    ─Selection_10          | 250.00  | cop[tikv] |                   | ge(
    ↪ test.t.x, 7), le(test.t.x, 14) |
|      ─TableFullScan_9     | 10000.00 | cop[tikv] | table:t, partition:p1 |
    ↪  keep order:false, stats:pseudo |
|  ─TableReader_14          | 250.00  | root     |                    | data:
    ↪ Selection_13              |
|    ─Selection_13          | 250.00  | cop[tikv] |                   | ge(
    ↪ test.t.x, 7), le(test.t.x, 14) |
|      ─TableFullScan_12    | 10000.00 | cop[tikv] | table:t, partition:p2 |
    ↪  keep order:false, stats:pseudo |
+--
    ↪ -------------------------+---------+----------+--------------------+-----
    ↪
```

Scenario three

Partition pruning applies to the scenario where the partition expression is in the simple form of `fn(col)`, the query condition is one of `>`, `<`, `=`, `>=`, and `<=`, and the `fn` function is monotonous.

If the `fn` function is monotonous, for any `x` and `y`, if `x > y`, then `fn(x) > fn(y)`. Then this `fn` function can be called strictly monotonous. For any `x` and `y`, if `x > y`, then `fn` ↪ `(x) >= fn(y)`. In this case, `fn` could also be called "monotonous". Theoretically, all monotonous functions, strictly or not, are supported by partition pruning. Currently, TiDB only supports the following monotonous functions:

- `UNIX_TIMESTAMP()`
- `TO_DAYS()`

For example, partition pruning takes effect when the partition expression is in the form of `fn(col)`, where the `fn` is monotonous function `to_days`:

```
create table t (id datetime) partition by range (to_days(id)) (
   partition p0 values less than (to_days('2020-04-01')),
   partition p1 values less than (to_days('2020-05-01')));
explain select * from t where id > '2020-04-18';
```

```
+--
   ↪ ----------------------+----------+-----------+----------------------+---------
   ↪
| id                    | estRows | task     | access object       | operator
   ↪ info                          |
+--
   ↪ ----------------------+----------+-----------+----------------------+---------
   ↪
| TableReader_8         | 3333.33 | root     |                      | data:
   ↪ Selection_7                       |
|  ─Selection_7         | 3333.33 | cop[tikv] |                      | gt(test.t
   ↪ .id, 2020-04-18 00:00:00.000000) |
|    ─TableFullScan_6   | 10000.00 | cop[tikv] | table:t, partition:p1 |
   ↪ keep order:false, stats:pseudo |
+--
   ↪ ----------------------+----------+-----------+----------------------+---------
   ↪
```

Inapplicable scenario in Range partitioned tables

Because the rule optimization of partition pruning is performed during the generation phase of the query plan, partition pruning is not suitable for scenarios where the filter conditions can be obtained only during the execution phase. For example:

```
create table t1 (x int) partition by range (x) (
   partition p0 values less than (5),
   partition p1 values less than (10));
create table t2 (x int);
explain select * from t2 where x < (select * from t1 where t2.x < t1.x and
   ↪ t2.x < 2);
```

```
+--
   ↪ ----------------------------------+----------+-----------+-------------------
   ↪
| id                                | estRows | task     | access object
   ↪       | operator info                                    |
+--
   ↪ ----------------------------------+----------+-----------+-------------------
   ↪
```

```
| Projection_13                    | 9990.00 | root    |
   ↪                     | test.t2.x
   ↪                                          |
|  ─Apply_15                       | 9990.00 | root    |
   ↪                     | CARTESIAN inner join, other cond:lt(test.t2.x,
   ↪ test.t1.x) |
|    ─TableReader_18(Build)        | 9990.00 | root    |
   ↪                     | data:Selection_17
   ↪                                       |
|      ─Selection_17               | 9990.00 | cop[tikv] |
   ↪                   | not(isnull(test.t2.x))
   ↪                                 |
|        ─TableFullScan_16         | 10000.00 | cop[tikv] | table:t2
   ↪         | keep order:false, stats:pseudo                    |
|    ─Selection_19(Probe)          | 0.80    | root    |
   ↪                     | not(isnull(test.t1.x))
   ↪                                 |
|      ─MaxOneRow_20               | 1.00    | root    |
   ↪                     |
   ↪                                                   |
|        ─Union_21                 | 2.00    | root    |
   ↪                     |
   ↪                                                   |
|          ─TableReader_24         | 2.00    | root    |
   ↪                     | data:Selection_23
   ↪                                       |
|            ─Selection_23         | 2.00    | cop[tikv] |
   ↪                   | lt(test.t2.x, 2), lt(test.t2.x, test.t1.x)
   ↪         |
|              ─TableFullScan_22   | 2.50    | cop[tikv] | table:t1,
   ↪ partition:p0 | keep order:false, stats:pseudo                |
|          ─TableReader_27         | 2.00    | root    |
   ↪                     | data:Selection_26
   ↪                                       |
|            ─Selection_26         | 2.00    | cop[tikv] |
   ↪                   | lt(test.t2.x, 2), lt(test.t2.x, test.t1.x)
   ↪         |
|              ─TableFullScan_25   | 2.50    | cop[tikv] | table:t1,
   ↪ partition:p1 | keep order:false, stats:pseudo                |
+--
   ↪ ----------------------------------+----------+-----------+--------------------
   ↪
14 rows in set (0.00 sec)
```

Each time this query reads a row from `t2`, it will query on the `t1` partitioned table.

Theoretically, the `t1.x> val` filter condition is met at this time, but in fact, partition pruning takes effect only in the generation phase of the query plan, not the execution phase.

11.3.3.2.8 TopN and Limit Operator Push Down

This document describes the implementation of TopN and Limit operator pushdown.

In the TiDB execution plan tree, the `LIMIT` clause in SQL corresponds to the Limit operator node, and the `ORDER BY` clause corresponds to the Sort operator node. The adjacent Limit operator and Sort operator are combined as the TopN operator node, which means that the top N records are returned according to a certain sorting rule. That is to say, a Limit operator is equivalent to a TopN operator node with a null sorting rule.

Similar to predicate pushdown, TopN and Limit are pushed down in the execution plan tree to a position as close to the data source as possible so that the required data is filtered at an early stage. In this way, the pushdown significantly reduces the overhead of data transmission and calculation.

To disable this rule, refer to Optimization Rules and Blocklist for Expression Pushdown.

Examples

This section illustrates TopN pushdown through some examples.

Example 1: Push down to the Coprocessors in the storage layer

```
create table t(id int primary key, a int not null);
explain select * from t order by a limit 10;
```

```
+-------------------------+---------+-----------+--------------+--------------------
   ↪
| id                      | estRows | task      | access object | operator
   ↪ info            |
+-------------------------+---------+-----------+--------------+--------------------
   ↪
| TopN_7                  | 10.00   | root      |              | test.t.a,
   ↪ offset:0, count:10 |
|  ─TableReader_15        | 10.00   | root      |              | data:TopN_14
   ↪                |
|    ─TopN_14             | 10.00   | cop[tikv] |              | test.t.a,
   ↪ offset:0, count:10 |
|      ─TableFullScan_13  | 10000.00 | cop[tikv] | table:t   | keep order:
   ↪ false, stats:pseudo |
+-------------------------+---------+-----------+--------------+--------------------
   ↪
4 rows in set (0.00 sec)
```

In this query, the TopN operator node is pushed down to TiKV for data filtering, and each Coprocessor returns only 10 records to TiDB. After TiDB aggregates the data, the final filtering is performed.

Example 2: TopN can be pushed down into Join (the sorting rule only depends on the columns in the outer table)

```
create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t left join s on t.a = s.a order by t.a limit 10;
```

```
+---------------------------------+----------+-----------+---------------+-------------
   ↪
| id                              | estRows | task      | access object |
   ↪ operator info                                |
+---------------------------------+----------+-----------+---------------+-------------
   ↪
| TopN_12                         | 10.00    | root      |               | test.t.a,
   ↪  offset:0, count:10                 |
|  ─HashJoin_17                    | 12.50    | root      |               | left
   ↪ outer join, equal:[eq(test.t.a, test.s.a)] |
|    ─TopN_18(Build)              | 10.00    | root      |               | test.t.a
   ↪ , offset:0, count:10                |
|      ─TableReader_26            | 10.00    | root      |               | data:
   ↪ TopN_25                               |
|        ─TopN_25                 | 10.00    | cop[tikv] |               | test.t.a
   ↪ , offset:0, count:10                |
|          ─TableFullScan_24      | 10000.00 | cop[tikv] | table:t       | keep
   ↪ order:false, stats:pseudo             |
|    ─TableReader_30(Probe)       | 10000.00 | root      |               | data:
   ↪ TableFullScan_29                      |
|      ─TableFullScan_29          | 10000.00 | cop[tikv] | table:s       | keep
   ↪ order:false, stats:pseudo             |
+---------------------------------+----------+-----------+---------------+-------------
   ↪
8 rows in set (0.01 sec)
```

In this query, the sorting rule of the TopN operator only depends on the columns in the outer table `t`, so a calculation can be performed before pushing down TopN to Join, to reduce the calculation cost of the Join operation. Besides, TiDB also pushes TopN down to the storage layer.

Example 3: TopN cannot be pushed down before Join

```
create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t join s on t.a = s.a order by t.id limit 10;
```

```
+-------------------------------+----------+-----------+---------------+---------------
   ↪
```

```
| id                           | estRows | task    | access object | operator
   ↪ info                         |
+------------------------------+---------+---------+---------------+---------------
   ↪
| TopN_12                      | 10.00   | root    |               | test.t.id,
   ↪ offset:0, count:10           |
|  ─HashJoin_16                | 12500.00 | root   |               | inner join,
   ↪  equal:[eq(test.t.a, test.s.a)] |
|    ─TableReader_21(Build)    | 10000.00 | root   |               | data:
   ↪ TableFullScan_20             |
|      ─TableFullScan_20       | 10000.00 | cop[tikv] | table:s     | keep order:
   ↪ false, stats:pseudo          |
|    ─TableReader_19(Probe)    | 10000.00 | root   |               | data:
   ↪ TableFullScan_18             |
|      ─TableFullScan_18       | 10000.00 | cop[tikv] | table:t     | keep order:
   ↪ false, stats:pseudo          |
+------------------------------+---------+---------+---------------+---------------
   ↪
6 rows in set (0.00 sec)
```

TopN cannot be pushed down before `Inner Join`. Taking the query above as an example, if you get 100 records after Join, then you can have 10 records left after TopN. However, if TopN is performed first to get 10 records, only 5 records are left after Join. In such cases, the pushdown results in different results.

Similarly, TopN can neither be pushed down to the inner table of Outer Join, nor can it be pushed down when its sorting rule is related to columns on multiple tables, such as `t.a+s.a`. Only when the sorting rule of TopN exclusively depends on columns on the outer table, can TopN be pushed down.

Example 4: Convert TopN to Limit

```
create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t left join s on t.a = s.a order by t.id limit 10;
```

```
+--------------------------------+---------+----------+--------------+-------------
   ↪
| id                             | estRows | task     | access object |
   ↪ operator info                          |
+--------------------------------+---------+----------+--------------+-------------
   ↪
| TopN_12                        | 10.00   | root     |               | test.t.id
   ↪ , offset:0, count:10                |
|  ─HashJoin_17                  | 12.50   | root     |               | left
   ↪ outer join, equal:[eq(test.t.a, test.s.a)] |
```

```
|   └─Limit_21(Build)           | 10.00   | root     |              | offset
    ↪ :0, count:10                          |
|     └─TableReader_31          | 10.00   | root     |              | data:
    ↪ Limit_30                              |
|       └─Limit_30              | 10.00   | cop[tikv] |             | offset
    ↪ :0, count:10                          |
|         └─TableFullScan_29    | 10.00   | cop[tikv] | table:t     | keep
    ↪ order:true, stats:pseudo              |
|   └─TableReader_35(Probe)     | 10000.00 | root    |              | data:
    ↪ TableFullScan_34                      |
|     └─TableFullScan_34        | 10000.00 | cop[tikv] | table:s   | keep
    ↪ order:false, stats:pseudo             |
+-------------------------------+----------+-----------+--------------+-------------
    ↪
8 rows in set (0.00 sec)
```

In the query above, TopN is first pushed to the outer table `t`. TopN needs to sort by `t ↪ .id`, which is the primary key and can be directly read in order (`keep order: true`) without extra sorting in TopN. Therefore, TopN is simplified as Limit.

11.3.3.2.9 Introduction to Join Reorder

In real application scenarios, it is common to join multiple tables. The execution efficiency of join is associated with the order in which each table joins.

For example:

```
SELECT * FROM t1, t2, t3 WHERE t1.a=t2.a AND t3.a=t2.a;
```

In this query, tables can be joined in the following two orders:

- t1 joins t2, and then joins t3
- t2 joins t3, and then joins t1

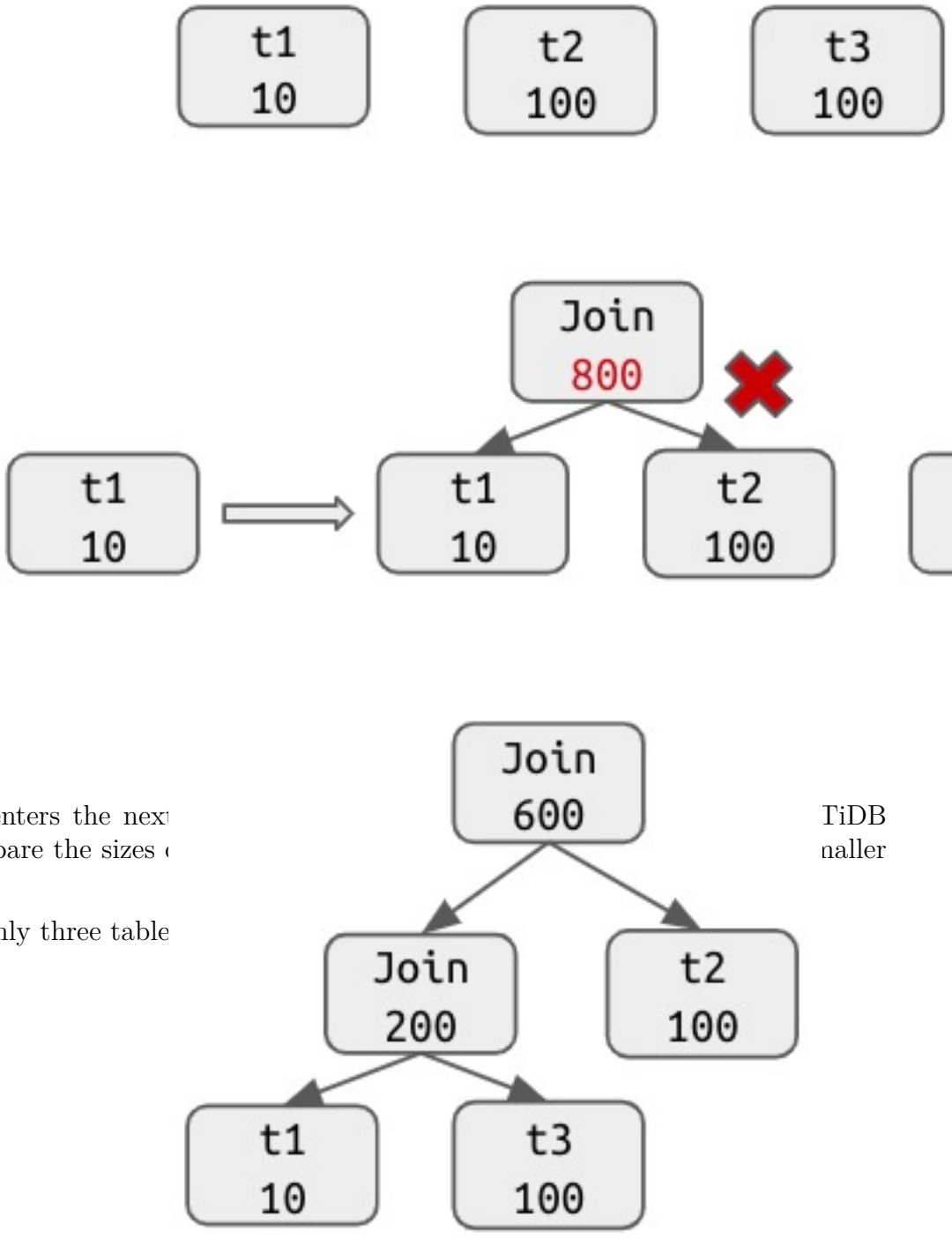
As t1 and t3 have different data volumes and distribution, these two execution orders might show different performances.

Therefore, the optimizer needs an algorithm to determine the join order. Currently, TiDB uses the Join Reorder algorithm, also known as the greedy algorithm.

Instance of Join Reorder algorithm

Take the three tables above (t1, t2, and t3) as an example.

First, TiDB obtains all the nodes that participates in the join operation, and sorts the nodes in the ascending order of row numbers.



After that, respectively. By smaller result se

Then TiDB enters the nex continues to compare the sizes result set.

In this case only three table

TiDB naller

Figure 182: join-reorder-3

The above process is the Join Reorder algorithm currently used in TiDB.

Limitations of Join Reorder algorithm

The current Join Reorder algorithm has the following limitations:

- Limited by the calculation methods of the result sets, the algorithm cannot ensure it selects the optimum join order.

- Currently, the Join Reorder algorithm's support for Outer Join is disabled by default. To enable it, set the value of the system variable `tidb_enable_outer_join_reorder` to `ON`.

Currently, the `STRAIGHT_JOIN` syntax is supported in TiDB to force a join order. For more information, refer to Description of the syntax elements.

11.3.3.3 Physical Optimization

11.3.3.3.1 SQL Physical Optimization

Physical optimization is cost-based optimization, which makes a physical execution plan for the logical execution plan generated in the previous stage. In this stage, the optimizer selects a specific physical implementation for each operator in the logical execution plan. Different physical implementations of logical operators have different time complexity, resource consumption and physical properties. In this process, the optimizer determines the cost of different physical implementations based on the statistics of the data, and selects the physical execution plan with the smallest overall cost.

Understand the Query Execution Plan has introduced some physical operators. This chapter focuses on the following aspects:

- In Index Selection, you will learn how to select the optimal index to access tables when TiDB has multiple indexes on a table.
- In Introduction to Statistics, you will learn what statistics TiDB collects to obtain the data distribution of a table.
- Wrong Index Solution introduces how to use the right index when you find the index is selected wrongly.
- Distinct Optimization introduces an optimization related to the `DISTINCT` keyword during physical optimization. In this section, you will learn its advantages and disadvantages and how to use it.
- Cost Model introduces how to choose an optimal execution plan based on the cost model during physical optimization.

11.3.3.3.2 Index Selection

Reading data from storage engines is one of the most time-consuming steps during the SQL execution. Currently, TiDB supports reading data from different storage engines and different indexes. Query execution performance depends largely on whether you select a suitable index or not.

This document introduces how to select an index to access a table, and some related ways to control index selection.

Access tables

Before introducing index selection, it is important to understand the ways TiDB accesses tables, what triggers each way, what differences each way makes, and what the pros and cons are.

Operators for accessing tables

Operator	Trigger Conditions	Applicable Scenarios	Explanations
PointGet / BatchPointGet	When accessing tables in one or more single point ranges.	Any scenario	If triggered, it is usually considered as the fastest operator, since it calls the kvget interface directly to perform the calculations rather than calls the coprocessor interface.

Operator	Trigger Conditions	Applicable Scenarios	Explanations
TableReader	None	Any scenario	It is generally considered as the least efficient operator that scans table data directly from the TiKV layer. It can be selected only if there is a range query on the `_tidb_rowid` column, or if there are no other operators for accessing tables to

Operator	Trigger Conditions	Applicable Scenarios	Explanations
TableReader	A table has a replica on the TiFlash node.	There are fewer columns to read, but many rows to evaluate.	Tiflash is column-based storage. If you need to calculate a small number of columns and a large number of rows, it is recommended to choose this operator.

Operator	Trigger Conditions	Applicable Scenarios	Explanations
IndexReader	A table has one or more indexes, and the columns needed for the calculation are included in the indexes.	When there is a smaller range query on the indexes, or when there is an order requirement for indexed columns.	When multiple indexes exist, a reasonable index is selected based on the cost estimation.

Operator	Trigger Conditions	Applicable Scenarios	Explanations
IndexLookUpReader	A table has one or more indexes, and the columns needed for calculation are not completely included in the indexes.	Same as IndexReader.	Since the index does not completely cover calculated columns, TiDB needs to retrieve rows from a table after reading indexes. There is an extra cost compared to the IndexReader operator.

Note:

The TableReader operator is based on the `_tidb_rowid` column index, and TiFlash uses a column storage index, so the selection of index is the selection

of an operator for accessing tables.

Index selection rules

TiDB selects indexes based on rules or cost. The based rules include pre-rules and skyline-pruning. When selecting an index, TiDB tries the pre-rule first. If an index satisfies a pre-rule, TiDB directly selects this index. Otherwise, TiDB uses skyline-pruning to exclude unsuitable indexes, and then selects the index with the lowest cost based on the cost estimation of each operator that accesses tables.

Rule-based selection

Pre-rules

TiDB uses the following heuristic pre-rules to select indexes:

- Rule 1: If an index satisfies "unique index with full match + no need to retrieve rows from a table (which means that the plan generated by the index is the IndexReader operator)", TiDB directly selects this index.
- Rule 2: If an index satisfies "unique index with full match + the need to retrieve rows from a table (which means that the plan generated by the index is the IndexLookupReader operator)", TiDB selects the index with the smallest number of rows to be retrieved from a table as a candidate index.
- Rule 3: If an index satisfies "ordinary index + no need to retrieve rows from a table + the number of rows to be read is less than the value of a certain threshold", TiDB selects the index with the smallest number of rows to be read as a candidate index.
- Rule 4: If only one candidate index is selected based on rule 2 and 3, select this candidate index. If two candidate indexes are respectively selected based on rule 2 and 3, select the index with the smaller number of rows to be read (the number of rows with index + the number of rows to be retrieved from a table).

The "index with full match" in the above rules means each indexed column has the equal condition. When executing the `EXPLAIN FORMAT = 'verbose' ...` statement, if the pre-rules match an index, TiDB outputs a NOTE-level warning indicating that the index matches the pre-rule.

In the following example, because the index `idx_b` meets the condition "unique index with full match + the need to retrieve rows from a table" in rule 2, TiDB selects the index `idx_b` as the access path, and `SHOW WARNING` returns a note indicating that the index `idx_b` matches the pre-rule.

```
mysql> CREATE TABLE t(a INT PRIMARY KEY, b INT, c INT, UNIQUE INDEX idx_b(b
    ↪ ));
```

```
Query OK, 0 rows affected (0.01 sec)

mysql> EXPLAIN FORMAT = 'verbose' SELECT b, c FROM t WHERE b = 3 OR b = 6;
+--
    ↪ ----------------+---------+---------+------+-------------------------+---------
    ↪
| id                | estRows | estCost | task | access object     | operator
    ↪ info            |
+--
    ↪ ----------------+---------+---------+------+-------------------------+---------
    ↪
| Batch_Point_Get_5 | 2.00 | 8.80  | root | table:t, index:idx_b(b) | keep
    ↪ order:false, desc:false |
+--
    ↪ ----------------+---------+---------+------+-------------------------+---------
    ↪
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+--
    ↪ -----+------+-----------------------------------------------------------------
    ↪
| Level | Code | Message
    ↪                                                                              |
+--
    ↪ -----+------+-----------------------------------------------------------------
    ↪
| Note | 1105 | unique index idx_b of t is selected since the path only has
    ↪  point ranges with double scan |
+--
    ↪ -----+------+-----------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

Skyline-pruning

Skyline-pruning is a heuristic filtering rule for indexes, which can reduce the probability of wrong index selection caused by wrong estimation. To judge an index, the following three dimensions are needed:

- How many access conditions are covered by the indexed columns. An "access condition" is a where condition that can be converted to a column range. And the more access conditions an indexed column set covers, the better it is in this dimension.
- Whether it needs to retrieve rows from a table when you select the index to access the table (that is, the plan generated by the index is IndexReader operator or In-

dexLookupReader operator). Indexes that do not retrieve rows from a table are better on this dimension than indexes that do. If both indexes need TiDB to retrieve rows from the table, compare how many filtering conditions are covered by the indexed columns. Filtering conditions mean the `where` condition that can be judged based on the index. If the column set of an index covers more access conditions, the smaller the number of retrieved rows from a table, and the better the index is in this dimension.

- Select whether the index satisfies a certain order. Because index reading can guarantee the order of certain column sets, indexes that satisfy the query order are superior to indexes that do not satisfy on this dimension.

For these three dimensions above, if the index `idx_a` performs no worse than the index `idx_b` in all three dimensions and performs better than `idx_b` in one dimension, then `idx_a` is preferred. When executing the `EXPLAIN FORMAT = 'verbose' ...` statement, if skyline-pruning excludes some indexes, TiDB outputs a NOTE-level warning listing the remaining indexes after the skyline-pruning exclusion.

In the following example, the indexes `idx_b` and `idx_e` are both inferior to `idx_b_c`, so they are excluded by skyline-pruning. The returned result of `SHOW WARNING` displays the remaining indexes after skyline-pruning.

```
mysql> CREATE TABLE t(a INT PRIMARY KEY, b INT, c INT, d INT, e INT, INDEX
    ↪ idx_b(b), INDEX idx_b_c(b, c), INDEX idx_e(e));
Query OK, 0 rows affected (0.01 sec)

mysql> EXPLAIN FORMAT = 'verbose' SELECT * FROM t WHERE b = 2 AND c > 4;
+--
    ↪ ----------------------------+---------+---------+----------+----------------
    ↪
| id                          | estRows | estCost | task   | access object
    ↪            | operator info                                  |
+--
    ↪ ----------------------------+---------+---------+----------+----------------
    ↪
| IndexLookUp_10              | 33.33   | 738.29 | root     |
    ↪                             |
    ↪                                                |
|  ─IndexRangeScan_8(Build)  | 33.33   | 2370.00 | cop[tikv] | table:t, index
    ↪ :idx_b_c(b, c) | range:(2 4,2 +inf], keep order:false, stats:pseudo |
|  ─TableRowIDScan_9(Probe)  | 33.33   | 2370.00 | cop[tikv] | table:t
    ↪                | keep order:false, stats:pseudo                |
+--
    ↪ ----------------------------+---------+---------+----------+----------------
    ↪
3 rows in set, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS;
+--
    ↪ -----+------+-----------------------------------------------------------------------
    ↪
| Level | Code | Message
    ↪                                                                                |
+--
    ↪ -----+------+-----------------------------------------------------------------------
    ↪
| Note | 1105 | [t,idx_b_c] remain after pruning paths for t given Prop{
    ↪ SortItems: [], TaskTp: rootTask} |
+--
    ↪ -----+------+-----------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

Cost estimation-based selection

After using the skyline-pruning rule to rule out inappropriate indexes, the selection of indexes is based entirely on the cost estimation. The cost estimation of accessing tables requires the following considerations:

- The average length of each row of the indexed data in the storage engine.
- The number of rows in the query range generated by the index.
- The cost for retrieving rows from a table.
- The number of ranges generated by index during the query execution.

According to these factors and the cost model, the optimizer selects an index with the lowest cost to access the table.

Common tuning problems with cost estimation based selection

1. The estimated number of rows is not accurate?

   This is usually due to stale or inaccurate statistics. You can re-execute the `analyze table` statement or modify the parameters of the `analyze table` statement.

2. Statistics are accurate, and reading from TiFlash is faster, but why does the optimizer choose to read from TiKV?

   At present, the cost model of distinguishing TiFlash from TiKV is still rough. You can decrease the value of `tidb_opt_seek_factor` parameter, then the optimizer prefers to choose TiFlash.

3. The statistics are accurate. Index A needs to retrieve rows from tables, but it actually executes faster than Index B that does not retrieve rows from tables. Why does the optimizer choose Index B?

In this case, the cost estimation may be too large for retrieving rows from tables. You can decrease the value of `tidb_opt_network_factor` parameter to reduce the cost of retrieving rows from tables.

Control index selection

The index selection can be controlled by a single query through Optimizer Hints.

- `USE_INDEX` / `IGNORE_INDEX` can force the optimizer to use / not use certain indexes. `FORCE_INDEX` and `USE_INDEX` have the same effect.
- `READ_FROM_STORAGE` can force the optimizer to choose the TiKV / TiFlash storage engine for certain tables to execute queries.

11.3.3.3.3 Introduction to Statistics

TiDB uses statistics to decide which index to choose. The `tidb_analyze_version` variable controls the statistics collected by TiDB. Currently, two versions of statistics are supported: `tidb_analyze_version = 1` and `tidb_analyze_version = 2`.

In versions earlier than v5.1.0, the default value of this variable is `1`. In v5.3.0 and later versions, the default value of this variable is `2`. If your cluster is upgraded from a version earlier than v5.3.0 to v5.3.0 or later, the default value of `tidb_analyze_version` does not change.

Note:

When `tidb_analyze_version = 2`, if memory overflow occurs after `ANALYZE` is executed, you need to set `tidb_analyze_version = 1` and perform one of the following operations:

- If the `ANALYZE` statement is executed manually, manually analyze every table to be analyzed.

```
sql   SELECT DISTINCT(CONCAT('ANALYZE TABLE ', table_schema
↪ , '.', table_name, ';'))FROM information_schema.tables,
↪ mysql.stats_histograms WHERE stats_ver = 2 AND table_id =
↪ tidb_table_id;
```

- If TiDB automatically executes the `ANALYZE` statement because the auto-analysis has been enabled, execute the following statement that generates the `DROP STATS` statement:

```
sql   SELECT DISTINCT(CONCAT('DROP STATS ', table_schema,
↪ '.', table_name, ';'))FROM information_schema.tables,
↪ mysql.stats_histograms WHERE stats_ver = 2 AND table_id =
↪  tidb_table_id;
```

- If the result of the preceding statement is too long to copy and paste, you can export the result to a temporary text file and then perform execution from the file like this:

```
sql   SELECT DISTINCT ... INTO OUTFILE '/tmp/sql.txt'; mysql -h
↪  XXX -u user -P 4000 ... < '/tmp/sql.txt';
```

These two versions include different information in TiDB:

Information	Version 1	Version 2
The total number of rows in the table	√	√
Column Count-Min Sketch	√	×
Index Count-Min Sketch	√	×

Information	Version 1	Version 2
Column Top-N	√	√ (Maintenance methods and precision are improved)
Index Top-N	√ (Insufficient maintenance precision might cause inaccuracy)	√ (Maintenance methods and precision are improved)
Column histogram	√	√ (The histogram does not include Top-N values.)

Information	Version 1	Version 2
Index histogram	√	√ (The histogram buckets record the number of different values in each bucket, and the histogram does not include Top-N values.)
The number of `NULL` ↪ s in the column	√	√

Information	Version 1	Version 2
The number of `NULL` ↪ s in the index	√	√
The average length of columns	√	√
The average length of indexes	√	√

Compared to Version 1, Version 2 statistics avoids the potential inaccuracy caused by hash collision when the data volume is huge. It also maintains the estimate precision in most scenarios.

This document briefly introduces the histogram, Count-Min Sketch, and Top-N, and details the collection and maintenance of statistics.

Histogram

A histogram is an approximate representation of the distribution of data. It divides the entire range of values into a series of buckets, and uses simple data to describe each bucket, such as the number of values falling in the bucket. In TiDB, an equal-depth histogram is created for the specific columns of each table. The equal-depth histogram can be used to estimate the interval query.

Here "equal-depth" means that the number of values falling into each bucket is as equal as possible. For example, for a given set {1.6, 1.9, 1.9, 2.0, 2.4, 2.6, 2.7, 2.7, 2.8, 2.9, 3.4, 3.5}, you want to generate 4 buckets. The equal-depth histogram is as follows. It contains

four buckets [1.6, 1.9], [2.0, 2.6], [2.7, 2.8], [2.9, 3.5]. The bucket depth is 3.

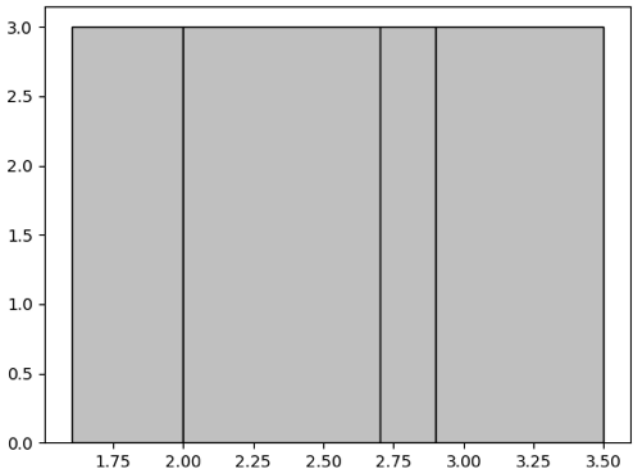


Figure 183: Equal-depth Histogram Example

For details about the parameter that determines the upper limit to the number of histogram buckets, refer to Manual Collection. When the number of buckets is larger, the accuracy of the histogram is higher; however, higher accuracy is at the cost of the usage of memory resources. You can adjust this number appropriately according to the actual scenario.

Count-Min Sketch

Count-Min Sketch is a hash structure. When an equivalence query contains `a = 1` or `IN` query (for example, `a in (1, 2, 3)`), TiDB uses this data structure for estimation.

A hash collision might occur since Count-Min Sketch is a hash structure. In the `EXPLAIN` statement, if the estimate of the equivalent query deviates greatly from the actual value, it can be considered that a larger value and a smaller value have been hashed together. In this case, you can take one of the following ways to avoid the hash collision:

- Modify the `WITH NUM TOPN` parameter. TiDB stores the high-frequency (top x) data separately, with the other data stored in Count-Min Sketch. Therefore, to prevent a larger value and a smaller value from being hashed together, you can increase the value of `WITH NUM TOPN`. In TiDB, its default value is 20. The maximum value is 1024. For more information about this parameter, see Full Collection.
- Modify two parameters `WITH NUM CMSKETCH DEPTH` and `WITH NUM CMSKETCH WIDTH`. Both affect the number of hash buckets and the collision probability. You can increase the values of the two parameters appropriately according to the actual scenario to reduce the probability of hash collision, but at the cost of higher memory usage of statistics. In TiDB, the default value of `WITH NUM CMSKETCH DEPTH` is 5, and the

default value of `WITH NUM CMSKETCH WIDTH` is 2048. For more information about the two parameters, see Full Collection.

Top-N values

Top-N values are values with the top N occurrences in a column or index. TiDB records the values and occurrences of Top-N values.

Collect statistics

Manual collection

You can run the `ANALYZE` statement to collect statistics.

Note:

The execution time of `ANALYZE TABLE` in TiDB is longer than that in MySQL or InnoDB. In InnoDB, only a small number of pages are sampled, while in TiDB a comprehensive set of statistics is completely rebuilt. Scripts that were written for MySQL may naively expect `ANALYZE TABLE` will be a short-lived operation.

For quicker analysis, you can set `tidb_enable_fast_analyze` to `1` to enable the Quick Analysis feature. The default value for this parameter is `0`.

After Quick Analysis is enabled, TiDB randomly samples approximately 10,000 rows of data to build statistics. Therefore, in the case of uneven data distribution or a relatively small amount of data, the accuracy of statistical information is relatively poor. It might lead to poor execution plans, such as choosing the wrong index. If the execution time of the normal `ANALYZE` statement is acceptable, it is recommended to disable the Quick Analysis feature.

`tidb_enable_fast_analyze` is an experimental feature, which currently **does not match exactly** with the statistical information of `tidb_analyze_version=2`. Therefore, you need to set the value of `tidb_analyze_version` to `1` when `tidb_enable_fast_analyze` is enabled.

Full collection

You can perform full collection using the following syntax.

- To collect statistics of all the tables in `TableNameList`:

```
ANALYZE TABLE TableNameList [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|
    ↪ CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];
```

- `WITH NUM BUCKETS` specifies the maximum number of buckets in the generated histogram.
- `WITH NUM TOPN` specifies the maximum number of the generated `TOPN`s.
- `WITH NUM CMSKETCH DEPTH` specifies the depth of the CM Sketch.
- `WITH NUM CMSKETCH WIDTH` specifies the width of the CM Sketch.
- `WITH NUM SAMPLES` specifies the number of samples.
- `WITH FLOAT_NUM SAMPLERATE` specifies the sampling rate.

`WITH NUM SAMPLES` and `WITH FLOAT_NUM SAMPLERATE` correspond to two different algorithms of collecting samples.

- `WITH NUM SAMPLES` specifies the size of the sampling set, which is implemented in the reservoir sampling method in TiDB. When a table is large, it is not recommended to use this method to collect statistics. Because the intermediate result set of the reservoir sampling contains redundant results, it causes additional pressure on resources such as memory.
- `WITH FLOAT_NUM SAMPLERATE` is a sampling method introduced in v5.3.0. With the value range `(0, 1]`, this parameter specifies the sampling rate. It is implemented in the way of Bernoulli sampling in TiDB, which is more suitable for sampling larger tables and performs better in collection efficiency and resource usage.

Before v5.3.0, TiDB uses the reservoir sampling method to collect statistics. Since v5.3.0, the TiDB Version 2 statistics uses the Bernoulli sampling method to collect statistics by default. To re-use the reservoir sampling method, you can use the `WITH NUM SAMPLES` statement.

The current sampling rate is calculated based on an adaptive algorithm. When you can observe the number of rows in a table using `SHOW STATS_META`, you can use this number of rows to calculate the sampling rate corresponding to 100,000 rows. If you cannot observe this number, you can use the `TABLE_KEYS` column in the `TABLE_STORAGE_STATS` table as another reference to calculate the sampling rate.

> **Note:**
>
> Normally, `STATS_META` is more credible than `TABLE_KEYS`. However, after importing data through the methods like TiDB Lightning physical import mode, the result of `STATS_META` is `0`. To handle this situation, you can use `TABLE_KEYS` to calculate the sampling rate when the result of `STATS_META` is much smaller than the result of `TABLE_KEYS`.

Collect statistics on some columns

In most cases, when executing SQL statements, the optimizer only uses statistics on some columns (such as columns in the `WHERE`, `JOIN`, `ORDER BY`, and `GROUP BY` statements). These columns are called `PREDICATE COLUMNS`.

If a table has many columns, collecting statistics on all the columns can cause a large overhead. To reduce the overhead, you can collect statistics on only specific columns or `PREDICATE COLUMNS` to be used by the optimizer.

> **Note:**
>
> Collecting statistics on some columns is only applicable for `tidb_analyze_version = 2`.

- To collect statistics on specific columns, use the following syntax:

```
ANALYZE TABLE TableName COLUMNS ColumnNameList [WITH NUM BUCKETS|TOPN|
    ↪ CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM
    ↪ SAMPLERATE];
```

  In the syntax, `ColumnNameList` specifies the name list of the target columns. If you need to specify more than one column, use comma `,` to separate the column names. For example, `ANALYZE table t columns a, b`. Besides collecting statistics on the specific columns in a specific table, this syntax collects statistics on the indexed columns and all indexes in that table at the same time.

> **Note:**
>
> The syntax above is a full collection. For example, after collecting statistics on columns `a` and `b` using this syntax, if you also want to collect statistics on column `c`, you need to specify all three columns using `ANALYZE table t columns a, b, c`, rather than only specifying the additional column `c` using `ANALYZE TABLE t COLUMNS c`.

- To collect statistics on `PREDICATE COLUMNS`, do the following:

> **Warning:**
>
> Currently, collecting statistics on `PREDICATE COLUMNS` is an experimental feature. It is not recommended that you use it in production environments.

1. Set the value of the tidb_enable_column_tracking system variable to ON to enable TiDB to collect PREDICATE COLUMNS.

   After the setting, TiDB writes the PREDICATE COLUMNS information to the mysql.column_stats_usage system table every 100 * stats-lease.

2. After the query pattern of your business is relatively stable, collect statistics on PREDICATE COLUMNS by using the following syntax:

```
ANALYZE TABLE TableName PREDICATE COLUMNS [WITH NUM BUCKETS|TOPN|
    ↪ CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH
    ↪ FLOATNUM SAMPLERATE];
```

   Besides collecting statistics on PREDICATE COLUMNS in a specific table, this syntax collects statistics on indexed columns and all indexes in that table at the same time.

Note:

- If the mysql.column_stats_usage system table does not contain any PREDICATE COLUMNS record for that table, the preceding syntax collects statistics on all columns and all indexes in that table.
- After using this syntax to collect statistics, when executing a new type of SQL query, the optimizer might temporarily use the old or pseudo column statistics for this time, and TiDB will collect the statistics on the used columns from the next time.

- To collect statistics on all columns and indexes, use the following syntax:

```
ANALYZE TABLE TableName ALL COLUMNS [WITH NUM BUCKETS|TOPN|CMSKETCH
    ↪ DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE
    ↪ ];
```

If you want to persist the column configuration in the ANALYZE statement (including COLUMNS ColumnNameList, PREDICATE COLUMNS, and ALL COLUMNS), set the value of the tidb_persist_analyze_options system variable to ON to enable the ANALYZE configuration persistence feature. After enabling the ANALYZE configuration persistence feature:

- When TiDB collects statistics automatically or when you manually collect statistics by executing the ANALYZE statement without specifying the column configuration, TiDB continues using the previously persisted configuration for statistics collection.
- When you manually execute the ANALYZE statement multiple times with column configuration specified, TiDB overwrites the previously recorded persistent configuration using the new configuration specified by the latest ANALYZE statement.

To locate PREDICATE COLUMNS and columns on which statistics have been collected, use the following syntax:

```
SHOW COLUMN_STATS_USAGE [ShowLikeOrWhere];
```

The `SHOW COLUMN_STATS_USAGE` statement returns the following 6 columns:

Column name	Description
`Db_name`	The database name
`Table_name`	The table name
`Partition_name`	The partition name
`Column_name`	The column name
`Last_used_at`	The last time when the column statistics were used in the query optimization
`Last_analyzed_at`	The last time when the column statistics were collected

In the following example, after executing `ANALYZE TABLE t PREDICATE COLUMNS;`, TiDB collects statistics on columns `b`, `c`, and `d`, where column `b` is a `PREDICATE COLUMN` and columns `c` and `d` are index columns.

```
SET GLOBAL tidb_enable_column_tracking = ON;
Query OK, 0 rows affected (0.00 sec)

CREATE TABLE t (a INT, b INT, c INT, d INT, INDEX idx_c_d(c, d));
Query OK, 0 rows affected (0.00 sec)

-- The optimizer uses the statistics on column b in this query.
SELECT * FROM t WHERE b > 1;
Empty set (0.00 sec)

-- After waiting for a period of time (100 * stats-lease), TiDB writes the
   ↪ collected `PREDICATE COLUMNS` to mysql.column_stats_usage.
```

```
-- Specify `last_used_at IS NOT NULL` to show the `PREDICATE COLUMNS`
    ↪ collected by TiDB.
SHOW COLUMN_STATS_USAGE WHERE db_name = 'test' AND table_name = 't' AND
    ↪ last_used_at IS NOT NULL;
+--
    ↪ -------+-----------+---------------+------------+-------------------+------
    ↪
| Db_name | Table_name | Partition_name | Column_name | Last_used_at |
    ↪ Last_analyzed_at |
+--
    ↪ -------+-----------+---------------+------------+-------------------+------
    ↪
| test   | t          |               | b          | 2022-01-05 17:21:33 | NULL
    ↪          |
+--
    ↪ -------+-----------+---------------+------------+-------------------+------
    ↪
1 row in set (0.00 sec)

ANALYZE TABLE t PREDICATE COLUMNS;
Query OK, 0 rows affected, 1 warning (0.03 sec)

-- Specify `last_analyzed_at IS NOT NULL` to show the columns for which
    ↪ statistics have been collected.
SHOW COLUMN_STATS_USAGE WHERE db_name = 'test' AND table_name = 't' AND
    ↪ last_analyzed_at IS NOT NULL;
+--
    ↪ -------+-----------+---------------+------------+-------------------+------
    ↪
| Db_name | Table_name | Partition_name | Column_name | Last_used_at |
    ↪ Last_analyzed_at |
+--
    ↪ -------+-----------+---------------+------------+-------------------+------
    ↪
| test   | t          |               | b          | 2022-01-05 17:21:33 |
    ↪ 2022-01-05 17:23:06 |
| test   | t          |               | c          | NULL                |
    ↪ 2022-01-05 17:23:06 |
| test   | t          |               | d          | NULL                |
    ↪ 2022-01-05 17:23:06 |
+--
    ↪ -------+-----------+---------------+------------+-------------------+------
    ↪
3 rows in set (0.00 sec)
```

Collect statistics on indexes

To collect statistics on all indexes in `IndexNameList` in `TableName`, use the following syntax:

```
ANALYZE TABLE TableName INDEX [IndexNameList] [WITH NUM BUCKETS|TOPN|
    ↪ CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM
    ↪ SAMPLERATE];
```

When `IndexNameList` is empty, this syntax collects statistics on all indexes in `TableName` ↪ .

> **Note:**
>
> To ensure that the statistical information before and after the collection is consistent, when `tidb_analyze_version` is `2`, this syntax collects statistics on the entire table (including all columns and indexes), instead of only on indexes.

Collect statistics on partitions

- To collect statistics on all partitions in `PartitionNameList` in `TableName`, use the following syntax:

```
ANALYZE TABLE TableName PARTITION PartitionNameList [WITH NUM BUCKETS|
    ↪ TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH
    ↪ FLOATNUM SAMPLERATE];
```

- To collect index statistics on all partitions in `PartitionNameList` in `TableName`, use the following syntax:

```
ANALYZE TABLE TableName PARTITION PartitionNameList INDEX [
    ↪ IndexNameList] [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH
    ↪ WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];
```

- If you only need to collect statistics on some columns of some partitions in a table, use the following syntax:

> **Warning:**
>
> Currently, collecting statistics on `PREDICATE COLUMNS` is an experimental feature. It is not recommended that you use it in production environments.

```
ANALYZE TABLE TableName PARTITION PartitionNameList [COLUMNS
    ↪ ColumnNameList|PREDICATE COLUMNS|ALL COLUMNS] [WITH NUM BUCKETS|
    ↪ TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH
    ↪ FLOATNUM SAMPLERATE];
```

Collect statistics of partitioned tables in dynamic pruning mode

When accessing partitioned tables in dynamic pruning mode, TiDB collects table-level statistics, which is called GlobalStats. Currently, GlobalStats is aggregated from statistics of all partitions. In dynamic pruning mode, a statistics update of any partitioned table can trigger the GlobalStats to be updated.

Note:

- When GlobalStats update is triggered:
  - If some partitions have no statistics (such as a new partition that has never been analyzed), GlobalStats generation is interrupted and a warning message is displayed saying that no statistics are available on partitions.
  - If statistics of some columns are absent in specific partitions (different columns are specified for analyzing in these partitions), GlobalStats generation is interrupted when statistics of these columns are aggregated, and a warning message is displayed saying that statistics of some columns are absent in specific partitions.
- In dynamic pruning mode, the Analyze configurations of partitions and tables should be the same. Therefore, if you specify the `COLUMNS` ↪ configuration following the `ANALYZE TABLE TableName PARTITION` ↪ `PartitionNameList` statement or the `OPTIONS` configuration following `WITH`, TiDB will ignore them and return a warning.

Incremental collection

To improve the speed of analysis after full collection, incremental collection could be used to analyze the newly added sections in monotonically non-decreasing columns such as time columns.

Note:

- Currently, the incremental collection is only provided for index.
- When using the incremental collection, you must ensure that only `INSERT` operations exist on the table, and that the newly inserted value on the index column is monotonically non-decreasing. Otherwise, the statistical information might be inaccurate, affecting the TiDB optimizer to select an appropriate execution plan.

You can perform incremental collection using the following syntax.

- To incrementally collect statistics on index columns in all `IndexNameLists` in `TableName`:

```
ANALYZE INCREMENTAL TABLE TableName INDEX [IndexNameList] [WITH NUM
    ↪ BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|
    ↪ WITH FLOATNUM SAMPLERATE];
```

- To incrementally collect statistics on index columns for partitions in all `PartitionNameLists` ↪ in `TableName`:

```
ANALYZE INCREMENTAL TABLE TableName PARTITION PartitionNameList INDEX [
    ↪ IndexNameList] [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH
    ↪ WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];
```

Automatic update

For the `INSERT`, `DELETE`, or `UPDATE` statements, TiDB automatically updates the number of rows and updated rows. TiDB persists this information regularly and the update cycle is 20 * `stats-lease`. The default value of `stats-lease` is `3s`. If you specify the value as `0`, it does not update automatically.

Three system variables related to automatic update of statistics are as follows:

System Variable	Default Value	Description
tidb_auto_analyze_ratio	0.5	The threshold value of automatic update
tidb_auto_analyze_start_time	00:00 +0000	The start time in a day when TiDB can perform automatic update
tidb_auto_analyze_end_time	23:59 +0000	The end time in a day when TiDB can perform automatic update

When the ratio of the number of modified rows to the total number of rows of `tbl` ↪ in a table is greater than `tidb_auto_analyze_ratio`, and the current time is between `tidb_auto_analyze_start_time` and `tidb_auto_analyze_end_time`, TiDB executes the `ANALYZE TABLE tbl` statement in the background to automatically update the statistics on this table.

To avoid the situation that modifying a small amount of data on a small table frequently triggers the automatic update, when a table has less than 1000 rows, such data modifying does not trigger the automatic update in TiDB. You can use the `SHOW STATS_META` statement to view the number of rows in a table.

> **Note:**
>
> Currently, the automatic update does not record the configuration items input at manual `ANALYZE`. Therefore, when you use the `WITH` syntax to control the collecting behavior of `ANALYZE`, you need to manually set scheduled tasks to collect statistics.

Before TiDB v5.0, when you execute a query, TiDB collects feedback with `feedback-` ↪ `probability` and updates the histogram and Count-Min Sketch based on the feedback. **Since v5.0, this feature is disabled by default. Since v5.4, this feature has been deprecated, and it is not recommended to enable this feature.**

Since TiDB v6.0, TiDB supports using the `KILL` statement to terminate an `ANALYZE` task running in the background. If you find that an `ANALYZE` task running in the background consumes a lot of resources and affects your application, you can terminate the `ANALYZE` task by taking the following steps:

1. Execute the following SQL statement:

   ```
   SHOW ANALYZE STATUS
   ```

   By checking the `instance` column and the `process_id` column in the result, you can get the TiDB instance address and the task `ID` of the background `ANALYZE` task.

2. Terminate the `ANALYZE` task that is running in the background.

   - If `enable-global-kill` is `true` (`true` by default), you can execute the `KILL` ↪ `TIDB ${id};` statement directly, where `${id}` is the `ID` of the background `ANALYZE` task obtained from the previous step.
   - If `enable-global-kill` is `false`, you need to use a client to connect to the TiDB instance that is executing the backend `ANALYZE` task, and then execute the `KILL` ↪ `TIDB ${id};` statement. If you use a client to connect to another TiDB instance, or if there is a proxy between the client and the TiDB cluster, the `KILL` statement cannot terminate the background `ANALYZE` task.

For more information on the `KILL` statement, see KILL.

Control `ANALYZE` concurrency

When you run the `ANALYZE` statement, you can adjust the concurrency using the following parameters, to control its effect on the system.

`tidb_build_stats_concurrency`

Currently, when you run the `ANALYZE` statement, the task is divided into multiple small tasks. Each task only works on one column or index. You can use the `tidb_build_stats_concurrency` parameter to control the number of simultaneous tasks. The default value is `4`.

`tidb_distsql_scan_concurrency`

When you analyze regular columns, you can use the `tidb_distsql_scan_concurrency` parameter to control the number of Region to be read at one time. The default value is `15`.

`tidb_index_serial_scan_concurrency`

When you analyze index columns, you can use the `tidb_index_serial_scan_concurrency` ↪ parameter to control the number of Region to be read at one time. The default value is `1`.

Persist ANALYZE configurations

Since v5.4.0, TiDB supports persisting some `ANALYZE` configurations. With this feature, the existing configurations can be easily reused for future statistics collection.

The following are the `ANALYZE` configurations that support persistence:

Configuration	Corresponding ANALYZE syntax
The number of histogram buckets	WITH NUM BUCKETS
The number of Top-N	WITH NUM TOPN

Configuration	Corresponding ANALYZE syntax
The number of samples	WITH NUM SAMPLES
The sampling rate	WITH FLOAT_NUM SAMPLERATE
The column type	AnalyzeColumnOption ANALYZE TABLE (‘ALL COLUMNS’ | ‘PREDICATE COLUMNS’ | ‘COLUMNS’ ColumnNameList)
The column	ColumnNameList ANALYZE TABLE Identifier (‘,’ Identifier)*

Enable ANALYZE configuration persistence

The `ANALYZE` configuration persistence feature is enabled by default (the system variable `tidb_analyze_version` is `2` and `tidb_persist_analyze_options` is `ON` by default).

You can use this feature to record the persistence configurations specified in the `ANALYZE` statement when executing the statement manually. Once recorded, the next time TiDB automatically updates statistics or you manually collect statistics without specifying these configuration, TiDB will collect statistics according to the recorded configurations.

To query the configuration persisted on a specific table used for auto analyze operations, you can use the following SQL statement:

```
SELECT sample_num, sample_rate, buckets, topn, column_choice, column_ids
    ↪ FROM mysql.analyze_options opt JOIN information_schema.tables tbl ON
    ↪ opt.table_id = tbl.tidb_table_id WHERE tbl.table_schema = '{db_name}'
    ↪  AND tbl.table_name = '{table_name}';
```

TiDB will overwrite the previously recorded persistent configuration using the new configurations specified by the latest `ANALYZE` statement. For example, if you run `ANALYZE TABLE t WITH 200 TOPN;`, it will set the top 200 values in the `ANALYZE` statement. Subsequently, executing `ANALYZE TABLE t WITH 0.1 SAMPLERATE;` will set both the top 200 values and a sampling rate of 0.1 for auto `ANALYZE` statements, similar to `ANALYZE TABLE t WITH 200 TOPN, 0.1 SAMPLERATE;`.

Disable ANALYZE configuration persistence

To disable the `ANALYZE` configuration persistence feature, set the `tidb_persist_analyze_options` system variable to `OFF`. Because the `ANALYZE` configuration persistence feature is not applicable to `tidb_analyze_version = 1`, setting `tidb_analyze_version = 1` can also disable the feature.

After disabling the `ANALYZE` configuration persistence feature, TiDB does not clear the persisted configuration records. Therefore, if you enable this feature again, TiDB continues to collect statistics using the previously recorded persistent configurations.

> **Note:**
>
> When you enable the `ANALYZE` configuration persistence feature again, if the previously recorded persistence configurations are no longer applicable to the latest data, you need to execute the `ANALYZE` statement manually and specify the new persistence configurations.

The memory quota for collecting statistics

> **Warning:**
>
> Currently, the `ANALYZE` memory quota is an experimental feature, and the memory statistics might be inaccurate in production environments.

Since TiDB v6.1.0, you can use the system variable `tidb_mem_quota_analyze` to control the memory quota for collecting statistics in TiDB.

To set a proper value of `tidb_mem_quota_analyze`, consider the data size of the cluster. When the default sampling rate is used, the main considerations are the number of columns, the size of column values, and the memory configuration of TiDB. Consider the following suggestions when you configure the maximum and minimum values:

> **Note:**
>
> The following suggestions are for reference only. You need to configure the values based on the real scenario.

- Minimum value: should be greater than the maximum memory usage when TiDB collects statistics from the table with the most columns. An approximate reference: when TiDB collects statistics from a table with 20 columns using the default configuration, the maximum memory usage is about 800 MiB; when TiDB collects statistics from a table with 160 columns using the default configuration, the maximum memory usage is about 5 GiB.
- Maximum value: should be less than the available memory when TiDB is not collecting statistics.

View `ANALYZE` state

When executing the `ANALYZE` statement, you can view the current state of `ANALYZE` using the following SQL statement:

```
SHOW ANALYZE STATUS [ShowLikeOrWhere]
```

This statement returns the state of `ANALYZE`. You can use `ShowLikeOrWhere` to filter the information you need.

Currently, the `SHOW ANALYZE STATUS` statement returns the following 11 columns:

Column name	Description
table_schema	The database name
table_name	The table name
partition_name	The partition name
job_info	The task information. If an index is analyzed, this information will include the index name. When `tidb_analyze_version =2`, this information will include configuration items such as sample rate.
processed_rows	The number of rows that have been analyzed
start_time	The time at which the task starts
state	The state of a task, including `pending`, `running`, `finished`, and `failed`
fail_reason	The reason why the task fails. If the execution is successful, the value is `NULL`.
instance	The TiDB instance that executes the task

Column name	Description
process_id	The process ID that executes the task

Starting from TiDB v6.1.0, the `SHOW ANALYZE STATUS` statement supports showing cluster-level tasks. Even after a TiDB restart, you can still view task records before the restart using this statement. Before TiDB v6.1.0, the `SHOW ANALYZE STATUS` statement can only show instance-level tasks, and task records are cleared after a TiDB restart.

`SHOW ANALYZE STATUS` shows the most recent task records only. Starting from TiDB v6.1.0, you can view the history tasks within the last 7 days through the system table `mysql.analyze_jobs`.

When `tidb_mem_quota_analyze` is set and an automatic `ANALYZE` task running in the TiDB background uses more memory than this threshold, the task will be retried. You can see failed and retried tasks in the output of the `SHOW ANALYZE STATUS` statement.

When `tidb_max_auto_analyze_time` is greater than 0 and an automatic `ANALYZE` task running in the TiDB background takes more time than this threshold, the task will be terminated.

```
mysql> SHOW ANALYZE STATUS [ShowLikeOrWhere];
+--
   ↪ ------------+------------+----------------+-----------------------------------
   ↪
| Table_schema | Table_name | Partition_name | Job_info
   ↪                                                                  |
   ↪ Processed_rows | Start_time     | End_time          | State   |
   ↪ Fail_reason                                                        |
+--
   ↪ ------------+------------+----------------+-----------------------------------
   ↪
| test         | sbtest1    |                | retry auto analyze table all
   ↪ columns with 100 topn, 0.055 samplerate          |       2000000 |
   ↪ 2022-05-07 16:41:09 | 2022-05-07 16:41:20 | finished | NULL
   ↪                                                              |
| test         | sbtest1    |                | auto analyze table all columns
   ↪ with 100 topn, 0.5 samplerate                    |            0 |
   ↪ 2022-05-07 16:40:50 | 2022-05-07 16:41:09 | failed | analyze panic
   ↪ due to memory quota exceeds, please try with smaller samplerate |
```

View statistics

You can view the statistics status using the following statements.

Metadata of tables

You can use the `SHOW STATS_META` statement to view the total number of rows and the number of updated rows.

```
SHOW STATS_META [ShowLikeOrWhere];
```

The syntax of `ShowLikeOrWhereOpt` is as follows:

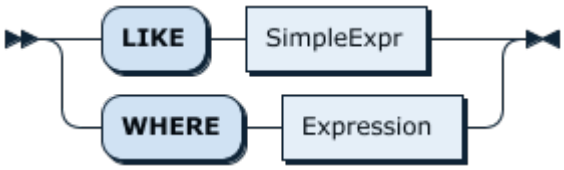


Figure 184: ShowLikeOrWhereOpt

Currently, the `SHOW STATS_META` statement returns the following 6 columns:

Column name	Description
`db_name`	The database name
`table_name`	The table name
`partition_name`	The partition name
`update_time`	The time of the update
`modify_count`	The number of modified rows
`row_count`	The total number of rows

Note:

When TiDB automatically updates the total number of rows and the number of modified rows according to DML statements, `update_time` is also updated. Therefore, `update_time` does not necessarily indicate the last time when the `ANALYZE` statement is executed.

Health state of tables

You can use the `SHOW STATS_HEALTHY` statement to check the health state of tables and roughly estimate the accuracy of the statistics. When `modify_count` >= `row_count`, the health state is 0; when `modify_count` < `row_count`, the health state is (1 - `modify_count` ↪ `/row_count`) * 100.

The syntax is as follows:

```
SHOW STATS_HEALTHY [ShowLikeOrWhere];
```

The synopsis of `SHOW STATS_HEALTHY` is:



Figure 185: ShowStatsHealthy

Currently, the `SHOW STATS_HEALTHY` statement returns the following 4 columns:

Column name	Description
`db_name`	The database name
`table_name`	The table name
`partition_name`	The partition name
`healthy`	The health state of tables

Metadata of columns

You can use the `SHOW STATS_HISTOGRAMS` statement to view the number of different values and the number of `NULL` in all the columns.

Syntax as follows:

```
SHOW STATS_HISTOGRAMS [ShowLikeOrWhere]
```

This statement returns the number of different values and the number of `NULL` in all the columns. You can use `ShowLikeOrWhere` to filter the information you need.

Currently, the `SHOW STATS_HISTOGRAMS` statement returns the following 10 columns:

Column name	Description
`db_name`	The database name
`table_name`	The table name
`partition_name`	The partition name
`column_name`	The column name (when `is_index` is `0`) or the index name (when `is_index` is `1`)
`is_index`	Whether it is an index column or not

Column name	Description
`update_time` ↪	The time of the update
`distinct_count` ↪	The number of different values
`null_count` ↪	The number of `NULL`
`avg_col_size` ↪	The average length of columns
correlation	The Pearson correlation coefficient of the column and the integer primary key, which indicates the degree of association between the two columns

Buckets of histogram

You can use the `SHOW STATS_BUCKETS` statement to view each bucket of the histogram.

The syntax is as follows:

```
SHOW STATS_BUCKETS [ShowLikeOrWhere]
```

The diagram is as follows:



Figure 186: SHOW STATS_BUCKETS

This statement returns information about all the buckets. You can use `ShowLikeOrWhere` to filter the information you need.

Currently, the `SHOW STATS_BUCKETS` statement returns the following 11 columns:

Column name	Description
`db_name`	The database name
`table_name`	The table name
`partition_name`	The partition name
`column_name`	The column name (when `is_index` is `0`) or the index name (when `is_index` is `1`)
`is_index`	Whether it is an index column or not
`bucket_id`	The ID of a bucket
`count`	The number of all the values that falls on the bucket and the previous buckets
`repeats`	The occurrence number of the maximum value
`lower_bound`	The minimum value
`upper_bound`	The maximum value
`ndv`	The number of different values in the bucket. When `tidb_analyze_version = 1`, `ndv` is always `0`, which has no actual meaning.

Top-N information

You can use the `SHOW STATS_TOPN` statement to view the Top-N information currently

collected by TiDB.

The syntax is as follows:

```
SHOW STATS_TOPN [ShowLikeOrWhere];
```

Currently, the `SHOW STATS_TOPN` statement returns the following 7 columns:

Column name	Description
`db_name`	The database name
`table_name`	The table name
`partition_name`	The partition name
`column_name`	The column name (when `is_index` is 0)
`is_index`	Whether it is an index column or not

Column name	Description
`value`	The value of this column
`count`	How many times the value appears

Delete statistics

You can run the `DROP STATS` statement to delete statistics.

```
DROP STATS TableName
```

The preceding statement deletes all statistics of `TableName`. If a partitioned table is specified, this statement will delete statistics of all partitions in this table as well as GlobalStats generated in dynamic pruning mode.

```
DROP STATS TableName PARTITION PartitionNameList;
```

This preceding statement only deletes statistics of the specified partitions in `PartitionNameList`.

```
DROP STATS TableName GLOBAL;
```

The preceding statement only deletes GlobalStats generated in dynamic pruning mode of the specified table.

Load statistics

By default, depending on the size of column statistics, TiDB loads statistics differently as follows:

- For statistics that consume small amounts of memory (such as count, distinctCount, and nullCount), as long as the column data is updated, TiDB automatically loads the corresponding statistics into memory for use in the SQL optimization stage.
- For statistics that consume large amounts of memory (such as histograms, TopN, and Count-Min Sketch), to ensure the performance of SQL execution, TiDB loads the statistics asynchronously on demand. Take histograms as an example. TiDB

loads histogram statistics on a column into memory only when the optimizer uses the histogram statistics on that column. On-demand asynchronous statistics loading does not affect the performance of SQL execution but might provide incomplete statistics for SQL optimization.

Since v5.4.0, TiDB introduces the synchronously loading statistics feature. This feature allows TiDB to synchronously load large-sized statistics (such as histograms, TopN, and Count-Min Sketch statistics) into memory when you execute SQL statements, which improves the completeness of statistics for SQL optimization.

> **Warning:**
>
> Currently, synchronously loading statistics is an experimental feature. It is not recommended that you use it in production environments.

The synchronously loading statistics feature is disabled by default. To enable this feature, set the value of the `tidb_stats_load_sync_wait` system variable to a timeout (in milliseconds) that SQL optimization can wait for at most to synchronously load complete column statistics. The default value of this variable is `0`, indicating that the feature is disabled.

After enabling the synchronously loading statistics feature, you can further configure the feature as follows:

- To control how TiDB behaves when the waiting time of SQL optimization reaches the timeout, modify the value of the `tidb_stats_load_pseudo_timeout` system variable. The default value of this variable is `OFF`, indicating that the SQL execution fails after the timeout. If you set this variable to `ON`, after the timeout, the SQL optimization process does not use any histogram, TopN, or CMSketch statistics on any columns, but gets back to using pseudo statistics.
- To specify the maximum number of columns that the synchronously loading statistics feature can process concurrently, modify the value of the `stats-load-concurrency` option in the TiDB configuration file. The default value is `5`.
- To specify the maximum number of column requests that the synchronously loading statistics feature can cache, modify the value of the `stats-load-queue-size` option in the TiDB configuration file. The default value is `1000`.

Import and export statistics

Export statistics

The interface to export statistics is as follows:

- To obtain the JSON format statistics of the `${table_name}` table in the `${db_name}` database:

```
http://${tidb-server-ip}:${tidb-server-status-port}/stats/dump/${
    ↪ db_name}/${table_name}
```

For example:

```
curl -s http://127.0.0.1:10080/stats/dump/test/t1 -o /tmp/t1.json
```

- To obtain the JSON format statistics of the `${table_name}` table in the `${db_name}` database at specific time:

```
http://${tidb-server-ip}:${tidb-server-status-port}/stats/dump/${
    ↪ db_name}/${table_name}/${yyyyMMddHHmmss}
```

Import statistics

> **Note:**
>
> When you start the MySQL client, use the `--local-infile=1` option.

Generally, the imported statistics refer to the JSON file obtained using the export interface.

Syntax:

```
LOAD STATS 'file_name'
```

`file_name` is the file name of the statistics to be imported.

See also

- LOAD STATS
- DROP STATS

11.3.3.3.4 Introduction to Extended Statistics

TiDB can collect the following two types of statistics:

- Basic statistics: statistics such as histograms and Count-Min Sketch. See Introduction to Statistics for details.
- Extended statistics: statistics filtered by tables and columns.

Tip:

Before reading this document, it is recommended that you read Introduction to Statistics first.

When the `ANALYZE` statement is executed manually or automatically, TiDB by default only collects the basic statistics and does not collect the extended statistics. This is because the extended statistics are only used for optimizer estimates in specific scenarios, and collecting them requires additional overhead.

Extended statistics are disabled by default. To collect extended statistics, you need to first enable the extended statistics, and then register each individual extended statistics object.

After the registration, the next time the `ANALYZE` statement is executed, TiDB collects both the basic statistics and the registered extended statistics.

Limitations

Extended statistics are not collected in the following scenarios:

- Statistics collection on indexes only
- Statistics collection using the `ANALYZE INCREMENTAL` command
- Statistics collection when the value of the system variable `tidb_enable_fast_analyze` is set to `true`

Common operations

Enable extended statistics

To enable extended statistics, set the system variable `tidb_enable_extended_stats` to `ON`:

```
SET GLOBAL tidb_enable_extended_stats = ON;
```

The default value of this variable is `OFF`. The setting of this system variable applies to all extended statistics objects.

Register extended statistics

The registration for extended statistics is not a one-time task, and you need repeat the registration for each extended statistics object.

To register extended statistics, use the SQL statement `ALTER TABLE ADD STATS_EXTENDED` ↪ . The syntax is as follows:

```
ALTER TABLE table_name ADD STATS_EXTENDED IF NOT EXISTS stats_name
    ↪ stats_type(column_name, column_name...);
```

In the syntax, you can specify the table name, statistics name, statistics type, and column name of the extended statistics to be collected.

- `table_name` specifies the name of the table from which the extended statistics are collected.
- `stats_name` specifies the name of the statistics object, which must be unique for each table.
- `stats_type` specifies the type of the statistics. Currently, only the correlation type is supported.
- `column_name` specifies the column group, which might have multiple columns. Currently, you can only specify two column names.

How it works

To improve access performance, each TiDB node maintains a cache in the system table `mysql.stats_extended` for extended statistics. After you register the extended statistics, the next time the `ANALYZE` statement is executed, TiDB will collect the extended statistics if the system table `mysql.stats_extended` has the corresponding objects.

Each row in the `mysql.stats_extended` table has a `version` column. Once a row is updated, the value of `version` is increased. In this way, TiDB loads the table into memory incrementally, instead of fully.

TiDB loads `mysql.stats_extended` periodically to ensure that the cache is kept the same as the data in the table.

Warning:

It is **NOT RECOMMENDED** to directly operate on the `mysql.stats_extended` system table. Otherwise, inconsistent caches occur on different TiDB nodes.

If you have mistakenly operated on the table, you can execute the following statement on each TiDB node. Then the current cache will be cleared and the `mysql.stats_extended` table will be fully reloaded:

```
ADMIN RELOAD STATS_EXTENDED;
```

Delete extended statistics

To delete an extended statistics object, use the following statement:

```
ALTER TABLE table_name DROP STATS_EXTENDED stats_name;
```

How it works

After you execute the statement, TiDB marks the value of the corresponding object in `mysql.stats_extended`'s column `status` to `2`, instead of deleting the object directly.

Other TiDB nodes will read this change and delete the object in their memory cache. The background garbage collection will delete the object eventually.

> **Warning:**
>
> It is **NOT RECOMMENDED** to directly operate on the `mysql.stats_extended` system table. Otherwise, inconsistent caches occur on different TiDB nodes.
>
> If you have mistakenly operated on the table, you can use the following statement on each TiDB node. Then the current cache will be cleared and the `mysql.stats_extended` table will be fully reloaded:
>
> ```
> ADMIN RELOAD STATS_EXTENDED;
> ```

Export and import extended statistics

The way of exporting or importing extended statistics is the same as exporting or importing basic statistics. See Introduction to Statistics - Import and export statistics for details.

Usage examples for correlation-type extended statistics

Currently, TiDB only supports the correlation-type extended statistics. This type is used to estimate the number of rows in the range query and improve index selection. The following example shows how the correlation-type extended statistics are used to estimate the number of rows in a range query.

Step 1. Define the table

Define a table `t` as follows:

```
CREATE TABLE t(col1 INT, col2 INT, KEY(col1), KEY(col2));
```

Suppose that `col1` and `col2` of table `t` both obey monotonically increasing constraints in row order. This means that the values of `col1` and `col2` are strictly correlated in order, and the correlation factor is `1`.

Step 2. Execute an example query without extended statistics

Execute the following query without using extended statistics:

```
SELECT * FROM t WHERE col1 > 1 ORDER BY col2 LIMIT 1;
```

For the execution of the preceding query, the TiDB optimizer has the following options to access table `t`:

- Uses the index on `col1` to access table `t` and then sorts the result by `col2` to calculate `Top-1`.
- Uses the index on `col2` to meet the first row that satisfies `col1 > 1`. The cost of this access method mainly depends on how many rows are filtered out when TiDB scans the table in `col2`'s order.

Without extended statistics, the TiDB optimizer only supposes that `col1` and `col2` are independent, which **leads to a significant estimation error**.

Step 3. Enable extended statistics

Set `tidb_enable_extended_stats` to `ON`, and register the extended statistics object for `col1` and `col2`:

```
ALTER TABLE t ADD STATS_EXTENDED s1 correlation(col1, col2);
```

When you execute `ANALYZE` after the registration, TiDB calculates the Pearson correlation coefficient of `col` and `col2` of table `t`, and writes the object into the `mysql.stats_extended` table.

Step 4. See how extended statistics make a difference

After TiDB has the extended statistics for correlation, the optimizer can estimate how many rows to be scanned more precisely.

At this time, for the query in Stage 2. Execute an example query without extended statistics, `col1` and `col2` are strictly correlated in order. If TiDB accesses table `t` by using the index on `col2` to meet the first row that satisfies `col1 > 1`, the TiDB optimizer will equivalently translate the row count estimation into the following query:

```
SELECT * FROM t WHERE col1 <= 1 OR col1 IS NULL;
```

The preceding query result plus one will be the final estimation for the row count. In this way, you do not need to use the independent assumption and **the significant estimation error is avoided**.

If the correlation factor (`1` in this example) is less than the value of the system variable `tidb_opt_correlation_threshold`, the optimizer will use the independent assumption, but it will also increase the estimation heuristically. The larger the value of `tidb_opt_correlation_exp_factor`, the larger the estimation result. The larger the absolute value of the correlation factor, the larger the estimation result.

11.3.3.3.5 Wrong Index Solution

If you find that the execution speed of some query does not reach the expectation, the optimizer might choose the wrong index to run the query.

You can first view the health state of tables in the statistics, and then solve this issue according to the different health states.

Low health state

The low health state means TiDB has not performed the`ANALYZE` statement for a long time. You can update the statistics by running the `ANALYZE` command. After the update, if the optimizer still uses the wrong index, refer to the next section.

Near 100% health state

The near 100% health state suggests that the `ANALYZE` statement is just completed or was completed a short time ago. In this case, the wrong index issue might be related to TiDB's estimation logic for the number of rows.

For equivalence queries, the cause might be Count-Min Sketch. You can check whether Count-Min Sketch is the cause and take corresponding solutions.

If the cause above does not apply to your problem, you can force-select indexes by using the `USE_INDEX` or `use index` optimzer hint (see USE_INDEX for details). Also, you can change the query behavior by using SQL Plan Management in a non-intrusive way.

Other situations

Apart from the aforementioned situations, the wrong index issue might also be caused by data updates which renders all the indexes no longer applicable. In such cases, you need to perform analysis on the conditions and data distribution to see whether new indexes can speed up the query. If so, you can add new indexes by running the `ADD INDEX` command.

11.3.3.3.6 Distinct Optimization

This document introduces the `distinct` optimization in the TiDB query optimizer, including `SELECT DISTINCT` and `DISTINCT` in the aggregate functions.

`DISTINCT` modifier in `SELECT` statements

The `DISTINCT` modifier specifies removal of duplicate rows from the result set. `SELECT DISTINCT` is transformed to `GROUP BY`, for example:

```
mysql> explain SELECT DISTINCT a from t;
+--
   ↪ ----------------------+---------+----------+---------------+----------------
   ↪
| id                    | estRows | task    | access object | operator info
   ↪                               |
+--
   ↪ ----------------------+---------+----------+---------------+----------------
   ↪
| HashAgg_6             | 2.40    | root     |               | group by:test.t.a
   ↪ , funcs:firstrow(test.t.a)->test.t.a |
| └─TableReader_11      | 3.00    | root     |               | data:
   ↪ TableFullScan_10                          |
```

```
|   └─TableFullScan_10   | 3.00   | cop[tikv] | table:t    | keep order:false
    ↪ , stats:pseudo                |
+--
    ↪ ----------------------+--------+----------+--------------+----------------
    ↪
3 rows in set (0.00 sec)
```

`DISTINCT` option in aggregate functions

Usually, aggregate functions with the `DISTINCT` option is executed in the TiDB layer in a single-threaded execution model.

The `tidb_opt_distinct_agg_push_down` system variable or the `distinct-agg-push ↪ -down` configuration item in TiDB controls whether to rewrite the distinct aggregate queries and push them to the TiKV or TiFlash Coprocessor.

Take the following queries as an example of this optimization. `tidb_opt_distinct_agg_push_down ↪` is disabled by default, which means the aggregate functions are executed in the TiDB layer. After enabling this optimization by setting its value to `1`, the `distinct a` part of `count(distinct a)` is pushed to TiKV or TiFlash Coprocessor: there is a HashAgg_5 to remove the duplicated values on column a in the TiKV Coprocessor. It might reduce the computation overhead of `HashAgg_8` in the TiDB layer.

```
mysql> desc select count(distinct a) from test.t;
+--
    ↪ ----------------------+----------+----------+--------------+----------------
    ↪
| id                    | estRows | task     | access object | operator info
    ↪                     |
+--
    ↪ ----------------------+----------+----------+--------------+----------------
    ↪
| StreamAgg_6           | 1.00    | root     |               | funcs:count(
    ↪ distinct test.t.a)->Column#4 |
|  └─TableReader_10     | 10000.00 | root    |               | data:
    ↪ TableFullScan_9               |
|    └─TableFullScan_9  | 10000.00 | cop[tikv] | table:t   | keep order:false
    ↪ , stats:pseudo      |
+--
    ↪ ----------------------+----------+----------+--------------+----------------
    ↪
3 rows in set (0.01 sec)

mysql> set session tidb_opt_distinct_agg_push_down = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> desc select count(distinct a) from test.t;
```

```
+--
   ↪ ------------------------+---------+----------+--------------+-------------
   ↪
| id                        | estRows | task     | access object | operator info
   ↪                          |
+--
   ↪ ------------------------+---------+----------+--------------+-------------
   ↪
| HashAgg_8                 | 1.00    | root     |               | funcs:count(
   ↪ distinct test.t.a)->Column#3 |
| ─TableReader_9            | 1.00    | root     |               | data:HashAgg_5
   ↪                           |
|   ─HashAgg_5              | 1.00    | cop[tikv] |              | group by:test.
   ↪ t.a,                      |
|     ─TableFullScan_7      | 10000.00 | cop[tikv] | table:t     | keep order:
   ↪ false, stats:pseudo       |
+--
   ↪ ------------------------+---------+----------+--------------+-------------
   ↪
4 rows in set (0.00 sec)
```

11.3.3.3.7 Cost Model

TiDB uses a cost model to choose an index and operator during physical optimization. The process is illustrated in the following diagram:

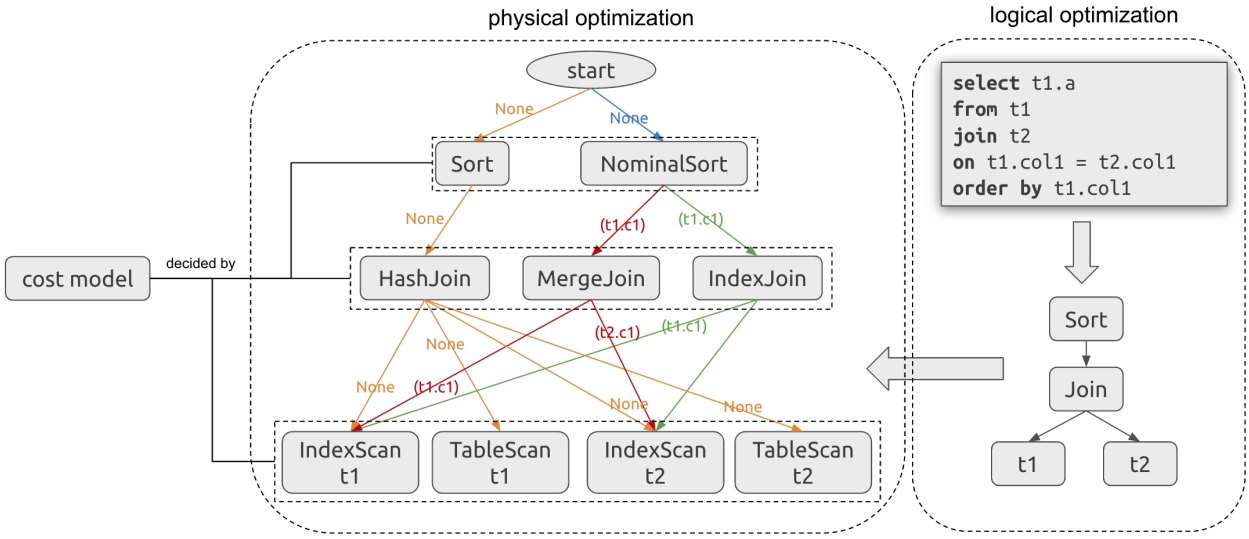


Figure 187: CostModel

TiDB calculates the access cost of each index and the execution cost of each physical

operator in plans (such as HashJoin and IndexJoin) and chooses the minimum cost plan.

The following is a simplified example to explain how the cost model works. Suppose that there is a table `t`:

```
mysql> SHOW CREATE TABLE t;
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| t     | CREATE TABLE `t` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `c` int(11) DEFAULT NULL,
  KEY `b` (`b`),
  KEY `c` (`c`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

When executing the `SELECT * FROM t WHERE b < 100 and c < 100` statement, suppose that TiDB estimates 20 rows meet the `b < 100` condition and 500 rows meet `c < 100`, and the length of `INT` type indexes is 8. Then TiDB calculates the cost for two indexes:

- The cost of index `b` = row count of `b < 100` * length of index `b` = 20 * 8 = 160
- The cost of index `c` = row count of `c < 100` * length of index `c` = 500 * 8 = 4000

Because the cost of index `b` is lower, TiDB chooses `b` as the index.

The preceding example is simplified and only used to explain the basic principle. In real SQL executions, the TiDB cost model is more complex.

11.3.3.4 SQL Prepare Execution Plan Cache

TiDB supports execution plan caching for `Prepare` and `Execute` queries. This includes both forms of prepared statements:

- Using the `COM_STMT_PREPARE` and `COM_STMT_EXECUTE` protocol features.
- Using the SQL statements `PREPARE` and `EXECUTE`.

The TiDB optimizer handles these two types of queries in the same way: when preparing, the parameterized query is parsed into an AST (Abstract Syntax Tree) and cached; in later execution, the execution plan is generated based on the stored AST and specific parameter values.

When the execution plan cache is enabled, in the first execution every `Prepare` statement checks whether the current query can use the execution plan cache, and if the query can use it, then put the generated execution plan into a cache implemented by LRU (Least Recently Used) linked list. In the subsequent `Execute` queries, the execution plan is obtained from the cache and checked for availability. If the check succeeds, the step of generating an execution plan is skipped. Otherwise, the execution plan is regenerated and saved in the cache.

In the current version of TiDB, if a `Prepare` statement meets any of the following conditions, the query or the plan is not cached:

- The query contains SQL statements other than `SELECT`, `UPDATE`, `INSERT`, `DELETE`, `Union`, `Intersect`, and `Except`.
- The query accesses partitioned tables or temporary tables, or a table that contains generated columns.
- The query contains sub-queries, such as `select * from t where a > (select ...)`.
- The query contains the `ignore_plan_cache` hint, such as `select /*+ ignore_plan_cache ()*/ * from t`.
- The query contains variables other than `?` (including system variables or user-defined variables), such as `select * from t where a>? and b>@x`.
- The query contains the functions that cannot be cached: `database()`, `current_user`, `current_role`, `user`, `connection_id`, `last_insert_id`, `row_count`, `version`, and `like`.
- The query contains `?` after `Limit`, such as `Limit ?` and `Limit 10, ?`. Such queries are not cached because the specific value of `?` has a great impact on query performance.
- The query contains `?` after `Order By`, such as `Order By ?`. Such queries sort data based on the column specified by `?`. If the queries targeting different columns use the same execution plan, the results will be wrong. Therefore, such queries are not cached. However, if the query is a common one, such as `Order By a+?`, it is cached.
- The query contains `?` after `Group By`, such as `Group By?`. Such queries group data based on the column specified by `?`. If the queries targeting different columns use the same execution plan, the results will be wrong. Therefore, such queries are not cached. However, if the query is a common one, such as `Group By a+?`, it is cached.
- The query contains `?` in the definition of the `Window Frame` window function, such as `(partition by year order by sale rows ? preceding)`. If `?` appears elsewhere in the window function, the query is cached.
- The query contains parameters for comparing `int` and `string`, such as `c_int >= ?` or `c_int in (?, ?)`, in which `?` indicates the string type, such as `set @x='123'`. To ensure that the query result is compatible with MySQL, parameters need to be adjusted in each query, so such queries are not cached.
- The plan attempts to access `TiFlash`.

- In most cases, the plan that contains `TableDual` is not cached, unless the current `Prepare` statement does not have parameters.

The LRU linked list is designed as a session-level cache because `Prepare` / `Execute` cannot be executed across sessions. Each element of the LRU list is a key-value pair. The value is the execution plan, and the key is composed of the following parts:

- The name of the database where `Execute` is executed
- The identifier of the `Prepare` statement, that is, the name after the `PREPARE` keyword
- The current schema version, which is updated after every successfully executed DDL statement
- The SQL mode when executing `Execute`
- The current time zone, which is the value of the `time_zone` system variable
- The value of the `sql_select_limit` system variable

Any change in the above information (for example, switching databases, renaming `Prepare` statement, executing DDL statements, or modifying the value of SQL mode / `time_zone`), or the LRU cache elimination mechanism causes the execution plan cache miss when executing.

After the execution plan cache is obtained from the cache, TiDB first checks whether the execution plan is still valid. If the current `Execute` statement is executed in an explicit transaction, and the referenced table is modified in the transaction pre-order statement, the cached execution plan accessing this table does not contain the `UnionScan` operator, then it cannot be executed.

After the validation test is passed, the scan range of the execution plan is adjusted according to the current parameter values, and then used to perform data querying.

There are several points worth noting about execution plan caching and query performance:

- No matter an execution plan is cached or not, it is affected by SQL bindings. For execution plans that have not been cached (the first `Execute`), these plans are affected by existing SQL bindings. For execution plans that have been cached, if new SQL Bindings are created, these plans become invalid.
- Cached plans are not affected by changes in statistics, optimization rules, and blocklist pushdown by expressions.
- Considering that the parameters of `Execute` are different, the execution plan cache prohibits some aggressive query optimization methods that are closely related to specific parameter values to ensure adaptability. This causes that the query plan may not be optimal for certain parameter values. For example, the filter condition of the query is `where a > ? And a < ?`, the parameters of the first `Execute` statement are `2` and `1` respectively. Considering that these two parameters maybe be `1` and `2` in the next execution time, the optimizer does not generate the optimal `TableDual` execution plan that is specific to current parameter values;

- If cache invalidation and elimination are not considered, an execution plan cache is applied to various parameter values, which in theory also results in non-optimal execution plans for certain values. For example, if the filter condition is `where a < ?` and the parameter value used for the first execution is `1`, then the optimizer generates the optimal `IndexScan` execution plan and puts it into the cache. In the subsequent executions, if the value becomes `10000`, the `TableScan` plan might be the better one. But due to the execution plan cache, the previously generated `IndexScan` is used for execution. Therefore, the execution plan cache is more suitable for application scenarios where the query is simple (the ratio of compilation is high) and the execution plan is relatively fixed.

Since v6.1.0, the execution plan cache is enabled by default. You can control prepared plan cache via the system variable `tidb_enable_prepared_plan_cache`.

> **Note:**
>
> The execution plan cache feature applies only to `Prepare` / `Execute` queries and does not take effect for normal queries.

After the execution plan cache feature is enabled, you can use the session-level system variable `last_plan_from_cache` to see whether the previous `Execute` statement used the cached execution plan, for example:

```
MySQL [test]> create table t(a int);
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> prepare stmt from 'select * from t where a = ?';
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> set @a = 1;
Query OK, 0 rows affected (0.00 sec)

-- The first execution generates an execution plan and saves it in the
   ↪ cache.
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+------------------------+
| @@last_plan_from_cache |
+------------------------+
| 0                      |
+------------------------+
1 row in set (0.00 sec)
```

```
-- The second execution hits the cache.
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+------------------------+
| @@last_plan_from_cache |
+------------------------+
| 1                      |
+------------------------+
1 row in set (0.00 sec)
```

If you find that a certain set of `Prepare` / `Execute` has unexpected behavior due to the execution plan cache, you can use the `ignore_plan_cache()` SQL hint to skip using the execution plan cache for the current statement. Still, use the above statement as an example:

```
MySQL [test]> prepare stmt from 'select /*+ ignore_plan_cache() */ * from t
    ↪  where a = ?';
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> set @a = 1;
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+------------------------+
| @@last_plan_from_cache |
+------------------------+
| 0                      |
+------------------------+
1 row in set (0.00 sec)
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+------------------------+
| @@last_plan_from_cache |
+------------------------+
| 0                      |
+------------------------+
1 row in set (0.00 sec)
```

11.3.3.4.1 Memory management of Prepared Plan Cache

Using Prepared Plan Cache has some memory overhead. In internal tests, each cached plan consumes an average of 100 KiB of memory. Because Plan Cache is currently at the

SESSION level, the total memory consumption is approximately the number of sessions * the average number of cached plans in a session * 100 KiB.

For example, the current TiDB instance has 50 sessions in concurrency and each session has approximately 100 cached plans. The total memory consumption is approximately 50 * 100 * 100 KiB = 512 MB.

You can control the maximum number of plans that can be cached in each session by configuring the system variable tidb_prepared_plan_cache_size. For different environments, the recommended value is as follows:

- When the memory threshold of the TiDB server instance is <= 64 GiB, set tidb_prepared_plan_cache_size to 50.
- When the memory threshold of the TiDB server instance is > 64 GiB, set tidb_prepared_plan_cache_size to 100.

When the unused memory of the TiDB server is less than a certain threshold, the memory protection mechanism of plan cache is triggered, through which some cached plans will be evicted.

You can control the threshold by configuring the system variable tidb_prepared_plan_cache_memory_ . The threshold is 0.1 by default, which means when the unused memory of the TiDB server is less than 10% of the total memory (90% of the memory is used), the memory protection mechanism is triggered.

Due to memory limit, plan cache might be missed sometimes. You can check the status by viewing the Plan Cache Miss OPS metric in the Grafana dashboard.

11.3.3.4.2 Clear execution plan cache

You can clear execution plan cache by executing the ADMIN FLUSH [SESSION | INSTANCE] PLAN_CACHE statement.

In this statement, [SESSION | INSTANCE]specifies whether the plan cache is cleared for the current session or the whole TiDB instance. If the scope is not specified, the statement above applies to the SESSION cache by default.

The following is an example of clearing the SESSION execution plan cache:

```
MySQL [test]> create table t (a int);
Query OK, 0 rows affected (0.00 sec)

MySQL [test]> prepare stmt from 'select * from t';
Query OK, 0 rows affected (0.00 sec)

MySQL [test]> execute stmt;
Empty set (0.00 sec)

MySQL [test]> execute stmt;
```

```
Empty set (0.00 sec)

MySQL [test]> select @@last_plan_from_cache; -- Select the cached plan
+------------------------+
| @@last_plan_from_cache |
+------------------------+
|                      1 |
+------------------------+
1 row in set (0.00 sec)

MySQL [test]> admin flush session plan_cache; -- Clear the cached plan of
    ↪ the current session
Query OK, 0 rows affected (0.00 sec)

MySQL [test]> execute stmt;
Empty set (0.00 sec)

MySQL [test]> select @@last_plan_from_cache; -- The cached plan cannot be
    ↪ selected again, because it has been cleared
+------------------------+
| @@last_plan_from_cache |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.00 sec)
```

Currently, TiDB does not support clearing `GLOBAL` execution plan cache. That means you cannot clear the cached plan of the whole TiDB cluster. The following error is reported if you try to clear the `GLOBAL` execution plan cache:

```
MySQL [test]> admin flush global plan_cache;
ERROR 1105 (HY000): Do not support the 'admin flush global scope.'
```

11.3.3.4.3 Ignore the `COM_STMT_CLOSE` command and the `DEALLOCATE PREPARE` statement

To reduce the syntax parsing cost of SQL statements, it is recommended that you run `prepare stmt` once, then `execute stmt` multiple times before running `deallocate prepare`:

```
MySQL [test]> prepare stmt from '...'; -- Prepare once
MySQL [test]> execute stmt using ...; -- Execute once
MySQL [test]> ...
MySQL [test]> execute stmt using ...; -- Execute multiple times
MySQL [test]> deallocate prepare stmt; -- Release the prepared statement
```

In real practice, you may be used to running `deallocate prepare` each time after running `execute stmt`, as shown below:

```
MySQL [test]> prepare stmt from '...'; -- Prepare once
MySQL [test]> execute stmt using ...;
MySQL [test]> deallocate prepare stmt; -- Release the prepared statement
MySQL [test]> prepare stmt from '...'; -- Prepare twice
MySQL [test]> execute stmt using ...;
MySQL [test]> deallocate prepare stmt; -- Release the prepared statement
```

In such practice, the plan obtained by the first executed statement cannot be reused by the second executed statement.

To address the problem, you can set the system varible `tidb_ignore_prepared_cache_close_stmt` ↪ to `ON` so TiDB ignores commands to close `prepare stmt`:

```
mysql> set @@tidb_ignore_prepared_cache_close_stmt=1; -- Enable the
    ↪ variable
Query OK, 0 rows affected (0.00 sec)

mysql> prepare stmt from 'select * from t'; -- Prepare once
Query OK, 0 rows affected (0.00 sec)

mysql> execute stmt;                        -- Execute once
Empty set (0.00 sec)

mysql> deallocate prepare stmt;             -- Release after the first execute
Query OK, 0 rows affected (0.00 sec)

mysql> prepare stmt from 'select * from t'; -- Prepare twice
Query OK, 0 rows affected (0.00 sec)

mysql> execute stmt;                        -- Execute twice
Empty set (0.00 sec)

mysql> select @@last_plan_from_cache;  -- Reuse the last plan
+------------------------+
| @@last_plan_from_cache |
+------------------------+
|                      1 |
+------------------------+
1 row in set (0.00 sec)
```

Monitoring

In the Grafana dashboard on the TiDB page in the **Executor** section, there are the "Queries Using Plan Cache OPS" and "Plan Cache Miss OPS" graphs. These graphs can be

used to check if both TiDB and the application are configured correctly to allow the SQL Plan Cache to work correctly. The **Server** section on the same page provides the "Prepared Statement Count" graph. This graph shows a non-zero value if the application uses prepared statements, which is required for the SQL Plan Cache to function correctly.



Figure 188: `sql_plan_cache`

11.3.4 Control Execution Plans

11.3.4.1 Control Execution Plan

The first two chapters of SQL Tuning introduce how to understand TiDB's execution plan and how TiDB generates an execution plan. This chapter introduces what methods can be used to control the generation of the execution plan when you determine the problems with the execution plan. This chapter mainly includes the following three aspects:

- In Optimizer Hints, you will learn how to use hints to guide TiDB to generate an execution plan.
- But hints change the SQL statement intrusively. In some scenarios, hints cannot be simply inserted. In SQL Plan Management, you will know how TiDB uses another syntax to non-intrusively control the generation of execution plans, and the methods of automatic execution plan evolution in the background to alleviate the execution plan instability caused by reasons such as version upgrades, which degrades cluster performance.
- Finally, you will learn how to use the blocklist in Blocklist of Optimization Rules and Expression Pushdown.

11.3.4.2 Optimizer Hints

TiDB supports optimizer hints, which are based on the comment-like syntax introduced in MySQL 5.7. For example, one of the common syntaxes is `/*+ HINT_NAME([t1_name [, t2_name] ...])*/`. Use of optimizer hints is recommended in cases where the TiDB optimizer selects a less optimal query plan.

If you encounter a situation where hints do not take effect, see Troubleshoot common issues that hints do not take effect.

11.3.4.2.1 Syntax

Optimizer hints are case insensitive and specified within `/*+ ... */` comments following the `SELECT`, `UPDATE` or `DELETE` keyword in a SQL statement. Optimizer hints are not currently supported for `INSERT` statements.

Multiple hints can be specified by separating with commas. For example, the following query uses three different hints:

```
SELECT /*+ USE_INDEX(t1, idx1), HASH_AGG(), HASH_JOIN(t1) */ count(*) FROM
    ↪  t t1, t t2 WHERE t1.a = t2.b;
```

How optimizer hints affect query execution plans can be observed in the output of `EXPLAIN` and `EXPLAIN ANALYZE`.

An incorrect or incomplete hint will not result in a statement error. This is because hints are intended to have only a *hint* (suggestion) semantic to query execution. Similarly, TiDB will at most return a warning if a hint is not applicable.

> **Note:**
>
> If the comments do not follow behind the specified keywords, they will be treated as common MySQL comments. The comments do not take effect, and no warning is reported.

Currently, TiDB supports two categories of hints, which are different in scope. The first category of hints takes effect in the scope of query blocks, such as `/*+ HASH_AGG()*/`; the second category of hints takes effect in the whole query, such as `/*+ MEMORY_QUOTA(1024 ↪ MB)*/`.

Each query or sub-query in a statement corresponds to a different query block, and each query block has its own name. For example:

```
SELECT * FROM (SELECT * FROM t) t1, (SELECT * FROM t) t2;
```

The above query statement has three query blocks: the outermost `SELECT` corresponds to the first query block, whose name is `sel_1`; the two `SELECT` sub-queries correspond to the second and the third query block, whose names are `sel_2` and `sel_3`, respectively. The sequence of the numbers is based on the appearance of `SELECT` from left to right. If you replace the first `SELECT` with `DELETE` or `UPDATE`, then the corresponding query block names are `del_1` or `upd_1`.

11.3.4.2.2 Hints that take effect in query blocks

This category of hints can follow behind **any** `SELECT`, `UPDATE` or `DELETE` keywords. To control the effective scope of the hint, use the name of the query block in the hint. You can make the hint parameters clear by accurately identifying each table in the query (in case of duplicated table names or aliases). If no query block is specified in the hint, the hint takes effect in the current block by default.

For example:

```
SELECT /*+ HASH_JOIN(@sel_1 t1@sel_1, t3) */ * FROM (SELECT t1.a, t1.b
    ↪ FROM t t1, t t2 WHERE t1.a = t2.a) t1, t t3 WHERE t1.b = t3.b;
```

This hint takes effect in the `sel_1` query block, and its parameters are the `t1` and `t3` tables in `sel_1` (`sel_2` also contains a `t1` table).

As described above, you can specify the name of the query block in the hint in the following ways:

- Set the query block name as the first parameter of the hint, and separate it from other parameters with a space. In addition to `QB_NAME`, all the hints listed in this section also have another optional hidden parameter `@QB_NAME`. By using this parameter, you can specify the effective scope of this hint.
- Append `@QB_NAME` to a table name in the parameter to explicitly specify which query block this table belongs to.

> **Note:**
>
> You must put the hint in or before the query block where the hint takes effect. If the hint is put after the query block, it cannot take effect.

QB_NAME

If the query statement is a complicated statement that includes multiple nested queries, the ID and name of a certain query block might be mistakenly identified. The hint `QB_NAME` can help us in this regard.

`QB_NAME` means Query Block Name. You can specify a new name to a query block. The specified `QB_NAME` and the previous default name are both valid. For example:

```
SELECT /*+ QB_NAME(QB1) */ * FROM (SELECT * FROM t) t1, (SELECT * FROM t)
    ↪ t2;
```

This hint specifies the outer `SELECT` query block's name to `QB1`, which makes `QB1` and the default name `sel_1` both valid for the query block.

> **Note:**
>
> In the above example, if the hint specifies the `QB_NAME` to `sel_2` and does not specify a new `QB_NAME` for the original second `SELECT` query block, then `sel_2` becomes an invalid name for the second `SELECT` query block.

MERGE_JOIN(t1_name [, tl_name ...])

The `MERGE_JOIN(t1_name [, tl_name ...])` hint tells the optimizer to use the sort-merge join algorithm for the given table(s). Generally, this algorithm consumes less memory but takes longer processing time. If there is a very large data volume or insufficient system memory, it is recommended to use this hint. For example:

```
select /*+ MERGE_JOIN(t1, t2) */ * from t1, t2 where t1.id = t2.id;
```

> **Note:**
>
> `TIDB_SMJ` is the alias for `MERGE_JOIN` in TiDB 3.0.x and earlier versions. If you are using any of these versions, you must apply the `TIDB_SMJ(t1_name [, tl_name ...])` syntax for the hint. For the later versions of TiDB, `TIDB_SMJ` and `MERGE_JOIN` are both valid names for the hint, but `MERGE_JOIN` is recommended.

INL_JOIN(t1_name [, tl_name ...])

The `INL_JOIN(t1_name [, tl_name ...])` hint tells the optimizer to use the index nested loop join algorithm for the given table(s). This algorithm might consume less system resources and take shorter processing time in some scenarios and might produce an opposite result in other scenarios. If the result set is less than 10,000 rows after the outer table is filtered by the `WHERE` condition, it is recommended to use this hint. For example:

```
select /*+ INL_JOIN(t1, t2) */ * from t1, t2 where t1.id = t2.id;
```

The parameter(s) given in `INL_JOIN()` is the candidate table for the inner table when you create the query plan. For example, `INL_JOIN(t1)` means that TiDB only considers using `t1` as the inner table to create a query plan. If the candidate table has an alias, you must use the alias as the parameter in `INL_JOIN()`; if it does not has an alias, use the table's original name as the parameter. For example, in the `select /*+ INL_JOIN(t1)*/ * from t t1, t t2 where t1.a = t2.b;` query, you must use the `t` table's alias `t1` or `t2` rather than `t` as `INL_JOIN()`'s parameter.

> **Note:**
>
> `TIDB_INLJ` is the alias for `INL_JOIN` in TiDB 3.0.x and earlier versions. If you are using any of these versions, you must apply the `TIDB_INLJ(t1_name [, tl_name ...])` syntax for the hint. For the later versions of TiDB, `TIDB_INLJ` and `INL_JOIN` are both valid names for the hint, but `INL_JOIN` is recommended.

INL_HASH_JOIN

The `INL_HASH_JOIN(t1_name [, tl_name])` hint tells the optimizer to use the index nested loop hash join algorithm. The conditions for using this algorithm are the same with the conditions for using the index nested loop join algorithm. The difference between the two algorithms is that `INL_JOIN` creates a hash table on the joined inner table, but `INL_HASH_JOIN` creates a hash table on the joined outer table. `INL_HASH_JOIN` has a fixed limit on memory usage, while the memory used by `INL_JOIN` depends on the number of rows matched in the inner table.

HASH_JOIN(t1_name [, tl_name ...])

The `HASH_JOIN(t1_name [, tl_name ...])` hint tells the optimizer to use the hash join algorithm for the given table(s). This algorithm allows the query to be executed concurrently with multiple threads, which achieves a higher processing speed but consumes more memory. For example:

```
select /*+ HASH_JOIN(t1, t2) */ * from t1, t2 where t1.id = t2.id;
```

> **Note:**
>
> `TIDB_HJ` is the alias for `HASH_JOIN` in TiDB 3.0.x and earlier versions. If you are using any of these versions, you must apply the `TIDB_HJ(t1_name [, tl_name ...])` syntax for the hint. For the later versions of TiDB, `TIDB_HJ` and `HASH_JOIN` are both valid names for the hint, but `HASH_JOIN` is recommended.

HASH_AGG()

The `HASH_AGG()` hint tells the optimizer to use the hash aggregation algorithm in all the aggregate functions in the specified query block. This algorithm allows the query to be executed concurrently with multiple threads, which achieves a higher processing speed but consumes more memory. For example:

```
select /*+ HASH_AGG() */ count(*) from t1, t2 where t1.a > 10 group by t1.
    ↪ id;
```

STREAM_AGG()

The `STREAM_AGG()` hint tells the optimizer to use the stream aggregation algorithm in all the aggregate functions in the specified query block. Generally, this algorithm consumes less memory but takes longer processing time. If there is a very large data volume or insufficient system memory, it is recommended to use this hint. For example:

```
select /*+ STREAM_AGG() */ count(*) from t1, t2 where t1.a > 10 group by t1
    ↪ .id;
```

USE_INDEX(t1_name, idx1_name [, idx2_name ...])

The `USE_INDEX(t1_name, idx1_name [, idx2_name ...])` hint tells the optimizer to use only the given index(es) for a specified `t1_name` table. For example, applying the following hint has the same effect as executing the `select * from t t1 use index(idx1, idx2);` statement.

```
SELECT /*+ USE_INDEX(t1, idx1, idx2) */ * FROM t1;
```

> **Note:**
>
> If you specify only the table name but not index name in this hint, the execution does not consider any index but scan the entire table.

FORCE_INDEX(t1_name, idx1_name [, idx2_name ...])

The `FORCE_INDEX(t1_name, idx1_name [, idx2_name ...])` hint tells the optimizer to use only the given index(es).

The usage and effect of `FORCE_INDEX(t1_name, idx1_name [, idx2_name ...])` are the same as the usage and effect of `USE_INDEX(t1_name, idx1_name [, idx2_name ...])`.

The following 4 queries have the same effect:

```
SELECT /*+ USE_INDEX(t, idx1) */ * FROM t;
SELECT /*+ FORCE_INDEX(t, idx1) */ * FROM t;
SELECT * FROM t use index(idx1);
SELECT * FROM t force index(idx1);
```

IGNORE_INDEX(t1_name, idx1_name [, idx2_name ...])

The `IGNORE_INDEX(t1_name, idx1_name [, idx2_name ...])` hint tells the optimizer to ignore the given index(es) for a specified `t1_name` table. For example, applying the

following hint has the same effect as executing the `select * from t t1 ignore index(idx1, idx2);` statement.

```
select /*+ IGNORE_INDEX(t1, idx1, idx2) */ * from t t1;
```

AGG_TO_COP()

The `AGG_TO_COP()` hint tells the optimizer to push down the aggregate operation in the specified query block to the coprocessor. If the optimizer does not push down some aggregate function that is suitable for pushdown, then it is recommended to use this hint. For example:

```
select /*+ AGG_TO_COP() */ sum(t1.a) from t t1;
```

LIMIT_TO_COP()

The `LIMIT_TO_COP()` hint tells the optimizer to push down the `Limit` and `TopN` operators in the specified query block to the coprocessor. If the optimizer does not perform such an operation, it is recommended to use this hint. For example:

```
SELECT /*+ LIMIT_TO_COP() */ * FROM t WHERE a = 1 AND b > 10 ORDER BY c
    ↪ LIMIT 1;
```

READ_FROM_STORAGE(TIFLASH[t1_name [, tl_name ...]], TIKV[t2_name [, tl_name ...]])

The `READ_FROM_STORAGE(TIFLASH[t1_name [, tl_name ...]], TIKV[t2_name [, tl_name ...]])` hint tells the optimizer to read specific table(s) from specific storage engine(s). Currently, this hint supports two storage engine parameters - `TIKV` and `TIFLASH`. If a table has an alias, use the alias as the parameter of `READ_FROM_STORAGE()`; if the table does not has an alias, use the table's original name as the parameter. For example:

```
select /*+ READ_FROM_STORAGE(TIFLASH[t1], TIKV[t2]) */ t1.a from t t1, t
    ↪ t2 where t1.a = t2.a;
```

USE_INDEX_MERGE(t1_name, idx1_name [, idx2_name ...])

The `USE_INDEX_MERGE(t1_name, idx1_name [, idx2_name ...])` hint tells the optimizer to access a specific table with the index merge method. The given list of indexes are optional parameters. If you explicitly specify the list, TiDB selects indexes from the list to build index merge; if you do not give the list of indexes, TiDB selects indexes from all available indexes to build index merge. For example:

```
SELECT /*+ USE_INDEX_MERGE(t1, idx_a, idx_b, idx_c) */ * FROM t1 WHERE t1.
    ↪ a > 10 OR t1.b > 10;
```

When multiple `USE_INDEX_MERGE` hints are made to the same table, the optimizer tries to select the index from the union of the index sets specified by these hints.

> **Note:**
>
> The parameters of `USE_INDEX_MERGE` refer to index names, rather than column names. The index name of the primary key is `primary`.

This hint takes effect on strict conditions, including:

- If the query can select a single index scan in addition to full table scan, the optimizer does not select index merge.

LEADING(t1_name [, tl_name ...])

The `LEADING(t1_name [, tl_name ...])` hint reminds the optimizer that, when generating the execution plan, to determine the order of multi-table joins according to the order of table names specified in the hint. For example:

```
SELECT /*+ LEADING(t1, t2) */ * FROM t1, t2, t3 WHERE t1.id = t2.id and t2.
    ↪ id = t3.id;
```

In the above query with multi-table joins, the order of joins is determined by the order of table names specified in the `LEADING()` hint. The optimizer will first join `t1` and `t2` and then join the result with `t3`. This hint is more general than `STRAIGHT_JOIN`.

The `LEADING` hint does not take effect in the following situations:

- Multiple `LEADING` hints are specified.
- The table name specified in the `LEADING` hint does not exist.
- A duplicated table name is specified in the `LEADING` hint.
- The optimizer cannot perform join operations according to the order as specified by the `LEADING` hint.
- The `straight_join()` hint already exists.
- The query contains an outer join.
- Any of the `MERGE_JOIN`, `INL_JOIN`, `INL_HASH_JOIN`, and `HASH_JOIN` hints is used at the same time.

In the above situations, a warning is generated.

```
-- Multiple `LEADING` hints are specified.
SELECT /*+ LEADING(t1, t2) LEADING(t3) */ * FROM t1, t2, t3 WHERE t1.id =
    ↪ t2.id and t2.id = t3.id;

-- To learn why the `LEADING` hint fails to take effect, execute `show
    ↪ warnings`.
SHOW WARNINGS;
```

```
+--
   ↪ -------+------+-----------------------------------------------------------------
   ↪
| Level   | Code | Message
   ↪
   ↪ |
+--
   ↪ -------+------+-----------------------------------------------------------------
   ↪
| Warning | 1815 | We can only use one leading hint at most, when multiple
   ↪ leading hints are used, all leading hints will be invalid |
+--
   ↪ -------+------+-----------------------------------------------------------------
   ↪
```

11.3.4.2.3 Hints that take effect in the whole query

This category of hints can only follow behind the **first** `SELECT`, `UPDATE` or `DELETE` keyword, which is equivalent to modifying the value of the specified system variable when this query is executed. The priority of the hint is higher than that of existing system variables.

> **Note:**
>
> This category of hints also has an optional hidden variable `@QB_NAME`, but the hint takes effect in the whole query even if you specify the variable.

NO_INDEX_MERGE()

The `NO_INDEX_MERGE()` hint disables the index merge feature of the optimizer.

For example, the following query will not use index merge:

```
select /*+ NO_INDEX_MERGE() */ * from t where t.a > 0 or t.b > 0;
```

In addition to this hint, setting the `tidb_enable_index_merge` system variable also controls whether to enable this feature.

> **Note:**
>
> - `NO_INDEX_MERGE` has a higher priority over `USE_INDEX_MERGE`. When both hints are used, `USE_INDEX_MERGE` does not take effect.

- For a subquery, `NO_INDEX_MERGE` only takes effect when it is placed at the outermost level of the subquery.

USE_TOJA(boolean_value)

The `boolean_value` parameter can be `TRUE` or `FALSE`. The `USE_TOJA(TRUE)` hint enables the optimizer to convert an `in` condition (containing a sub-query) to join and aggregation operations. Comparatively, the `USE_TOJA(FALSE)` hint disables this feature.

For example, the following query will convert `in (select t2.a from t2)subq` to corresponding join and aggregation operations:

```
select /*+ USE_TOJA(TRUE) */ t1.a, t1.b from t1 where t1.a in (select t2.a
    ↪ from t2) subq;
```

In addition to this hint, setting the `tidb_opt_insubq_to_join_and_agg` system variable also controls whether to enable this feature.

MAX_EXECUTION_TIME(N)

The `MAX_EXECUTION_TIME(N)` hint places a limit `N` (a timeout value in milliseconds) on how long a statement is permitted to execute before the server terminates it. In the following hint, `MAX_EXECUTION_TIME(1000)` means that the timeout is 1000 milliseconds (that is, 1 second):

```
select /*+ MAX_EXECUTION_TIME(1000) */ * from t1 inner join t2 where t1.id
    ↪  = t2.id;
```

In addition to this hint, the `global.max_execution_time` system variable can also limit the execution time of a statement.

MEMORY_QUOTA(N)

The `MEMORY_QUOTA(N)` hint places a limit `N` (a threshold value in MB or GB) on how much memory a statement is permitted to use. When a statement's memory usage exceeds this limit, TiDB produces a log message based on the statement's over-limit behavior or just terminates it.

In the following hint, `MEMORY_QUOTA(1024 MB)` means that the memory usage is limited to 1024 MB:

```
select /*+ MEMORY_QUOTA(1024 MB) */ * from t;
```

In addition to this hint, the `tidb_mem_quota_query` system variable can also limit the memory usage of a statement.

READ_CONSISTENT_REPLICA()

The `READ_CONSISTENT_REPLICA()` hint enables the feature of reading consistent data from the TiKV follower node. For example:

```
select /*+ READ_CONSISTENT_REPLICA() */ * from t;
```

In addition to this hint, setting the `tidb_replica_read` environment variable to `'follower'` or `'leader'` also controls whether to enable this feature.

IGNORE_PLAN_CACHE()

The `IGNORE_PLAN_CACHE()` hint reminds the optimizer not to use the Plan Cache when handling the current `prepare` statement.

This hint is used to temporarily disable the Plan Cache for a certain type of queries when prepare-plan-cache is enabled.

In the following example, the Plan Cache is forcibly disabled when executing the `prepare` statement.

```
prepare stmt from 'select /*+ IGNORE_PLAN_CACHE() */ * from t where t.id =
    ↪ ?';
```

STRAIGHT_JOIN()

The `STRAIGHT_JOIN()` hint reminds the optimizer to join tables in the order of table names in the `FROM` clause when generating the join plan.

```
SELECT /*+ STRAIGHT_JOIN() */ * FROM t t1, t t2 WHERE t1.a = t2.a;
```

Note:

- `STRAIGHT_JOIN` has higher priority over `LEADING`. When both hints are used, `LEADING` does not take effect.
- It is recommended to use the `LEADING` hint, which is more general than the `STRAIGHT_JOIN` hint.

NTH_PLAN(N)

The `NTH_PLAN(N)` hint reminds the optimizer to select the `N`th physical plan found during the physical optimization. `N` must be a positive integer.

If the specified `N` is beyond the search range of the physical optimization, TiDB will return a warning and select the optimal physical plan based on the strategy that ignores this hint.

This hint does not take effect when the cascades planner is enabled.

In the following example, the optimizer is forced to select the third physical plan found during the physical optimization:

```
SELECT /*+ NTH_PLAN(3) */ count(*) from t where a > 5;
```

> **Note:**
>
> `NTH_PLAN(N)` is mainly used for testing, and its compatibility is not guaranteed in later versions. Use this hint **with caution**.

11.3.4.2.4 Troubleshoot common issues that hints do not take effect

Hints do not take effect because your MySQL command-line client strips hints

MySQL command-line clients earlier than 5.7.7 strip optimizer hints by default. If you want to use the Hint syntax in these earlier versions, add the `--comments` option when starting the client. For example: `mysql -h 127.0.0.1 -P 4000 -uroot --comments`.

Hints do not take effect because the database name is not specified

If you do not specify the database name when creating a connection, hints might not take effect. For example:

When connecting to TiDB, you use the `mysql -h127.0.0.1 -P4000 -uroot` command without the `-D` option, and then execute the following SQL statements:

```
SELECT /*+ use_index(t, a) */ a FROM test.t;
SHOW WARNINGS;
```

Because TiDB cannot identify the database for table `t`, the `use_index(t, a)` hint does not take effect.

```
+--
   ↪ -------+------+-----------------------------------------------------------
   ↪
| Level   | Code | Message
   ↪                                                                |
+--
   ↪ -------+------+-----------------------------------------------------------
   ↪
| Warning | 1815 | use_index(.t, a) is inapplicable, check whether the
   ↪ table(.t) exists |
+--
   ↪ -------+------+-----------------------------------------------------------
   ↪
1 row in set (0.00 sec)
```

Hints do not take effect because the database name is not explicitly specified in cross-table queries

When executing cross-table queries, you need to explicitly specify database names. Otherwise, hints might not take effect. For example:

```
USE test1;
CREATE TABLE t1(a INT, KEY(a));
USE test2;
CREATE TABLE t2(a INT, KEY(a));
SELECT /*+ use_index(t1, a) */ * FROM test1.t1, t2;
SHOW WARNINGS;
```

In the preceding statements, because table `t1` is not in the current `test2` database, the `use_index(t1, a)` hint does not take effect.

```
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
| Level   | Code | Message
    ↪                                                                              |
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
| Warning | 1815 | use_index(test2.t1, a) is inapplicable, check whether
    ↪ the table(test2.t1) exists |
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

In this case, you need to specify the database name explicitly by using `use_index(test1
↪ .t1, a)` instead of `use_index(t1, a)`.

Hints do not take effect because they are placed in wrong locations

Hints cannot take effect if they are not placed directly after the specific keywords. For example:

```
SELECT * /*+ use_index(t, a) */ FROM t;
SHOW WARNINGS;
```

The warning is as follows:

```
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
| Level   | Code | Message
    ↪
    ↪ |
```

```
+--
    ↪ -------+------+-------------------------------------------------------------
    ↪
| Warning | 1064 | You have an error in your SQL syntax; check the manual
    ↪ that corresponds to your TiDB version for the right syntax to use [
    ↪ parser:8066]Optimizer hint can only be followed by certain keywords
    ↪ like SELECT, INSERT, etc. |
+--
    ↪ -------+------+-------------------------------------------------------------
    ↪
1 row in set (0.01 sec)
```

In this case, you need to place the hint directly after the `SELECT` keyword. For more details, see the Syntax section.

INL_JOIN hint does not take effect due to collation incompatibility

When the collation of the join key is incompatible between two tables, the `IndexJoin` operator cannot be utilized to execute the query. In this case, the `INL_JOIN` hint does not take effect. For example:

```
CREATE TABLE t1 (k varchar(8), key(k)) COLLATE=utf8mb4_general_ci;
CREATE TABLE t2 (k varchar(8), key(k)) COLLATE=utf8mb4_bin;
EXPLAIN SELECT /*+ tidb_inlj(t1) */ * FROM t1, t2 WHERE t1.k=t2.k;
```

The execution plan is as follows:

```
+--
    ↪ --------------------------+---------+----------+--------------------+------
    ↪
| id                        | estRows | task     | access object      |
    ↪ operator info                              |
+--
    ↪ --------------------------+---------+----------+--------------------+------
    ↪
| HashJoin_19               | 12487.50 | root    |                    | inner
    ↪ join, equal:[eq(test.t1.k, test.t2.k)] |
| ─IndexReader_24(Build)  | 9990.00 | root     |                     | index:
    ↪ IndexFullScan_23                  |
|   ─IndexFullScan_23     | 9990.00 | cop[tikv] | table:t2, index:k(k) |
    ↪ keep order:false, stats:pseudo    |
| ─IndexReader_22(Probe)  | 9990.00 | root     |                     | index:
    ↪ IndexFullScan_21                  |
|   ─IndexFullScan_21     | 9990.00 | cop[tikv] | table:t1, index:k(k) |
    ↪ keep order:false, stats:pseudo    |
+--
    ↪ --------------------------+---------+----------+--------------------+------
```

```
    ↪
5 rows in set, 1 warning (0.00 sec)
```

In the preceding statements, the collations of `t1.k` and `t2.k` are incompatible (`utf8mb4_general_ci` and `utf8mb4_bin` respectively), which prevents the `INL_JOIN` or `TIDB_INLJ` hint from taking effect.

```
SHOW WARNINGS;
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
| Level   | Code | Message
    ↪                                                                 |
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
| Warning | 1815 | Optimizer Hint /*+ INL_JOIN(t1) */ or /*+ TIDB_INLJ(t1)
    ↪ */ is inapplicable |
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

`INL_JOIN` hint does not take effect because of join order

The `INL_JOIN(t1, t2)` or `TIDB_INLJ(t1, t2)` hint semantically instructs `t1` and `t2` to act as inner tables in an `IndexJoin` operator to join with other tables, rather than directly joining them using an `IndexJoin`operator. For example:

```
EXPLAIN SELECT /*+ inl_join(t1, t3) */ * FROM t1, t2, t3 WHERE t1.id = t2.
    ↪ id AND t2.id = t3.id AND t1.id = t3.id;
+--
    ↪ ------------------------------+----------+-----------+---------------+---------
    ↪
| id                            | estRows | task     | access object | operator
    ↪  info
    ↪
    ↪ |
+--
    ↪ ------------------------------+----------+-----------+---------------+---------
    ↪
| IndexJoin_16                  | 15625.00 | root     |               | inner join
    ↪ , inner:TableReader_13, outer key:test.t2.id, test.t1.id, inner key:
    ↪ test.t3.id, test.t3.id, equal cond:eq(test.t1.id, test.t3.id), eq(
    ↪ test.t2.id, test.t3.id) |
| ─IndexJoin_34(Build)          | 12500.00 | root     |               | inner
    ↪ join, inner:TableReader_31, outer key:test.t2.id, inner key:test.t1.
```

```
    ↪ id, equal cond:eq(test.t2.id, test.t1.id)
    ↪                                  |
|    ─TableReader_40(Build)    | 10000.00 | root    |              | data:
    ↪ TableFullScan_39
    ↪
    ↪ |
|      ─TableFullScan_39       | 10000.00 | cop[tikv] | table:t2 | keep
    ↪ order:false, stats:pseudo
    ↪
    ↪ |
|    ─TableReader_31(Probe)    | 10000.00 | root    |              | data:
    ↪ TableRangeScan_30
    ↪
    ↪ |
|      ─TableRangeScan_30      | 10000.00 | cop[tikv] | table:t1 | range:
    ↪ decided by [test.t2.id], keep order:false, stats:pseudo
    ↪
    ↪ |
|  ─TableReader_13(Probe)      | 12500.00 | root    |              | data:
    ↪ TableRangeScan_12
    ↪
    ↪ |
|    ─TableRangeScan_12        | 12500.00 | cop[tikv] | table:t3 | range:
    ↪ decided by [test.t2.id test.t1.id], keep order:false, stats:pseudo
    ↪
    ↪ |
+--
    ↪ ------------------------------+----------+-----------+--------------+---------
    ↪
```

In the preceding example, `t1` and `t3` are not directly joined together by an `IndexJoin`.

To perform a direct `IndexJoin` between `t1` and `t3`, you can first use `LEADING(t1, t3)` hint to specify the join order of `t1` and `t3`, and then use the `INL_JOIN` hint to specify the join algorithm. For example:

```
EXPLAIN SELECT /*+ leading(t1, t3), inl_join(t3) */ * FROM t1, t2, t3 WHERE
    ↪  t1.id = t2.id AND t2.id = t3.id AND t1.id = t3.id;
+--
    ↪ ------------------------------+----------+-----------+--------------+---------
    ↪
| id                           | estRows | task    | access object | operator
    ↪  info
    ↪
    ↪ |
+--
```

```
    ↪ ------------------------------+----------+-----------+--------------+---------
    ↪
| Projection_12                | 15625.00 | root      |              | test.t1.id
    ↪ , test.t1.name, test.t2.id, test.t2.name, test.t3.id, test.t3.name
    ↪                            |
|  ─HashJoin_21                | 15625.00 | root      |              | inner
    ↪ join, equal:[eq(test.t1.id, test.t2.id) eq(test.t3.id, test.t2.id)]
    ↪                              |
|    ─TableReader_36(Build)    | 10000.00 | root      |              | data:
    ↪ TableFullScan_35
    ↪
    ↪ |
|      ─TableFullScan_35       | 10000.00 | cop[tikv] | table:t2 | keep
    ↪ order:false, stats:pseudo
    ↪
    ↪ |
|    ─IndexJoin_28(Probe)      | 12500.00 | root      |              | inner
    ↪ join, inner:TableReader_25, outer key:test.t1.id, inner key:test.t3.
    ↪ id, equal cond:eq(test.t1.id, test.t3.id) |
|      ─TableReader_34(Build)  | 10000.00 | root      |              | data:
    ↪ TableFullScan_33
    ↪
    ↪ |
|        ─TableFullScan_33     | 10000.00 | cop[tikv] | table:t1 | keep
    ↪ order:false, stats:pseudo
    ↪
    ↪ |
|      ─TableReader_25(Probe)  | 10000.00 | root      |              | data:
    ↪ TableRangeScan_24
    ↪
    ↪ |
|        ─TableRangeScan_24    | 10000.00 | cop[tikv] | table:t3 | range:
    ↪ decided by [test.t1.id], keep order:false, stats:pseudo
    ↪                                        |
+--
    ↪ ------------------------------+----------+-----------+--------------+---------
    ↪
9 rows in set (0.01 sec)
```

11.3.4.3 SQL Plan Management (SPM)

SQL Plan Management is a set of functions that execute SQL bindings to manually interfere with SQL execution plans. These functions include SQL binding, baseline capturing, and baseline evolution.

11.3.4.3.1 SQL binding

An SQL binding is the basis of SPM. The Optimizer Hints document introduces how to select a specific execution plan using hints. However, sometimes you need to interfere with execution selection without modifying SQL statements. With SQL bindings, you can select a specified execution plan without modifying SQL statements.

> **Note:**
>
> To use SQL bindings, you need to have the `SUPER` privilege. If TiDB prompts that you do not have sufficient privileges, see Privilege Management to add the required privileges.

Create a binding

```
CREATE [GLOBAL | SESSION] BINDING FOR BindableStmt USING BindableStmt
```

This statement binds SQL execution plans at the GLOBAL or SESSION level. Currently, supported bindable SQL statements (BindableStmt) in TiDB include `SELECT`, `DELETE`, `UPDATE`, and `INSERT` / `REPLACE` with `SELECT` subqueries.

> **Note:**
>
> Bindings have higher priority over manually added hints. Therefore, when you execute a statement containing a hint while a corresponding binding is present, the hint controlling the behavior of the optimizer does not take effect. However, other types of hints are still effective.

Specifically, two types of these statements cannot be bound to execution plans due to syntax conflicts. See the following examples:

```
-- Type one: Statements that get the Cartesian product by using the `JOIN`
   ↪  keyword and not specifying the associated columns with the `USING`
   ↪ keyword.
CREATE GLOBAL BINDING for
    SELECT * FROM t t1 JOIN t t2
USING
    SELECT * FROM t t1 JOIN t t2;

-- Type two: `DELETE` statements that contain the `USING` keyword.
CREATE GLOBAL BINDING for
```

```
    DELETE FROM t1 USING t1 JOIN t2 ON t1.a = t2.a
USING
    DELETE FROM t1 USING t1 JOIN t2 ON t1.a = t2.a;
```

You can bypass syntax conflicts by using equivalent statements. For example, you can rewrite the above statements in the following ways:

```
-- First rewrite of type one statements: Add a `USING` clause for the `
    ↪ JOIN` keyword.
CREATE GLOBAL BINDING for
    SELECT * FROM t t1 JOIN t t2 USING (a)
USING
    SELECT * FROM t t1 JOIN t t2 USING (a);

-- Second rewrite of type one statements: Delete the `JOIN` keyword.
CREATE GLOBAL BINDING for
    SELECT * FROM t t1, t t2
USING
    SELECT * FROM t t1, t t2;

-- Rewrite of type two statements: Remove the `USING` keyword from the `
    ↪ delete` statement.
CREATE GLOBAL BINDING for
    DELETE t1 FROM t1 JOIN t2 ON t1.a = t2.a
using
    DELETE t1 FROM t1 JOIN t2 ON t1.a = t2.a;
```

Note:

When creating execution plan bindings for `INSERT` / `REPLACE` statements with `SELECT` subqueries, you need to specify the optimizer hints you want to bind in the `SELECT` subquery, not after the `INSERT` / `REPLACE` keyword. Otherwise, the optimizer hints do not take effect as intended.

Here are two examples:

```
-- The hint takes effect in the following statement.
CREATE GLOBAL BINDING for
    INSERT INTO t1 SELECT * FROM t2 WHERE a > 1 AND b = 1
using
    INSERT INTO t1 SELECT /*+ use_index(@sel_1 t2, a) */ * FROM t2 WHERE a
        ↪ > 1 AND b = 1;
```

```
-- The hint cannot take effect in the following statement.
CREATE GLOBAL BINDING for
    INSERT INTO t1 SELECT * FROM t2 WHERE a > 1 AND b = 1
using
    INSERT /*+ use_index(@sel_1 t2, a) */ INTO t1 SELECT * FROM t2 WHERE a
        ↪ > 1 AND b = 1;
```

If you do not specify the scope when creating an execution plan binding, the default scope is SESSION. The TiDB optimizer normalizes bound SQL statements and stores them in the system table. When processing SQL queries, if a normalized statement matches one of the bound SQL statements in the system table and the system variable `tidb_use_plan_baselines` is set to `on` (the default value is `on`), TiDB then uses the corresponding optimizer hint for this statement. If there are multiple matchable execution plans, the optimizer chooses the least costly one to bind.

`Normalization` is a process that converts a constant in an SQL statement to a variable parameter and explicitly specifies the database for tables referenced in the query, with standardized processing on the spaces and line breaks in the SQL statement. See the following example:

```
SELECT * FROM t WHERE a > 1
-- After normalization, the above statement is as follows:
SELECT * FROM test . t WHERE a > ?
```

Note:

Multiple constants joined by commas `,` are normalized as `...` instead of `?`.

For example:

```
SELECT * FROM t limit 10
SELECT * FROM t limit 10, 20
SELECT * FROM t WHERE a IN (1)
SELECT * FROM t WHERE a IN (1,2,3)
-- After normalization, the above statements are as follows:
SELECT * FROM test . t limit ?
SELECT * FROM test . t limit ...
SELECT * FROM test . t WHERE a IN ( ? )
SELECT * FROM test . t WHERE a IN ( ... )
```

When bindings are created, TiDB treats SQL statements that contain a single constant and SQL statements that contain multiple constants joined by commas differently. Therefore, you need to create bindings for the two SQL types separately.

For example:

```
CREATE TABLE t(a INT, b INT, KEY idx(a));
CREATE SESSION BINDING for SELECT * FROM t WHERE a IN (?)
   ↪ USING SELECT /*+ use_index(t, idx) */ * FROM t WHERE a
   ↪ in (?);
SHOW BINDINGS;
+--
   ↪ ----------------------------------------------+-------------------------
   ↪
| Original_sql                                  | Bind_sql
   ↪                                                                    |
   ↪ Default_db | Status | Create_time        | Update_time
   ↪          | Charset | Collation      | Source | Sql_digest
   ↪                                                  |
   ↪ Plan_digest |
+--
   ↪ ----------------------------------------------+-------------------------
   ↪
| SELECT * FROM `test` . `t` WHERE `a` IN ( ? ) | SELECT /*+
   ↪ use_index(`t` `idx`)*/ * FROM `test`.`t` WHERE `a` IN
   ↪ (?) | test | enabled | 2023-08-23 14:15:31.472 |
   ↪ 2023-08-23 14:15:31.472 | utf8mb4 | utf8mb4_general_ci |
   ↪ manual | 8
   ↪ b9c4e6ab8fad5ba29b034311dcbfc8a8ce57dde2e2d5d5b65313b90ebcdebf7
   ↪  | |
+--
   ↪ ----------------------------------------------+-------------------------
   ↪
SELECT * FROM t WHERE a IN (1);
SELECT @@LAST_PLAN_FROM_BINDING;
+--------------------------+
| @@LAST_PLAN_FROM_BINDING |
+--------------------------+
|                        1 |
+--------------------------+
SELECT * FROM t WHERE a IN (1,2);
SELECT @@LAST_PLAN_FROM_BINDING;
+--------------------------+
| @@LAST_PLAN_FROM_BINDING |
+--------------------------+
|                        0 |
+--------------------------+
CREATE SESSION BINDING for SELECT * FROM t WHERE a IN (?,?)
   ↪ USING SELECT /*+ use_index(t, idx) */ * FROM t WHERE a
```

```
    ↪ IN (?,?);
show bindings;
+--
    ↪ ---------------------------------------------+-----------------------
    ↪
| Original_sql                                  | Bind_sql
    ↪                                                                    |
    ↪ Default_db | Status | Create_time         | Update_time
    ↪          | Charset | Collation       | Source | Sql_digest
    ↪                                                  |
    ↪ Plan_digest |
+--
    ↪ ---------------------------------------------+-----------------------
    ↪
| SELECT * FROM `test` . `t` WHERE `a` IN ( ... ) | SELECT /*+
    ↪ use_index(`t` `idx`)*/ * FROM `test`.`t` WHERE `a` IN
    ↪ (?,?) | test | enabled | 2023-08-23 14:16:30.762 |
    ↪ 2023-08-23 14:16:30.762 | utf8mb4 | utf8mb4_general_ci |
    ↪ manual |
    ↪ da38bf216db4a53e1a1e01c79ffa42306419442ad7238480bb7ac510723c8bdf
    ↪  | |
| SELECT * FROM `test` . `t` WHERE `a` IN ( ? ) | SELECT /*+
    ↪ use_index(`t` `idx`)*/ * FROM `test`.`t` WHERE `a` IN
    ↪ (?) | test | enabled | 2023-08-23 14:15:31.472 |
    ↪ 2023-08-23 14:15:31.472 | utf8mb4 | utf8mb4_general_ci |
    ↪ manual | 8
    ↪ b9c4e6ab8fad5ba29b034311dcbfc8a8ce57dde2e2d5d5b65313b90ebcdebf7
    ↪  | |
+--
    ↪ ---------------------------------------------+-----------------------
    ↪
SELECT * FROM t WHERE a IN (1,2);
SELECT @@LAST_PLAN_FROM_BINDING;
+--------------------------+
| @@LAST_PLAN_FROM_BINDING |
+--------------------------+
|                        1 |
+--------------------------+
SELECT * FROM t WHERE a IN (1,2,3);
SELECT @@LAST_PLAN_FROM_BINDING;
+--------------------------+
| @@LAST_PLAN_FROM_BINDING |
+--------------------------+
```

```
|                     1 |
+-----------------------+
DROP SESSION BINDING for SELECT * FROM t WHERE a IN (?);
SHOW BINDINGS;
+--
   ↪ ------------------------------------------------+-----------------------
   ↪
| Original_sql                                  | Bind_sql
   ↪                                                                       |
   ↪ Default_db | Status | Create_time        | Update_time
   ↪          | Charset | Collation      | Source | Sql_digest
   ↪                                                    |
   ↪ Plan_digest |
+--
   ↪ ------------------------------------------------+-----------------------
   ↪
| SELECT * FROM `test` . `t` WHERE `a` IN ( ... ) | SELECT /*+
   ↪ use_index(`t` `idx`)*/ * FROM `test`.`t` WHERE `a` IN
   ↪ (?,?) | test | enabled | 2023-08-23 14:16:30.762 |
   ↪ 2023-08-23 14:16:30.762 | utf8mb4 | utf8mb4_general_ci |
   ↪ manual |
   ↪ da38bf216db4a53e1a1e01c79ffa42306419442ad7238480bb7ac510723c8bdf
   ↪  | |
+--
   ↪ ------------------------------------------------+-----------------------
   ↪
SELECT * FROM t WHERE a IN (1);
SELECT @@LAST_PLAN_FROM_BINDING;
+--------------------------+
| @@LAST_PLAN_FROM_BINDING |
+--------------------------+
|                        0 |
+--------------------------+
```

When a SQL statement has bound execution plans in both GLOBAL and SESSION scopes, because the optimizer ignores the bound execution plan in the GLOBAL scope when it encounters the SESSION binding, the bound execution plan of this statement in the SESSION scope shields the execution plan in the GLOBAL scope.

For example:

```
-- Creates a GLOBAL binding and specifies using `sort merge join` in this
   ↪ binding.
```

```
CREATE GLOBAL BINDING for
    SELECT * FROM t1, t2 WHERE t1.id = t2.id
USING
    SELECT /*+ merge_join(t1, t2) */ * FROM t1, t2 WHERE t1.id = t2.id;

-- The execution plan of this SQL statement uses the `sort merge join`
    ↪ specified in the GLOBAL binding.
explain SELECT * FROM t1, t2 WHERE t1.id = t2.id;

-- Creates another SESSION binding and specifies using `hash join` in this
    ↪  binding.
CREATE BINDING for
    SELECT * FROM t1, t2 WHERE t1.id = t2.id
USING
    SELECT /*+ hash_join(t1, t2) */ * FROM t1, t2 WHERE t1.id = t2.id;

-- In the execution plan of this statement, `hash join` specified in the
    ↪ SESSION binding is used, instead of `sort merge join` specified in
    ↪ the GLOBAL binding.
explain SELECT * FROM t1, t2 WHERE t1.id = t2.id;
```

When the first `SELECT` statement is being executed, the optimizer adds the `sm_join(` ↪ `t1, t2)` hint to the statement through the binding in the GLOBAL scope. The top node of the execution plan in the `explain` result is MergeJoin. When the second `SELECT` statement is being executed, the optimizer uses the binding in the SESSION scope instead of the binding in the GLOBAL scope and adds the `hash_join(t1, t2)` hint to the statement. The top node of the execution plan in the `explain` result is HashJoin.

Each standardized SQL statement can have only one binding created using `CREATE` ↪ `BINDING` at a time. When multiple bindings are created for the same standardized SQL statement, the last created binding is retained, and all previous bindings (created and evolved) are marked as deleted. But session bindings and global bindings can coexist and are not affected by this logic.

In addition, when you create a binding, TiDB requires that the session is in a database context, which means that a database is specified when the client is connected or `use ${` ↪ `database}` is executed.

The original SQL statement and the bound statement must have the same text after normalization and hint removal, or the binding will fail. Take the following examples:

- This binding can be created successfully because the texts before and after parameterization and hint removal are the same: `SELECT * FROM test . t WHERE a > ?`

  ```
  sql  CREATE BINDING FOR SELECT * FROM t WHERE a > 1 USING SELECT * FROM
  ↪  t use index (idx)WHERE a > 2
  ```

- This binding will fail because the original SQL statement is processed as `SELECT * FROM test . t WHERE a > ?`, while the bound SQL statement is processed differently as `SELECT * FROM test . t WHERE b > ?`.

  ```
  sql CREATE BINDING FOR SELECT * FROM t WHERE a > 1 USING SELECT * FROM t use index(idx)WHERE b > 2
  ```

> **Note:**
>
> For `PREPARE` / `EXECUTE` statements and for queries executed with binary protocols, you need to create execution plan bindings for the real query statements, not for the `PREPARE` / `EXECUTE` statements.

Remove binding

```
DROP [GLOBAL | SESSION] BINDING FOR BindableStmt;
```

This statement removes a specified execution plan binding at the GLOBAL or SESSION level. The default scope is SESSION.

Generally, the binding in the SESSION scope is mainly used for test or in special situations. For a binding to take effect in all TiDB processes, you need to use the GLOBAL binding. A created SESSION binding shields the corresponding GLOBAL binding until the end of the SESSION, even if the SESSION binding is dropped before the session closes. In this case, no binding takes effect and the plan is selected by the optimizer.

The following example is based on the example in create binding in which the SESSION binding shields the GLOBAL binding:

```
-- Drops the binding created in the SESSION scope.
drop session binding for SELECT * FROM t1, t2 WHERE t1.id = t2.id;

-- Views the SQL execution plan again.
explain SELECT * FROM t1,t2 WHERE t1.id = t2.id;
```

In the example above, the dropped binding in the SESSION scope shields the corresponding binding in the GLOBAL scope. The optimizer does not add the `sm_join(t1, t2)` hint to the statement. The top node of the execution plan in the `explain` result is not fixed to MergeJoin by this hint. Instead, the top node is independently selected by the optimizer according to the cost estimation.

> **Note:**

Executing `DROP GLOBAL BINDING` drops the binding in the current tidb-server instance cache and changes the status of the corresponding row in the system table to 'deleted'. This statement does not directly delete the records in the system table, because other tidb-server instances need to read the 'deleted' status to drop the corresponding binding in their cache. For the records in these system tables with the status of 'deleted', at every 100 `bind-info-lease` (the default value is `3s`, and `300s` in total) interval, the background thread triggers an operation of reclaiming and clearing on the bindings of `update_time` before 10 `bind-info-lease` (to ensure that all tidb-server instances have read the 'deleted' status and updated the cache).

11.3.4.3.2 Change binding status

```
SET BINDING [ENABLED | DISABLED] FOR BindableStmt;
```

You can execute this statement to change the status of a binding. The default status is ENABLED. The effective scope is GLOBAL by default and cannot be modified.

When executing this statement, you can only change the status of a binding from `Disabled` to `Enabled` or from `Enabled` to `Disabled`. If no binding is available for status changes, a warning message is returned, saying `There are no bindings can be set the status. Please check the SQL text`. Note that a binding in `Disabled` status is not used by any query.

View bindings

```
SHOW [GLOBAL | SESSION] BINDINGS [ShowLikeOrWhere]
```

This statement outputs the execution plan bindings at the GLOBAL or SESSION level according to the order of binding update time from the latest to earliest. The default scope is SESSION. Currently `SHOW BINDINGS` outputs eight columns, as shown below:

Column Name	Note
original_sql	Original SQL statement after parameterization
bind_sql	Bound SQL statement with hints
default_db	Default database

Column Name	Note
status	Status including `enabled` (replacing the `using` status from v6.0), `disabled`, `deleted`, `invalid`, `rejected`, and `pending` ↪ `verify`
create_time	Creating time
update_time	Updating time
charset	Character set
collation	Ordering rule
source	The way in which a binding is created, including `manual` (created by the `create` ↪ `[global]` ↪ `binding` SQL statement), `capture` (captured automatically by TiDB), and `evolve` (evolved automatically by TiDB)

Troubleshoot a binding

You can use either of the following methods to troubleshoot a binding:

- Use the system variable `last_plan_from_binding` to show whether the execution plan used by the last executed statement is from the binding.

```
-- Create a global binding
CREATE GLOBAL BINDING for
    SELECT * FROM t
```

```
USING
    SELECT /*+ USE_INDEX(t, idx_a) */ * FROM t;

SELECT * FROM t;
SELECT @@[SESSION.]last_plan_from_binding;
```

```
+--------------------------+
| @@last_plan_from_binding |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.00 sec)
```

- Use the `explain format = 'verbose'` statement to view the query plan of a SQL statement. If the SQL statement uses a binding, you can run `show warnings` to check which binding is used in the SQL statement.

```
-- Create a global binding

CREATE GLOBAL BINDING for
    SELECT * FROM t
USING
    SELECT /*+ USE_INDEX(t, idx_a) */ * FROM t;

-- Use explain format = 'verbose' to view the execution plan of a SQL
    ↪  statement

explain format = 'verbose' SELECT * FROM t;

-- Run `show warnings` to view the binding used in the query.

show warnings;
```

```
+--
    ↪ -----+------+-----------------------------------------------------------
    ↪
| Level | Code | Message
    ↪                                                                      |
+--
    ↪ -----+------+-----------------------------------------------------------
    ↪
| Note | 1105 | Using the bindSQL: SELECT /*+ USE_INDEX(`t` `idx_a`)*/
    ↪  * FROM `test`.`t` |
+--
    ↪ -----+------+-----------------------------------------------------------
```

```
    ↪
1 row in set (0.01 sec)
```

Cache bindings

Each TiDB instance has a least recently used (LRU) cache for bindings. The cache capacity is controlled by the system variable `tidb_mem_quota_binding_cache`. You can view bindings that are cached in the TiDB instance.

To view the cache status of bindings, run the `SHOW binding_cache status` statement. In this statement, the effective scope is GLOBAL by default and cannot be modified. This statement returns the number of available bindings in the cache, the total number of available bindings in the system, memory usage of all cached bindings, and the total memory for the cache.

```
SHOW binding_cache status;
```

```
+------------------+------------------+-------------+-------------+
| bindings_in_cache | bindings_in_table | memory_usage | memory_quota |
+------------------+------------------+-------------+-------------+
|                1 |                1 | 159 Bytes   | 64 MB       |
+------------------+------------------+-------------+-------------+
1 row in set (0.00 sec)
```

11.3.4.4.3.3 Baseline capturing

Used for preventing regression of execution plans during an upgrade, this feature captures queries that meet capturing conditions and creates bindings for these queries.

Enable capturing

To enable baseline capturing, set `tidb_capture_plan_baselines` to `on`. The default value is `off`.

> **Note:**
>
> Because the automatic binding creation function relies on Statement Summary, make sure to enable Statement Summary before using automatic binding.

After automatic binding creation is enabled, the historical SQL statements in the Statement Summary are traversed every `bind-info-lease` (the default value is `3s`), and a binding

is automatically created for SQL statements that appear at least twice. For these SQL statements, TiDB automatically binds the execution plan recorded in Statement Summary.

However, TiDB does not automatically capture bindings for the following types of SQL statements:

- `EXPLAIN` and `EXPLAIN ANALYZE` statements.
- SQL statements executed internally in TiDB, such as `SELECT` queries used for automatically loading statistical information.
- Statements that contain `Enabled` or `Disabled` bindings.
- Statements that are filtered out by capturing conditions.

> **Note:**
>
> Currently, a binding generates a group of hints to fix an execution plan generated by a query statement. In this way, for the same query, the execution plan does not change. For most OLTP queries, including queries using the same index or Join algorithm (such as HashJoin and IndexJoin), TiDB guarantees plan consistency before and after the binding. However, due to the limitations of hints, TiDB cannot guarantee plan consistency for some complex queries, such as Join of more than two tables, MPP queries, and complex OLAP queries.

For `PREPARE` / `EXECUTE` statements and for queries executed with binary protocols, TiDB automatically captures bindings for the real query statements, not for the `PREPARE` / `EXECUTE` statements.

> **Note:**
>
> Because TiDB has some embedded SQL statements to ensure the correctness of some features, baseline capturing by default automatically shields these SQL statements.

Filter out bindings

This feature allows you to configure a blocklist to filter out queries whose bindings you do not want to capture. A blocklist has three dimensions, table name, frequency, and user name.

Usage

Insert filtering conditions into the system table `mysql.capture_plan_baselines_blacklist` ↪ . Then the filtering conditions take effect in the entire cluster immediately.

```
-- Filter by table name
 INSERT INTO mysql.capture_plan_baselines_blacklist(filter_type,
    ↪ filter_value) VALUES('table', 'test.t');

-- Filter by database name and table name through wildcards
 INSERT INTO mysql.capture_plan_baselines_blacklist(filter_type,
    ↪ filter_value) VALUES('table', 'test.table_*');
 INSERT INTO mysql.capture_plan_baselines_blacklist(filter_type,
    ↪ filter_value) VALUES('table', 'db_*.table_*');

-- Filter by frequency
 INSERT INTO mysql.capture_plan_baselines_blacklist(filter_type,
    ↪ filter_value) VALUES('frequency', '2');

-- Filter by user name
 INSERT INTO mysql.capture_plan_baselines_blacklist(filter_type,
    ↪ filter_value) VALUES('user', 'user1');
```

Dimension name	Description	Remarks
table	Filter by table name. Each filtering rule is in the `db.table` format. The supported filtering syntax includes Plain table names and Wildcards.	Case insensitive. If a table name contains illegal characters, the log returns a warning message `[sql-bind] failed to load mysql.capture_plan_baselines_blacklist .`
frequency	Filter by frequency. SQL statements executed more than once are captured by default. You can set a high frequency to capture statements that are frequently executed.	Setting frequency to a value smaller than 1 is considered illegal, and the log returns a warning message `[sql-bind] frequency threshold is less than 1, ignore it`. If multiple frequency filter rules are inserted, the value with the highest frequency prevails.
user	Filter by user name. Statements executed by blocklisted users are not captured.	If multiple users execute the same statement and their user names are all in the blocklist, this statement is not captured.

Note:

- Modifying a blocklist requires the super privilege.
- If a blocklist contains illegal filters, TiDB returns the warning message `[sql-bind] unknown capture filter type, ignore it` in the log.

Prevent regression of execution plans during an upgrade

Before upgrading a TiDB cluster, you can use baseline capturing to prevent regression of execution plans by performing the following steps:

1. Enable baseline capturing and keep it working.

> **Note:**
> Test data shows that long-term working of baseline capturing has a slight impact on the performance of the cluster load. Therefore, it is recommended to enable baseline capturing as long as possible so that important plans (appear twice or above) are captured.

2. Upgrade the TiDB cluster. After the upgrade, TiDB uses those captured bindings to ensure execution plan consistency.
3. After the upgrade, delete bindings as required.

    - Check the binding source by running the `SHOW GLOBAL BINDINGS` statement.

      In the output, check the `Source` field to see whether a binding is captured (`capture`) or manually created (`manual`).
    - Determine whether to retain the captured bindings:

```
-- View the plan with the binding enabled
SET @@SESSION.TIDB_USE_PLAN_BASELINES = true;
EXPLAIN FORMAT='VERBOSE' SELECT * FROM t1 WHERE ...;

-- View the plan with the binding disabled
SET @@SESSION.TIDB_USE_PLAN_BASELINES = false;
EXPLAIN FORMAT='VERBOSE' SELECT * FROM t1 WHERE ...;
```

        - If the execution plan is consistent, you can delete the binding safely.
        - If the execution plan is inconsistent, you need to identify the cause, for example, by checking statistics. In this case, you need to retain the binding to ensure plan consistency.

11.3.4.3.4 Baseline evolution

Baseline evolution is an important feature of SPM introduced in TiDB v4.0.

As data updates, the previously bound execution plan might no longer be optimal. The baseline evolution feature can automatically optimize the bound execution plan.

In addition, baseline evolution, to a certain extent, can also avoid the jitter brought to the execution plan caused by the change of statistical information.

Usage

Use the following statement to enable automatic binding evolution:

```
SET GLOBAL tidb_evolve_plan_baselines = ON;
```

The default value of `tidb_evolve_plan_baselines` is `off`.

> **Warning:**
>
> - Baseline evolution is an experimental feature. Unknown risks might exist. It is **NOT** recommended that you use it in the production environment.
> - This variable is forcibly set to `off` until the baseline evolution feature becomes generally available (GA). If you try to enable this feature, an error is returned. If you have already used this feature in a production environment, disable it as soon as possible. If you find that the binding status is not as expected, get support from PingCAP or the community.

After the automatic binding evolution feature is enabled, if the optimal execution plan selected by the optimizer is not among the binding execution plans, the optimizer marks the plan as an execution plan that waits for verification. At every `bind-info-lease` (the default value is `3s`) interval, an execution plan to be verified is selected and compared with the binding execution plan that has the least cost in terms of the actual execution time. If the plan to be verified has shorter execution time (the current criterion for the comparison is that the execution time of the plan to be verified is no longer than 2/3 that of the binding execution plan), this plan is marked as a usable binding. The following example describes the process above.

Assume that table `t` is defined as follows:

```
CREATE TABLE t(a INT, b INT, KEY(a), KEY(b));
```

Perform the following query on table `t`:

```
SELECT * FROM t WHERE a < 100 AND b < 100;
```

In the table defined above, few rows meet the `a < 100` condition. But for some reason, the optimizer mistakenly selects the full table scan instead of the optimal execution plan that uses index `a`. You can first use the following statement to create a binding:

```
CREATE GLOBAL BINDING for SELECT * FROM t WHERE a < 100 AND b < 100 using
    ↪ SELECT * FROM t use index(a) WHERE a < 100 AND b < 100;
```

When the query above is executed again, the optimizer selects index `a` (influenced by the binding created above) to reduce the query time.

Assuming that as insertions and deletions are performed on table `t`, an increasing number of rows meet the `a < 100` condition and a decreasing number of rows meet the `b < 100` condition. At this time, using index `a` under the binding might no longer be the optimal plan.

The binding evolution can address this kind of issues. When the optimizer recognizes data change in a table, it generates an execution plan for the query that uses index `b`. However, because the binding of the current plan exists, this query plan is not adopted and executed. Instead, this plan is stored in the backend evolution list. During the evolution process, if this plan is verified to have an obviously shorter execution time than that of the current execution plan that uses index `a`, index `b` is added into the available binding list. After this, when the query is executed again, the optimizer first generates the execution plan that uses index `b` and makes sure that this plan is in the binding list. Then the optimizer adopts and executes this plan to reduce the query time after data changes.

To reduce the impact that the automatic evolution has on clusters, use the following configurations:

- Set `tidb_evolve_plan_task_max_time` to limit the maximum execution time of each execution plan. The default value is `600s`. In the actual verification process, the maximum execution time is also limited to no more than twice the time of the verified execution plan.
- Set `tidb_evolve_plan_task_start_time` (`00:00 +0000` by default) and `tidb_evolve_plan_task_e`
  ↪ (`23:59 +0000` by default) to limit the time window.

Notes

Because the baseline evolution automatically creates a new binding, when the query environment changes, the automatically created binding might have multiple behavior choices. Pay attention to the following notes:

- Baseline evolution only evolves standardized SQL statements that have at least one global binding.
- Because creating a new binding deletes all previous bindings (for a standardized SQL statement), the automatically evolved binding will be deleted after manually creating a new binding.

- All hints related to the calculation process are retained during the evolution. These hints are as follows:

Hint	Description
`memory_quota` ↪	The maximum memory that can be used for a query.
`use_toja`	Whether the optimizer transforms sub-queries to Join.
`use_cascades` ↪	Whether to use the cascades optimizer.
`no_index_merge` ↪	Whether the optimizer uses Index Merge as an option for reading tables.
`read_consistent_replica` ↪	Whether to forcibly enable Follower Read when reading tables.
`max_execution_time` ↪	The longest duration for a query.

- `read_from_storage` is a special hint in that it specifies whether to read data from TiKV or from TiFlash when reading tables. Because TiDB provides isolation reads, when the isolation condition changes, this hint has a great influence on the evolved execution plan. Therefore, when this hint exists in the initially created binding, TiDB ignores all its evolved bindings.

11.3.4.3.5 Upgrade checklist

During cluster upgrade, SQL Plan Management (SPM) might cause compatibility issues and make the upgrade fail. To ensure a successful upgrade, you need to include the following list for upgrade precheck:

- When you upgrade from a version earlier than v5.2.0 (that is, v4.0, v5.0, and v5.1) to the current version, make sure that `tidb_evolve_plan_baselines` is disabled before the upgrade. To disable this variable, perform the following steps.

```
-- Check whether `tidb_evolve_plan_baselines` is disabled in the
   ↪ earlier version.
```

```
SELECT @@global.tidb_evolve_plan_baselines;

-- If `tidb_evolve_plan_baselines` is still enabled, disable it.

SET GLOBAL tidb_evolve_plan_baselines = OFF;
```

- Before you upgrade from v4.0 to the current version, you need to check whether the syntax of all queries corresponding to the available SQL bindings is correct in the new version. If any syntax errors exist, delete the corresponding SQL binding. To do that, perform the following steps.

```
-- Check the query corresponding to the available SQL binding in the
   ↪ version to be upgraded.

SELECT bind_sql FROM mysql.bind_info WHERE status = 'using';

-- Verify the result from the above SQL query in the test environment
   ↪  of the new version.

bind_sql_0;
bind_sql_1;
...

-- In the case of a syntax error (ERROR 1064 (42000): You have an
   ↪ error in your SQL syntax), delete the corresponding binding.
-- For any other errors (for example, tables are not found), it means
   ↪  that the syntax is compatible. No other operation is needed.
```

11.3.4.4 The Blocklist of Optimization Rules and Expression Pushdown

This document introduces how to use the blocklist of optimization rules and the blocklist of expression pushdown to control the behavior of TiDB.

11.3.4.4.1 The blocklist of optimization rules

The blocklist of optimization rules is one way to tune optimization rules, mainly used to manually disable some optimization rules.

Important optimization rules

Optimization Rule	Rule Name	Description
Column pruning	column_prune	Prune operator will prune the column if it is not needed by the upper executor.
Decorrelate subquery	decorrelate	Tries to rewrite the correlated subquery to non-correlated join or aggregation.

Optimization Rule	Rule Name	Description
Aggregation elimination	aggregation_eliminate	To remove unnecessary aggregation operators from the execution plan.
Projection elimination	projection_eliminate	To remove unnecessary projection operators from the execution plan.

Optimization Rule	Rule Name	Description
Max/Min elimination	max_min_eliminate	Revelimnate some max/min functions in aggregation to the `order` ↪ ↪ `by` ↪ + `limit` ↪ ↪ `1` form.
Predicate push-down	predicate_push_down	Tries to push predicates down to the operator that is closer to the data source.

Optimization Rule	Rule Name	Description
Outer join elimination	outer_join_elimination	Tries to remove the unnecessary left join or right join from the execution plan.

Optimization Rule	Rule Name	Description
Partition pruning	partition_processor	Prunes partitions which are rejected by the predicates and rewrite partitioned table query to the `UnionAll` ↪ ↪ `+` ↪ ↪ `Partition` ↪ ↪ `Datasource` ↪ form.
Aggregation push-down	aggregation_push_down	Tries to push aggregations down to their children.

Optimization Rule	Rule Name	Description
TopN push-down	topn_push_down	This rule is to push the TopN operator to the place closer to the data source.
Join reorder	join_reorder	Decides the order of multi-table joins.

Disable optimization rules

You can use the blocklist of optimization rules to disable some of them if some rules lead to a sub-optimal execution plan for special queries.

Usage

> **Note:**
>
> All the following operations need the `super privilege` privilege of the database. Each optimization rule has a name. For example, the name of column pruning is `column_prune`. The names of all optimization rules can be found in the second column of the table Important Optimization Rules.

- If you want to disable some rules, write its name to the `mysql.opt_rule_blacklist` table. For example:

```
INSERT INTO mysql.opt_rule_blacklist VALUES("join_reorder"), ("
    ↪ topn_push_down");
```

Executing the following SQL statement can make the above operation take effect immediately. The effective range includes all old connections of the corresponding TiDB server:

```
admin reload opt_rule_blacklist;
```

> **Note:**
>
> `admin reload opt_rule_blacklist` only takes effect on the TiDB server where the above statement has been run. If you want all TiDB servers of the cluster to take effect, run this command on each TiDB server.

- If you want to re-enable a rule, delete the corresponding data in the table, and then run the `admin reload` statement:

```
DELETE FROM mysql.opt_rule_blacklist WHERE name IN ("join_reorder", "
    ↪ topn_push_down");
```

```
admin reload opt_rule_blacklist;
```

11.3.4.4.2 The blocklist of expression pushdown

The blocklist of expression pushdown is one way to tune the expression pushdown, mainly used to manually disable some expressions of some specific data types.

Expressions which are supported to be pushed down

Expression Classification	Concrete Operations
Logical operations	AND (&&), OR (||), NOT (!)
Comparison functions and operators	<, <=, =, != (<>), >, >=, <=>, IN(), IS NULL, LIKE, IS TRUE, IS FALSE, COALESCE()
Numeric functions and operators	+, -, *, /, ABS(), CEIL(), CEILING(), FLOOR()
Control flow functions	CASE, IF(), IFNULL()

Expression Classification	Concrete Operations
JSON functions	JSON_TYPE(json_val), JSON_EXTRACT(json_doc, path[, path] ...), JSON_UNQUOTE(json_val), JSON_OBJECT(key, val[, key, val] ...), JSON_ARRAY([val[, val] ...]), JSON_MERGE(json_doc, json_doc[, json_doc] ...), JSON_SET(json_doc, path, val[, path, val] ...), JSON_INSERT(json_doc, path, val[, path, val] ...), JSON_REPLACE(json_doc, path, val[, path, val] ...), JSON_REMOVE(json_doc, path[, path] ...)
Date and time functions	DATE_FORMAT()

Disable the pushdown of specific expressions

When you get wrong results due to the expression pushdown, you can use the blocklist to make a quick recovery for the application. More specifically, you can add some of the supported functions or operators to the `mysql.expr_pushdown_blacklist` table to disable the pushdown of specific expressions.

The schema of `mysql.expr_pushdown_blacklist` is shown as follows:

```
DESC mysql.expr_pushdown_blacklist;
```

```
+------------+-------------+------+------+-------------------+-------+
| Field      | Type        | Null | Key | Default           | Extra |
+------------+-------------+------+------+-------------------+-------+
| name       | char(100)   | NO   |     | NULL              |       |
| store_type | char(100)   | NO   |     | tikv,tiflash,tidb |       |
| reason     | varchar(200) | YES |    | NULL              |       |
+------------+-------------+------+------+-------------------+-------+
3 rows in set (0.00 sec)
```

Here is the description of each field above:

- `name`: The name of the function that is disabled to be pushed down.
- `store_type`: To specify the component that you want to prevent the function from being pushed down to for computing. Available components are `tidb`, `tikv`, and `tiflash`. The `store_type` is case-insensitive. If you need to specify multiple components, use a comma to separate each component.

  - When `store_type` is `tidb`, it indicates whether the function can be executed in other TiDB servers while the TiDB memory table is being read.
  - When `store_type` is `tikv`, it indicates whether the function can be executed in TiKV server's Coprocessor component.
  - When `store_type` is `tiflash`, it indicates whether the function can be executed in TiFlash Server's Coprocessor component.
- `reason`: To record the reason why this function is added to the blocklist.

Usage

This section describes how to use the blocklist of expression pushdown.

Add to the blocklist

To add one or more expressions (functions or operators) to the blocklist, perform the following steps:

1. Insert the corresponding function name or operator name, and the set of components you want to disable the pushdown, to the `mysql.expr_pushdown_blacklist` table.
2. Execute `admin reload expr_pushdown_blacklist`.

Remove from the blocklist

To remove one or more expressions from the blocklist, perform the following steps:

1. Delete the corresponding function name or operator name, and the set of components you want to disable the pushdown, from the `mysql.expr_pushdown_blacklist` table.
2. Execute `admin reload expr_pushdown_blacklist`.

> **Note:**
>
> `admin reload expr_pushdown_blacklist` only takes effect on the TiDB server where this statement is run. If you want all TiDB servers of the cluster to take effect, run this command on each TiDB server.

11.3.4.4.3 Expression blocklist usage example

In the following example, the `<` and `>` operators are added to the blocklist, and then the `>` operator is removed from the blocklist.

To judge whether the blocklist takes effect, observe the results of `EXPLAIN` (See TiDB Query Execution Plan Overview).

1. The predicates `a < 2` and `a > 2` in the `WHERE` clause of the following SQL statement can be pushed down to TiKV.

```
EXPLAIN SELECT * FROM t WHERE a < 2 AND a > 2;
```

```
+--
    ↪ ----------------------+---------+----------+--------------+-----------
    ↪
| id                    | estRows | task     | access object | operator
    ↪ info                  |
+--
    ↪ ----------------------+---------+----------+--------------+-----------
    ↪
| TableReader_7         | 0.00    | root     |               | data:
    ↪ Selection_6               |
|  ─Selection_6         | 0.00    | cop[tikv] |              | gt(ssb_1.t.
    ↪ a, 2), lt(ssb_1.t.a, 2) |
|    ─TableFullScan_5   | 10000.00 | cop[tikv] | table:t   | keep order:
    ↪ false, stats:pseudo |
+--
    ↪ ----------------------+---------+----------+--------------+-----------
    ↪
3 rows in set (0.00 sec)
```

2. Insert the expression to the `mysql.expr_pushdown_blacklist` table and execute `admin reload expr_pushdown_blacklist`.

```
INSERT INTO mysql.expr_pushdown_blacklist VALUES('<','tikv',''), ('>',
    ↪ 'tikv','');
```

```
Query OK, 2 rows affected (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
admin reload expr_pushdown_blacklist;
```

```
Query OK, 0 rows affected (0.00 sec)
```

3. Observe the execution plan again and you will find that both the `<` and `>` operators are not pushed down to TiKV Coprocessor.

```
EXPLAIN SELECT * FROM t WHERE a < 2 and a > 2;
```

```
+--
    ↪ ----------------------+---------+----------+--------------+-----------
    ↪
```

```
| id                    | estRows | task     | access object | operator
    ↪ info                |
+--
    ↪ ----------------------+---------+----------+---------------+----------
    ↪
| Selection_7           | 10000.00 | root    |               | gt(ssb_1.t.a
    ↪ , 2), lt(ssb_1.t.a, 2) |
|  ─TableReader_6       | 10000.00 | root    |               | data:
    ↪ TableFullScan_5           |
|    ─TableFullScan_5   | 10000.00 | cop[tikv] | table:t   | keep order:
    ↪ false, stats:pseudo |
+--
    ↪ ----------------------+---------+----------+---------------+----------
    ↪
3 rows in set (0.00 sec)
```

4. Remove one expression (here is >) from the blocklist and execute `admin reload ↪ expr_pushdown_blacklist`.

```
DELETE FROM mysql.expr_pushdown_blacklist WHERE name = '>';
```

```
Query OK, 1 row affected (0.01 sec)
```

```
admin reload expr_pushdown_blacklist;
```

```
Query OK, 0 rows affected (0.00 sec)
```

5. Observe the execution plan again and you will find that < is not pushed down while > is pushed down to TiKV Coprocessor.

```
EXPLAIN SELECT * FROM t WHERE a < 2 AND a > 2;
```

```
+--
    ↪ ------------------------+---------+----------+---------------+----------
    ↪
| id                      | estRows | task     | access object | operator
    ↪  info              |
+--
    ↪ ------------------------+---------+----------+---------------+----------
    ↪
| Selection_8             | 0.00    | root     |               | lt(ssb_1.t
    ↪ .a, 2)             |
|  ─TableReader_7         | 0.00    | root     |               | data:
    ↪ Selection_6           |
```

```
|   └─Selection_6         | 0.00    | cop[tikv] |           | gt(ssb_1.
    ↪ t.a, 2)           |
|     └─TableFullScan_5   | 10000.00 | cop[tikv] | table:t   | keep
    ↪ order:false, stats:pseudo |
+--
    ↪ -------------------------+----------+-----------+---------------+----------
    ↪
4 rows in set (0.00 sec)
```

12 Tutorials

12.1 Multiple Availability Zones in One Region Deployment

As a distributed SQL database, TiDB combines the best features of the traditional relational database and the scalability of the NoSQL database, and is highly available across availability zones (AZs). This document introduces the deployment of multiple AZs in one region.

The term "region" in this document refers to a geographic area, while the capitalized "Region" refers to a basic unit of data storage in TiKV. "AZ" refers to an isolated location within a region, and each region has multiple AZs. The solution described in this document also applies to the scenario where multiple data centers are located in a single city.

12.1.1 Raft protocol

Raft is a distributed consensus algorithm. Using this algorithm, both PD and TiKV, among components of the TiDB cluster, achieve disaster recovery of data, which is implemented through the following mechanisms:

- The essential role of Raft members is to perform log replication and act as a state machine. Among Raft members, data replication is implemented by replicating logs. Raft members change their own states in different conditions to elect a leader to provide services.
- Raft is a voting system that follows the majority protocol. In a Raft group, if a member gets the majority of votes, its membership changes to leader. In other words, when the majority of nodes remain in the Raft group, a leader can be elected to provide services.

To take advantage of Raft's reliability, the following conditions must be met in a real deployment scenario:

- Use at least three servers in case one server fails.
- Use at least three racks in case one rack fails.

- Use at least three AZs in case one AZ fails.
- Deploy TiDB in at least three regions in case data safety issue occurs in one region.

The native Raft protocol does not have good support for an even number of replicas. Considering the impact of cross-region network latency, three AZs in the same region might be the most suitable solution to a highly available and disaster tolerant Raft deployment.

12.1.2 Three AZs in one region deployment

TiDB clusters can be deployed in three AZs in the same region. In this solution, data replication across the three AZs is implemented using the Raft protocol within the cluster. These three AZs can provide read and write services at the same time. Data consistency is not affected even if one AZ fails.

12.1.2.1 Simple architecture

TiDB, TiKV, and PD are distributed among three AZs, which is the most common deployment with the highest availability.

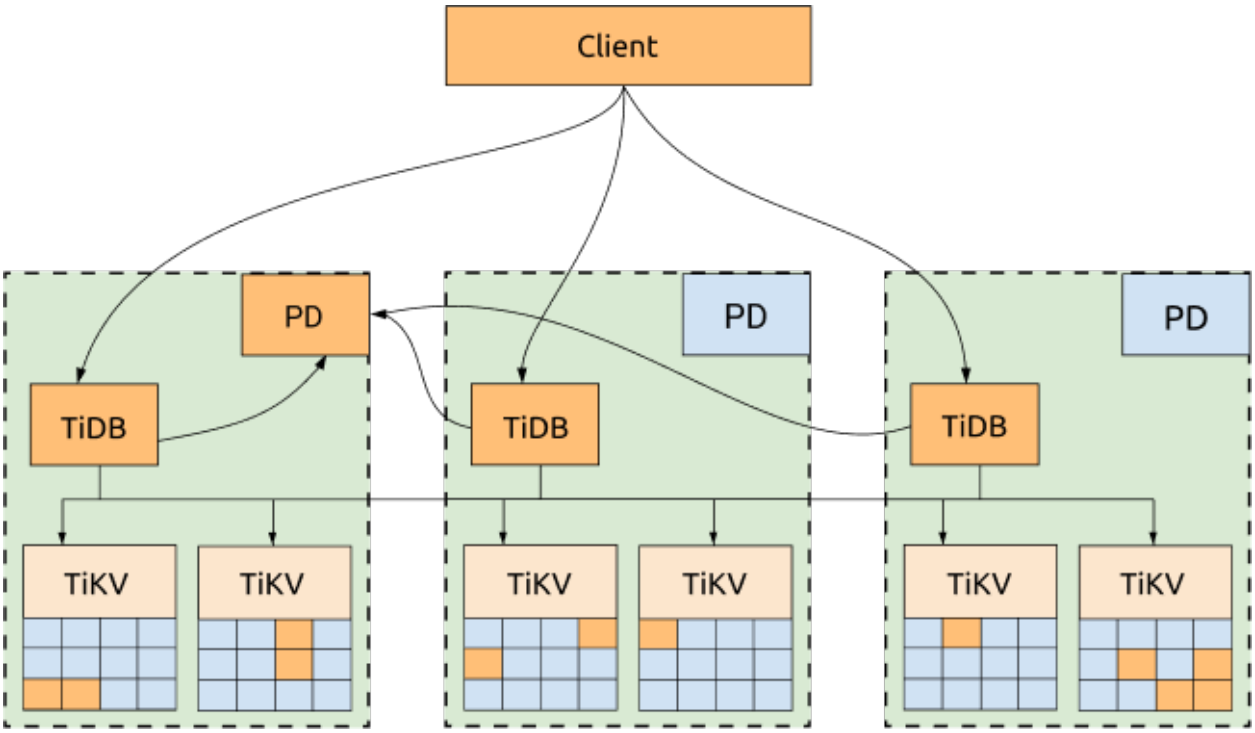


Figure 189: 3-AZ Deployment Architecture

Advantages:

- All replicas are distributed among three AZs, with high availability and disaster recovery capability.
- No data will be lost if one AZ is down (RPO = 0).
- Even if one AZ is down, the other two AZs will automatically start leader election and automatically resume services within a certain period (within 20 seconds in most cases). See the following diagram for more information:

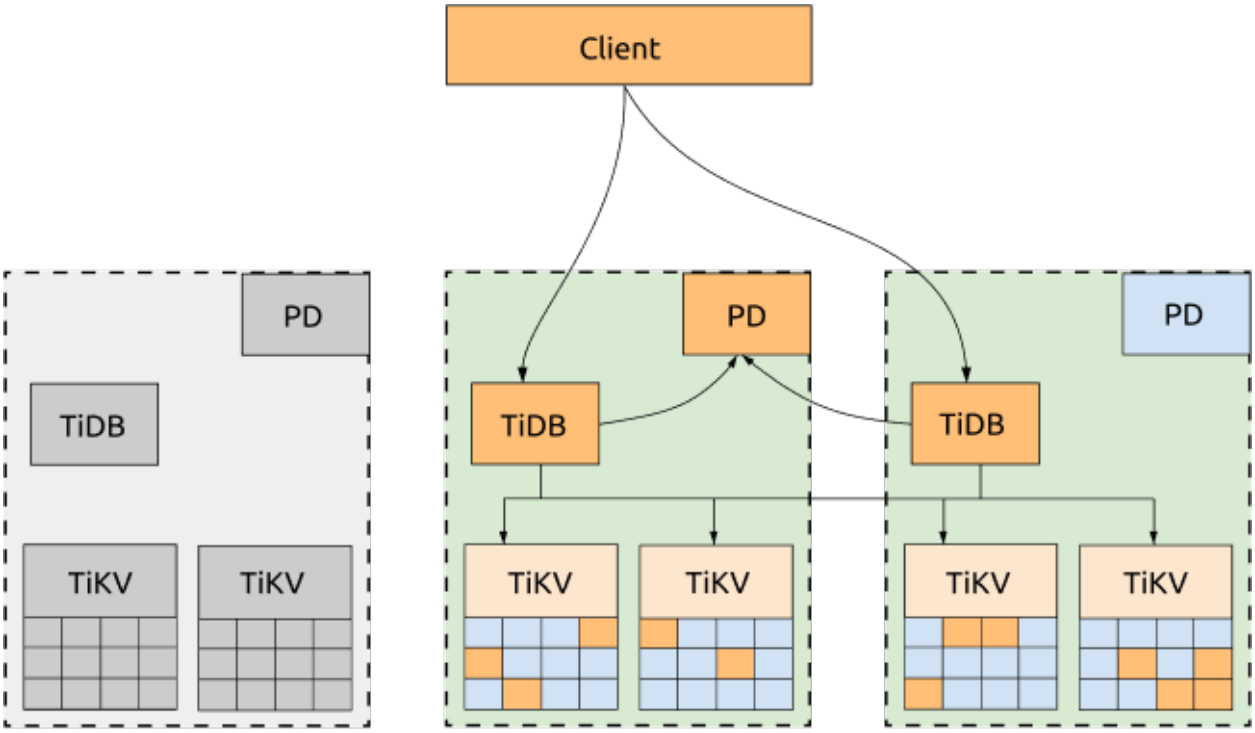


Figure 190: Disaster Recovery for 3-AZ Deployment

Disadvantages:

The performance can be affected by the network latency.

- For writes, all the data has to be replicated to at least two AZs. Because TiDB uses a two-phase commit for writes, the write latency is at least twice the latency of the network between two AZs.
- The read performance will also be affected by the network latency if the leader is not in the same AZ with the TiDB node that sends the read request.
- Each TiDB transaction needs to obtain TimeStamp Oracle (TSO) from the PD leader. So if the TiDB and PD leaders are not in the same AZ, the performance of the transactions will also be affected by the network latency because each transaction with the write request has to obtain TSO twice.

12.1.2.2 Optimized architecture

If not all of the three AZs need to provide services to the applications, you can dispatch all the requests to one AZ and configure the scheduling policy to migrate the TiKV Region leader and PD leader to the same AZ. In this way, neither obtaining TSO nor reading TiKV Regions will be impacted by the network latency across AZs. If this AZ is down, the PD leader and TiKV Region leader will be automatically elected in other surviving AZs, and you just need to switch the requests to the AZs that are still alive.

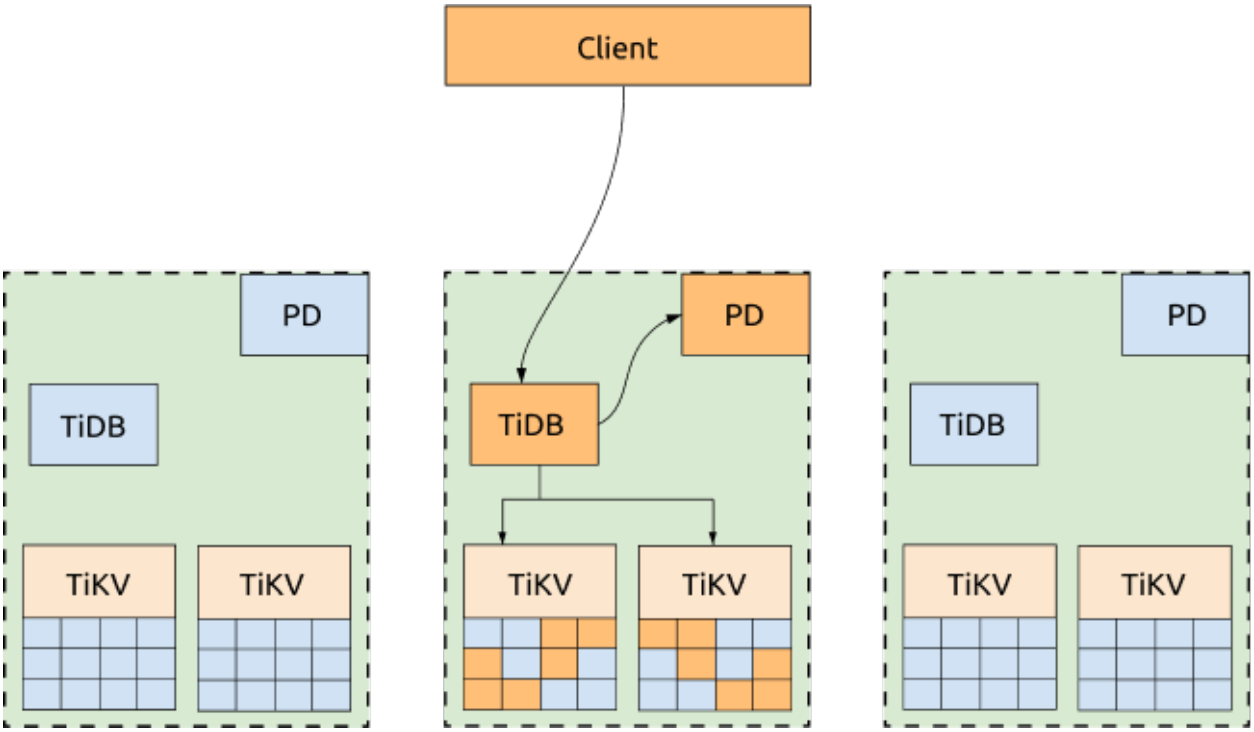


Figure 191: Read Performance Optimized 3-AZ Deployment

Advantages:

The cluster's read performance and the capability to get TSO are improved. A configuration template of scheduling policy is as follows:

```
-- Evicts all leaders of other AZs to the AZ that provides services to the
   ↪ application.
config set label-property reject-leader LabelName labelValue

-- Migrates PD leaders and sets priority.
member leader transfer pdName1
member leader_priority pdName1 5
member leader_priority pdName2 4
member leader_priority pdName3 3
```

> **Note:**
>
> Starting from TiDB v5.2, the `label-property` configuration is not supported by default. To set the replica policy, use the placement rules.

Disadvantages:

- Write scenarios are still affected by network latency across AZs. This is because Raft follows the majority protocol and all written data must be replicated to at least two AZs.
- The TiDB server that provides services is only in one AZ.
- All application traffic is processed by one AZ and the performance is limited by the network bandwidth pressure of that AZ.
- The capability to get TSO and the read performance are affected by whether the PD server and TiKV server are up in the AZ that processes application traffic. If these servers are down, the application is still affected by the cross-center network latency.

12.1.2.3 Deployment example

This section provides a topology example, and introduces TiKV labels and TiKV labels planning.

12.1.2.3.1 Topology example

The following example assumes that three AZs (AZ1, AZ2, and AZ3) are located in one region; each AZ has two sets of racks and each rack has three servers. The example ignores the hybrid deployment or the scenario where multiple instances are deployed on one machine. The deployment of a TiDB cluster (three replicas) on three AZs in one region is as follows:

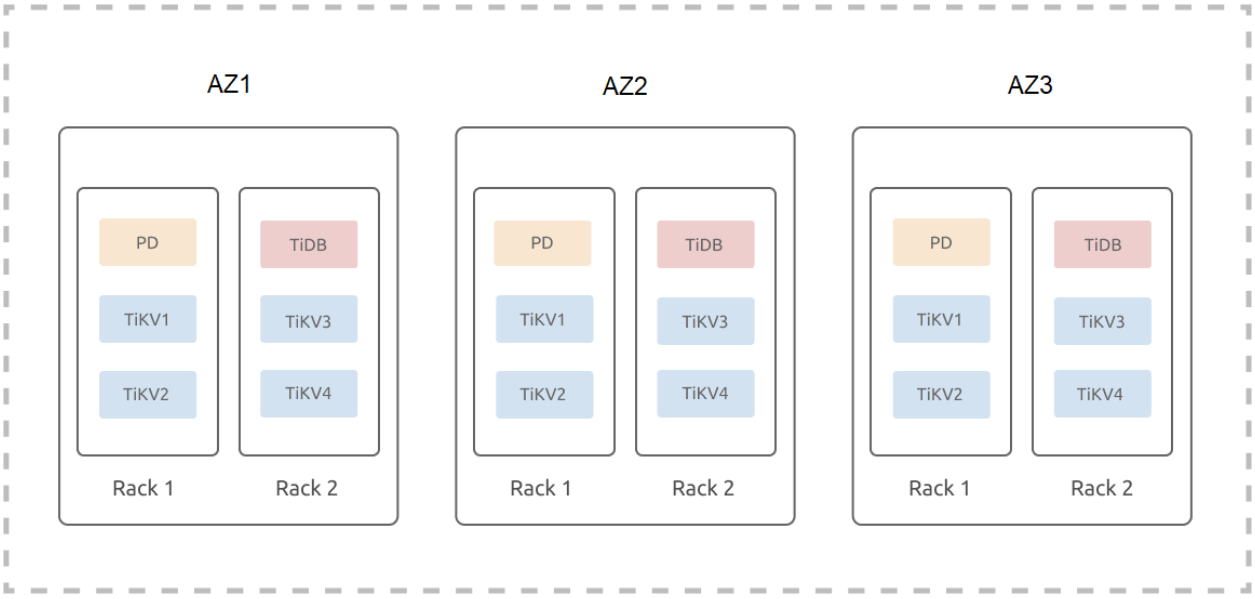


Figure 192: 3-AZ in One Region

12.1.2.3.2 TiKV labels

TiKV is a Multi-Raft system where data is divided into Regions and the size of each Region is 96 MB by default. Three replicas of each Region form a Raft group. For a TiDB cluster of three replicas, because the number of Region replicas is independent of the TiKV instance numbers, three replicas of a Region are only scheduled to three TiKV instances. This means that even if the cluster is scaled out to have N TiKV instances, it is still a cluster of three replicas.

Because a Raft group of three replicas tolerates only one replica failure, even if the cluster is scaled out to have N TiKV instances, this cluster still tolerates only one replica failure. Two failed TiKV instances might cause some Regions to lose replicas and the data in this cluster is no longer complete. SQL requests that access data from these Regions will fail. The probability of two simultaneous failures among N TiKV instances is much higher than the probability of two simultaneous failures among three TiKV instances. This means that the more TiKV instances the Multi-Raft system is scaled out to have, the less the availability of the system.

Because of the preceding limitation, `label` is used to describe the location information of TiKV. The label information is refreshed to the TiKV startup configuration file with deployment or rolling upgrade operations. The started TiKV reports its latest label information to PD. Based on the user-registered label name (the label metadata) and the TiKV topology, PD optimally schedules Region replicas and improves the system availability.

12.1.2.3.3 TiKV labels planning example

To improve the availability and disaster recovery of the system, you need to design and

plan TiKV labels according to your existing physical resources and the disaster recovery capability. You also need to edit the cluster initialization configuration file according to the planned topology:

```
server_configs:
  pd:
    replication.location-labels: ["zone","az","rack","host"]

tikv_servers:
  - host: 10.63.10.30
    config:
      server.labels: { zone: "z1", az: "az1", rack: "r1", host: "30" }
  - host: 10.63.10.31
    config:
      server.labels: { zone: "z1", az: "az1", rack: "r1", host: "31" }
  - host: 10.63.10.32
    config:
      server.labels: { zone: "z1", az: "az1", rack: "r2", host: "32" }
  - host: 10.63.10.33
    config:
      server.labels: { zone: "z1", az: "az1", rack: "r2", host: "33" }

  - host: 10.63.10.34
    config:
      server.labels: { zone: "z2", az: "az2", rack: "r1", host: "34" }
  - host: 10.63.10.35
    config:
      server.labels: { zone: "z2", az: "az2", rack: "r1", host: "35" }
  - host: 10.63.10.36
    config:
      server.labels: { zone: "z2", az: "az2", rack: "r2", host: "36" }
  - host: 10.63.10.37
    config:
      server.labels: { zone: "z2", az: "az2", rack: "r2", host: "37" }

  - host: 10.63.10.38
    config:
      server.labels: { zone: "z3", az: "az3", rack: "r1", host: "38" }
  - host: 10.63.10.39
    config:
      server.labels: { zone: "z3", az: "az3", rack: "r1", host: "39" }
  - host: 10.63.10.40
    config:
      server.labels: { zone: "z3", az: "az3", rack: "r2", host: "40" }
  - host: 10.63.10.41
```

```
config:
  server.labels: { zone: "z3", az: "az3", rack: "r2", host: "41" }
```

In the preceding example, `zone` is the logical availability zone layer that controls the isolation of replicas (three replicas in the example cluster).

Considering that the AZs might be scaled out in the future, the three-layer label structure (`az`, `rack`, and `host`) is not directly adopted. Assuming that `AZ2`, `AZ3`, and `AZ4` are to be scaled out, you only need to scale out the AZs in the corresponding availability zone and scale out the racks in the corresponding AZ.

If this three-layer label structure is directly adopted, after scaling out an AZ, you might need to apply new labels and the data in TiKV needs to be rebalanced.

12.1.2.4 High availability and disaster recovery analysis

The multiple AZs in one region deployment can guarantee that if one AZ fails, the cluster can automatically recover services without manual intervention. Data consistency is also guaranteed. Note that scheduling policies are used to optimize performance, but when a failure occurs, these policies prioritize availability over performance.

12.2 Three Availability Zones in Two Regions Deployment

This document introduces the architecture and configuration of the three availability zones (AZs) in two regions deployment.

The term "region" in this document refers to a geographic area, while the capitalized "Region" refers to a basic unit of data storage in TiKV. "AZ" refers to an isolated location within a region, and each region has multiple AZs. The solution described in this document also applies to the scenario where multiple data centers are located in a single city.

12.2.1 Overview

The architecture of three AZs in two regions is a highly available and disaster tolerant deployment solution that provides a production data AZ, a disaster recovery AZ in the same region, and a disaster recovery AZ in another region. In this mode, the three AZs in two regions are interconnected. If one AZ fails or suffers from a disaster, other AZs can still operate well and take over the key applications or all applications. Compared with the multi-AZ in one region deployment, this solution has the advantage of cross-region high availability and can survive region-level natural disasters.

The distributed database TiDB natively supports the three-AZ-in-two-region architecture by using the Raft algorithm, and guarantees the consistency and high availability of data within a database cluster. Because the network latency across AZs in the same region is relatively low, the application traffic can be dispatched to two AZs in the same region, and the traffic load can be shared by these two AZs by controlling the distribution of TiKV Region leaders and PD leaders.

12.2.2 Deployment architecture

This section takes the example of Seattle and San Francisco to explain the deployment mode of three AZs in two regions for the distributed database of TiDB.

In this example, two AZs (AZ1 and AZ2) are located in Seattle and another AZ (AZ3) is located in San Francisco. The network latency between AZ1 and AZ2 is lower than 3 milliseconds. The network latency between AZ3 and AZ1/AZ2 in Seattle is about 20 milliseconds (ISP dedicated network is used).

The architecture of the cluster deployment is as follows:

- The TiDB cluster is deployed to three AZs in two regions: AZ1 in Seattle, AZ2 in Seattle, and AZ3 in San Francisco.
- The cluster has five replicas, two in AZ1, two in AZ2, and one in AZ3. For the TiKV component, each rack has a label, which means that each rack has a replica.
- The Raft protocol is adopted to ensure consistency and high availability of data, which is transparent to users.

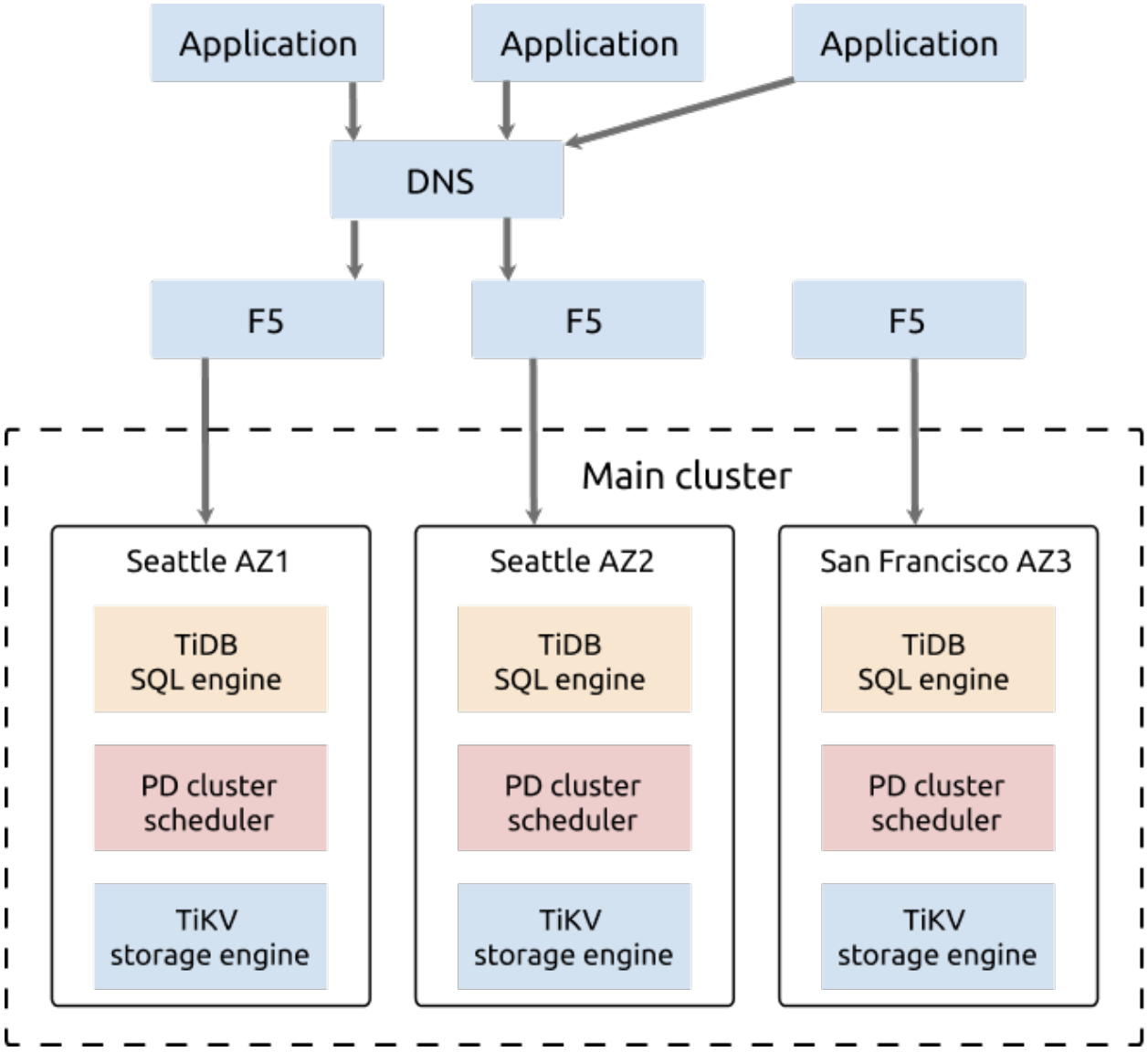


Figure 193: 3-AZ-in-2-region architecture

This architecture is highly available. The distribution of Region leaders is restricted to the two AZs (AZ1 and AZ2) that are in the same region (Seattle). Compared with the three-AZ solution in which the distribution of Region leaders is not restricted, this architecture has the following advantages and disadvantages:

- **Advantages**
  - Region leaders are in AZs of the same region with low latency, so the write is faster.
  - The two AZs can provide services at the same time, so the resource usage rate is higher.

- If one AZ fails, services are still available and data safety is ensured.

- **Disadvantages**

  - Because the data consistency is achieved by the Raft algorithm, when two AZs in the same region fail at the same time, only one surviving replica remains in the disaster recovery AZ in another region (San Francisco). This cannot meet the requirement of the Raft algorithm that most replicas survive. As a result, the cluster can be temporarily unavailable. Maintenance staff needs to recover the cluster from the one surviving replica and a small amount of hot data that has not been replicated will be lost. But this case is a rare occurrence.
  - Because the ISP dedicated network is used, the network infrastructure of this architecture has a high cost.
  - Five replicas are configured in three AZs in two regions, data redundancy increases, which brings a higher storage cost.

12.2.2.1 Deployment details

The configuration of the three AZs in two regions (Seattle and San Francisco) deployment plan is illustrated as follows:

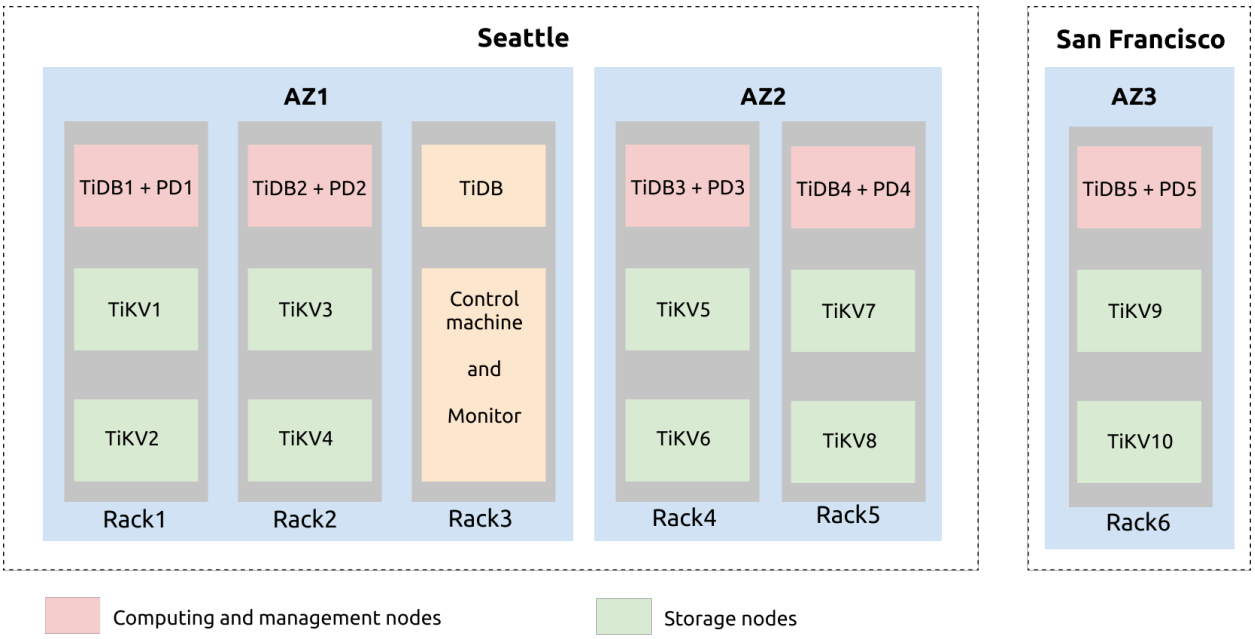


Figure 194: 3-AZ-2-region

From the preceding illustration, you can see that Seattle has two AZs: AZ1 and AZ2. AZ1 has three sets of racks: rac1, rac2, and rac3. AZ2 has two racks: rac4 and rac5. The AZ3 in San Francisco has the rac6 rack.

In the rac1 of AZ1, one server is deployed with TiDB and PD services, and the other two servers are deployed with TiKV services. Each TiKV server is deployed with two TiKV instances (tikv-server). This is similar to rac2, rac4, rac5, and rac6.

The TiDB server, the control machine, and the monitoring server are on rac3. The TiDB server is deployed for regular maintenance and backup. Prometheus, Grafana, and the restore tools are deployed on the control machine and monitoring machine.

Another backup server can be added to deploy Drainer. Drainer saves binlog data to a specified location by outputting files, to achieve incremental backup.

12.2.3 Configuration

12.2.3.1 Example

See the following `tiup topology.yaml` yaml file for example:

```
## # Global variables are applied to all deployments and used as the default
   ↪ value of
## # the deployments if a specific deployment value is missing.
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/data/tidb_cluster/tidb-deploy"
  data_dir: "/data/tidb_cluster/tidb-data"

server_configs:
  tikv:
    server.grpc-compression-type: gzip
  pd:
    replication.location-labels: ["az","replication zone","rack","host"]

pd_servers:
  - host: 10.63.10.10
    name: "pd-10"
  - host: 10.63.10.11
    name: "pd-11"
  - host: 10.63.10.12
    name: "pd-12"
  - host: 10.63.10.13
    name: "pd-13"
  - host: 10.63.10.14
    name: "pd-14"

tidb_servers:
  - host: 10.63.10.10
  - host: 10.63.10.11
```

```
  - host: 10.63.10.12
  - host: 10.63.10.13
  - host: 10.63.10.14

tikv_servers:
  - host: 10.63.10.30
    config:
      server.labels: { az: "1", replication zone: "1", rack: "1", host: "30"
          ↪ }
  - host: 10.63.10.31
    config:
      server.labels: { az: "1", replication zone: "2", rack: "2", host: "31"
          ↪ }
  - host: 10.63.10.32
    config:
      server.labels: { az: "2", replication zone: "3", rack: "3", host: "32"
          ↪ }
  - host: 10.63.10.33
    config:
      server.labels: { az: "2", replication zone: "4", rack: "4", host: "33"
          ↪ }
  - host: 10.63.10.34
    config:
      server.labels: { az: "3", replication zone: "5", rack: "5", host: "34"
          ↪ }
      raftstore.raft-min-election-timeout-ticks: 1000
      raftstore.raft-max-election-timeout-ticks: 1200

monitoring_servers:
  - host: 10.63.10.60

grafana_servers:
  - host: 10.63.10.60

alertmanager_servers:
  - host: 10.63.10.60
```

12.2.3.2 Labels design

In the deployment of three AZs in two regions, the label design requires taking availability and disaster recovery into account. It is recommended that you define the four levels (`az`, `replication zone`, `rack`, and `host`) based on the physical structure of the deployment.

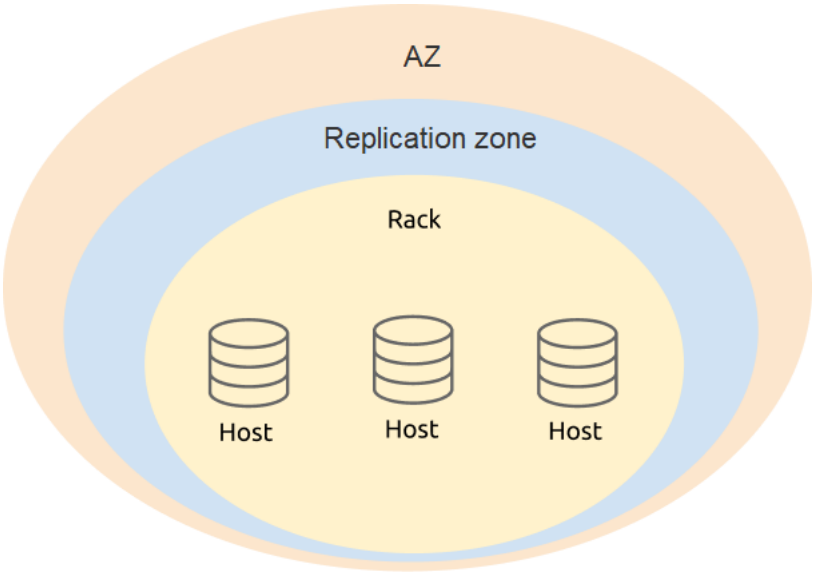


Figure 195: Label logical definition

In the PD configuration, add level information of TiKV labels:

```
server_configs:
  pd:
    replication.location-labels: ["az","replication zone","rack","host"]
```

The configuration of `tikv_servers` is based on the label information of the real physical deployment location of TiKV, which makes it easier for PD to perform global management and scheduling.

```
tikv_servers:
  - host: 10.63.10.30
    config:
      server.labels: { az: "1", replication zone: "1", rack: "1", host: "30"
          ↪ }
  - host: 10.63.10.31
    config:
      server.labels: { az: "1", replication zone: "2", rack: "2", host: "31"
          ↪ }
  - host: 10.63.10.32
    config:
      server.labels: { az: "2", replication zone: "3", rack: "3", host: "32"
          ↪ }
  - host: 10.63.10.33
    config:
      server.labels: { az: "2", replication zone: "4", rack: "4", host: "33"
          ↪ }
  - host: 10.63.10.34
```

```
config:
  server.labels: { az: "3", replication zone: "5", rack: "5", host: "34"
     ↪ }
```

12.2.3.3 Optimize parameter configuration

In the deployment of three AZs in two regions, to optimize performance, you need to not only configure regular parameters, but also adjust component parameters.

- Enable gRPC message compression in TiKV. Because data of the cluster is transmitted in the network, you can enable the gRPC message compression to lower the network traffic.

```
server.grpc-compression-type: gzip
```

- Optimize the network configuration of the TiKV node in another region (San Francisco). Modify the following TiKV parameters for AZ3 in San Francisco and try to prevent the replica in this TiKV node from participating in the Raft election.

```
raftstore.raft-min-election-timeout-ticks: 1000
raftstore.raft-max-election-timeout-ticks: 1200
```

- Configure scheduling. After the cluster is enabled, use the `tiup ctl:<cluster-version> pd` tool to modify the scheduling policy. Modify the number of TiKV Raft replicas. Configure this number as planned. In this example, the number of replicas is five.

```
config set max-replicas 5
```

- Forbid scheduling the Raft leader to AZ3. Scheduling the Raft leader to another region (AZ3) causes unnecessary network overhead between AZ1/AZ2 in Seattle and AZ3 in San Francisco. The network bandwidth and latency also affect the performance of the TiDB cluster.

```
config set label-property reject-leader dc 3
```

> **Note:**
>
> Starting from TiDB v5.2, the `label-property` configuration is not supported by default. To set the replica policy, use the placement rules.

- Configure the priority of PD. To avoid the situation where the PD leader is in another region (AZ3), you can increase the priority of local PD (in Seattle) and decrease the priority of PD in another region (San Francisco). The larger the number, the higher the priority.

```
member leader_priority PD-10 5
member leader_priority PD-11 5
member leader_priority PD-12 5
member leader_priority PD-13 5
member leader_priority PD-14 1
```

12.3 Two Availability Zones in One Region Deployment

This document introduces the deployment mode of two availability zones (AZs) in one region, including the architecture, configuration, how to enable this deployment mode, and how to use replicas in this mode.

The term "region" in this document refers to a geographic area, while the capitalized "Region" refers to a basic unit of data storage in TiKV. "AZ" refers to an isolated location within a region, and each region has multiple AZs. The solution described in this document also applies to the scenario where multiple data centers are located in a single city.

12.3.1 Introduction

TiDB usually adopts the multi-AZ deployment solution to ensure high availability and disaster recovery capability. The multi-AZ deployment solution includes multiple deployment modes, such as multiple AZs in one region and multiple AZs in two regions. This document introduces the deployment mode of two AZs in one region. Deployed in this mode, TiDB can also meet the requirements of high availability and disaster recovery, with a lower cost. This deployment solution adopts Data Replication Auto Synchronous mode, or the DR Auto-Sync mode.

Under the mode of two AZs in one region, the two AZs are less than 50 kilometers apart. They are usually located in the same region or in two adjacent regions. The network latency between the two AZs is lower than 1.5 milliseconds and the bandwidth is higher than 10 Gbps.

12.3.2 Deployment architecture

This section takes the example of a region where two availability zones AZ1 and AZ2 are located respectively in the east and west. AZ1 is the primary AZ and AZ2 is the disaster recovery (DR) AZ.

The architecture of the cluster deployment is as follows:

- The cluster has four replicas: two Voter replicas in AZ1, one Voter replica, and one Learner replica in AZ2. For the TiKV component, each rack has a proper label.
- The Raft protocol is adopted to ensure consistency and high availability of data, which is transparent to users.

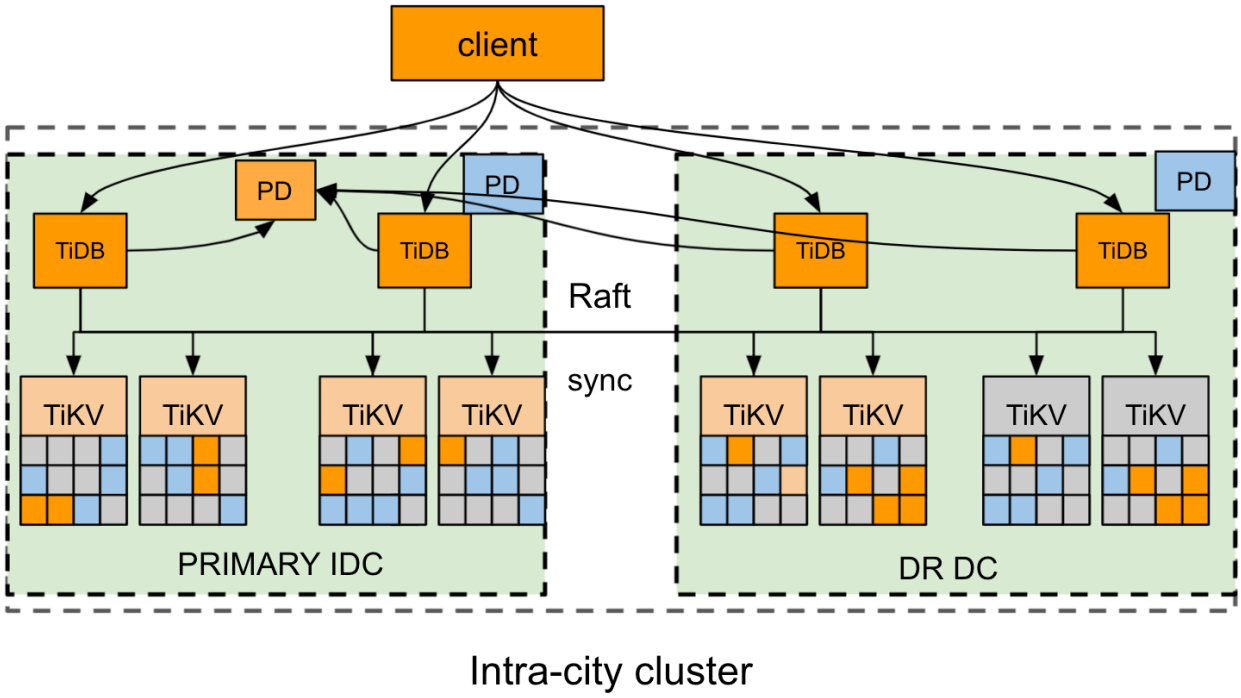


Figure 196: 2-AZ-in-1-region architecture

This deployment solution defines three statuses to control and identify the replication status of the cluster, which restricts the replication mode of TiKV. The replication mode of the cluster can automatically and adaptively switch between the three statuses. For details, see the Status switch section.

- **sync**: Synchronous replication mode. In this mode, at least one replica in the disaster recovery AZ synchronizes with the primary AZ. The Raft algorithm ensures that each log is replicated to the DR based on the label.
- **async**: Asynchronous replication mode. In this mode, the disaster recovery AZ is not fully synchronized with the primary AZ. The Raft algorithm follows the majority protocol to replicate logs.
- **sync-recover**: Synchronous recovery mode. In this mode, the disaster recovery AZ is not fully synchronized with the primary AZ. Raft gradually switches to the label replication mode and then reports the label information to PD.

12.3.3 Configuration

12.3.3.1 Example

The following `tiup topology.yaml` example file is a typical topology configuration for the two availability zones in one region deployment mode:

```
## # Global variables are applied to all deployments and used as the default
   ↪ value of
## # the deployments if a specific deployment value is missing.
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/data/tidb_cluster/tidb-deploy"
  data_dir: "/data/tidb_cluster/tidb-data"
server_configs:
  pd:
    replication.location-labels: ["az","rack","host"]
pd_servers:
  - host: 10.63.10.10
    name: "pd-10"
  - host: 10.63.10.11
    name: "pd-11"
  - host: 10.63.10.12
    name: "pd-12"
tidb_servers:
  - host: 10.63.10.10
  - host: 10.63.10.11
  - host: 10.63.10.12
tikv_servers:
  - host: 10.63.10.30
    config:
      server.labels: { az: "east", rack: "east-1", host: "30" }
  - host: 10.63.10.31
    config:
      server.labels: { az: "east", rack: "east-2", host: "31" }
  - host: 10.63.10.32
    config:
      server.labels: { az: "east", rack: "east-3", host: "32" }
  - host: 10.63.10.33
    config:
      server.labels: { az: "west", rack: "west-1", host: "33" }
  - host: 10.63.10.34
    config:
      server.labels: { az: "west", rack: "west-2", host: "34" }
  - host: 10.63.10.35
```

```
    config:
      server.labels: { az: "west", rack: "west-3", host: "35" }
monitoring_servers:
  - host: 10.63.10.60
grafana_servers:
  - host: 10.63.10.60
alertmanager_servers:
  - host: 10.63.10.60
```

12.3.3.2 Placement Rules

To deploy a cluster based on the planned topology, you need to use Placement Rules to determine the locations of the cluster replicas. Taking the deployment of four replicas (two Voter replicas are at the primary AZ, one Voter replica, and one Learner replica are at the disaster recovery AZ) as an example, you can use the Placement Rules to configure the replicas as follows:

```
cat rule.json
[
  {
    "group_id": "pd",
    "group_index": 0,
    "group_override": false,
    "rules": [
      {
        "group_id": "pd",
        "id": "az-east",
        "start_key": "",
        "end_key": "",
        "role": "voter",
        "count": 3,
        "label_constraints": [
          {
            "key": "az",
            "op": "in",
            "values": [
              "east"
            ]
          }
        ],
        "location_labels": [
          "az",
          "rack",
          "host"
        ]
```

```
  },
  {
    "group_id": "pd",
    "id": "az-west",
    "start_key": "",
    "end_key": "",
    "role": "follower",
    "count": 2,
    "label_constraints": [
      {
        "key": "az",
        "op": "in",
        "values": [
          "west"
        ]
      }
    ],
    "location_labels": [
      "az",
      "rack",
      "host"
    ]
  },
  {
    "group_id": "pd",
    "id": "az-west",
    "start_key": "",
    "end_key": "",
    "role": "learner",
    "count": 1,
    "label_constraints": [
      {
        "key": "az",
        "op": "in",
        "values": [
          "west"
        ]
      }
    ],
    "location_labels": [
      "az",
      "rack",
      "host"
    ]
  }
```

```
    ]
  }
]
```

To use the configurations in `rule.json`, run the following command to back up the existing configuration to the `default.json` file and overwrite the existing configuration with `rule.json`:

```
pd-ctl config placement-rules rule-bundle load --out="default.json"
pd-ctl config placement-rules rule-bundle save --in="rule.json"
```

If you need to roll back to the previous configuration, you can restore the backup file `default.json` or write the following JSON file manually and overwrite the current configuration with this JSON file:

```
cat default.json
[
  {
    "group_id": "pd",
    "group_index": 0,
    "group_override": false,
    "rules": [
      {
        "group_id": "pd",
        "id": "default",
        "start_key": "",
        "end_key": "",
        "role": "voter",
        "count": 5
      }
    ]
  }
]
```

12.3.3.3 Enable the DR Auto-Sync mode

The replication mode is controlled by PD. You can configure the replication mode in the PD configuration file using one of the following methods:

- Method 1: Configure the PD configuration file, and then deploy a cluster.

```
[replication-mode]
replication-mode = "dr-auto-sync"
[replication-mode.dr-auto-sync]
label-key = "az"
primary = "east"
```

```
dr = "west"
primary-replicas = 3
dr-replicas = 2
wait-store-timeout = "1m"
```

- Method 2: If you have deployed a cluster, use pd-ctl commands to modify the configurations of PD.

```
config set replication-mode dr-auto-sync
config set replication-mode dr-auto-sync label-key az
config set replication-mode dr-auto-sync primary east
config set replication-mode dr-auto-sync dr west
config set replication-mode dr-auto-sync primary-replicas 3
config set replication-mode dr-auto-sync dr-replicas 2
```

Descriptions of configuration items:

- `replication-mode` is the replication mode to be enabled. In the preceding example, it is set to `dr-auto-sync`. By default, the majority protocol is used.
- `label-key` is used to distinguish different AZs and needs to match Placement Rules. In this example, the primary AZ is "east" and the disaster recovery AZ is "west".
- `primary-replicas` is the number of Voter replicas in the primary AZ.
- `dr-replicas` is the number of Voter replicas in the disaster recovery AZ.
- `wait-store-timeout` is the waiting time for switching to asynchronous replication mode when network isolation or failure occurs. If the time of network failure exceeds the waiting time, asynchronous replication mode is enabled. The default waiting time is 60 seconds.

To check the current replication status of the cluster, use the following API:

```
curl http://pd_ip:pd_port/pd/api/v1/replication_mode/status
```

```
{
  "mode": "dr-auto-sync",
  "dr-auto-sync": {
    "label-key": "az",
    "state": "sync"
  }
}
```

12.3.3.3.1 Status switch

The replication mode of a cluster can automatically and adaptively switch between three statuses:

- When the cluster is normal, the synchronous replication mode is enabled to maximize the data integrity of the disaster recovery AZ.
- When the network connection between the two AZs fails or the disaster recovery AZ breaks down, after a pre-set protective interval, the cluster enables the asynchronous replication mode to ensure the availability of the application.
- When the network reconnects or the disaster recovery AZ recovers, the TiKV node joins the cluster again and gradually replicates the data. Finally, the cluster switches to the synchronous replication mode.

The details for the status switch are as follows:

1. **Initialization**: At the initialization stage, the cluster is in the synchronous replication mode. PD sends the status information to TiKV, and all TiKV nodes strictly follow the synchronous replication mode to work.

2. **Switch from sync to async**: PD regularly checks the heartbeat information of TiKV to judge whether the TiKV node fails or is disconnected. If the number of failed nodes exceeds the number of replicas of the primary AZ (`primary-replicas`) and the disaster recovery AZ (`dr-replicas`), the synchronous replication mode can no longer serve the data replication and it is necessary to switch the status. When the failure or disconnect time exceeds the time set by `wait-store-timeout`, PD switches the status of the cluster to the async mode. Then PD sends the status of async to all TiKV nodes, and the replication mode for TiKV switches from two-availability-zone replication to the native Raft majority.

3. **Switch from async to sync**: PD regularly checks the heartbeat information of TiKV to judge whether the TiKV node is reconnected. If the number of failed nodes is less than the number of replicas of the primary AZ (`primary-replicas`) and the disaster recovery AZ (`dr-replicas`), the synchronous replication mode can be enabled again. PD first switches the status of the cluster to sync-recover and sends the status information to all TiKV nodes. All Regions of TiKV gradually switch to the two-availability-zone synchronous replication mode and then report the heartbeat information to PD. PD records the status of TiKV Regions and calculates the recovery progress. When all TiKV Regions finish the switching, PD switches the replication mode to sync.

12.3.3.4 Disaster recovery

This section introduces the disaster recovery solution of the two AZs in one region deployment.

When a disaster occurs to a cluster in the synchronous replication mode, you can perform data recovery with `RPO = 0`:

- If the primary AZ fails and most of the Voter replicas are lost, but complete data exists in the disaster recovery AZ, the lost data can be recovered from the disaster recovery AZ. At this time, manual intervention is required with professional tools. You can get support from PingCAP or the community for a recovery solution.

- If the disaster recovery AZ fails and a few Voter replicas are lost, the cluster automatically switches to the asynchronous replication mode.

When a disaster occurs to a cluster that is not in the synchronous replication mode and you cannot perform data recovery with `RPO = 0`:

- If most of the Voter replicas are lost, manual intervention is required with professional tools. You can get support from PingCAP or the community for a recovery solution.

12.4 Read Historical Data

12.4.1 Use Stale Read (Recommended)

12.4.1.1 Usage Scenarios of Stale Read

This document describes the usage scenarios of Stale Read. Stale Read is a mechanism that TiDB applies to read historical versions of data stored in TiDB. Using this mechanism, you can read the corresponding historical data of a specific point in time or within a specified time range, and thus save the latency brought by data replication between storage nodes.

When you are using Stale Read, TiDB will randomly select a replica for data reading, which means that all replicas are available for data reading. If your application cannot tolerate reading non-real-time data, do not use Stale Read; otherwise, the data read from the replica might not be the latest data written into TiDB.

12.4.1.1.1 Scenario examples

- Scenario one: If a transaction only involves read operations and is tolerant of data staleness to some extent, you can use Stale Read to get historical data. Using Stale Read, TiDB makes the query requests sent to any replica at the expense of some real-time performance, and thus increases the throughput of query executions. Especially in some scenarios where small tables are queried, if strongly consistent reads are used, leader might be concentrated on a certain storage node, causing the query pressure to be concentrated on that node as well. Therefore, that node might become a bottleneck for the whole query. Stale Read, however, can improve the overall query throughput and significantly improve the query performance.
- Scenario two: In some scenarios of geo-distributed deployment, if strongly consistent follower reads are used, to make sure that the data read from the Followers is consistent with that stored in the Leader, TiDB requests `Readindex` from different data centers for verification, which increases the access latency for the whole query process. With Stale Read, TiDB accesses the replica in the current data center to read the corresponding data at the expense of some real-time performance, which avoids network latency brought by cross-center connection and reduces the access latency for the entire query. For more information, see Local Read under Three Data Centers Deployment.

12.4.1.1.2 Usages

TiDB provides the methods of performing Stale Read at the statement level and the session level as follows:

- Statement level
  - Specifying an exact point in time (**recommended**): If you need TiDB to read data that is globally consistent from a specific point in time without violating the isolation level, you can specify the corresponding timestamp of that point in time in the query statement. For detailed usage, see `AS OF TIMESTAMP` clause.
  - Specifying a time range: If you need TiDB to read the data as new as possible within a time range without violating the isolation level, you can specify the time range in the query statement. Within the specified time range, TiDB selects a suitable timestamp to read the corresponding data. "Suitable" means there are no transactions that start before this timestamp and have not been committed on the accessed replica, that is, TiDB can perform read operations on the accessed replica and the read operations are not blocked. For detailed usage, refer to the introduction of the `AS OF TIMESTAMP` Clause and the `TIDB_BOUNDED_STALENESS` function.
- Session level
  - Specifying a time range: In a session, if you need TiDB to read the data as new as possible within a time range in subsequent queries without violating the isolation level, you can specify the time range by setting the `tidb_read_staleness` system variable. For detailed usage, refer to `tidb_read_staleness`.

12.4.1.2 Read Historical Data Using the `AS OF TIMESTAMP` Clause

This document describes how to perform the Stale Read feature using the `AS OF TIMESTAMP` clause to read historical data in TiDB, including specific usage examples and strategies for saving historical data.

> **Warning:**
>
> Currently, you cannot use Stale Read together with TiFlash. If your SQL query contains the `AS OF TIMESTAMP` clause and TiDB might read data from TiFlash replicas, you might encounter an error with a message like `ERROR 1105 (HY000): stale requests require tikv backend`.
>
> To fix the problem, disable TiFlash replicas for your Stale Read query. To do that, perform one of the following operations:
>
> - Use the `set session tidb_isolation_read_engines='tidb,tikv'` variable.
> - Use the hint to enforce TiDB to read data from TiKV.

TiDB supports reading historical data through a standard SQL interface, which is the `AS OF TIMESTAMP` SQL clause, without the need for special clients or drivers. After data is updated or deleted, you can read the historical data before the update or deletion using this SQL interface.

> **Note:**
>
> When reading historical data, TiDB returns the data with the old table structure even if the current table structure is different.

12.4.1.2.1 Syntax

You can use the `AS OF TIMESTAMP` clause in the following three ways:

- `SELECT ... FROM ... AS OF TIMESTAMP`
- `START TRANSACTION READ ONLY AS OF TIMESTAMP`
- `SET TRANSACTION READ ONLY AS OF TIMESTAMP`

If you want to specify an exact point of time, you can set a datetime value or use a time function in the `AS OF TIMESTAMP` clause. The format of datetime is like "2016-10-08 16:45:26.999", with millisecond as the minimum time unit, but for most of the time, the time unit of second is enough for specifying a datetime, such as "2016-10-08 16:45:26". You can also get the current time to the millisecond using the `NOW(3)` function. If you want to read the data of several seconds ago, it is **recommended** to use an expression such as `NOW() - INTERVAL 10 SECOND`.

If you want to specify a time range, you can use the `TIDB_BOUNDED_STALENESS()` function in the clause. When this function is used, TiDB selects a suitable timestamp within the specified time range. "Suitable" means there are no transactions that start before this timestamp and have not been committed on the accessed replica, that is, TiDB can perform read operations on the accessed replica and the read operations are not blocked. You need to use `TIDB_BOUNDED_STALENESS(t1, t2)` to call this function. `t1` and `t2` are the two ends of the time range, which can be specified using either datetime values or time functions.

Here are some examples of the `AS OF TIMESTAMP` clause:

- `AS OF TIMESTAMP '2016-10-08 16:45:26'`: Tells TiDB to read the latest data stored at 16:45:26 on October 8, 2016.
- `AS OF TIMESTAMP NOW()- INTERVAL 10 SECOND`: Tells TiDB to read the latest data stored 10 seconds ago.
- `AS OF TIMESTAMP TIDB_BOUNDED_STALENESS('2016-10-08 16:45:26', '2016-10-08 16:45:29')`: Tells TiDB to read the data as new as possible within the time range of 16:45:26 to 16:45:29 on October 8, 2016.

- `AS OF TIMESTAMP TIDB_BOUNDED_STALENESS(NOW()- INTERVAL 20 SECOND, NOW() ↪ )`: Tells TiDB to read the data as new as possible within the time range of 20 seconds ago to the present.

> **Note:**
>
> In addition to specifying a timestamp, the most common use of the `AS OF ↪ TIMESTAMP` clause is to read data that is several seconds old. If this approach is used, it is recommended to read historical data older than 5 seconds.
>
> You need to deploy the NTP service for your TiDB and PD nodes when you use Stale Read. This avoids the situation where the specified timestamp used by TiDB goes ahead of the latest TSO allocating progress (such as a timestamp several seconds ahead), or is later than the GC safe point timestamp. When the specified timestamp goes beyond the service scope, TiDB returns an error.

12.4.1.2.2 Usage examples

This section describes different ways to use the `AS OF TIMESTAMP` clause with several examples. It first introduces how to prepare the data for recovery, and then shows how to use `AS OF TIMESTAMP` in `SELECT`, `START TRANSACTION READ ONLY AS OF TIMESTAMP`, and `SET TRANSACTION READ ONLY AS OF TIMESTAMP` respectively.

Prepare data sample

To prepare data for recovery, create a table first and insert several rows of data:

```
create table t (c int);
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
insert into t values (1), (2), (3);
```

```
Query OK, 3 rows affected (0.00 sec)
```

View the data in the table:

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
```

```
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

View the current time:

```
select now();
```

```
+---------------------+
| now()               |
+---------------------+
| 2021-05-26 16:45:26 |
+---------------------+
1 row in set (0.00 sec)
```

Update the data in a row:

```
update t set c=22 where c=2;
```

```
Query OK, 1 row affected (0.00 sec)
```

Confirm that the data of the row is updated:

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

Read historical data using the `SELECT` statement

You can use the `SELECT ... FROM ... AS OF TIMESTAMP` statement to read data from a time point in the past.

```
select * from t as of timestamp '2021-05-26 16:45:26';
```

```
+------+
| c    |
+------+
|    1 |
|    2 |
```

```
|   3 |
+------+
3 rows in set (0.00 sec)
```

> **Note:**
>
> When reading multiple tables using one `SELECT` statement, you need to make sure that the format of TIMESTAMP EXPRESSIONs is consistent. For example, `select * from t as of timestamp NOW()- INTERVAL 2 SECOND, c as of timestamp NOW()- INTERVAL 2 SECOND;`. In addition, you must specify the `AS OF` information for the relevant table in the `SELECT` statement; otherwise, the `SELECT` statement reads the latest data by default.

Read historical data using the `START TRANSACTION READ ONLY AS OF TIMESTAMP` statement

You can use the `START TRANSACTION READ ONLY AS OF TIMESTAMP` statement to start a read-only transaction based on a time point in the past. The transaction reads historical data of the given time.

```
start transaction read only as of timestamp '2021-05-26 16:45:26';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

```
commit;
```

```
Query OK, 0 rows affected (0.00 sec)
```

After the transaction is committed, you can read the latest data.

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

> **Note:**
>
> If you start a transaction with the statement `START TRANSACTION READ ONLY AS OF TIMESTAMP`, it is a read-only transaction. Write operations are rejected in this transaction.

Read historical data using the `SET TRANSACTION READ ONLY AS OF TIMESTAMP` statement

You can use the `SET TRANSACTION READ ONLY AS OF TIMESTAMP` statement to set the next transaction as a read-only transaction based on a specified time point in the past. The transaction reads historical data of the given time.

```
set transaction read only as of timestamp '2021-05-26 16:45:26';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
begin;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

```
commit;
```

```
Query OK, 0 rows affected (0.00 sec)
```

After the transaction is committed, you can read the latest data.

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

> **Note:**
>
> If you start a transaction with the statement `SET TRANSACTION READ ONLY AS OF TIMESTAMP`, it is a read-only transaction. Write operations are rejected in this transaction.

12.4.1.3 Read Historical Data Using the `tidb_read_staleness` System Variable

To support reading the historical data, in v5.4, TiDB introduces a new system variable `tidb_read_staleness`. This document describes how to read historical data through this system variable, including detailed operating procedures.

12.4.1.3.1 Feature description

The `tidb_read_staleness` system variable is used to set the time range of historical data that TiDB can read in the current session. The data type of this variable is int type, and the scope of it is `SESSION`. After setting the value, TiDB selects a timestamp as new as possible from the range allowed by this variable, and all subsequent read operations are performed against this timestamp. For example, if the value of this variable is set to `-5`, on the condition that TiKV has the corresponding historical version's data, TiDB selects a timestamp as new as possible within a 5-second time range.

After enabling `tidb_read_staleness`, you still can perform the following operations:

- Insert, modify, delete data or perform DML operations in the current session. These statements are not affected by `tidb_read_staleness`.

- Start an interactive transaction in the current session. Queries within this transaction still read the latest data.

After reading the historical data, you can read the latest data in the following two ways:

- Start a new session.
- Set the value of the `tidb_read_staleness` variable to `""` using the `SET` statement.

12.4.1.3.2 Usage examples

This section describes how to use `tidb_read_staleness` with examples.

1. Create a table, and insert a few rows of data into the table:

```
create table t (c int);
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
insert into t values (1), (2), (3);
```

```
Query OK, 3 rows affected (0.00 sec)
```

2. Check out the data in the table:

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

3. Update the data in a row:

```
update t set c=22 where c=2;
```

```
Query OK, 1 row affected (0.00 sec)
```

4. Confirm that the data has been updated:

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

5. Set the `tidb_read_staleness` system variable.

   The scope of this variable is `SESSION`. After setting its value, TiDB reads the latest version data before the time set by the value.

   The following setting indicates that TiDB selects a timestamp as new as possible within the time range from 5 seconds ago to now and uses it as the timestamp for reading historical data:

```
set @@tidb_read_staleness="-5";
```

```
Query OK, 0 rows affected (0.00 sec)
```

Note:

- Use `@@` instead of `@` before `tidb_read_staleness`. `@@` means system variables, and `@` means user variables.
- You need to set the historical time range (the value of `tidb_read_staleness`) according to the total time that you spent in step 3 and step 4. Otherwise, the latest data will be displayed in the query results, not the historical data. Therefore, you need to adjust this time range according to the time you spent on operations. For example, in this example, since the set time range is 5 seconds, you need to complete step 3 and step 4 within 5 seconds.

The data read here is the data before the update, that is, the historical data:

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
```

```
+------+
3 rows in set (0.00 sec)
```

6. After un-setting this variable as follows, TiDB can read the latest data:

```
set @@tidb_read_staleness="";
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select * from t;
```

```
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

12.4.2 Read Historical Data Using the System Variable `tidb_snapshot`

This document describes how to read data from the history versions using the system variable `tidb_snapshot`, including specific usage examples and strategies for saving historical data.

> **Note:**
>
> You can also use the Stale Read feature to read historical data, which is more recommended.

12.4.2.1 Feature description

TiDB implements a feature to read history data using the standard SQL interface directly without special clients or drivers.

> **Note:**
>
> - Even when data is updated or removed, its history versions can be read using the SQL interface.

- When reading historical data, TiDB returns the data with the old table structure even if the current table structure is different.

12.4.2.2 How TiDB reads data from history versions

The `tidb_snapshot` system variable is introduced to support reading history data. About the `tidb_snapshot` variable:

- The variable is valid in the `SESSION` scope.
- Its value can be modified using the `SET` statement.
- The data type for the variable is text.
- The variable accepts TSO (Timestamp Oracle) and datetime. TSO is a globally unique time service, which is obtained from PD. The acceptable datetime format is "2016-10-08 16:45:26.999". Generally, the datetime can be set using second precision, for example "2016-10-08 16:45:26".
- When the variable is set, TiDB creates a Snapshot using its value as the timestamp, just for the data structure and there is no any overhead. After that, all the `SELECT` operations will read data from this Snapshot.

Note:

Because the timestamp in TiDB transactions is allocated by Placement Driver (PD), the version of the stored data is also marked based on the timestamp allocated by PD. When a Snapshot is created, the version number is based on the value of the `tidb_snapshot` variable. If there is a large difference between the local time of the TiDB server and the PD server, use the time of the PD server.

After reading data from history versions, you can read data from the latest version by ending the current Session or using the `SET` statement to set the value of the `tidb_snapshot` variable to "" (empty string).

12.4.2.3 How TiDB manages the data versions

TiDB implements Multi-Version Concurrency Control (MVCC) to manage data versions. The history versions of data are kept because each update/removal creates a new version of the data object instead of updating/removing the data object in-place. But not all the versions are kept. If the versions are older than a specific time, they will be removed completely to reduce the storage occupancy and the performance overhead caused by too many history versions.

In TiDB, Garbage Collection (GC) runs periodically to remove the obsolete data versions. For GC details, see TiDB Garbage Collection (GC)

Pay special attention to the following:

- `tidb_gc_life_time`: This system variable is used to configure the retention time of earlier modifications (default: `10m0s`).
- The output of `SELECT * FROM mysql.tidb WHERE variable_name = 'tikv_gc_safe_point ↪ '`. This is the current `safePoint` where you can read historical data up to. It is updated every time the garbage collection process is run.

12.4.2.4 Example

1. At the initial stage, create a table and insert several rows of data:

```
mysql> create table t (c int);
Query OK, 0 rows affected (0.01 sec)

mysql> insert into t values (1), (2), (3);
Query OK, 3 rows affected (0.00 sec)
```

2. View the data in the table:

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

3. View the timestamp of the table:

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2016-10-08 16:45:26 |
+---------------------+
1 row in set (0.00 sec)
```

4. Update the data in one row:

```
mysql> update t set c=22 where c=2;
Query OK, 1 row affected (0.00 sec)
```

5. Make sure the data is updated:

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

6. Set the `tidb_snapshot` variable whose scope is Session. The variable is set so that the latest version before the value can be read.

> **Note:**
> In this example, the value is set to be the time before the update operation.

```
mysql> set @@tidb_snapshot="2016-10-08 16:45:26";
Query OK, 0 rows affected (0.00 sec)
```

> **Note:**
> You should use `@@` instead of `@` before `tidb_snapshot` because `@@` is used to denote the system variable while `@` is used to denote the user variable.

Result: The read from the following statement is the data before the update operation, which is the history data.

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

7. Set the `tidb_snapshot` variable to be "" (empty string) and you can read the data from the latest version:

```
mysql> set @@tidb_snapshot="";
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

> **Note:**
>
> You should use `@@` instead of `@` before `tidb_snapshot` because `@@` is used to denote the system variable while `@` is used to denote the user variable.

12.4.2.5 How to restore historical data

Before you restore data from an older version, make sure that Garbage Collection (GC) does not clear the history data while you are working on it. This can be done by setting the `tidb_gc_life_time` variable as the following example shows. Do not forget to set the variable back to the previous value after the restore.

```
SET GLOBAL tidb_gc_life_time="60m";
```

> **Note:**
>
> Increasing the GC life time from the default 10 minutes to half an hour or more will result in additional versions of rows being retained, which might require more disk space. This might also affect the performance of certain operations such as scans when TiDB needs to skip these additional versions of the same rows during data reads.

To restore data from an older version, you can use one of the following methods:

- For simple cases, use `SELECT` after setting the `tidb_snapshot` variable and copy-paste the output, or use `SELECT ... INTO LOCAL OUTFLE` and use `LOAD DATA` to import the data later on.

- Use Dumpling to export a historical snapshot. Dumpling performs well in exporting larger sets of data.

12.5 Best Practices

12.5.1 TiDB Best Practices

This document summarizes the best practices of using TiDB, including the use of SQL and optimization tips for Online Analytical Processing (OLAP) and Online Transactional Processing (OLTP) scenarios, especially the optimization options specific for TiDB.

Before you read this document, it is recommended that you read three blog posts that introduce the technical principles of TiDB:

- TiDB Internal (I) - Data Storage
- TiDB Internal (II) - Computing
- TiDB Internal (III) - Scheduling

12.5.1.1 Preface

Database is a generic infrastructure system. It is important to consider various user scenarios during the development process and to modify the data parameters or the way to use according to actual situations in specific business scenarios.

TiDB is a distributed database compatible with the MySQL protocol and syntax. But with the internal implementation and supporting of distributed storage and transactions, the way of using TiDB is different from MySQL.

12.5.1.2 Basic concepts

The best practices are closely related to its implementation principles. It is recommended that you learn some of the basic mechanisms, including the Raft consensus algorithm, distributed transactions, data sharding, load balancing, the mapping solution from SQL to Key-Value (KV), the implementation method of secondary indexing, and distributed execution engines.

This section is an introduction to these concepts. For detailed information, refer to PingCAP blog posts.

12.5.1.2.1 Raft

Raft is a consensus algorithm that ensures data replication with strong consistency. At the bottom layer, TiDB uses Raft to replicate data. TiDB writes data to the majority of the replicas before returning the result of success. In this way, even though a few replicas might get lost, the system still has the latest data. For example, if there are three replicas, the system does not return the result of success until data has been written to two replicas. Whenever a replica is lost, at least one of the remaining two replicas have the latest data.

To store three replicas, compared with the replication of Source-Replica, Raft is more efficient. The write latency of Raft depends on the two fastest replicas, instead of the slowest one. Therefore, the implementation of geo-distributed and multiple active data centers becomes possible by using the Raft replication. In the typical scenario of three data centers distributing in two sites, to guarantee the data consistency, TiDB just needs to successfully write data into the local data center and the closer one, instead of writing to all three data centers. However, this does not mean that cross-data center deployment can be implemented in any scenario. When the amount of data to be written is large, the bandwidth and latency between data centers become the key factors. If the write speed exceeds the bandwidth or the latency is too high, the Raft replication mechanism still cannot work well.

12.5.1.2.2 Distributed transactions

TiDB provides complete distributed transactions and the model has some optimizations on the basis of Google Percolator. This document introduces the following features:

- Optimistic transaction model

  TiDB's optimistic transaction model does not detect conflicts until the commit phase. If there are conflicts, the transaction needs retry. But this model is inefficient if the conflict is severe, because operations before retry are invalid and need to repeat.

  Assume that the database is used as a counter. High access concurrency might lead to severe conflicts, resulting in multiple retries or even timeouts. Therefore, in the scenario of severe conflicts, it is recommended to use the pessimistic transaction mode or to solve problems at the system architecture level, such as placing counter in Redis. Nonetheless, the optimistic transaction model is efficient if the access conflict is not very severe.

- Pessimistic transaction mode

  In TiDB, the pessimistic transaction mode has almost the same behavior as in MySQL. The transaction applies a lock during the execution phase, which avoids retries in conflict situations and ensures a higher success rate. By applying the pessimistic locking, you can also lock data in advance using `SELECT FOR UPDATE`.

  However, if the application scenario has fewer conflicts, the optimistic transaction model has better performance.

- Transaction size limit

  As distributed transactions need to conduct two-phase commit and the bottom layer performs Raft replication, if a transaction is very large, the commit process would be quite slow, and the following Raft replication process is thus stuck. To avoid this problem, the transaction size is limited:

  - A transaction is limited to 5,000 SQL statements (by default)
  - Each Key-Value entry is no more than 6 MB (by default)
  - The total size of Key-Value entries is no more than 10 GB.

  You can find similar limits in Google Cloud Spanner.

12.5.1.2.3 Data sharding

TiKV automatically shards bottom-layered data according to the range of keys. Each Region is a range of keys, which is a left-closed and right-open interval, `[StartKey, EndKey ↪ )`. When the amount of Key-Value pairs in a Region exceeds a certain value, the Region automatically splits into two.

12.5.1.2.4 Load balancing

Placement Driver (PD) balances the load of the cluster according to the status of the entire TiKV cluster. The unit of scheduling is Region and the logic is the strategy configured by PD.

12.5.1.2.5 SQL on KV

TiDB automatically maps the SQL structure into Key-Value structure. For details, see TiDB Internal (II) - Computing.

Simply put, TiDB performs the following operations:

- A row of data is mapped to a Key-Value pair. The key is prefixed with `TableID` and suffixed with the row ID.
- An index is mapped as a Key-Value pair. The key is prefixed with `TableID+IndexID` and suffixed with the index value.

The data or indexes in the same table have the same prefix. These Key-Values are at adjacent positions in the key space of TiKV. Therefore, when the amount of data to be written is large and all is written to one table, the write hotspot is created. The situation gets worse when some index values of the continuous written data is also continuous (for example, fields that increase with time, like `update time`), which creates a few write hotspots and becomes the bottleneck of the entire system.

Similarly, if all data is read from a focused small range (for example, the continuous tens or hundreds of thousands of rows of data), an access hotspot of data is likely to occur.

12.5.1.2.6 Secondary index

TiDB supports the complete secondary indexes, which are also global indexes. Many queries can be optimized by index. Thus, it is important for applications to make good use of secondary indexes.

Lots of MySQL experience is also applicable to TiDB. It is noted that TiDB has its unique features. The following are a few notes when using secondary indexes in TiDB.

- The more secondary indexes, the better?

  Secondary indexes can speed up queries, but adding an index has side effects. The previous section introduces the storage model of indexes. For each additional index,

there will be one more Key-Value when inserting a row. Therefore, the more indexes, the slower the writing speed and the more space it takes up.

In addition, too many indexes affects the runtime of the optimizer, and inappropriate indexes mislead the optimizer. Thus, more secondary indexes does not mean better performance.

- Which columns should create indexes?

  As is mentioned above, index is important but the number of indexes should be proper. You must create appropriate indexes according to the application characteristics. In principle, you need to create an index on the columns involved in the query to improve the performance. The following are situations that need to create indexes:

  - For columns with a high degree of differentiation, filtered rows are remarkably reduced through indexes.
  - If there are multiple query criteria, you can choose composite indexes. Note to put the columns with the equivalent condition before composite indexes.

  For example, if a commonly used query is `select * from t where c1 = 10 and c2 = 100 and c3 > 10`, you can create a composite index `Index cidx (c1, c2, c3)`. In this way, you can use the query condition to create an index prefix and then scan.

- The difference between querying through indexes and directly scanning the table

  TiDB has implemented global indexes, so indexes and data of the table are not necessarily on the same data sharding. When querying through indexes, it should firstly scan indexes to get the corresponding row ID and then use the row ID to get the data. Thus, this method involves two network requests and has a certain performance overhead.

  If the query involves lots of rows, scanning index proceeds concurrently. When the first batch of results is returned, getting the data of the table can then proceed. Therefore, this is a parallel + pipeline model. Though the two accesses create overhead, the latency is not high.

  The following two conditions do not have the problem of two accesses:

  - Columns of the index have already met the query requirement. Assume that the `c` column on the `t` table has an index and the query is `select c from t where c > 10;`. At this time, all needed data can be obtained if you access the index. This situation is called `Covering Index`. But if you focus more on the query performance, you can put into index a portion of columns that do not need to be filtered but need to be returned in the query result, creating composite index. Take `select c1, c2 from t where c1 > 10;` as an example. You can optimize this query by creating composite index `Index c12 (c1, c2)`.

    - The primary key of the table is integer. In this case, TiDB uses the value of the primary key as row ID. Thus, if the query condition is on the primary key, you can directly construct the range of the row ID, scan the table data, and get the result.

- Query concurrency

  As data is distributed across many Regions, queries run in TiDB concurrently. But the concurrency by default is not high in case it consumes lots of system resources. Besides, the OLTP query usually does not involve a large amount of data and the low concurrency is enough. But for the OLAP query, the concurrency is high and TiDB modifies the query concurrency through the following system variables:

    - `tidb_distsql_scan_concurrency`:

      The concurrency of scanning data, including scanning the table and index data.
    - `tidb_index_lookup_size`:

      If it needs to access the index to get row IDs before accessing the table data, it uses a batch of row IDs as a single request to access the table data. This parameter sets the size of a batch. The larger batch increases latency, while the smaller one might lead to more queries. The proper size of this parameter is related to the amount of data that the query involves. Generally, no modification is required.
    - `tidb_index_lookup_concurrency`:

      If it needs to access the index to get row IDs before accessing the table data, the concurrency of getting data through row IDs every time is modified through this parameter.

- Ensure the order of results through indexes

  You can use indexes to filter or sort data. Firstly, get row IDs according to the index order. Then, return the row content according to the return order of row IDs. In this way, the returned results are ordered according to the index column. It has been mentioned earlier that the model of scanning index and getting row is parallel + pipeline. If the row is returned according to the index order, a high concurrency between two queries does not reduce latency. Thus, the concurrency is low by default, but it can be modified through the `tidb_index_serial_scan_concurrency` variable.

- Reverse index scan

  TiDB supports scanning an ascending index in reverse order, at a speed slower than normal scan by 20%. If the data is changed frequently and thus too many versions exist, the performance overhead might be higher. It is recommended to avoid reverse index scans as much as possible.

12.5.1.3 Scenarios and practices

In the last section, we discussed some basic implementation mechanisms of TiDB and their influence on usage. This section introduces specific usage scenarios and operation practices, from deployment to application usage.

12.5.1.3.1 Deployment

Before deployment, read Software and Hardware Requirements.

It is recommended to deploy the TiDB cluster using TiUP. This tool can deploy, stop, destroy, and upgrade the whole cluster, which is quite convenient. It is not recommended to manually deploy the TiDB cluster, which might be troublesome to maintain and upgrade later.

12.5.1.3.2 Data import

To improve the write performance during the import process, you can tune TiKV's parameters as stated in Tune TiKV Memory Parameter Performance.

12.5.1.3.3 Write

As mentioned before, TiDB limits the size of a single transaction in the Key-Value layer. As for the SQL layer, a row of data is mapped to a Key-Value entry. For each additional index, one more Key-Value entry is added.

> **Note:**
>
> When you set the size limit for transactions, you need to consider the overhead of TiDB encoding and the extra transaction key. It is recommended that **the number of rows of each transaction is less than 200 and the data size of a single row is less than 100 KB**; otherwise, the performance is bad.

It is recommended to split statements into batches or add a limit to the statements, whether they are `INSERT`, `UPDATE` or `DELETE` statements.

When deleting a large amount of data, it is recommended to use `Delete from t where xx limit 5000;`. It deletes through the loop and use `Affected Rows == 0` as a condition to end the loop.

If the amount of data that needs to be deleted at a time is large, this loop method gets slower and slower because each deletion traverses backward. After deleting the previous data, lots of deleted flags remain for a short period (then all is cleared by Garbage Collection) and affect the following `DELETE` statement. If possible, it is recommended to refine the `WHERE` condition. Assume that you need to delete all data on `2017-05-26`, you can use the following statements:

```
for i from 0 to 23:
    while affected_rows > 0:
        delete from t where insert_time >= i:00:00 and insert_time < (i+1)
            ↪ :00:00 limit 5000;
```

```
    affected_rows = select affected_rows()
```

This pseudocode means to split huge chunks of data into small ones and then delete, so that the earlier `Delete` statements do not affect the later ones.

12.5.1.3.4 Query

For query requirements and specific statements, refer to System Variables.

You can control the concurrency of SQL execution through the `SET` statement and the selection of the `Join` operator through hints.

In addition, you can also use MySQL's standard index selection, the hint syntax, or control the optimizer to select indexes through `Use Index/Ignore Index hint`.

If the application scenario has both OLTP and OLAP workloads, you can send the OLTP request and OLAP request to different TiDB servers, diminishing the impact of OLAP on OLTP. It is recommended to use machines with high-performance hardware (for example, more processor cores and larger memory) for the TiDB server that processes OLAP workloads.

To completely isolate OLTP and OLAP workloads, it is recommended to run OLAP applications on TiFlash. TiFlash is a columnar storage engine with great performance on OLAP workloads. TiFlash can achieve physical isolation on the storage layer and guarantees consistent reads.

12.5.1.3.5 Monitoring and log

The monitoring metrics is the best method to learn the status of the system. It is recommended that you deploy the monitoring system along with your TiDB cluster.

TiDB uses Grafana + Prometheus to monitor the system status. The monitoring system is automatically deployed and configured if you deploy TiDB using TiUP.

There are lots of items in the monitoring system, the majority of which are for TiDB developers. You do not have to understand these items without an in-depth knowledge of the source code. Some items that are related to applications or to the state of system key components are selected and put in a separate `overview` panel for users.

In addition to monitoring, you can also view the system logs. The three components of TiDB, tidb-server, tikv-server, and pd-server, each has a `--log-file` parameter. If this parameter has been configured when the cluster is started, logs are stored in the file configured by the parameter and log files are automatically archived on a daily basis. If the `--log-file` parameter has not been configured, the log is output to `stderr`.

Starting from TiDB 4.0, TiDB provides TiDB Dashboard UI to improve usability. You can access TiDB Dashboard by visiting http://$%7BPD_IP%7D:$%7BPD_PORT%7D/dashboard in your browser. TiDB Dashboard provides features such as viewing cluster status, performance analysis, traffic visualization, cluster diagnostics, and log searching.

12.5.1.3.6 Documentation

The best way to learn about a system or solve the problem is to read its documentation and understand its implementation principles.

TiDB has a large number of official documents both in Chinese and English. If you have met an issue, you can start from FAQ and TiDB Cluster Troubleshooting Guide. You can also search the issue list or create an issue in TiDB repository on GitHub.

TiDB also has many useful migration tools. See Migration Tool Overview for details.

For more articles on the technical details of TiDB, see the PingCAP official blog site.

12.5.1.4 Best scenarios for TiDB

TiDB is suitable for the following scenarios:

- The data volume is too large for a standalone database
- You do not want to do sharding
- The access mode has no obvious hotspot
- Transactions, strong consistency, and disaster recovery are required
- You hope to have real-time Hybrid Transaction/Analytical Processing (HTAP) analytics and reduce storage links

12.5.2 Best Practices for Developing Java Applications with TiDB

This document introduces the best practice for developing Java applications to better use TiDB. Based on some common Java application components that interact with the backend TiDB database, this document also provides the solutions to commonly encountered issues during development.

12.5.2.1 Database-related components in Java applications

Common components that interact with the TiDB database in Java applications include:

- Network protocol: A client interacts with a TiDB server via the standard MySQL protocol.
- JDBC API and JDBC drivers: Java applications usually use the standard JDBC (Java Database Connectivity) API to access a database. To connect to TiDB, you can use a JDBC driver that implements the MySQL protocol via the JDBC API. Such common JDBC drivers for MySQL include MySQL Connector/J and MariaDB Connector/J.
- Database connection pool: To reduce the overhead of creating a connection each time it is requested, applications usually use a connection pool to cache and reuse connections. JDBC DataSource defines a connection pool API. You can choose from different open-source connection pool implementations as needed.
- Data access framework: Applications usually use a data access framework such as MyBatis and Hibernate to further simplify and manage the database access operations.

- Application implementation: The application logic controls when to send what commands to the database. Some applications use Spring Transaction aspects to manage transactions' start and commit logics.

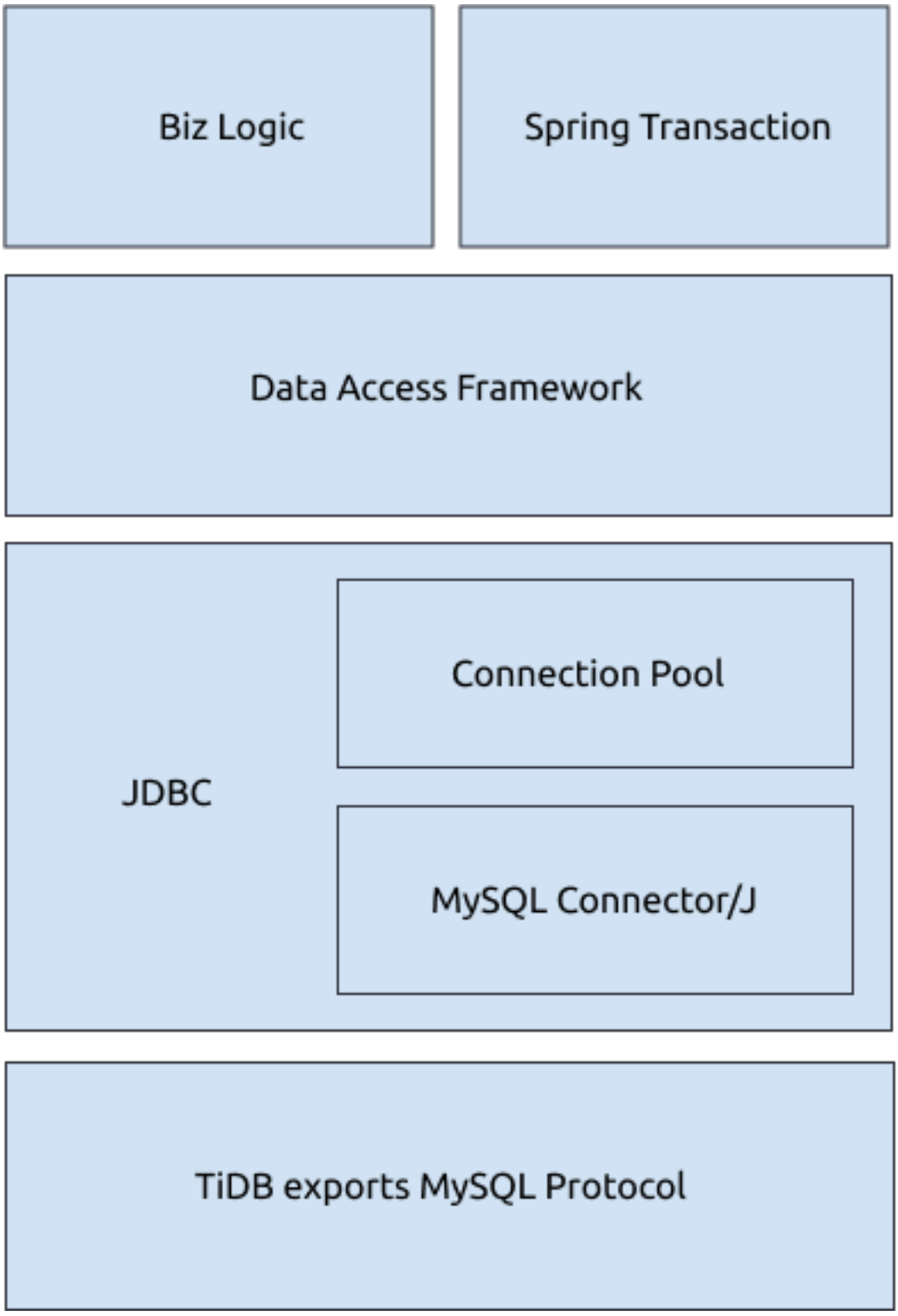


Figure 197: Java application components

From the above diagram, you can see that a Java application might do the following things:

- Implement the MySQL protocol via the JDBC API to interact with TiDB.
- Get a persistent connection from the connection pool.
- Use a data access framework such as MyBatis to generate and execute SQL statements.
- Use Spring Transaction to automatically start or stop a transaction.

The rest of this document describes the issues and their solutions when you develop a Java application using the above components.

12.5.2.2 JDBC

Java applications can be encapsulated with various frameworks. In most of the frameworks, JDBC API is called on the bottommost level to interact with the database server. For JDBC, it is recommended that you focus on the following things:

- JDBC API usage choice
- API Implementer's parameter configuration

12.5.2.2.1 JDBC API

For JDBC API usage, see JDBC official tutorial. This section covers the usage of several important APIs.

Use Prepare API

For OLTP (Online Transactional Processing) scenarios, the SQL statements sent by the program to the database are several types that can be exhausted after removing parameter changes. Therefore, it is recommended to use Prepared Statements instead of regular execution from a text file and reuse Prepared Statements to execute directly. This avoids the overhead of repeatedly parsing and generating SQL execution plans in TiDB.

At present, most upper-level frameworks call the Prepare API for SQL execution. If you use the JDBC API directly for development, pay attention to choosing the Prepare API.

In addition, with the default implementation of MySQL Connector/J, only client-side statements are preprocessed, and the statements are sent to the server in a text file after `?` is replaced on the client. Therefore, in addition to using the Prepare API, you also need to configure `useServerPrepStmts = true` in JDBC connection parameters before you perform statement preprocessing on the TiDB server. For detailed parameter configuration, see MySQL JDBC parameters.

Use Batch API

For batch inserts, you can use the `addBatch/executeBatch` API. The `addBatch()` method is used to cache multiple SQL statements first on the client, and then send them to the database server together when calling the `executeBatch` method.

Note:

In the default MySQL Connector/J implementation, the sending time of the SQL statements that are added to batch with `addBatch()` is delayed to the time when `executeBatch()` is called, but the statements will still be sent one by one during the actual network transfer. Therefore, this method usually does not reduce the amount of communication overhead.

If you want to batch network transfer, you need to configure `rewriteBatchedStatements = true` in the JDBC connection parameters. For the detailed parameter configuration, see Batch-related parameters.

Use `StreamingResult` to get the execution result

In most scenarios, to improve execution efficiency, JDBC obtains query results in advance and save them in client memory by default. But when the query returns a super large result set, the client often wants the database server to reduce the number of records returned at a time, and waits until the client's memory is ready and it requests for the next batch.

Usually, there are two kinds of processing methods in JDBC:

- Set `FetchSize` to `Integer.MIN_VALUE` to ensure that the client does not cache. The client will read the execution result from the network connection through `StreamingResult`.

  When the client uses the streaming read method, it needs to finish reading or close `resultset` before continuing to use the statement to make a query. Otherwise, the error `No statements may be issued when any streaming result sets are open and in use on a given connection. Ensure that you have called .close()on any active streaming result sets before attempting more queries.` is returned.

  To avoid such an error in queries before the client finishes reading or closes `resultset`, you can add the `clobberStreamingResults=true` parameter in the URL. Then, `resultset` is automatically closed but the result set to be read in the previous streaming query is lost.

- To use Cursor Fetch, first set `FetchSize` as a positive integer and configure `useCursorFetch=true` in the JDBC URL.

TiDB supports both methods, but it is preferred that you use the first method, because it is a simpler implementation and has a better execution efficiency.

12.5.2.2.2 MySQL JDBC parameters

JDBC usually provides implementation-related configurations in the form of JDBC URL parameters. This section introduces MySQL Connector/J's parameter configurations (If

you use MariaDB, see MariaDB's parameter configurations). Because this document cannot cover all configuration items, it mainly focuses on several parameters that might affect performance.

Prepare-related parameters

This section introduces parameters related to `Prepare`.

`useServerPrepStmts`

`useServerPrepStmts` is set to `false` by default, that is, even if you use the Prepare API, the "prepare" operation will be done only on the client. To avoid the parsing overhead of the server, if the same SQL statement uses the Prepare API multiple times, it is recommended to set this configuration to `true`.

To verify that this setting already takes effect, you can do:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > CPS By Instance**.
- If `COM_QUERY` is replaced by `COM_STMT_EXECUTE` or `COM_STMT_PREPARE` in the request, it means this setting already takes effect.

`cachePrepStmts`

Although `useServerPrepStmts=true` allows the server to execute Prepared Statements, by default, the client closes the Prepared Statements after each execution and does not reuse them. This means that the "prepare" operation is not even as efficient as text file execution. To solve this, it is recommended that after setting `useServerPrepStmts=true`, you should also configure `cachePrepStmts=true`. This allows the client to cache Prepared Statements.

To verify that this setting already takes effect, you can do:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > CPS By Instance**.
- If the number of `COM_STMT_EXECUTE` in the request is far more than the number of `COM_STMT_PREPARE`, it means this setting already takes effect.

In addition, configuring `useConfigs=maxPerformance` will configure multiple parameters at the same time, including `cachePrepStmts=true`.

`prepStmtCacheSqlLimit`

After configuring `cachePrepStmts`, also pay attention to the `prepStmtCacheSqlLimit` configuration (the default value is `256`). This configuration controls the maximum length of the Prepared Statements cached on the client.

The Prepared Statements that exceed this maximum length will not be cached, so they cannot be reused. In this case, you may consider increasing the value of this configuration depending on the actual SQL length of the application.

You need to check whether this setting is too small if you:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > CPS By Instance**.
- And find that `cachePrepStmts=true` has been configured, but `COM_STMT_PREPARE` is still mostly equal to `COM_STMT_EXECUTE` and `COM_STMT_CLOSE` exists.

`prepStmtCacheSize`

`prepStmtCacheSize` controls the number of cached Prepared Statements (the default value is `25`). If your application requires "preparing" many types of SQL statements and wants to reuse Prepared Statements, you can increase this value.

To verify that this setting already takes effect, you can do:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > CPS By Instance**.
- If the number of `COM_STMT_EXECUTE` in the request is far more than the number of `COM_STMT_PREPARE`, it means this setting already takes effect.

Batch-related parameters

While processing batch writes, it is recommended to configure `rewriteBatchedStatements=true`. After using `addBatch()` or `executeBatch()`, JDBC still sends SQL one by one by default, for example:

```
pstmt = prepare("insert into t (a) values(?)");
pstmt.setInt(1, 10);
pstmt.addBatch();
pstmt.setInt(1, 11);
pstmt.addBatch();
pstmt.setInt(1, 12);
pstmt.executeBatch();
```

Although `Batch` methods are used, the SQL statements sent to TiDB are still individual `INSERT` statements:

```
insert into t(a) values(10);
insert into t(a) values(11);
insert into t(a) values(12);
```

But if you set `rewriteBatchedStatements=true`, the SQL statements sent to TiDB will be a single `INSERT` statement:

```
insert into t(a) values(10),(11),(12);
```

Note that the rewrite of the `INSERT` statements is to concatenate the values after multiple "values" keywords into a whole SQL statement. If the `INSERT` statements have other differences, they cannot be rewritten, for example:

```
insert into t (a) values (10) on duplicate key update a = 10;
insert into t (a) values (11) on duplicate key update a = 11;
insert into t (a) values (12) on duplicate key update a = 12;
```

The above `INSERT` statements cannot be rewritten into one statement. But if you change the three statements into the following ones:

```
insert into t (a) values (10) on duplicate key update a = values(a);
insert into t (a) values (11) on duplicate key update a = values(a);
insert into t (a) values (12) on duplicate key update a = values(a);
```

Then they meet the rewrite requirement. The above `INSERT` statements will be rewritten into the following one statement:

```
insert into t (a) values (10), (11), (12) on duplicate key update a =
    ↪ values(a);
```

If there are three or more updates during the batch update, the SQL statements will be rewritten and sent as multiple queries. This effectively reduces the client-to-server request overhead, but the side effect is that a larger SQL statement is generated. For example:

```
update t set a = 10 where id = 1; update t set a = 11 where id = 2; update
    ↪ t set a = 12 where id = 3;
```

In addition, because of a client bug, if you want to configure `rewriteBatchedStatements ↪ =true` and `useServerPrepStmts=true` during batch update, it is recommended that you also configure the `allowMultiQueries=true` parameter to avoid this bug.

Integrate parameters

Through monitoring, you might notice that although the application only performs `INSERT` operations to the TiDB cluster, there are a lot of redundant `SELECT` statements. Usually this happens because JDBC sends some SQL statements to query the settings, for example, `select @@session.transaction_read_only`. These SQL statements are useless for TiDB, so it is recommended that you configure `useConfigs=maxPerformance` to avoid extra overhead.

`useConfigs=maxPerformance` includes a group of configurations. To get the detailed configurations in MySQL Connector/J 8.0 and those in MySQL Connector/J 5.1, see mysql-connector-j 8.0 and mysql-connector-j 5.1 respectively.

After it is configured, you can check the monitoring to see a decreased number of `SELECT` statements.

Timeout-related parameters

TiDB provides two MySQL-compatible parameters that controls the timeout: `wait_timeout` and `max_execution_time`. These two parameters respectively control the connection idle timeout with the Java application and the timeout of the SQL execution

in the connection; that is to say, these parameters control the longest idle time and the longest busy time for the connection between TiDB and the Java application. The default value of both parameters is `0`, which by default allows the connection to be infinitely idle and infinitely busy (an infinite duration for one SQL statement to execute).

However, in an actual production environment, idle connections and SQL statements with excessively long execution time negatively affect databases and applications. To avoid idle connections and SQL statements that are executed for too long, you can configure these two parameters in your application's connection string. For example, set `sessionVariables ↪ =wait_timeout=3600` (1 hour) and `sessionVariables=max_execution_time=300000` (5 minutes).

12.5.2.3 Connection pool

Building TiDB (MySQL) connections is relatively expensive (for OLTP scenarios at least), because in addition to building a TCP connection, connection authentication is also required. Therefore, the client usually saves the TiDB (MySQL) connections to the connection pool for reuse.

Java has many connection pool implementations such as HikariCP, tomcat-jdbc, druid, c3p0, and dbcp. TiDB does not limit which connection pool you use, so you can choose whichever you like for your application.

12.5.2.3.1 Configure the number of connections

It is a common practice that the connection pool size is well adjusted according to the application's own needs. Take HikariCP as an example:

- `maximumPoolSize`: The maximum number of connections in the connection pool. If this value is too large, TiDB consumes resources to maintain useless connections. If this value is too small, the application gets slow connections. So configure this value for your own good. For details, see About Pool Sizing.
- `minimumIdle`: The minimum number of idle connections in the connection pool. It is mainly used to reserve some connections to respond to sudden requests when the application is idle. You can also configure it according to your application needs.

The application needs to return the connection after finishing using it. It is also recommended that the application use the corresponding connection pool monitoring (such as `metricRegistry`) to locate the connection pool issue in time.

12.5.2.3.2 Probe configuration

The connection pool maintains persistent connections from clients to TiDB as follows:

- Before v5.4, TiDB does not proactively close client connections by default (unless an error is reported).

- Starting from v5.4, TiDB automatically closes client connections after `28800` seconds (this is, `8` hours) of inactivity by default. You can control this timeout setting using the TiDB and MySQL compatible `wait_timeout` variable. For more information, see JDBC Query Timeout.

Moreover, there might be network proxies such as LVS or HAProxy between clients and TiDB. These proxies typically proactively clean up connections after a specific idle period (determined by the proxy's idle configuration). In addition to monitoring the proxy's idle configuration, connection pools also need to maintain or probe connections for keep-alive.

If you often see the following error in your Java application:

```
The last packet sent successfully to the server was 3600000 milliseconds ago
    ↪ . The driver has not received any packets from the server. com.mysql.
    ↪ jdbc.exceptions.jdbc4.CommunicationsException: Communications link
    ↪ failure
```

If `n` in `n milliseconds ago` is `0` or a very small value, it is usually because the executed SQL operation causes TiDB to exit abnormally. To find the cause, it is recommended to check the TiDB stderr log.

If `n` is a very large value (such as `3600000` in the above example), it is likely that this connection was idle for a long time and then closed by the intermediate proxy. The usual solution is to increase the value of the proxy's idle configuration and allow the connection pool to:

- Check whether the connection is available before using the connection every time
- Regularly check whether the connection is available using a separate thread.
- Send a test query regularly to keep alive connections

Different connection pool implementations might support one or more of the above methods. You can check your connection pool documentation to find the corresponding configuration.

12.5.2.4 Data access framework

Applications often use some kind of data access framework to simplify database access.

12.5.2.4.1 MyBatis

MyBatis is a popular Java data access framework. It is mainly used to manage SQL queries and complete the mapping between result sets and Java objects. MyBatis is highly compatible with TiDB. MyBatis rarely has problems based on its historical issues.

Here this document mainly focuses on the following configurations.

Mapper parameters

MyBatis Mapper supports two parameters:

- `select 1 from t where id = #{param1}` will be converted to `select 1 from t where id =?` as a Prepared Statement and be "prepared", and the actual parameter will be used for reuse. You can get the best performance when using this parameter with the previously mentioned Prepare connection parameters.
- `select 1 from t where id = ${param2}` will be replaced with `select 1 from t where id = 1` as a text file and be executed. If this statement is replaced with different parameters and is executed, MyBatis will send different requests for "preparing" the statements to TiDB. This might cause TiDB to cache a large number of Prepared Statements, and executing SQL operations this way has injection security risks.

Dynamic SQL Batch

Dynamic SQL - foreach

To support the automatic rewriting of multiple `INSERT` statements into the form of `insert ... values(...), (...), ...`, in addition to configuring `rewriteBatchedStatements=true` in JDBC as mentioned before, MyBatis can also use dynamic SQL to semi-automatically generate batch inserts. Take the following mapper as an example:

```
<insert id="insertTestBatch" parameterType="java.util.List" fetchSize="1">
  insert into test
   (id, v1, v2)
  values
  <foreach item="item" index="index" collection="list" separator=",">
  (
   #{item.id}, #{item.v1}, #{item.v2}
  )
  </foreach>
  on duplicate key update v2 = v1 + values(v1)
</insert>
```

This mapper generates an `insert on duplicate key update` statement. The number of `(?,?,?)` following "values" is determined by the number of passed lists. Its final effect is similar to using `rewriteBatchStatements=true`, which also effectively reduces communication overhead between the client and TiDB.

As mentioned before, you also need to note that the Prepared Statements will not be cached after their maximum length exceeds the value of `prepStmtCacheSqlLimit`.

Streaming result

A previous section introduces how to stream read execution results in JDBC. In addition to the corresponding configurations of JDBC, if you want to read a super large result set in MyBatis, you also need to note that:

- You can set `fetchSize` for a single SQL statement in the mapper configuration (see the previous code block). Its effect is equivalent to calling `setFetchSize` in JDBC.

- You can use the query interface with `ResultHandler` to avoid getting the entire result set at once.
- You can use the `Cursor` class for stream reading.

If you configure mappings using XML, you can stream read results by configuring `fetchSize="-2147483648"(Integer.MIN_VALUE)` in the mapping's `<select>` section.

```
<select id="getAll" resultMap="postResultMap" fetchSize="-2147483648">
  select * from post;
</select>
```

If you configure mappings using code, you can add the `@Options(fetchSize = Integer .MIN_VALUE)` annotation and keep the type of results as `Cursor` so that the SQL results can be read in streaming.

```
@Select("select * from post")
@Options(fetchSize = Integer.MIN_VALUE)
Cursor<Post> queryAllPost();
```

12.5.2.4.2 ExecutorType

You can choose `ExecutorType` during `openSession`. MyBatis supports three types of executors:

- Simple: The Prepared Statements are called to JDBC for each execution (if the JDBC configuration item `cachePrepStmts` is enabled, repeated Prepared Statements will be reused)
- Reuse: The Prepared Statements are cached in `executor`, so that you can reduce duplicate calls for Prepared Statements without using the JDBC `cachePrepStmts`
- Batch: Each update operation (`INSERT`/`DELETE`/`UPDATE`) will first be added to the batch, and will be executed until the transaction commits or a `SELECT` query is performed. If `rewriteBatchStatements` is enabled in the JDBC layer, it will try to rewrite the statements. If not, the statements will be sent one by one.

Usually, the default value of `ExecutorType` is `Simple`. You need to change `ExecutorType` when calling `openSession`. If it is the batch execution, you might find that in a transaction the `UPDATE` or `INSERT` statements are executed pretty fast, but it is slower when reading data or committing the transaction. This is actually normal, so you need to note this when troubleshooting slow SQL queries.

12.5.2.5 Spring Transaction

In the real world, applications might use Spring Transaction and AOP aspects to start and stop transactions.

By adding the `@Transactional` annotation to the method definition, AOP starts the transaction before the method is called, and commits the transaction before the method returns the result. If your application has a similar need, you can find `@Transactional` in code to determine when the transaction is started and closed.

Pay attention to a special case of embedding. If it occurs, Spring will behave differently based on the Propagation configuration. Because TiDB does not support savepoint, nested transactions are not supported yet.

12.5.2.6 Misc

This section introduces some useful tools for Java to help you troubleshoot issues.

12.5.2.6.1 Troubleshooting tools

Using the powerful troubleshooting tools of JVM is recommended when an issue occurs in your Java application and you do not know the application logic. Here are a few common tools:

jstack

jstack is similar to pprof/goroutine in Go, which can easily troubleshoot the process stuck issue.

By executing `jstack pid`, you can output the IDs and stack information of all threads in the target process. By default, only the Java stack is output. If you want to output the C++ stack in the JVM at the same time, add the `-m` option.

By using jstack multiple times, you can easily locate the stuck issue (for example, a slow query from application's view due to using Batch ExecutorType in Mybatis) or the application deadlock issue (for example, the application does not send any SQL statement because it is preempting a lock before sending it).

In addition, `top -p $ PID -H` or Java swiss knife are common methods to view the thread ID. Also, to locate the issue of "a thread occupies a lot of CPU resources and I don't know what it is executing", do the following steps:

- Use `printf "%x\n" pid` to convert the thread ID to hexadecimal.
- Go to the jstack output to find the stack information of the corresponding thread.

jmap & mat

Unlike pprof/heap in Go, jmap dumps the memory snapshot of the entire process (in Go, it is the sampling of the distributor), and then the snapshot can be analyzed by another tool mat.

Through mat, you can see the associated information and attributes of all objects in the process, and you can also observe the running status of the thread. For example, you can use mat to find out how many MySQL connection objects exist in the current application, and what is the address and status information of each connection object.

Note that mat only handles reachable objects by default. If you want to troubleshoot young GC issues, you can adjust mat configuration to view unreachable objects. In addition, for investigating the memory allocation of young GC issues (or a large number of short-lived objects), using Java Flight Recorder is more convenient.

trace

Online applications usually do not support modifying the code, but it is often desired that dynamic instrumentation is performed in Java to locate issues. Therefore, using btrace or arthas trace is a good option. They can dynamically insert trace code without restarting the application process.

Flame graph

Obtaining flame graphs in Java applications is tedious. For details, see Java Flame Graphs Introduction: Fire For Everyone!.

12.5.2.7 Conclusion

Based on commonly used Java components that interact with databases, this document describes the common problems and solutions for developing Java applications with TiDB. TiDB is highly compatible with the MySQL protocol, so most of the best practices for MySQL-based Java applications also apply to TiDB.

Join us at TiDB Community slack channel, and share with broad TiDB user group about your experience or problems when you develop Java applications with TiDB.

12.5.3 Best Practices for Using HAProxy in TiDB

This document describes best practices for configuration and usage of HAProxy in TiDB. HAProxy provides load balancing for TCP-based applications. From TiDB clients, you can manipulate data just by connecting to the floating virtual IP address provided by HAProxy, which helps to achieve load balance in the TiDB server layer.

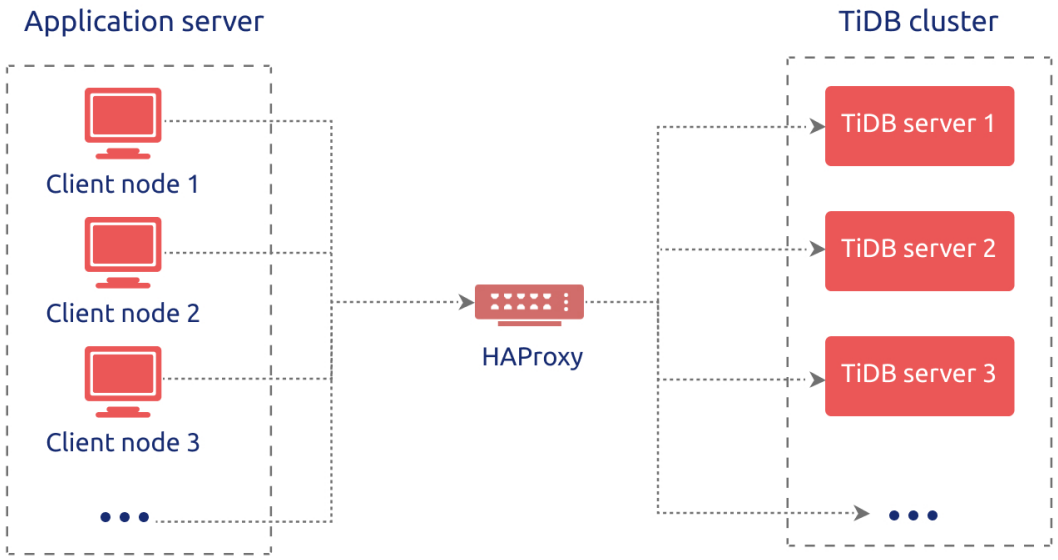


Figure 198: HAProxy Best Practices in TiDB

Note:

The minimum version of HAProxy that works with all versions of TiDB is v1.5. Between v1.5 and v2.1, you need to set the `post-41` option in `mysql-check`. It is recommended to use HAProxy v2.2 or newer.

12.5.3.1 HAProxy overview

HAProxy is free, open-source software written in C language that provides a high availability load balancer and proxy server for TCP and HTTP-based applications. Because of its fast and efficient use of CPU and memory, HAProxy is now widely used by many well-known websites such as GitHub, Bitbucket, Stack Overflow, Reddit, Tumblr, Twitter, Tuenti, and AWS (Amazon Web Services).

HAProxy is written in the year 2000 by Willy Tarreau, the core contributor to the Linux kernel, who is still responsible for the maintenance of the project and provides free software updates in the open-source community. In this guide, HAProxy 2.6 is used. It is recommended to use the latest stable version. See the released version of HAProxy for details.

12.5.3.2 Basic features

- High Availability: HAProxy provides high availability with support for a graceful shutdown and a seamless switchover;
- Load Balancing: Two major proxy modes are supported: TCP, also known as layer 4, and HTTP, also known as layer 7. No less than 9 load balancing algorithms are supported, such as roundrobin, leastconn and random;
- Health Check: HAProxy periodically checks the status of HTTP or TCP mode of the server;
- Sticky Session: HAProxy can stick a client to a specific server for the duration when the application does not support sticky sessions;
- SSL: HTTPS communication and resolution are supported;
- Monitoring and Statistics: Through the web page, you can monitor the service state and traffic flow in real time.

12.5.3.3 Before you begin

Before you deploy HAProxy, make sure that you meet the hardware and software requirements.

12.5.3.3.1 Hardware requirements

For your server, it is recommended to meet the following hardware requirements. You can also improve server specifications according to the load balancing environment.

Hardware resource	Minimum specification
CPU	2 cores, 3.5 GHz
Memory	16 GB
Storage	50 GB (SATA)
Network Interface Card	10G Network Card

12.5.3.3.2 Software requirements

You can use the following operating systems and make sure the required dependencies are installed. If you use yum to install HAProxy, the dependencies are installed along with it and you do not need to separately install them again.

Operating systems

Linux distribution	Version
Red Hat Enterprise Linux	7 or 8
CentOS	7 or 8
Oracle Enterprise Linux	7 or 8
Ubuntu LTS	18.04 or later versions

Note:

- For more information about other supported operating systems, see HAProxy documentation.

Dependencies

- epel-release
- gcc
- systemd-devel

To install the dependencies above, run the following command:

```
yum -y install epel-release gcc systemd-devel
```

12.5.3.4 Deploy HAProxy

You can easily use HAProxy to configure and set up a load-balanced database environment. This section shows general deployment operations. You can customize the configuration file based on your actual scenario.

12.5.3.4.1 Install HAProxy

1. Download the package of the HAProxy 2.6.2 source code:

```
wget https://www.haproxy.org/download/2.6/src/haproxy-2.6.2.tar.gz
```

2. Extract the package:

```
tar zxf haproxy-2.6.2.tar.gz
```

3. Compile the application from the source code:

```
cd haproxy-2.6.2
make clean
make -j 8 TARGET=linux-glibc USE_THREAD=1
make PREFIX=${/app/haproxy} SBINDIR=${/app/haproxy/bin} install #
    ↪ Replace `${/app/haproxy}` and `${/app/haproxy/bin}` with your
    ↪ custom directories.
```

4. Reconfigure the profile:

```
echo 'export PATH=/app/haproxy/bin:$PATH' >> /etc/profile
. /etc/profile
```

5. Check whether the installation is successful:

```
which haproxy
```

HAProxy commands

Execute the following command to print a list of keywords and their basic usage:

```
haproxy --help
```

Option	Description
`-v`	Reports the version and build date.
`-vv`	Displays the version, build options, libraries versions and usable pollers.
`-d`	Enables debug mode.
`-db`	Disables background mode and multi-process mode.

Option	Description
`-dM [< ↪ byte ↪ >]`	Forces memory poisoning, which means that each and every memory region allocated with malloc() or pool_alloc2() will be filled with `<byte>` before being passed to the caller.
`-V`	Enables verbose mode (disables quiet mode).
`-D`	Starts as a daemon.
`-C <dir>`	Changes to directory `<dir>` before loading configuration files.
`-W`	Master-worker mode.

Option	Description
`-q`	Sets "quiet" mode: This disables some messages during the configuration parsing and during startup.
`-c`	Only performs a check of the configuration files and exits before trying to bind.
`-n <limit>`	Limits the per-process connection limit to `<limit>`.
`-m <limit>`	Limits the total allocatable memory to `<limit>` megabytes across all processes.

Option	Description
`-N <limit>`	Sets the default per-proxy maxconn to `<limit>` instead of the builtin default value (usually 2000).
`-L <name>`	Changes the local peer name to `<name>`, which defaults to the local hostname.
`-p <file>`	Writes all processes' PIDs into `<file>` during startup.
`-de`	Disables the use of epoll(7). epoll(7) is available only on Linux 2.6 and some custom Linux 2.4 systems.
`-dp`	Disables the use of poll(2). select(2) might be used instead.

Option	Description
`-dS`	Disables the use of splice(2), which is broken on older kernels.
`-dR`	Disables SO_REUSEPORT usage.
`-dr`	Ignores server address resolution failures.
`-dV`	Disables SSL verify on the server side.

Option	Description
`-sf <pidlist>`	Sends the "finish" signal to the PIDs in pidlist after startup. The processes which receive this signal wait for all sessions to finish before exiting. This option must be specified last, followed by any number of PIDs. Technically speaking, SIGTTOU and SIGUSR1 are sent.

Option	Description
`-st <` ↪ `pidlist` ↪ `>`	Sends the “terminate” signal to the PIDs in pidlist after startup. The processes which receive this signal terminate immediately, closing all active sessions. This option must be specified last, followed by any number of PIDs. Technically speaking, SIGTTOU and SIGTERM are sent.

Option	Description
`-x <unix_socket>`	Connects to the specified socket and retrieves all the listening sockets from the old process. Then, these sockets are used instead of binding new ones.
`-S <bind>[,<bind_options>...]`	In master-worker mode, creates a master CLI. This CLI enables access to the CLI of every worker. Useful for debugging, it's a convenient way of accessing a leaving process.

For more details on HAProxy command line options, refer to Management Guide of HAProxy and General Commands Manual of HAProxy.

12.5.3.4.2 Configure HAProxy

A configuration template is generated when you use yum to install HAProxy. You can also customize the following configuration items according to your scenario.

```
global                                  # Global configuration.
   log         127.0.0.1 local2         # Global syslog servers (up to two).
   chroot      /var/lib/haproxy         # Changes the current directory and
      ↪ sets superuser privileges for the startup process to improve
      ↪ security.
   pidfile     /var/run/haproxy.pid     # Writes the PIDs of HAProxy processes
      ↪  into this file.
   maxconn     4096                     # The maximum number of concurrent
      ↪ connections for a single HAProxy process. It is equivalent to the
      ↪ command-line argument "-n".
   nbthread    48                       # The maximum number of threads. (The
      ↪ upper limit is equal to the number of CPUs)
   user        haproxy                  # Same with the UID parameter.
   group       haproxy                  # Same with the GID parameter. A
      ↪ dedicated user group is recommended.
   daemon                               # Makes the process fork into
      ↪ background. It is equivalent to the command line "-D" argument. It
      ↪ can be disabled by the command line "-db" argument.
   stats socket /var/lib/haproxy/stats # The directory where statistics
      ↪ output is saved.

defaults                                # Default configuration.
   log global                           # Inherits the settings of the global
      ↪ configuration.
   retries 2                            # The maximum number of retries to
      ↪ connect to an upstream server. If the number of connection attempts
      ↪  exceeds the value, the backend server is considered unavailable.
   timeout connect 2s                   # The maximum time to wait for a
      ↪ connection attempt to a backend server to succeed. It should be set
      ↪  to a shorter time if the server is located on the same LAN as
      ↪ HAProxy.
   timeout client 30000s                # The maximum inactivity time on the
      ↪ client side.
   timeout server 30000s                # The maximum inactivity time on the
      ↪ server side.

listen admin_stats                      # The name of the Stats page reporting
   ↪  information from frontend and backend. You can customize the name
   ↪ according to your needs.
   bind 0.0.0.0:8080                    # The listening port.
   mode http                            # The monitoring mode.
   option httplog                       # Enables HTTP logging.
   maxconn 10                           # The maximum number of concurrent
      ↪ connections.
```

```
  stats refresh 30s                   # Automatically refreshes the Stats
     ↪ page every 30 seconds.
  stats uri /haproxy                  # The URL of the Stats page.
  stats realm HAProxy                 # The authentication realm of the
     ↪ Stats page.
  stats auth admin:pingcap123         # User name and password in the Stats
     ↪ page. You can have multiple user names.
  stats hide-version                  # Hides the version information of
     ↪ HAProxy on the Stats page.
  stats admin if TRUE                 # Manually enables or disables the
     ↪ backend server (supported in HAProxy 1.4.9 or later versions).

listen tidb-cluster                   # Database load balancing.
  bind 0.0.0.0:3390                   # The Floating IP address and
     ↪ listening port.
  mode tcp                            # HAProxy uses layer 4, the transport
     ↪ layer.
  balance leastconn                   # The server with the smallest number
     ↪ of connections receives the connection. "leastconn" is recommended
     ↪ where long sessions are expected, such as LDAP, SQL and TSE, rather
     ↪  than protocols using short sessions, such as HTTP. The algorithm
     ↪ is dynamic, which means that server weights might be adjusted on
     ↪ the fly for slow starts for instance.
  server tidb-1 10.9.18.229:4000 check inter 2000 rise 2 fall 3 # Detects
     ↪ port 4000 at a frequency of once every 2000 milliseconds. If it is
     ↪ detected as successful twice, the server is considered available;
     ↪ if it is detected as failed three times, the server is considered
     ↪ unavailable.
  server tidb-2 10.9.39.208:4000 check inter 2000 rise 2 fall 3
  server tidb-3 10.9.64.166:4000 check inter 2000 rise 2 fall 3
```

To check the source IP address using `SHOW PROCESSLIST`, you need to configure the PROXY protocol to connect to TiDB.

```
  server tidb-1 10.9.18.229:4000 send-proxy check inter 2000 rise 2 fall 3
  server tidb-2 10.9.39.208:4000 send-proxy check inter 2000 rise 2 fall 3
  server tidb-3 10.9.64.166:4000 send-proxy check inter 2000 rise 2 fall 3
```

Note:

Before using the PROXY protocol, you need to configure `proxy-protocol.networks` in the configuration file of the TiDB server.

12.5.3.4.3 Start HAProxy

To start HAProxy, run `haproxy`. `/etc/haproxy/haproxy.cfg` is read by default (recommended).

```
haproxy -f /etc/haproxy/haproxy.cfg
```

12.5.3.4.4 Stop HAProxy

To stop HAProxy, use the `kill -9` command.

1. Run the following command:

```
ps -ef | grep haproxy
```

2. Terminate the process of HAProxy:

```
kill -9 ${haproxy.pid}
```

12.5.4 Highly Concurrent Write Best Practices

This document describes best practices for handling highly-concurrent write-heavy workloads in TiDB, which can help to facilitate your application development.

12.5.4.1 Target audience

This document assumes that you have a basic understanding of TiDB. It is recommended that you first read the following three blog articles that explain TiDB fundamentals, and TiDB Best Practices:

- Data Storage
- Computing
- Scheduling

12.5.4.2 Highly-concurrent write-intensive scenario

The highly concurrent write scenario often occurs when you perform batch tasks in applications, such as clearing and settlement. This scenario has the following features:

- A huge volume of data
- The need to import historical data into database in a short time
- The need to read a huge volume of data from database in a short time

These features pose these challenges to TiDB:

- The write or read capacity must be linearly scalable.
- Database performance is stable and does not decrease as a huge volume of data is written concurrently.

For a distributed database, it is important to make full use of the capacity of all nodes and to prevent a single node from becoming the bottleneck.

12.5.4.3 Data distribution principles in TiDB

To address the above challenges, it is necessary to start with the data segmentation and scheduling principle of TiDB. Refer to Scheduling for more details.

TiDB splits data into Regions, each representing a range of data with a size limit of 96M by default. Each Region has multiple replicas, and each group of replicas is called a Raft Group. In a Raft Group, the Region Leader executes the read and write tasks (TiDB supports Follower-Read) within the data range. The Region Leader is automatically scheduled by the Placement Driver (PD) component to different physical nodes evenly to distribute the read and write pressure.

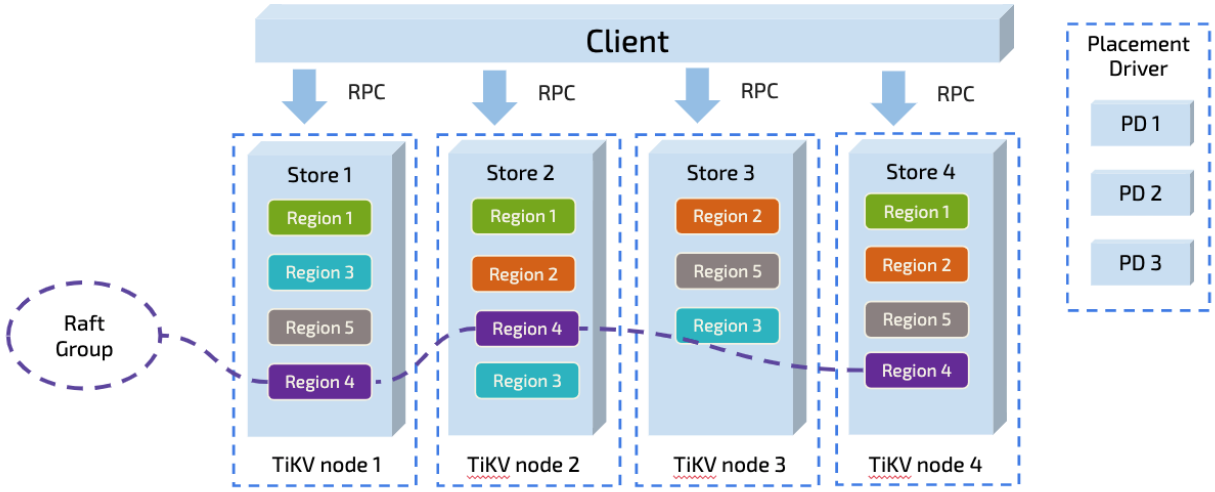


Figure 199: TiDB Data Overview

In theory, if an application has no write hotspot, TiDB, by the virtue of its architecture, can not only linearly scale its read and write capacities, but also make full use of the distributed resources. From this point of view, TiDB is especially suitable for the high-concurrent and write-intensive scenario.

However, the actual situation often differs from the theoretical assumption.

> **Note:**
>
> No write hotspot in an application means the write scenario does not have any `AUTO_INCREMENT` primary key or monotonically increasing index.

12.5.4.4 Hotspot case

The following case explains how a hotspot is generated. Take the table below as an example:

```
CREATE TABLE IF NOT EXISTS TEST_HOTSPOT(
     id        BIGINT PRIMARY KEY,
     age       INT,
     user_name VARCHAR(32),
     email     VARCHAR(128)
)
```

This table is simple in structure. In addition to `id` as the primary key, no secondary index exists. Execute the following statement to write data into this table. `id` is discretely generated as a random number.

```
SET SESSION cte_max_recursion_depth = 1000000;
INSERT INTO TEST_HOTSPOT
SELECT
  n,                                    -- ID
  RAND()*80,                            -- Number between 0 and 80
  CONCAT('user-',n),
  CONCAT(
    CHAR(65 + (RAND() * 25) USING ascii), -- Number between 65 and 65+25,
       ↪ converted to a character, A-Z
    '-user-',
    n,
    '@example.com'
  )
FROM
  (WITH RECURSIVE nr(n) AS
    (SELECT 1                           -- Start CTE at 1
      UNION ALL SELECT n + 1            -- increase n with 1 every loop
      FROM nr WHERE n < 1000000         -- stop loop at 1_000_000
    ) SELECT n FROM nr
  ) a;
```

The load comes from executing the above statement intensively in a short time.

In theory, the above operation seems to comply with the TiDB best practices, and no hotspot is caused in the application. The distributed capacity of TiDB can be fully used with adequate machines. To verify whether it is truly in line with the best practices, a test is conducted in the experimental environment, which is described as follows:

For the cluster topology, 2 TiDB nodes, 3 PD nodes and 6 TiKV nodes are deployed. Ignore the QPS performance, because this test is to clarify the principle rather than for benchmark.

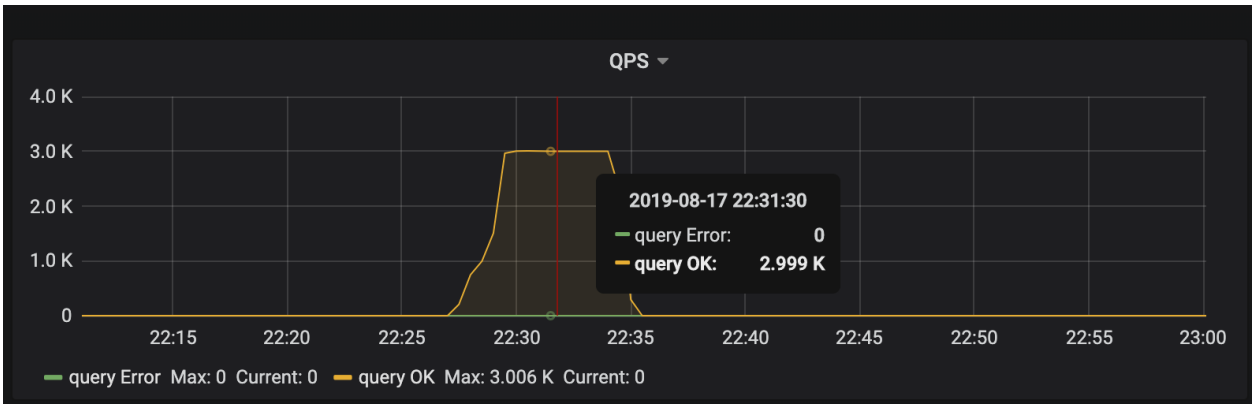


Figure 200: QPS1

The client starts "intensive" write requests in a short time, which is 3K QPS received by TiDB. In theory, the load pressure should be evenly distributed to 6 TiKV nodes. However, from the CPU usage of each TiKV node, the load distribution is uneven. The `tikv-3` node is the write hotspot.

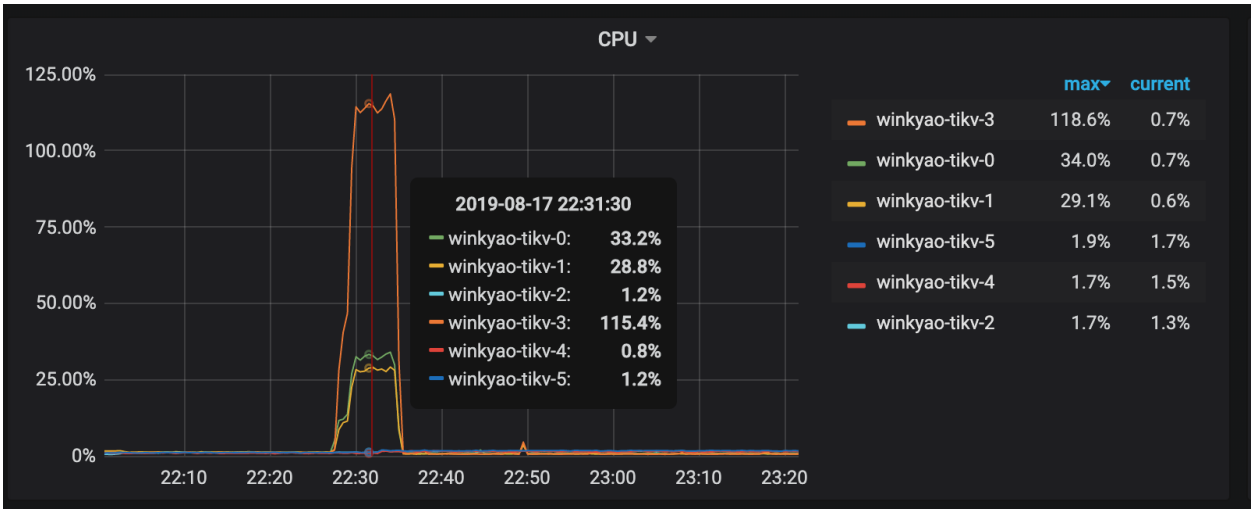


Figure 201: QPS2

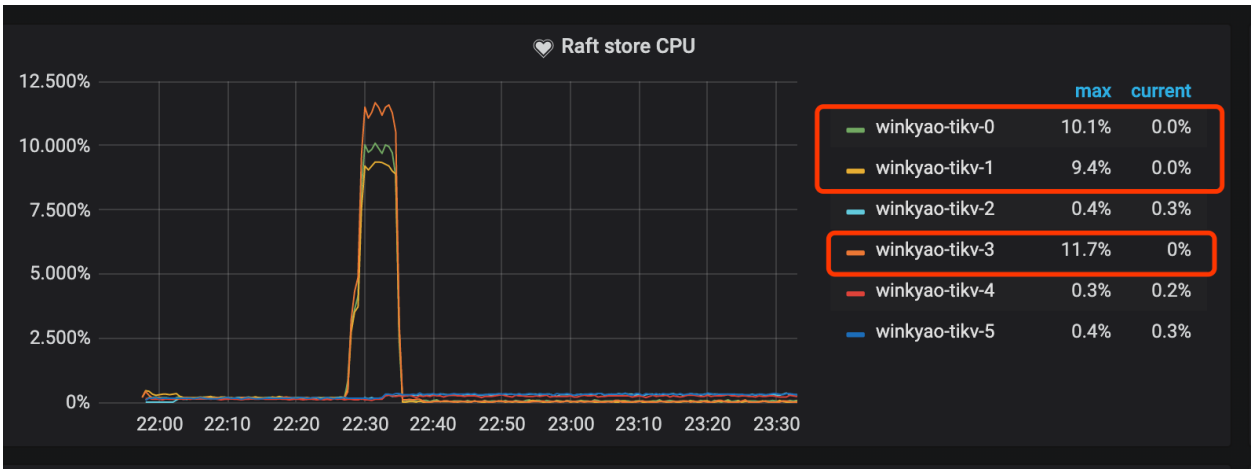


Figure 202: QPS3

Raft store CPU is the CPU usage rate for the `raftstore` thread, usually representing the write load. In this scenario, `tikv-3` is the Leader of this Raft Group; `tikv-0` and `tikv-1` are the followers. The loads of other nodes are almost empty.

The monitoring metrics of PD also confirms that hotspot has been caused.

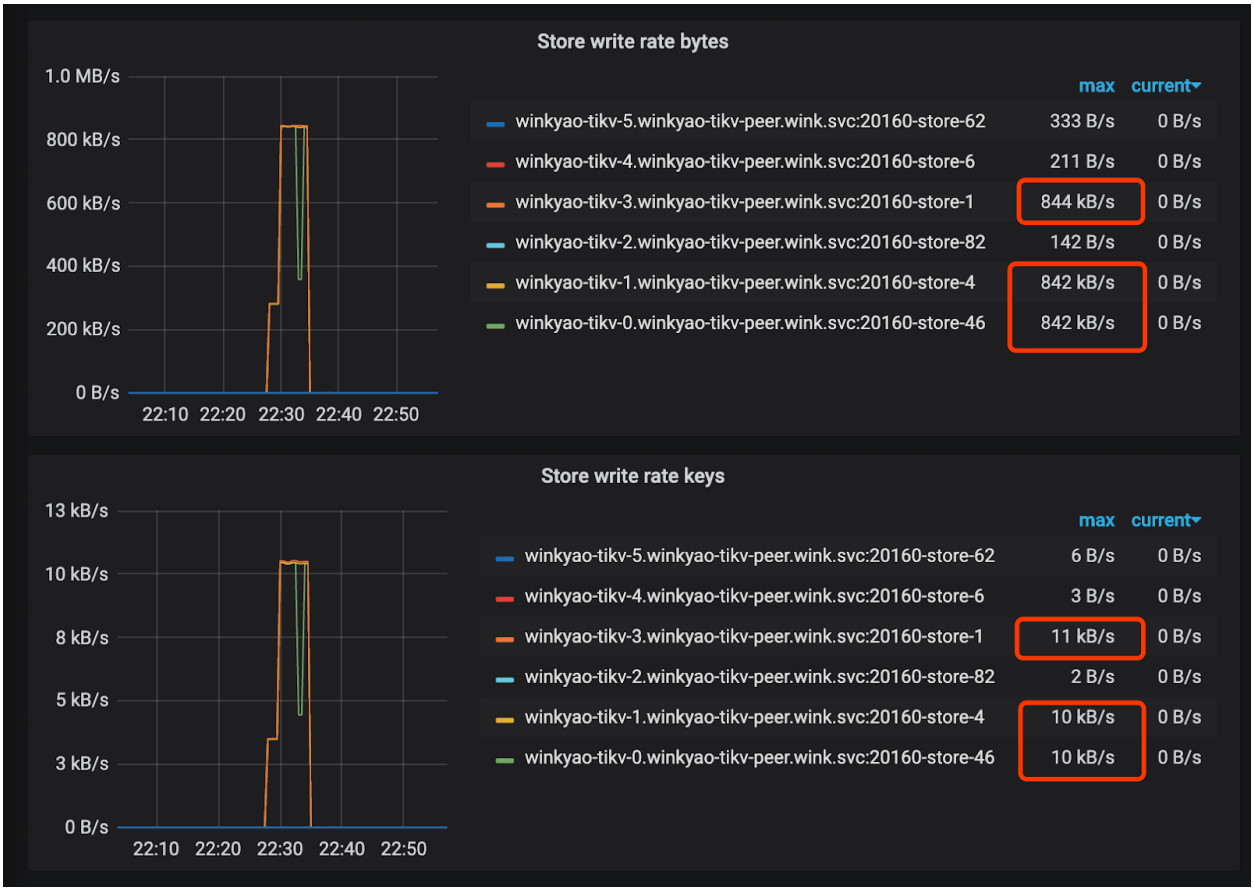


Figure 203: QPS4

12.5.4.5 Hotspot causes

In the above test, the operation does not reach the ideal performance expected in the best practices. This is because only one Region is split by default to store the data of each newly created table in TiDB, with the following data range:

```
[CommonPrefix + TableID, CommonPrefix + TableID + 1)
```

In a short period of time, a huge volume of data is continuously written to the same Region.

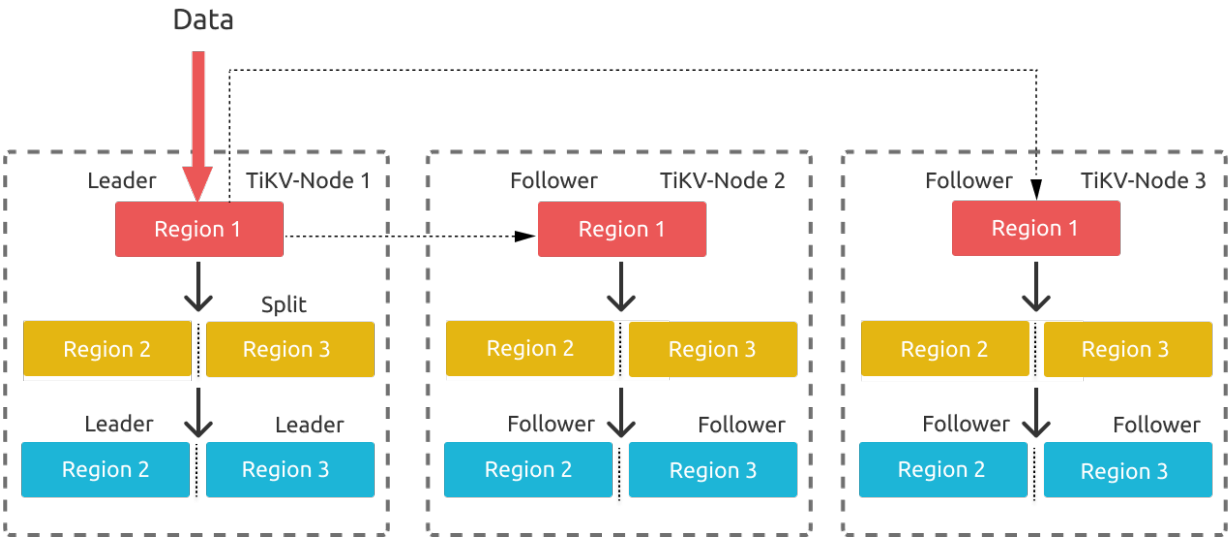


Figure 204: TiKV Region Split

The above diagram illustrates the Region splitting process. As data is continuously written into TiKV, TiKV splits a Region into multiple Regions. Because the leader election is started on the original store where the Region Leader to be split is located, the leaders of the two newly split Regions might be still on the same store. This splitting process might also happen on the newly split Region 2 and Region 3. In this way, write pressure is concentrated on TiKV-Node 1.

During the continuous write process, after finding that hotspot is caused on Node 1, PD evenly distributes the concentrated Leaders to other nodes. If the number of TiKV nodes is more than the number of Region replicas, TiKV will try to migrate these Regions to idle nodes. These two operations during the write process are also reflected in the PD's monitoring metrics:

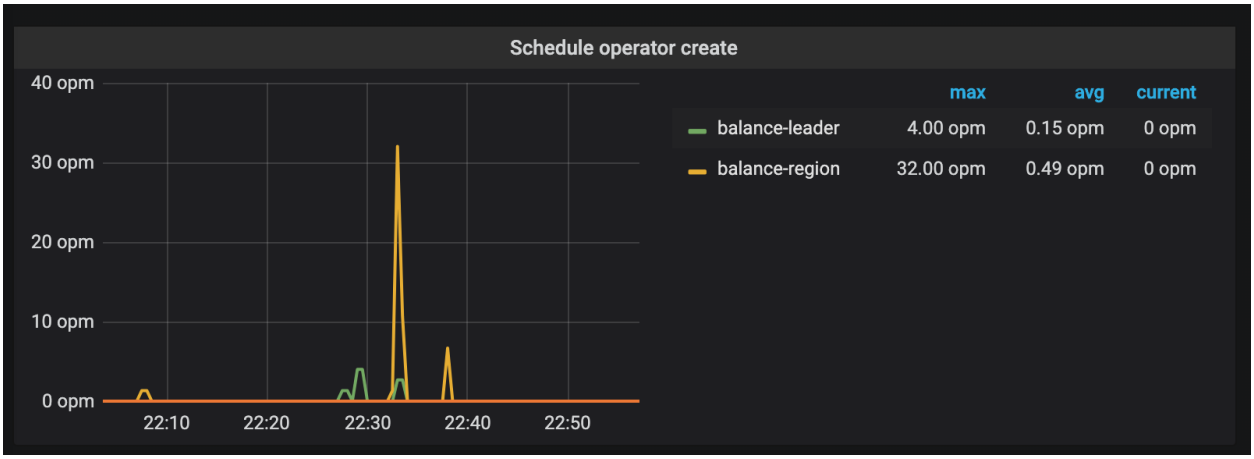


Figure 205: QPS5

After a period of continuous writes, PD automatically schedules the entire TiKV cluster to a state where pressure is evenly distributed. By that time, the capacity of the whole cluster can be fully used.

In most cases, the above process of causing a hotspot is normal, which is the Region warm-up phase of database. However, you need to avoid this phase in highly-concurrent write-intensive scenarios.

12.5.4.6 Hotspot solution

To achieve the ideal performance expected in theory, you can skip the warm-up phase by directly splitting a Region into the desired number of Regions and scheduling these Regions in advance to other nodes in the cluster.

In v3.0.x, v2.1.13 and later versions, TiDB supports a new feature called Split Region. This new feature provides the following new syntaxes:

```
SPLIT TABLE table_name [INDEX index_name] BETWEEN (lower_value) AND (
    ↪ upper_value) REGIONS region_num
```

```
SPLIT TABLE table_name [INDEX index_name] BY (value_list) [, (value_list)]
```

However, TiDB does not automatically perform this pre-split operation. The reason is related to the data distribution in TiDB.

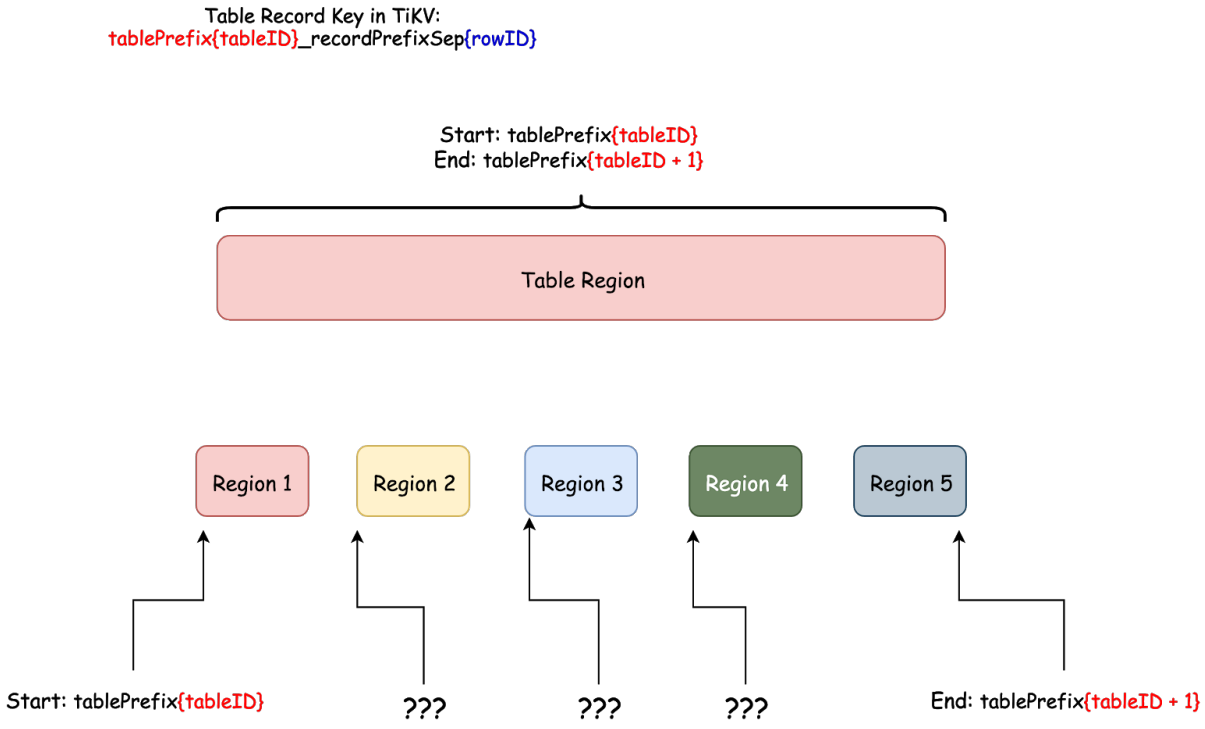


Figure 206: Table Region Range

From the diagram above, according to the encoding rule of a row's key, the `rowID` is the only variable part. In TiDB, `rowID` is an `Int64` integer. However, you might not need to evenly split the `Int64` integer range to the desired number of ranges and then to distribute these ranges to different nodes, because Region split must also be based on the actual situation.

If the write of `rowID` is completely discrete, the above method will not cause hotspots. If the row ID or index has a fixed range or prefix (for example, discretely insert data into the range of `[2000w, 5000w)`), no hotspot will be caused either. However, if you split a Region using the above method, data might still be written to the same Region at the beginning.

TiDB is a database for general usage and does not make assumptions about the data distribution. So it uses only one Region at the beginning to store the data of a table and automatically splits the Region according to the data distribution after real data is inserted.

Given this situation and the need to avoid the hotspot problem, TiDB offers the `Split Region` syntax to optimize performance for the highly-concurrent write-heavy scenario. Based on the above case, now scatter Regions using the `Split Region` syntax and observe the load distribution.

Because the data to be written in the test is entirely discrete within the positive range, you can use the following statement to pre-split the table into 128 Regions within the range of `minInt64` and `maxInt64`:

```
SPLIT TABLE TEST_HOTSPOT BETWEEN (0) AND (9223372036854775807) REGIONS 128;
```

After the pre-split operation, execute the `SHOW TABLE test_hotspot REGIONS;` statement to check the status of Region scattering. If the values of the `SCATTERING` column are all `0`, the scheduling is successful.

You can also check the Region leader distribution using the following SQL statement. You need to replace `table_name` with the actual table name.

```
SELECT
    p.STORE_ID,
    COUNT(s.REGION_ID) PEER_COUNT
FROM
    INFORMATION_SCHEMA.TIKV_REGION_STATUS s
    JOIN INFORMATION_SCHEMA.TIKV_REGION_PEERS p ON s.REGION_ID = p.REGION_ID
WHERE
    TABLE_NAME = 'table_name'
    AND p.is_leader = 1
GROUP BY
    p.STORE_ID
ORDER BY
    PEER_COUNT DESC;
```

Then operate the write load again:

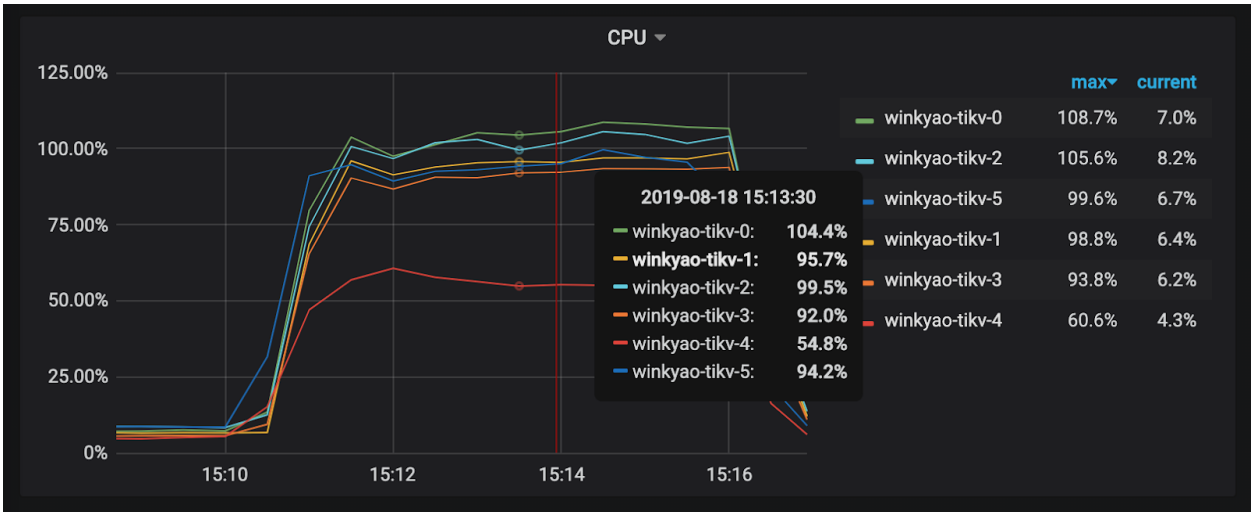


Figure 207: QPS6

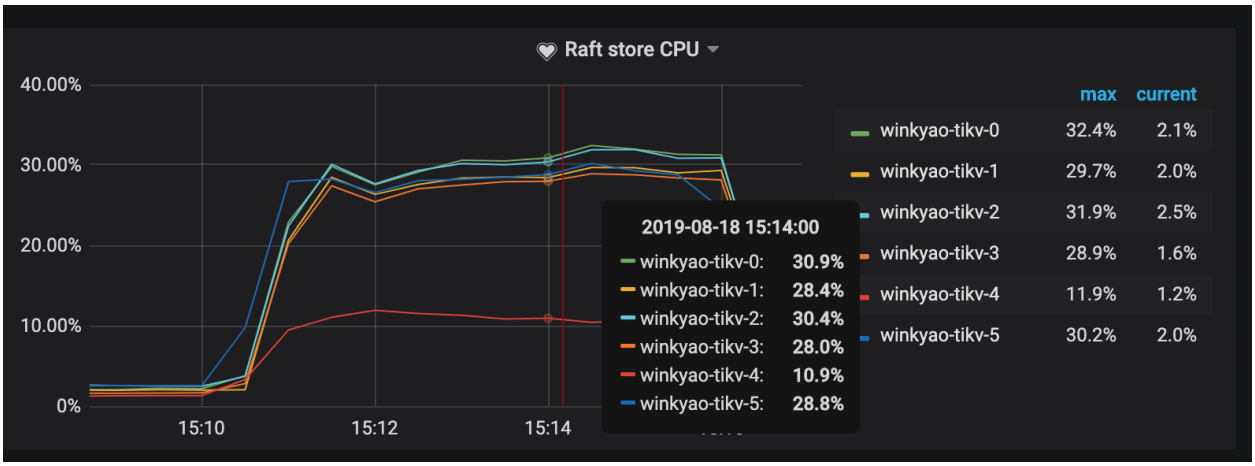


Figure 208: QPS7

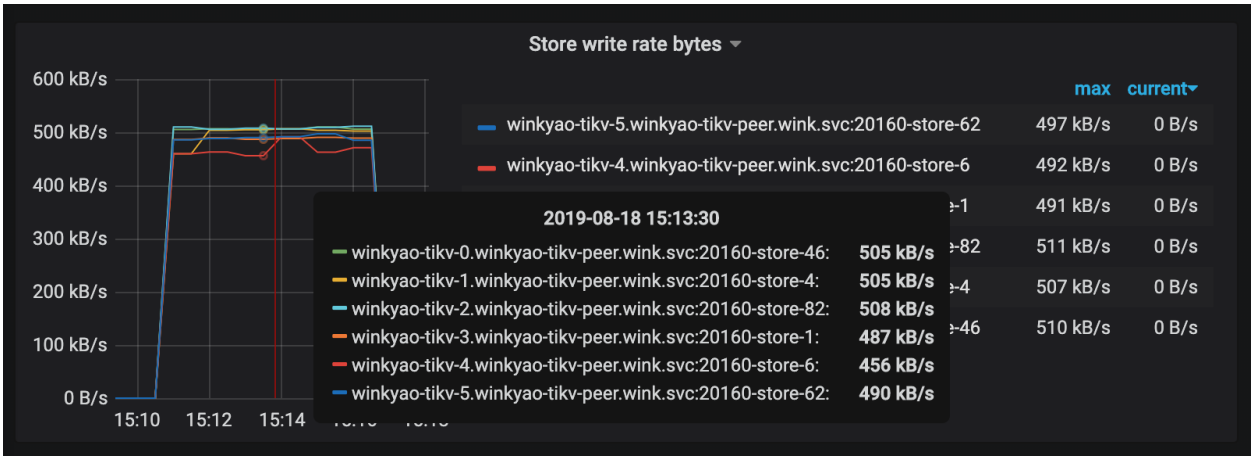


Figure 209: QPS8

You can see that the apparent hotspot problem has been resolved now.

In this case, the table is simple. In other cases, you might also need to consider the hotspot problem of index. For more details on how to pre-split the index Region, refer to Split Region.

12.5.4.7 Complex hotspot problems

Problem one:

If a table does not have a primary key, or the primary key is not the `Int` type and you do not want to generate a randomly distributed primary key ID, TiDB provides an implicit `_tidb_rowid` column as the row ID. Generally, when you do not use the `SHARD_ROW_ID_BITS` parameter, the values of the `_tidb_rowid` column are also monotonically increasing, which might causes hotspots too. Refer to `SHARD_ROW_ID_BITS` for more details.

To avoid the hotspot problem in this situation, you can use `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` when creating a table. For more details about `PRE_SPLIT_REGIONS`, refer to Pre-split Regions.

`SHARD_ROW_ID_BITS` is used to randomly scatter the row ID generated in the `_tidb_rowid` column. `PRE_SPLIT_REGIONS` is used to pre-split the Region after a table is created.

> **Note:**
>
> The value of `PRE_SPLIT_REGIONS` must be smaller than or equal to that of `SHARD_ROW_ID_BITS`.

Example:

```
create table t (a int, b int) SHARD_ROW_ID_BITS = 4 PRE_SPLIT_REGIONS=3;
```

- `SHARD_ROW_ID_BITS = 4` means that the values of `tidb_rowid` will be randomly distributed into 16 (16=2^4) ranges.
- `PRE_SPLIT_REGIONS=3` means that the table will be pre-split into 8 (2^3) Regions after it is created.

When data starts to be written into table `t`, the data is written into the pre-split 8 Regions, which avoids the hotspot problem that might be caused if only one Region exists after table creation.

> **Note:**
>
> The `tidb_scatter_region` global variable affects the behavior of `PRE_SPLIT_REGIONS`.
>
> This variable controls whether to wait for Regions to be pre-split and scattered before returning results after the table creation. If there are intensive writes after creating the table, you need to set the value of this variable to `1`, then TiDB will not return the results to the client until all the Regions are split and scattered. Otherwise, TiDB writes data before the scattering is completed, which will have a significant impact on write performance.

Problem two:

If a table's primary key is an integer type, and if the table uses `AUTO_INCREMENT` to ensure the uniqueness of the primary key (not necessarily continuous or incremental), you cannot use `SHARD_ROW_ID_BITS` to scatter the hotspot on this table because TiDB directly uses the row values of the primary key as `_tidb_rowid`.

To address the problem in this scenario, you can replace `AUTO_INCREMENT` with `AUTO_RANDOM` (a column attribute) when inserting data. Then TiDB automatically assigns values to the integer primary key column, which eliminates the continuity of the row ID and scatters the hotspot.

12.5.4.8 Parameter configuration

In v2.1, the latch mechanism is introduced in TiDB to identify transaction conflicts in advance in scenarios where write conflicts frequently appear. The aim is to reduce the retry of transaction commits in TiDB and TiKV caused by write conflicts. Generally, batch tasks use the data already stored in TiDB, so the write conflicts of transaction do not exist. In this situation, you can disable the latch in TiDB to reduce memory allocation for small objects:

```
[txn-local-latches]
enabled = false
```

12.5.5 Best Practices for Monitoring TiDB Using Grafana

When you deploy a TiDB cluster using TiUP and have added Grafana and Prometheus in the topology configuration, a set of Grafana + Prometheus monitoring platform is deployed simultaneously to collect and display metrics for various components and machines in the TiDB cluster. This document describes best practices for monitoring TiDB using Grafana. It aims to help you use metrics to analyze the status of the TiDB cluster and diagnose problems.

12.5.5.1 Monitoring architecture

Prometheus is a time series database with a multi-dimensional data model and a flexible query language. Grafana is an open source monitoring system for analyzing and visualizing metrics.

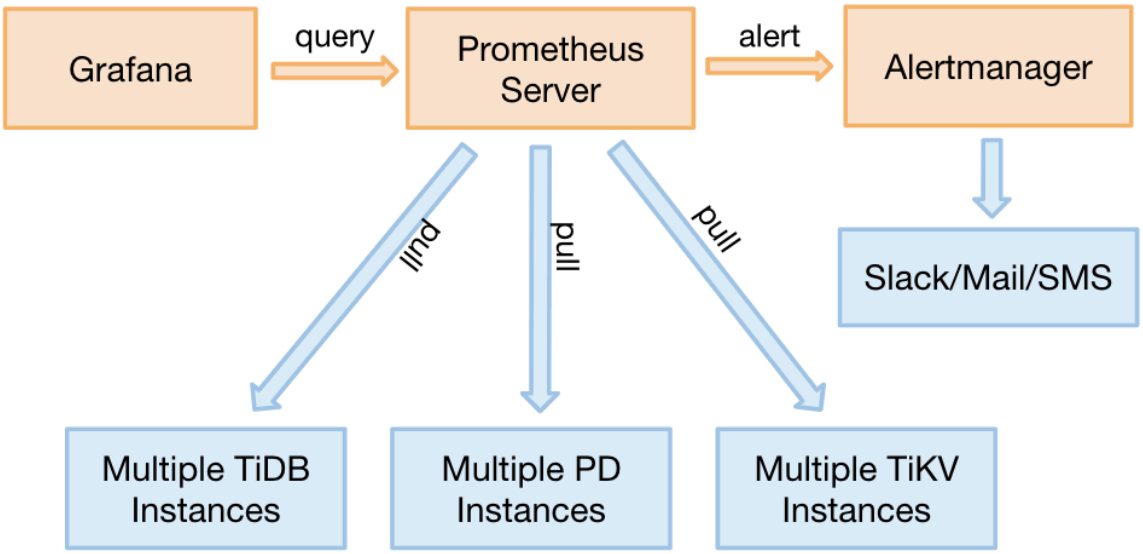


Figure 210: The monitoring architecture in the TiDB cluster

For TiDB 2.1.3 or later versions, TiDB monitoring supports the pull method. It is a good adjustment with the following benefits:

- There is no need to restart the entire TiDB cluster if you need to migrate Prometheus. Before adjustment, migrating Prometheus requires restarting the entire cluster because the target address needs to be updated.

- You can deploy 2 separate sets of Grafana + Prometheus monitoring platforms (not highly available) to prevent a single point of monitoring.
- The Pushgateway which might become a single point of failure is removed.

12.5.5.2 Source and display of monitoring data

The three core components of TiDB (TiDB server, TiKV server and PD server) obtain metrics through the HTTP interface. These metrics are collected from the program code, and the default ports are as follows:

Component	Port
TiDB server	10080
TiKV server	20180
PD server	2379

Execute the following command to check the QPS of a SQL statement through the HTTP interface. Take the TiDB server as an example:

```
curl http://__tidb_ip__:10080/metrics |grep tidb_executor_statement_total
```

```
### Check the real-time QPS of different types of SQL statements. The
    ↪ numbers below are the cumulative values of counter type (scientific
    ↪ notation).
tidb_executor_statement_total{type="Delete"} 520197
tidb_executor_statement_total{type="Explain"} 1
tidb_executor_statement_total{type="Insert"} 7.20799402e+08
tidb_executor_statement_total{type="Select"} 2.64983586e+08
tidb_executor_statement_total{type="Set"} 2.399075e+06
tidb_executor_statement_total{type="Show"} 500531
tidb_executor_statement_total{type="Use"} 466016
```

The data above is stored in Prometheus and displayed on Grafana. Right-click the panel and then click the **Edit** button (or directly press the E key) shown in the following figure:

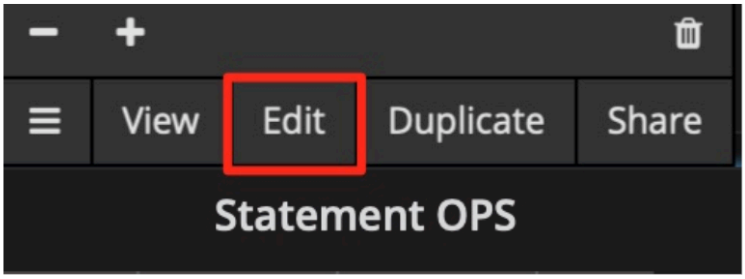


Figure 211: The Edit entry for the Metrics tab

After clicking the **Edit** button, you can see the query expression with the `tidb_executor_statement_t` ↪ metric name on the Metrics tab. The meanings of some items on the panel are as follows:

- `rate[1m]`: The growth rate in one minute. It can only be used for the data of counter type.
- `sum`: The sum of values.
- `by type`: The summed data is grouped by type in the original metric value.
- `Legend format`: The format of the metric name.
- `Resolution`: The step width defaults to 15 seconds. Resolution means whether to generate one data point for multiple pixels.

The query expression on the **Metrics** tab is as follows:



Figure 212: The query expression on the Metrics tab

Prometheus supports many query expressions and functions. For more details, refer to Prometheus official website.

12.5.5.3 Grafana tips

This section introduces seven tips for efficiently using Grafana to monitor and analyze the metrics of TiDB.

12.5.5.3.1 Tip 1: Check all dimensions and edit the query expression

In the example shown in the source and display of monitoring data section, the data is grouped by type. If you want to know whether you can group by other dimensions and quickly check which dimensions are available, you can use the following method: **Only keep**

the metric name on the query expression, no calculation, and leave the `Legend format` field blank. In this way, the original metrics are displayed. For example, the following figure shows that there are three dimensions (`instance`, `job` and `type`):

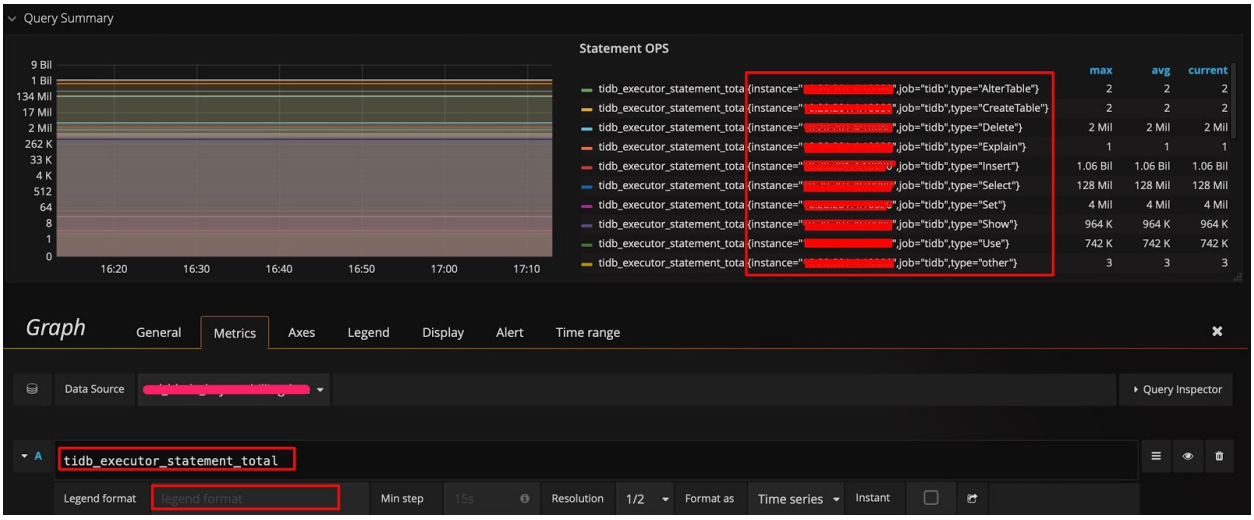


Figure 213: Edit query expression and check all dimensions

Then you can modify the query expression by adding the `instance` dimension after `type`, and adding `{{instance}}` to the `Legend format` field. In this way, you can check the QPS of different types of SQL statements that are executed on each TiDB server:

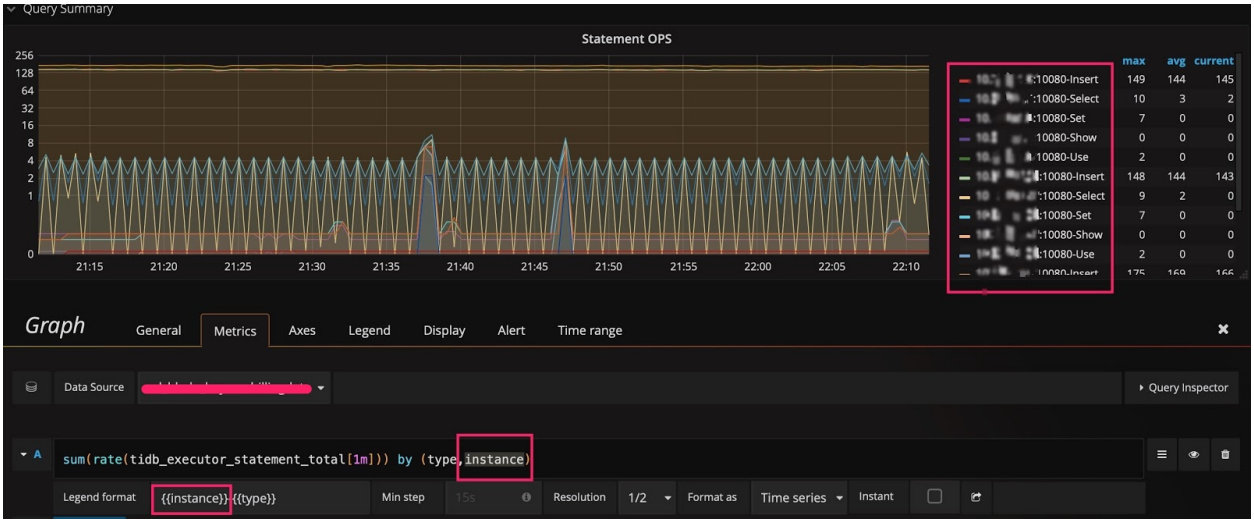


Figure 214: Add an instance dimension to the query expression

12.5.5.3.2 Tip 2: Switch the scale of the Y-axis

Take Query Duration as an example, the Y-axis defaults to be on a binary logarithmic scale (log2n), which narrows the gap in display. To amplify changes, you can switch it to a linear scale. Comparing the following two figures, you can easily notice the difference in display, and locate the time when an SQL statement runs slowly.

Of course, a linear scale is not suitable for all situations. For example, if you observe the performance trend for the duration of a month, there might be noises with a linear scale, making it hard to observe.

The Y-axis uses a binary logarithmic scale by default:

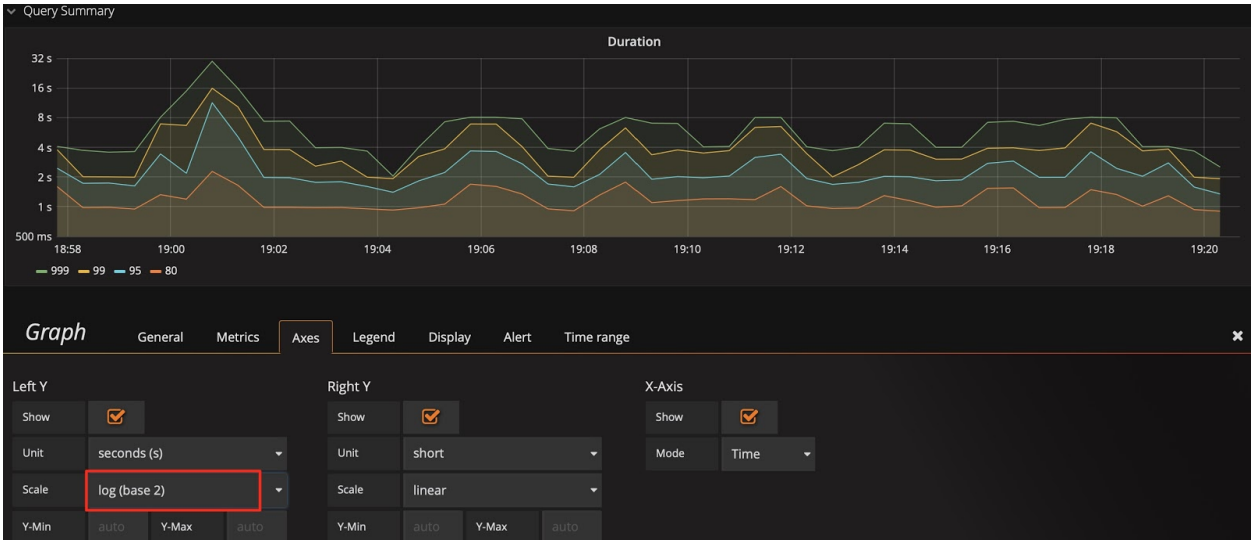


Figure 215: The Y-axis uses a binary logarithmic scale

Switch the Y-axis to a linear scale:

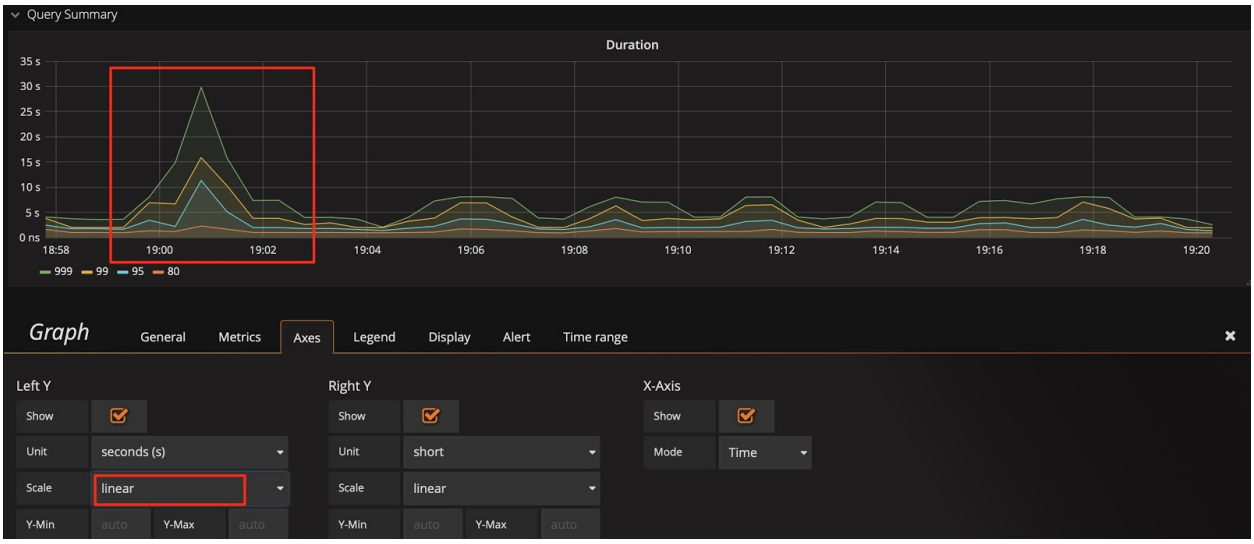


Figure 216: Switch to a linear scale

Tip:

Combining tip 2 with tip 1, you can find a `sql_type` dimension to help you immediately analyze whether the `SELECT` statement or the `UPDATE` statement is slow; you can even locate the instance with slow SQL statements.

12.5.5.3.3 Tip 3: Modify the baseline of the Y-axis to amplify changes

You might still cannot see the trend after switching to the linear scale. For example, in the following figure, you want to observe the real-time change of `Store size` after scaling the cluster, but due to the large baseline, small changes are not visible. In this situation, you can modify the baseline of the Y-axis from `0` to `auto` to zoom in the upper part. Check the two figures below, you can see data migration begins.

The baseline defaults to `0`:

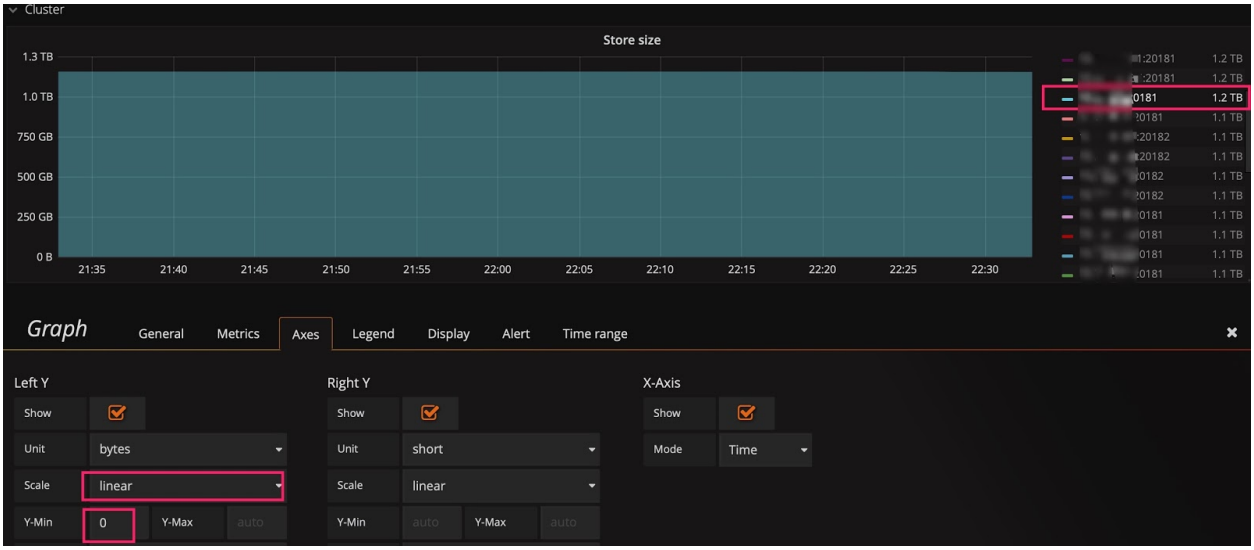


Figure 217: Baseline defaults to 0

Change the baseline to `auto`:



Figure 218: Change the baseline to auto

12.5.5.3.4 Tip 4: Use Shared crosshair or Tooltip

In the **Settings** panel, there is a **Graph Tooltip** panel option which defaults to **Default**.

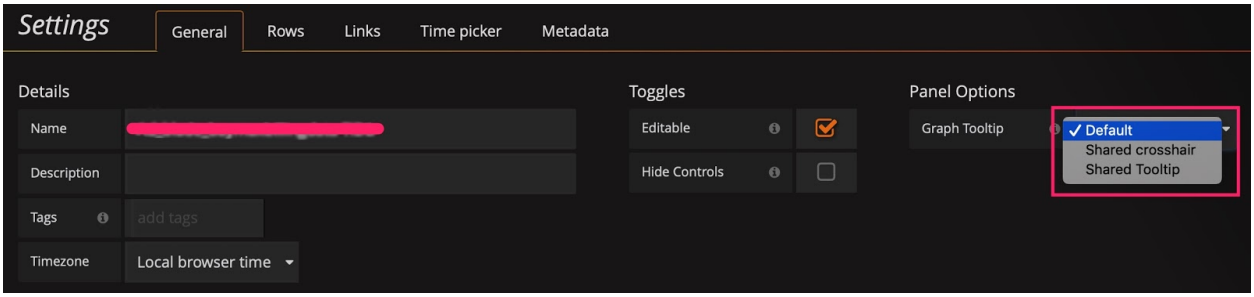


Figure 219: Graphic presentation tools

You can use **Shared crosshair** and **Shared Tooltip** respectively to test the effect as shown in the following figures. Then, the scales are displayed in linkage, which is convenient to confirm the correlation of two metrics when diagnosing problems.

Set the graphic presentation tool to **Shared crosshair**:

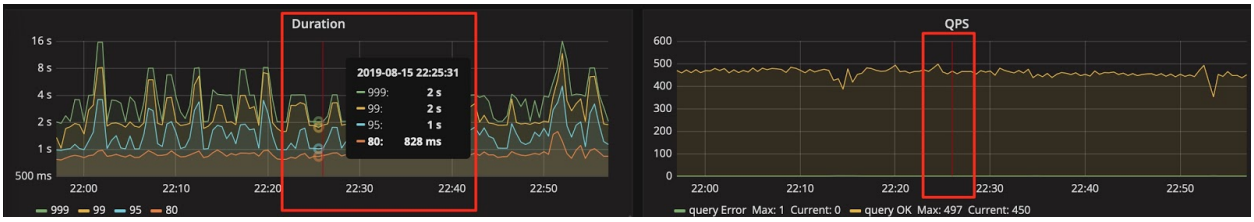


Figure 220: Set the graphical presentation tool to Shared crosshair

Set the graphical presentation tool to **Shared Tooltip**:

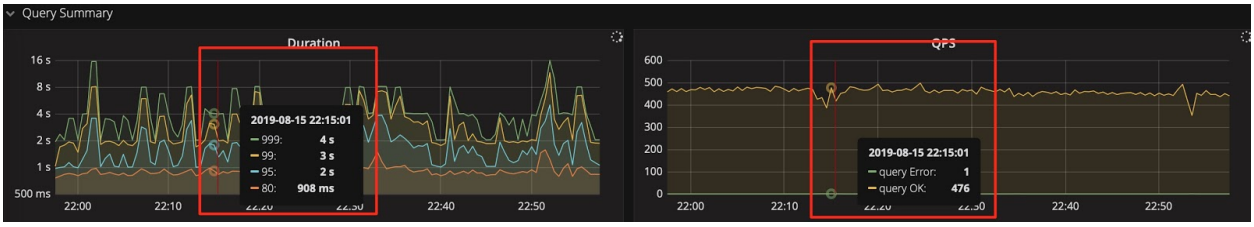


Figure 221: Set the graphic presentation tool to Shared Tooltip

12.5.5.3.5 Tip 5: Enter `IP address:port number` to check the metrics in history

PD's dashboard only shows the metrics of the current leader. If you want to check the status of a PD leader in history and it no longer exists in the drop-down list of the `instance` field, you can manually enter `IP address:2379` to check the data of the leader.

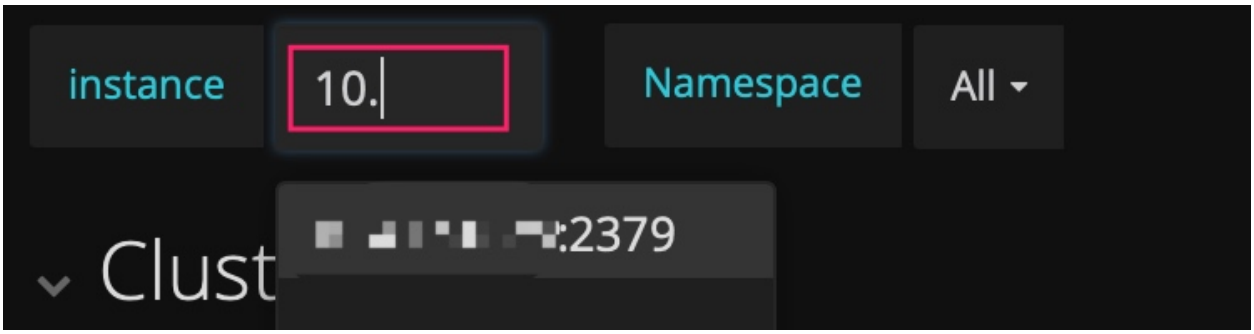


Figure 222: Check the metrics in history

12.5.5.3.6 Tip 6: Use the `Avg` function

Generally, only `Max` and `Current` functions are available in the legend by default. When the metrics fluctuate greatly, you can add other summary functions such as the `Avg` function to the legend to check the overall trend for the duration of time.

Add summary functions such as the `Avg` function:

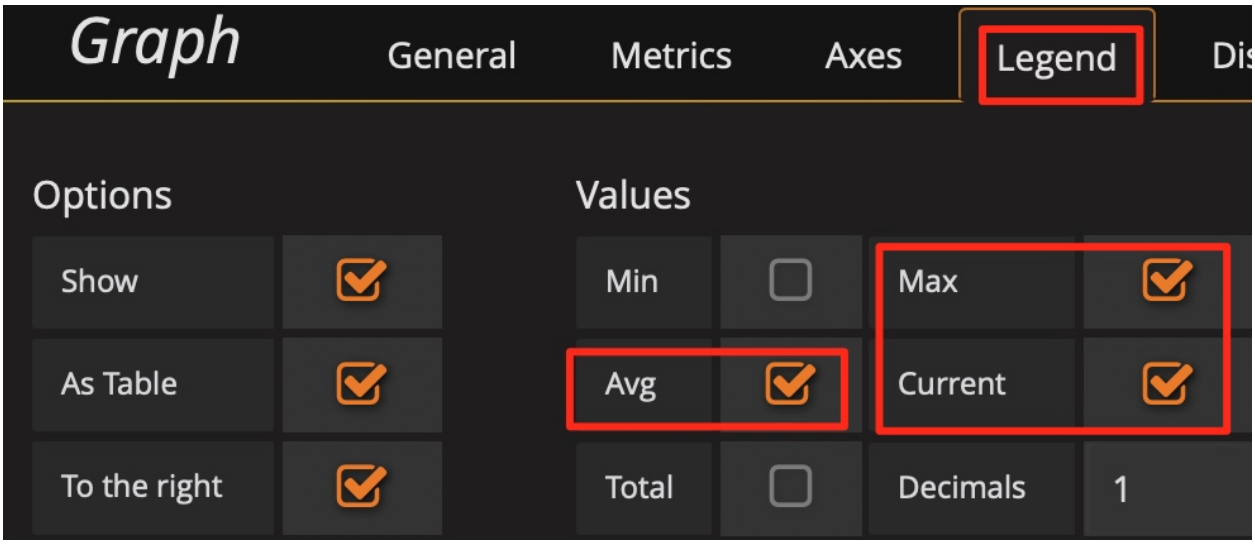


Figure 223: Add summary functions such as Avg

Then check the overall trend:

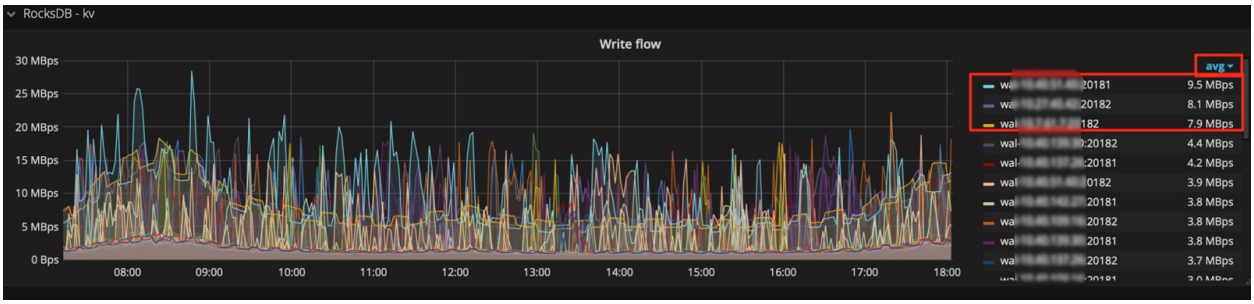


Figure 224: Add Avg function to check the overall trend

12.5.5.3.7 Tip 7: Use the API of Prometheus to obtain the result of query expressions

Grafana obtains data through the API of Prometheus and you can use this API to obtain information as well. In addition, it also has the following usages:

- Automatically obtains information such as the cluster size and status.
- Makes minor changes to the expression to provide information for the report, such as counting the total amount of QPS per day, the peak value of QPS per day, and the response time per day.
- Performs regular health inspection on the important metrics.

The API of Prometheus is shown as follows:

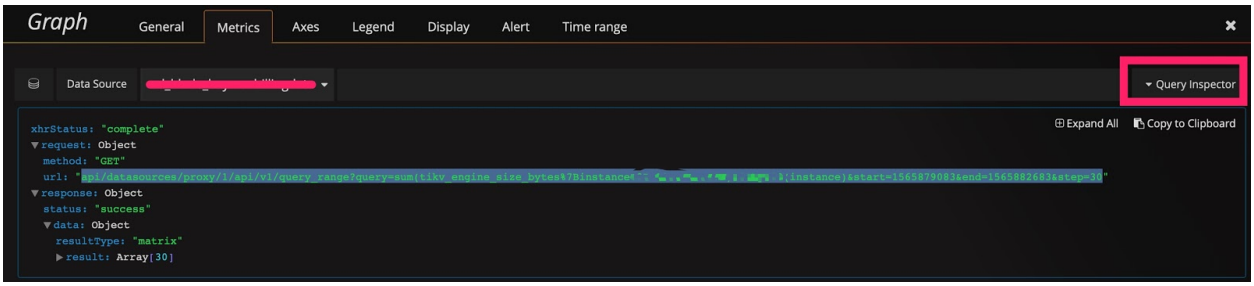


Figure 225: The API of Prometheus

```
curl -u user:pass 'http://__grafana_ip__:3000/api/datasources/proxy/1/api/v1
    ↪ /query_range?query=sum(tikv_engine_size_bytes%7
    ↪ Binstancexxxxxxxxx20180%22%7D)%20by%20(instance)&start=1565879269&end
    ↪ =1565882869&step=30' |python -m json.tool
```

```
{
    "data": {
```

```
        "result": [
            {
                "metric": {
                    "instance": "xxxxxxxxxx:20180"
                },
                "values": [
                    [
                        1565879269,
                        "1006046235280"
                    ],
                    [
                        1565879299,
                        "1006057877794"
                    ],
                    [
                        1565879329,
                        "1006021550039"
                    ],
                    [
                        1565879359,
                        "1006021550039"
                    ],
                    [
                        1565882869,
                        "1006132630123"
                    ]
                ]
            }
        ],
        "resultType": "matrix"
    },
    "status": "success"
}
```


12.5.5.4 Summary

The Grafana + Prometheus monitoring platform is a very powerful tool. Making good use of it can improve efficiency, saving you a lot of time on analyzing the status of the TiDB cluster. More importantly, it can help you diagnose problems. This tool is very useful in the operation and maintenance of TiDB clusters, especially when there is a large amount of data.

12.5.6 PD Scheduling Best Practices

This document details the principles and strategies of PD scheduling through common scenarios to facilitate your application. This document assumes that you have a basic understanding of TiDB, TiKV and PD with the following core concepts:

- leader/follower/learner
- operator
- operator step
- pending/down
- region/peer/Raft group
- region split
- scheduler
- store

> **Note:**
>
> This document initially targets TiDB 3.0. Although some features are not supported in earlier versions (2.x), the underlying mechanisms are similar and this document can still be used as a reference.

12.5.6.1 PD scheduling policies

This section introduces the principles and processes involved in the scheduling system.

12.5.6.1.1 Scheduling process

The scheduling process generally has three steps:

1. Collect information

   Each TiKV node periodically reports two types of heartbeats to PD:

   - `StoreHeartbeat`: Contains the overall information of stores, including disk capacity, available storage, and read/write traffic
   - `RegionHeartbeat`: Contains the overall information of regions, including the range of each region, peer distribution, peer status, data volume, and read/write traffic

   PD collects and restores this information for scheduling decisions.

2. Generate operators

   Different schedulers generate the operators based on their own logic and requirements, with the following considerations:

- Do not add peers to a store in abnormal states (disconnected, down, busy, low space)
- Do not balance regions in abnormal states
- Do not transfer a leader to a pending peer
- Do not remove a leader directly
- Do not break the physical isolation of various region peers
- Do not violate constraints such as label property

3. Execute operators

   To execute the operators, the general procedure is:

   1. The generated operator first joins a queue managed by `OperatorController`.
   2. `OperatorController` takes the operator out of the queue and executes it with a certain amount of concurrency based on the configuration. This step is to assign each operator step to the corresponding region leader.
   3. The operator is marked as "finish" or "timeout" and removed from the queue.

12.5.6.1.2 Load balancing

Region primarily relies on `balance-leader` and `balance-region` schedulers to achieve load balance. Both schedulers target distributing regions evenly across all stores in the cluster but with separate focuses: `balance-leader` deals with region leader to balance incoming client requests, whereas `balance-region` concerns itself with each region peer to redistribute the pressure of storage and avoid exceptions like out of storage space.

`balance-leader` and `balance-region` share a similar scheduling process:

1. Rate stores according to their resource availability.
2. `balance-leader` or `balance-region` constantly transfer leaders or peers from stores with high scores to those with low scores.

However, their rating methods are different. `balance-leader` uses the sum of all region sizes corresponding to leaders in a store, whereas the way of `balance-region` is relatively complicated. Depending on the specific storage capacity of each node, the rating method of `balance-region` might:

- based on the amount of data when there is sufficient storage (to balance data distribution among nodes).
- based on the available storage when there is insufficient storage (to balance the storage availability on different nodes).
- based on the weighted sum of the two factors above when neither of the situations applies.

Because different nodes might differ in performance, you can also set the weight of load balancing for different stores. `leader-weight` and `region-weight` are used to control the leader weight and region weight respectively ("1" by default for both). For example, when the `leader-weight` of a store is set to "2", the number of leaders on the node is about twice as many as that of other nodes after the scheduling stabilizes. Similarly, when the `leader-weight` of a store is set to "0.5", the number of leaders on the node is about half as many as that of other nodes.

12.5.6.1.3 Hot regions scheduling

For hot regions scheduling, use `hot-region-scheduler`. Since TiDB v3.0, the process is performed as follows:

1. Count hot regions by determining read/write traffic that exceeds a certain threshold for a certain period based on the information reported by stores.
2. Redistribute these regions in a similar way to load balancing.

For hot write regions, `hot-region-scheduler` attempts to redistribute both region peers and leaders; for hot read regions, `hot-region-scheduler` only redistributes region leaders.

12.5.6.1.4 Cluster topology awareness

Cluster topology awareness enables PD to distribute replicas of a region as much as possible. This is how TiKV ensures high availability and disaster recovery capability. PD continuously scans all regions in the background. When PD finds that the distribution of regions is not optimal, it generates an operator to replace peers and redistribute regions.

The component to check region distribution is `replicaChecker`, which is similar to a scheduler except that it cannot be disabled. `replicaChecker` schedules based on the the configuration of `location-labels`. For example, `[zone,rack,host]` defines a three-tier topology for a cluster. PD attempts to schedule region peers to different zones first, or to different racks when zones are insufficient (for example, 2 zones for 3 replicas), or to different hosts when racks are insufficient.

12.5.6.1.5 Scale-down and failure recovery

Scale-down refers to the process when you take a store offline and mark it as "offline" using a command. PD replicates the regions on the offline node to other nodes by scheduling. Failure recovery applies when stores failed and cannot be recovered. In this case, regions with peers distributed on the corresponding store might lose replicas, which requires PD to replenish on other nodes.

The processes of scale-down and failure recovery are basically the same. `replicaChecker` ↪ finds a region peer in abnormal states, and then generates an operator to replace the abnormal peer with a new one on a healthy store.

12.5.6.1.6 Region merge

Region merge refers to the process of merging adjacent small regions. It serves to avoid unnecessary resource consumption by a large number of small or even empty regions after data deletion. Region merge is performed by `mergeChecker`, which processes in a similar way to `replicaChecker`: PD continuously scans all regions in the background, and generates an operator when contiguous small regions are found.

Specifically, when a newly split Region exists for more than the value of `split-merge-interval` (`1h` by default), if the following conditions occur at the same time, this Region triggers the Region merge scheduling:

- The size of this Region is smaller than the value of the `max-merge-region-size` (20 MiB by default)
- The number of keys in this Region is smaller than the value of `max-merge-region-keys` (200,000 by default).

12.5.6.2 Query scheduling status

You can check the status of scheduling system through metrics, pd-ctl and logs. This section briefly introduces the methods of metrics and pd-ctl. Refer to PD Monitoring Metrics and PD Control for details.

12.5.6.2.1 Operator status

The **Grafana PD/Operator** page shows the metrics about operators, among which:

- Schedule operator create: Operator creating information
- Operator finish duration: Execution time consumed by each operator
- Operator step duration: Execution time consumed by the operator step

You can query operators using pd-ctl with the following commands:

- `operator show`: Queries all operators generated in the current scheduling task
- `operator show [admin | leader | region]`: Queries operators by type

12.5.6.2.2 Balance status

The **Grafana PD/Statistics - Balance** page shows the metrics about load balancing, among which:

- Store leader/region score: Score of each store
- Store leader/region count: The number of leaders/regions in each store
- Store available: Available storage on each store

You can use store commands of pd-ctl to query balance status of each store.

12.5.6.2.3 Hot Region status

The **Grafana PD/Statistics - hotspot** page shows the metrics about hot regions, among which:

- Hot write region's leader/peer distribution: the leader/peer distribution in hot write regions
- Hot read region's leader distribution: the leader distribution in hot read regions

You can also query the status of hot regions using pd-ctl with the following commands:

- `hot read`: Queries hot read regions
- `hot write`: Queries hot write regions
- `hot store`: Queries the distribution of hot regions by store
- `region topread [limit]`: Queries the region with top read traffic
- `region topwrite [limit]`: Queries the region with top write traffic

12.5.6.2.4 Region health

The **Grafana PD/Cluster/Region health** panel shows the metrics about regions in abnormal states.

You can query the list of regions in abnormal states using pd-ctl with region check commands:

- `region check miss-peer`: Queries regions without enough peers
- `region check extra-peer`: Queries regions with extra peers
- `region check down-peer`: Queries regions with down peers
- `region check pending-peer`: Queries regions with pending peers

12.5.6.3 Control scheduling strategy

You can use pd-ctl to adjust the scheduling strategy from the following three aspects. Refer to PD Control for more details.

12.5.6.3.1 Add/delete scheduler manually

PD supports dynamically adding and removing schedulers directly through pd-ctl. For example:

- `scheduler show`: Shows currently running schedulers in the system
- `scheduler remove balance-leader-scheduler`: Removes (disable) balance-leader-scheduler
- `scheduler add evict-leader-scheduler 1`: Adds a scheduler to remove all leaders in Store 1

12.5.6.3.2 Add/delete Operators manually

PD also supports adding or removing operators directly through pd-ctl. For example:

- `operator add add-peer 2 5`: Adds peers to Region 2 in Store 5
- `operator add transfer-leader 2 5`: Migrates the leader of Region 2 to Store 5
- `operator add split-region 2`: Splits Region 2 into two regions evenly in size
- `operator remove 2`: Removes currently pending operator in Region 2

12.5.6.3.3 Adjust scheduling parameter

You can check the scheduling configuration using the `config show` command in pd-ctl, and adjust the values using `config set {key} {value}`. Common adjustments include:

- `leader-schedule-limit`: Controls the concurrency of transferring leader scheduling
- `region-schedule-limit`: Controls the concurrency of adding/deleting peer scheduling
- `enable-replace-offline-replica`: Determines whether to enable the scheduling to take nodes offline
- `enable-location-replacement`: Determines whether to enable the scheduling that handles the isolation level of regions
- `max-snapshot-count`: Controls the maximum concurrency of sending/receiving snapshots for each store

12.5.6.4 PD scheduling in common scenarios

This section illustrates the best practices of PD scheduling strategies through several typical scenarios.

12.5.6.4.1 Leaders/regions are not evenly distributed

The rating mechanism of PD determines that leader count and region count of different stores cannot fully reflect the load balancing status. Therefore, it is necessary to confirm whether there is load imbalancing from the actual load of TiKV or storage usage.

Once you have confirmed that leaders/region are not evenly distributed, you need to check the rating of different stores.

If the scores of different stores are close, it means PD mistakenly believes that leaders/regions are evenly distributed. Possible reasons are:

- There are hot regions that cause load imbalancing. In this case, you need to analyze further based on hot regions scheduling.
- There are a large number of empty regions or small regions, which leads to a great difference in the number of leaders in different stores and high pressure on Raft store. This is the time for a region merge scheduling.

- Hardware and software environment varies among stores. To control the distribution of leader/region, you can refer to Load balancing and adjust the values of `leader-weight` and `region-weight`.
- Other unknown reasons. Still you can adjust the values of `leader-weight` and `region-weight` to control the distribution of leader/region.

If there is a big difference in the rating of different stores, you need to examine the operator-related metrics, with special focus on the generation and execution of operators. There are two main situations:

- When operators are generated normally but the scheduling process is slow, it is possible that:
    - The scheduling speed is limited by default for load balancing purpose. You can adjust `leader-schedule-limit` or `region-schedule-limit` to larger values without significantly impacting regular services. In addition, you can also properly ease the restrictions specified by `max-pending-peer-count` and `max-snapshot-count`.
    - Other scheduling tasks are running concurrently, which slows down the balancing. In this case, if the balancing takes precedence over other scheduling tasks, you can stop other tasks or limit their speeds. For example, if you take some nodes offline when balancing is in progress, both operations consume the quota of `region-schedule-limit`. In this case, you can limit the speed of scheduler to remove nodes, or simply set `enable-replace-offline-replica = false` to temporarily disable it.
    - The scheduling process is too slow. You can check the **Operator step duration** metric to confirm the cause. Generally, steps that do not involve sending and receiving snapshots (such as `TransferLeader`, `RemovePeer`, `PromoteLearner`) should be completed in milliseconds, while steps that involve snapshots (such as `AddLearner` and `AddPeer`) are expected to be completed in tens of seconds. If the duration is obviously too long, it could be caused by high pressure on TiKV or bottleneck in network, which needs specific analysis.
- PD fails to generate the corresponding balancing scheduler. Possible reasons include:
    - The scheduler is not activated. For example, the corresponding scheduler is deleted, or its limit it set to "0".
    - Other constraints. For example, `evict-leader-scheduler` in the system prevents leaders from being migrating to the corresponding store. Or label property is set, which makes some stores reject leaders.
    - Restrictions from the cluster topology. For example, in a cluster of 3 replicas across 3 data centers, 3 replicas of each region are distributed in different data centers due to replica isolation. If the number of stores is different among these data centers, the scheduling can only reach a balanced state within each data center, but not balanced globally.

12.5.6.4.2 Taking nodes offline is slow

This scenario requires examining the generation and execution of operators through related metrics.

If operators are successfully generated but the scheduling process is slow, possible reasons are:

- The scheduling speed is limited by default. You can adjust `leader-schedule-limit` or `replica-schedule-limit` to larger value.s Similarly, you can consider loosening the limits on `max-pending-peer-count` and `max-snapshot-count`.
- Other scheduling tasks are running concurrently and racing for resources in the system. You can refer to the solution in Leaders/regions are not evenly distributed.
- When you take a single node offline, a number of region leaders to be processed (around 1/3 under the configuration of 3 replicas) are distributed on the node to remove. Therefore, the speed is limited by the speed at which snapshots are generated by this single node. You can speed it up by manually adding an `evict-leader-scheduler` to migrate leaders.

If the corresponding operator fails to generate, possible reasons are:

- The operator is stopped, or `replica-schedule-limit` is set to "0".
- There is no proper node for region migration. For example, if the available capacity size of the replacing node (of the same label) is less than 20%, PD will stop scheduling to avoid running out of storage on that node. In such case, you need to add more nodes or delete some data to free the space.

12.5.6.4.3 Bringing nodes online is slow

Currently, bringing nodes online is scheduled through the balance region mechanism. You can refer to Leaders/regions are not evenly distributed for troubleshooting.

12.5.6.4.4 Hot regions are not evenly distributed

Hot regions scheduling issues generally fall into the following categories:

- Hot regions can be observed via PD metrics, but the scheduling speed cannot keep up to redistribute hot regions in time.

  Solution: adjust `hot-region-schedule-limit` to a larger value, and reduce the limit quota of other schedulers to speed up hot regions scheduling. Or you can adjust `hot-region-cache-hits-threshold` to a smaller value to make PD more sensitive to traffic changes.

- Hotspot formed on a single region. For example, a small table is intensively scanned by a massive amount of requests. This can also be detected from PD metrics. Because you cannot actually distribute a single hotspot, you need to manually add a `split-region` operator to split such a region.

- The load of some nodes is significantly higher than that of other nodes from TiKV-related metrics, which becomes the bottleneck of the whole system. Currently, PD counts hotspots through traffic analysis only, so it is possible that PD fails to identify hotspots in certain scenarios. For example, when there are intensive point lookup requests for some regions, it might not be obvious to detect in traffic, but still the high QPS might lead to bottlenecks in key modules.

  Solutions: Firstly, locate the table where hot regions are formed based on the specific business. Then add a `scatter-range-scheduler` scheduler to make all regions of this table evenly distributed. TiDB also provides an interface in its HTTP API to simplify this operation. Refer to TiDB HTTP API for more details.

12.5.6.4.5 Region merge is slow

Similar to slow scheduling, the speed of region merge is most likely limited by the configurations of `merge-schedule-limit` and `region-schedule-limit`, or the region merge scheduler is competing with other schedulers. Specifically, the solutions are:

- If it is known from metrics that there are a large number of empty regions in the system, you can adjust `max-merge-region-size` and `max-merge-region-keys` to smaller values to speed up the merge. This is because the merge process involves replica migration, so the smaller the region to be merged, the faster the merge is. If the merge operators are already generated rapidly, to further speed up the process, you can set `patrol-region-interval` to `10ms` (the default value of this configuration item is `10ms` in v5.3.0 and later versions of TiDB). This makes region scanning faster at the cost of more CPU consumption.
- A lot of tables have been created and then emptied (including truncated tables). These empty Regions cannot be merged if the split table attribute is enabled. You can disable this attribute by adjusting the following parameters:
  - TiKV: Set `split-region-on-table` to `false`. You cannot modify the parameter dynamically.
  - PD: Use PD Control to set the parameters required by your cluster situation.
    - Suppose that your cluster has no TiDB instance, and the value of `key-type` is set to `raw` or `txn`. In this case, PD can merge Regions across tables, regardless of the value of `enable-cross-table-merge setting`. You can modify the `key-type` parameter dynamically.

```
config set key-type txn
```

    - Suppose that your cluster has a TiDB instance, and the value of `key-type` is set to `table`. In this case, PD can merge Regions across tables only if the value of `enable-cross-table-merge` is set to `true`. You can modify the `key-type` parameter dynamically.

```
config set enable-cross-table-merge true
```

If the modification does not take effect, refer to FAQ - Why the modified `toml` configuration for TiKV/PD does not take effect?.

> **Note:**
> After enabling Placement Rules, properly switch the value of `key-type` to avoid the failure of decoding.

For v3.0.4 and v2.1.16 or earlier, the `approximate_keys` of regions are inaccurate in specific circumstances (most of which occur after dropping tables), which makes the number of keys break the constraints of `max-merge-region-keys`. To avoid this problem, you can adjust `max-merge-region-keys` to a larger value.

12.5.6.4.6 Troubleshoot TiKV node

If a TiKV node fails, PD defaults to setting the corresponding node to the **down** state after 30 minutes (customizable by configuration item `max-store-down-time`), and rebalancing replicas for regions involved.

Practically, if a node failure is considered unrecoverable, you can immediately take it offline. This makes PD replenish replicas soon in another node and reduces the risk of data loss. In contrast, if a node is considered recoverable, but the recovery cannot be done in 30 minutes, you can temporarily adjust `max-store-down-time` to a larger value to avoid unnecessary replenishment of the replicas and resources waste after the timeout.

In TiDB v5.2.0, TiKV introduces the mechanism of slow TiKV node detection. By sampling the requests in TiKV, this mechanism works out a score ranging from 1 to 100. A TiKV node with a score higher than or equal to 80 is marked as slow. You can add `evict-slow-store-scheduler` to detect and schedule slow nodes. If only one TiKV is detected as slow, and the slow score reaches the upper limit (100 by default), the leader in this node will be evicted (similar to the effect of `evict-leader-scheduler`).

12.5.7 Best Practices for TiKV Performance Tuning with Massive Regions

In TiDB, data is split into Regions, each storing data for a specific key range. These Regions are distributed among multiple TiKV instances. As data is written into a cluster, millions of or even tens of millions of Regions are created. Too many Regions on a single TiKV instance can bring a heavy burden to the cluster and affect its performance.

This document introduces the workflow of Raftstore (a core module of TiKV), explains why a massive amount of Regions affect the performance, and offers methods for tuning TiKV performance.

12.5.7.1 Raftstore workflow

A TiKV instance has multiple Regions on it. The Raftstore module drives the Raft state machine to process Region messages. These messages include processing read or write requests on Regions, persisting or replicating Raft logs, and processing Raft heartbeats. However, an increasing number of Regions can affect performance of the whole cluster. To understand this, it is necessary to learn the workflow of Raftstore shown as follows:

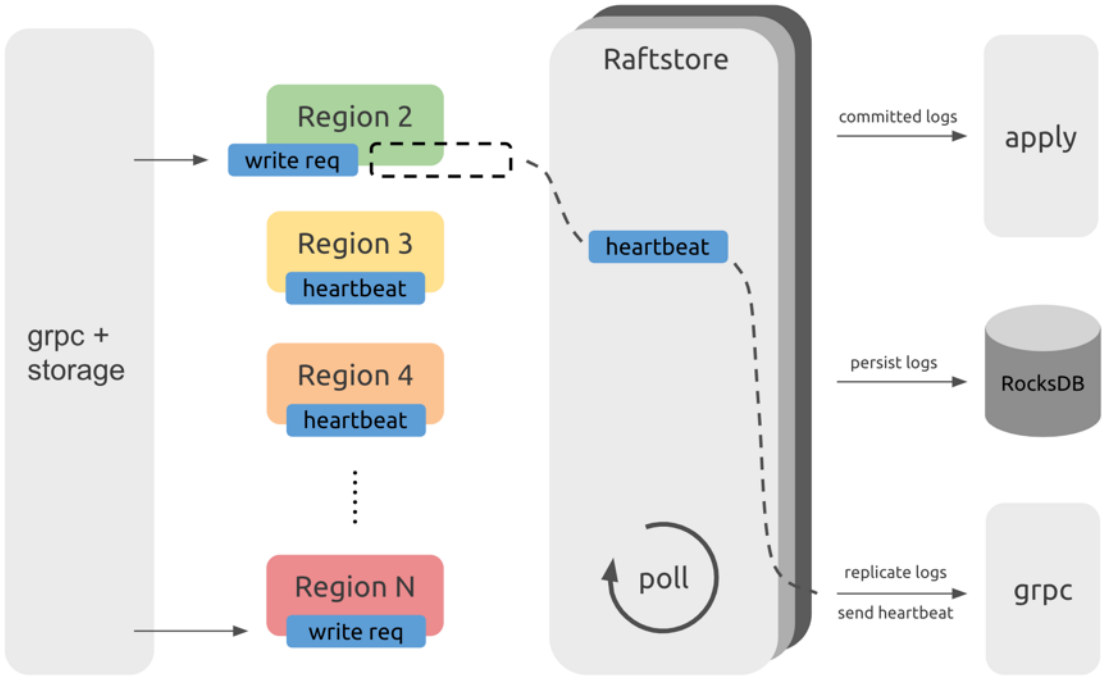


Figure 226: Raftstore Workflow

> **Note:**
>
> This diagram only illustrates the workflow of Raftstore and does not represent the actual code structure.

From the above diagram, you can see that requests from the TiDB servers, after passing through the gRPC and storage modules, become read and write messages of KV (key-value), and are sent to the corresponding Regions. These messages are not immediately processed but are temporarily stored. Raftstore polls to check whether each Region has messages to process. If a Region has messages to process, Raftstore drives the Raft state machine of this Region to process these messages and perform subsequent operations according to the state changes of these messages. For example, when write requests come in, the Raft state machine stores logs into disk and sends logs to other Region replicas; when the heartbeat interval is reached, the Raft state machine sends heartbeat information to other Region replicas.

12.5.7.2 Performance problem

From the Raftstore workflow diagram, messages in each Region are processed one by one. When a large number of Regions exist, it takes Raftstore some time to process the heartbeats of these Regions, which can cause some delay. As a result, some read and write requests are not processed in time. If read and write pressure is high, the CPU usage of the Raftstore thread might easily become the bottleneck, which further increases the delay and affects the performance.

Generally, if the CPU usage of the loaded Raftstore reaches 85% or higher, Raftstore goes into a busy state and becomes the bottleneck. At the same time, `propose wait duration` can be as high as hundreds of milliseconds.

Note:

- For the CPU usage of Raftstore as mentioned above, Raftstore is single-threaded. If Raftstore is multi-threaded, you can increase the CPU usage threshold (85%) proportionally.
- Because I/O operations exist in the Raftstore thread, CPU usage cannot reach 100%.

12.5.7.2.1 Performance monitoring

You can check the following monitoring metrics in Grafana's **TiKV Dashboard**:

- `Raft store CPU` in the **Thread-CPU** panel

  Reference value: lower than `raftstore.store-pool-size * 85%`.

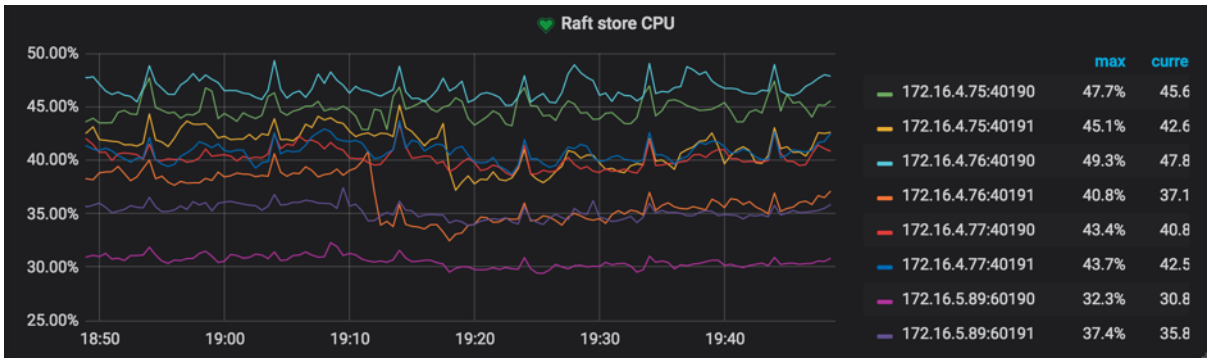


Figure 227: Check Raftstore CPU

- `Propose wait duration` in the **Raft Propose** panel

`Propose wait duration` is the delay between the time a request is sent to Raftstore and the time Raftstore actually starts processing the request. Long delay means that Raftstore is busy, or that processing the append log is time-consuming, making Raftstore unable to process the request in time.

Reference value: lower than 50~100 ms according to the cluster size

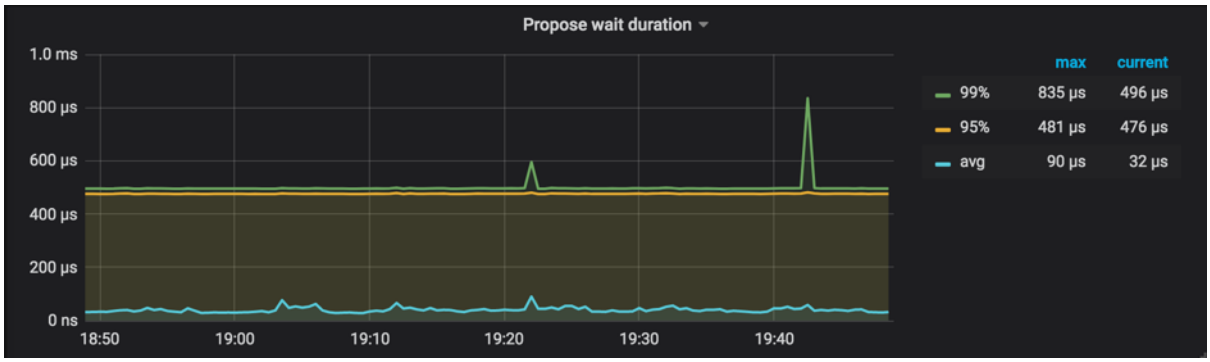


Figure 228: Check Propose wait duration

- `Commit log duration` in the **Raft IO** panel

  `Commit log duration` is the time Raftstore takes to commit Raft logs to the majority of members in the respective Region. The possible reasons for a high value of this metric with significant fluctuations include the following:

  - The workload on Raftstore is heavy.
  - The append log operation is slow.
  - Raft logs cannot be committed timely due to network congestion.

Reference value: lower than 200-500 ms.

```
![Check Commit log duration](./media/best-practices/commit-log-duration.png)
```

12.5.7.3 Performance tuning methods

After finding out the cause of a performance problem, try to solve it from the following two aspects:

- Reduce the number of Regions on a single TiKV instance
- Reduce the number of messages for a single Region

12.5.7.3.1 Method 1: Increase Raftstore concurrency

Raftstore has been upgraded to a multi-threaded module since TiDB v3.0, which greatly reduces the possibility that a Raftstore thread becomes the bottleneck.

By default, `raftstore.store-pool-size` is configured to `2` in TiKV. If a bottleneck occurs in Raftstore, you can properly increase the value of this configuration item according to the actual situation. But to avoid introducing unnecessary thread switching overhead, it is recommended that you do not set this value too high.

12.5.7.3.2 Method 2: Enable Hibernate Region

In the actual situation, read and write requests are not evenly distributed on every Region. Instead, they are concentrated on a few Regions. Then you can minimize the number of messages between the Raft leader and the followers for the temporarily idle Regions, which is the feature of Hibernate Region. In this feature, Raftstore doesn't send tick messages to the Raft state machines of idle Regions if not necessary. Then these Raft state machines will not be triggered to generate heartbeat messages, which can greatly reduce the workload of Raftstore.

Hibernate Region is enabled by default in TiKV master. You can configure this feature according to your needs. For details, refer to Configure Hibernate Region.

12.5.7.3.3 Method 3: Enable `Region Merge`

> **Note:**
>
> `Region Merge` is enabled by default since TiDB v3.0.

You can also reduce the number of Regions by enabling `Region Merge`. Contrary to `Region Split`, `Region Merge` is the process of merging adjacent small Regions through scheduling. After dropping data or executing the `Drop Table` or `Truncate Table` statement, you can merge small Regions or even empty Regions to reduce resource consumption.

Enable `Region Merge` by configuring the following parameters:

```
config set max-merge-region-size 20
config set max-merge-region-keys 200000
config set merge-schedule-limit 8
```

Refer to Region Merge and the following three configuration parameters in the PD configuration file for more details:

- `max-merge-region-size`
- `max-merge-region-keys`
- `merge-schedule-limit`

The default configuration of the `Region Merge` parameters is rather conservative. You can speed up the `Region Merge` process by referring to the method provided in PD Scheduling Best Practices.

12.5.7.3.4 Method 4: Increase the number of TiKV instances

If I/O resources and CPU resources are sufficient, you can deploy multiple TiKV instances on a single machine to reduce the number of Regions on a single TiKV instance; or you can increase the number of machines in the TiKV cluster.

12.5.7.3.5 Method 5: Adjust `raft-base-tick-interval`

In addition to reducing the number of Regions, you can also reduce pressure on Raftstore by reducing the number of messages for each Region within a unit of time. For example, you can properly increase the value of the `raft-base-tick-interval` configuration item:

```
[raftstore]
raft-base-tick-interval = "2s"
```

In the above configuration, `raft-base-tick-interval` is the time interval at which Raftstore drives the Raft state machine of each Region, which means at this time interval, Raftstore sends a tick message to the Raft state machine. Increasing this interval can effectively reduce the number of messages from Raftstore.

Note that this interval between tick messages also determines the intervals between `election timeout` and `heartbeat`. See the following example:

```
raft-election-timeout = raft-base-tick-interval * raft-election-timeout-
    ↪ ticks
raft-heartbeat-interval = raft-base-tick-interval * raft-heartbeat-ticks
```

If Region followers have not received the heartbeat from the leader within the `raft ↪ -election-timeout` interval, these followers determine that the leader has failed and start a new election. `raft-heartbeat-interval` is the interval at which a leader sends a heartbeat to followers. Therefore, increasing the value of `raft-base-tick-interval` can reduce the number of network messages sent from Raft state machines but also makes it longer for Raft state machines to detect the leader failure.

12.5.7.3.6 Method 6: Adjust Region size

The default size of a Region is 96 MiB, and you can reduce the number of Regions by setting Regions to a larger size. For more information, see Tune Region Performance.

> **Warning:**
>
> Currently, customized Region size is an experimental feature introduced in TiDB v6.1.0. It is not recommended that you use it in production environments. The risks are as follows:
>
> - Performance jitter might be caused.

- The query performance, especially for queries that deal with a large range of data, might decrease.
- The Region scheduling slows down.

12.5.7.3.7 Method 7: Increase the maximum number of connections for Raft communication

By default, the maximum number of connections used for Raft communication between TiKV nodes is 1. Increasing this number can help alleviate blockage issues caused by heavy communication workloads of a large number of Regions. For detailed instructions, see grpc ↪ -raft-conn-num.

Note:

To reduce unnecessary thread switching overhead and mitigate potential negative impacts from batch processing, it is recommended to set the number within the range of `[1, 4]`.

12.5.7.4 Other problems and solutions

This section describes some other problems and solutions.

12.5.7.4.1 Switching PD Leader is slow

PD needs to persist Region Meta information on etcd to ensure that PD can quickly resume to provide Region routing services after switching the PD Leader node. As the number of Regions increases, the performance problem of etcd appears, making it slower for PD to get Region Meta information from etcd when PD is switching the Leader. With millions of Regions, it might take more than ten seconds or even tens of seconds to get the meta information from etcd.

To address this problem, `use-region-storage` is enabled by default in PD since TiDB v3.0. With this feature enabled, PD stores Region Meta information on local LevelDB and synchronizes the information among PD nodes through other mechanisms.

12.5.7.4.2 PD routing information is not updated in time

In TiKV, pd-worker regularly reports Region Meta information to PD. When TiKV is restarted or switches the Region leader, PD needs to recalculate Region's `approximate ↪ size / keys` through statistics. Therefore, with a large number of Regions, the single-threaded pd-worker might become the bottleneck, causing tasks to be piled up and not

processed in time. In this situation, PD cannot obtain certain Region Meta information in time so that the routing information is not updated in time. This problem does not affect the actual reads and writes, but might cause inaccurate PD scheduling and require several round trips when TiDB updates Region cache.

You can check **Worker pending tasks** under **Task** in the **TiKV Grafana** panel to determine whether pd-worker has tasks piled up. Generally, `pending tasks` should be kept at a relatively low value.

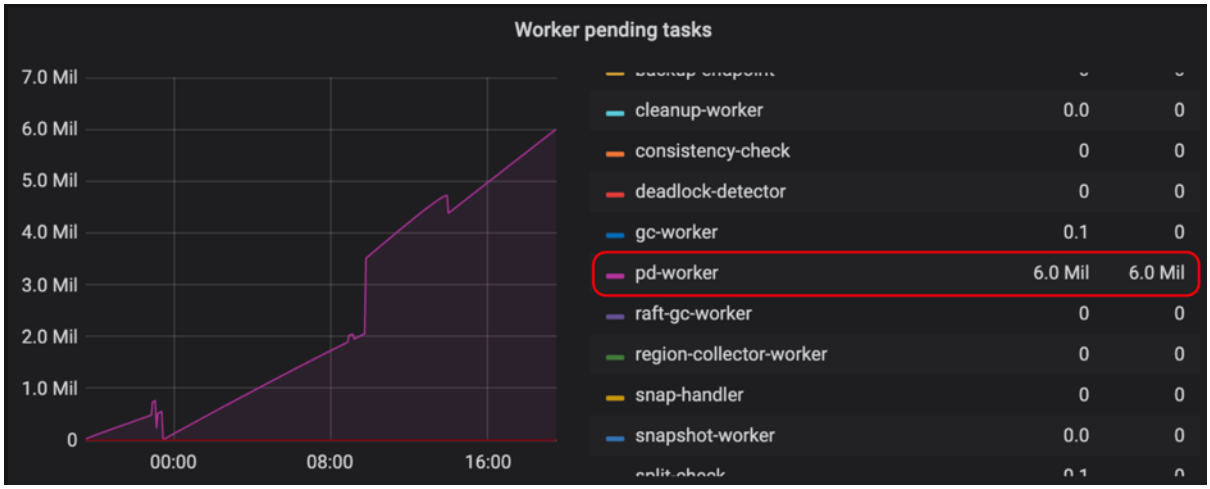


Figure 229: Check pd-worker

pd-worker has been optimized for better performance since v3.0.5. If you encounter a similar problem, it is recommended to upgrade to the latest version.

12.5.7.4.3 Prometheus is slow to query metrics

In a large-scale cluster, as the number of TiKV instances increases, Prometheus has greater pressure to query metrics, making it slower for Grafana to display these metrics. To ease this problem, metrics pre-calculation is configured since v3.0.

12.5.8 Best Practices for Three-Node Hybrid Deployment

For a TiDB cluster, if you have no requirements on high performance but need to control the cost, you can deploy the TiDB, TiKV, and PD components on three machines in a hybrid way.

This document offers an example of three-node hybrid deployment and a TPC-C test against the deployed cluster. Based on this example, this document offers best practices for the deployment scenario and its parameter adjustment.

12.5.8.1 Prerequisites for deployment and the test method

In this example, three physical machines are used for deployment, each with 16 CPU cores and 32 GB of memory. On each machine (node), one TiDB instance, one TiKV instance, and one PD instance are deployed in a hybrid way.

Because PD and TiKV both store information on the disk, the read and write latency of disk directly affects the latency of the PD and TiKV services. To avoid the situation that PD and TiKV compete for disk resources and affect each other, it is recommended to use different disks for PD and TiKV.

In this example, the TPC-C 5000 Warehouse data is used in TiUP bench and the test lasts 12 hours with the `terminals` parameter set to `128` (concurrency). Close attention is paid to metrics related to performance stability of the cluster.

The image below shows the QPS monitor of the cluster within 12 hours with the default parameter configuration. From the image, you can see an obvious performance jitter.



Figure 230: QPS with default config

After the parameter adjustment, the performance is improved.



Figure 231: QPS with modified config

12.5.8.2 Parameter adjustment

Performance jitter occurs in the image above, because the default thread pool configuration and the resource allocation to background tasks are for machines with sufficient resources. In the hybrid deployment scenario, the resources are shared among multiple components, so you need to limit the resource consumption via configuration parameters.

The final cluster configuration for this test is as follows:

```
tikv:
    readpool.unified.max-thread-count: 6
    server.grpc-concurrency: 2
    storage.scheduler-worker-pool-size: 2
    gc.max-write-bytes-per-sec: 300K
    rocksdb.max-background-jobs: 3
    rocksdb.max-sub-compactions: 1
    rocksdb.rate-bytes-per-sec: "200M"

  tidb:
    performance.max-procs: 8
```

The following sections introduce the meanings and the adjustment methods of these parameters.

12.5.8.2.1 Configuration of TiKV thread pool size

This section offers best practices for adjusting parameters that relate to the resource allocation of thread pools for foreground applications. Reducing these thread pool sizes will compromise performance, but in the hybrid deployment scenario with limited resources, the cluster itself is hard to achieve high performance. In this scenario, the overall stability of the cluster is preferred over performance.

If you conduct an actual load test, you can first use the default configuration and observe the actual resource usage of each thread pool. Then you can adjust the corresponding configuration items and reduce the sizes of the thread pools that have lower usage.

`readpool.unified.max-thread-count`

The default value of this parameter is 80% of the number of machine threads. In a hybrid deployment scenario, you need to manually calculate and specify this value. You can first set it to 80% of the expected number of CPU threads used by TiKV.

`server.grpc-concurrency`

This parameter defaults to `4`. Because in the existing deployment plan, the CPU resources are limited and the actual requests are few. You can observe the monitoring panel, lower the value of this parameter, and keep the usage rate below 80%.

In this test, the value of this parameter is set to `2`. Observe the **gRPC poll CPU** panel and you can see that the usage rate is just around 80%.

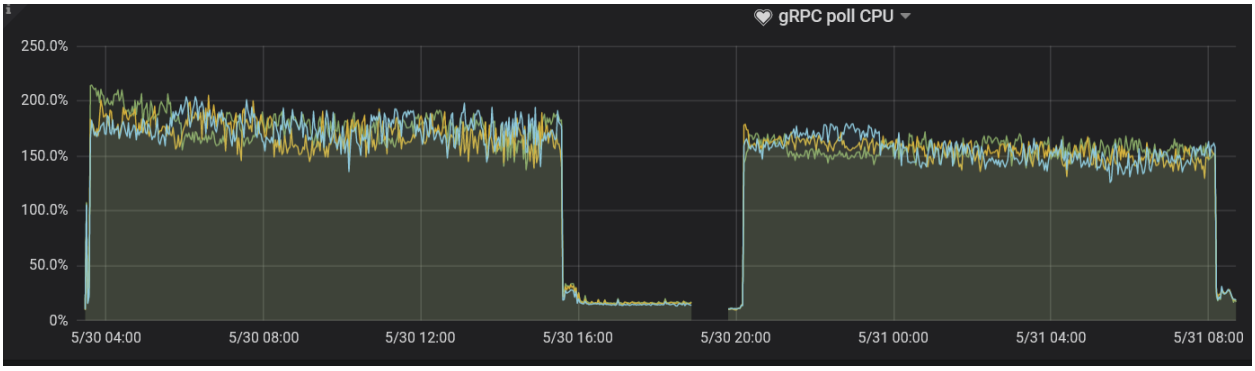


Figure 232: gRPC Pool CPU

`storage.scheduler-worker-pool-size`

When TiKV detects that the CPU core number of the machine is greater than or equal to `16`, this parameter value defaults to `8`. When the CPU core number is smaller than `16`, the parameter value defaults to `4`. This parameter is used when TiKV converts complex transaction requests to simple key-value reads or writes, but the scheduler thread pool does not performs any writes.

Ideally, the usage rate of the scheduler thread pool is kept between 50% and 75%. Similar to the gRPC thread pool, the `storage.scheduler-worker-pool-size` parameter defaults to a larger value during the hybrid deployment, which makes resource usage insufficient. In this test, the value of this parameter is set to `2`, which is in line with the best practices, a conclusion drawn by observing the corresponding metrics in the **Scheduler worker CPU** panel.



Figure 233: Scheduler Worker CPU

12.5.8.2.2 Resource configuration for TiKV background tasks

In addition to foreground tasks, TiKV regularly sorts data and cleans outdated data in background tasks. The default configuration allocates sufficient resources to these tasks for

the scenario of high-traffic writes.

However, in the hybrid deployment scenario, this default configuration is not in line with the best practices. You need to limit the resource usage of background tasks by adjusting the following parameters.

`rocksdb.max-background-jobs` and `rocksdb.max-sub-compactions`

The RocksDB thread pool is used to perform compaction and flush jobs. The default value of `rocksdb.max-background-jobs` is `8`, which obviously exceeds the resource that is in need. Therefore, the value should be adjusted to limit the resource usage.

`rocksdb.max-sub-compactions` indicates the number of concurrent sub-tasks allowed for a single compaction job, which defaults to `3`. You can lower this value when the write traffic is not high.

In the test, the `rocksdb.max-background-jobs` value is set to `3` and the `rocksdb.max-sub-compactions` value is set to `1`. No write stall occurs during the 12-hour test with the TPC-C load. When optimizing the two parameter values according to the actual load, you can lower the values gradually based on monitoring metrics:

- If write stall occurs, increase the value of `rocksdb.max-background-jobs`.
- If the write stall persists, set the value of `rocksdb.max-sub-compactions` to `2` or `3`.

`rocksdb.rate-bytes-per-sec`

This parameter is used to limit the disk traffic for the background compaction jobs. The default configuration has no limit for this parameter. To avoid the situation that compaction jobs occupy the resources of foreground services, you can adjust this parameter value according to the sequential read and write speed of the disk, which reserves enough disk bandwidth for foreground services.

The method of optimizing the RocksDB thread pool is similar to that of optimizing the compaction thread pool. You can determine whether the value you have adjusted is suitable according to whether write stall occurs.

`gc.max_write_bytes_per_sec`

Because TiDB uses the multi-version concurrency control (MVCC) model, TiKV periodically cleans old version data in the background. When the available resources are limited, this operation causes periodical performance jitter. You can use the `gc.max_write_bytes_per_sec` parameter to limit the resource usage of such an operation.



Figure 234: GC Impact

In addition to setting this parameter value in the configuration file, you can also dynamically adjust this value in tikv-ctl.

```
tiup ctl:<cluster-version> tikv --host=${ip:port} modify-tikv-config -n gc.
    ↪ max_write_bytes_per_sec -v ${limit}
```

Note:

In application scenarios with frequent updates, limiting GC traffic might cause the MVCC versions to pile up and affect read performance. Currently, to achieve a balance between performance jitter and performance decrease, you might need to try multiple times to adjust the value of this parameter.

12.5.8.2.3 TiDB parameter adjustment

Generally, you can adjust the TiDB parameters of execution operators using system variables such as `tidb_hash_join_concurrency` and `tidb_index_lookup_join_concurrency`.

In this test, these parameters are not adjusted. In the load test of your actual application, if the execution operators consume an excessive amount of CPU resources, you can limit the resource usage of specific operators according to your application scenario. For more details, see TiDB system variables.

`performance.max-procs`

This parameter is used to control how many CPU cores an entire Go process can use. By default, the value is equal to the number of CPU cores of the current machine or cgroups.

When Go is running, a proportion of threads is used for background tasks such as GC. If you do not limit the value of the `performance.max-procs` parameter, these background tasks will consume too much CPU.

12.5.9 Local Read under Three Data Centers Deployment

In the model of three data centers, a Region has three replicas which are isolated in each data center. However, due to the requirement of strongly consistent read, TiDB must access the Leader replica of the corresponding data for every query. If the query is generated in a data center different from that of the Leader replica, TiDB needs to read data from another data center, thus causing the access latency to increase.

This document describes how to use the Stale Read feature to avoid cross-center access and reduce the access latency at the expense of real-time data availability.

12.5.9.1 Deploy a TiDB cluster of three data centers

For the three-data-center deployment method, refer to Multiple Data Centers in One City Deployment.

Note that if both the TiKV and TiDB nodes have the configuration item `labels` configured, the TiKV and TiDB nodes in the same data center must have the same value for the `zone` label. For example, if a TiKV node and a TiDB node are both in the data center `dc-1`, then the two nodes need to be configured with the following label:

```
[labels]
zone=dc-1
```

12.5.9.2 Perform local read using Stale Read

Stale Read is a mechanism that TiDB provides for the users to read historical data. Using this mechanism, you can read the corresponding historical data of a specific point in time or within a specified time range, and thus save the latency brought by data replication between storage nodes. When using Stale Read in some scenarios of geo-distributed deployment, TiDB accesses the replica in the current data center to read the corresponding data at the expense of some real-time performance, which avoids network latency brought by cross-center connection and reduces the access latency for the entire query process.

When TiDB receives a Stale Read query, if the `zone` label of that TiDB node is configured, then TiDB sends the request to the TiKV node with the same `zone` label where the corresponding data replica resides.

For how to perform Stale Read, see Perform Stale Read using the `AS OF TIMESTAMP` clause.

12.5.10 UUID Best Practices

12.5.10.1 Overview of UUIDs

When used as a primary key, instead of an `AUTO_INCREMENT` integer value, a universally unique identifier (UUID) delivers the following benefits:

- UUIDs can be generated on multiple systems without risking conflicts. In some cases, this means that the number of network trips to TiDB can be reduced, leading to improved performance.
- UUIDs are supported by most programming languages and database systems.
- When used as a part of a URL, a UUID is not vulnerable to enumeration attacks. In comparison, with an `auto_increment` number, it is possible to guess the invoice IDs or user IDs.

12.5.10.2 Best practices

12.5.10.2.1 Store as binary

The textual UUID format looks like this: `ab06f63e-8fe7-11ec-a514-5405db7aad56`, which is a string of 36 characters. By using `UUID_TO_BIN()`, the textual format can be converted into a binary format of 16 bytes. This allows you to store the text in a `BINARY (16)` column. When retrieving the UUID, you can use the `BIN_TO_UUID()` function to get back to the textual format.

12.5.10.2.2 UUID format binary order and a clustered PK

The `UUID_TO_BIN()` function can be used with one argument, the UUID or with two arguments where the second argument is a `swap_flag`. It is recommended to not set the `swap_flag` with TiDB to avoid hotspots.

You can also explicitly set the `CLUSTERED` option for UUID based primary keys to avoid hotspots.

To demonstrate the effect of the `swap_flag`, here are two tables with an identical structure. The difference is that the data inserted into `uuid_demo_1` uses `UUID_TO_BIN(?, 0)` and `uuid_demo_2` uses `UUID_TO_BIN(?, 1)`.

In the screenshot of the Key Visualizer below, you can see that writes are concentrated in a single region of the `uuid_demo_2` table that has the order of the fields swapped in the binary format.

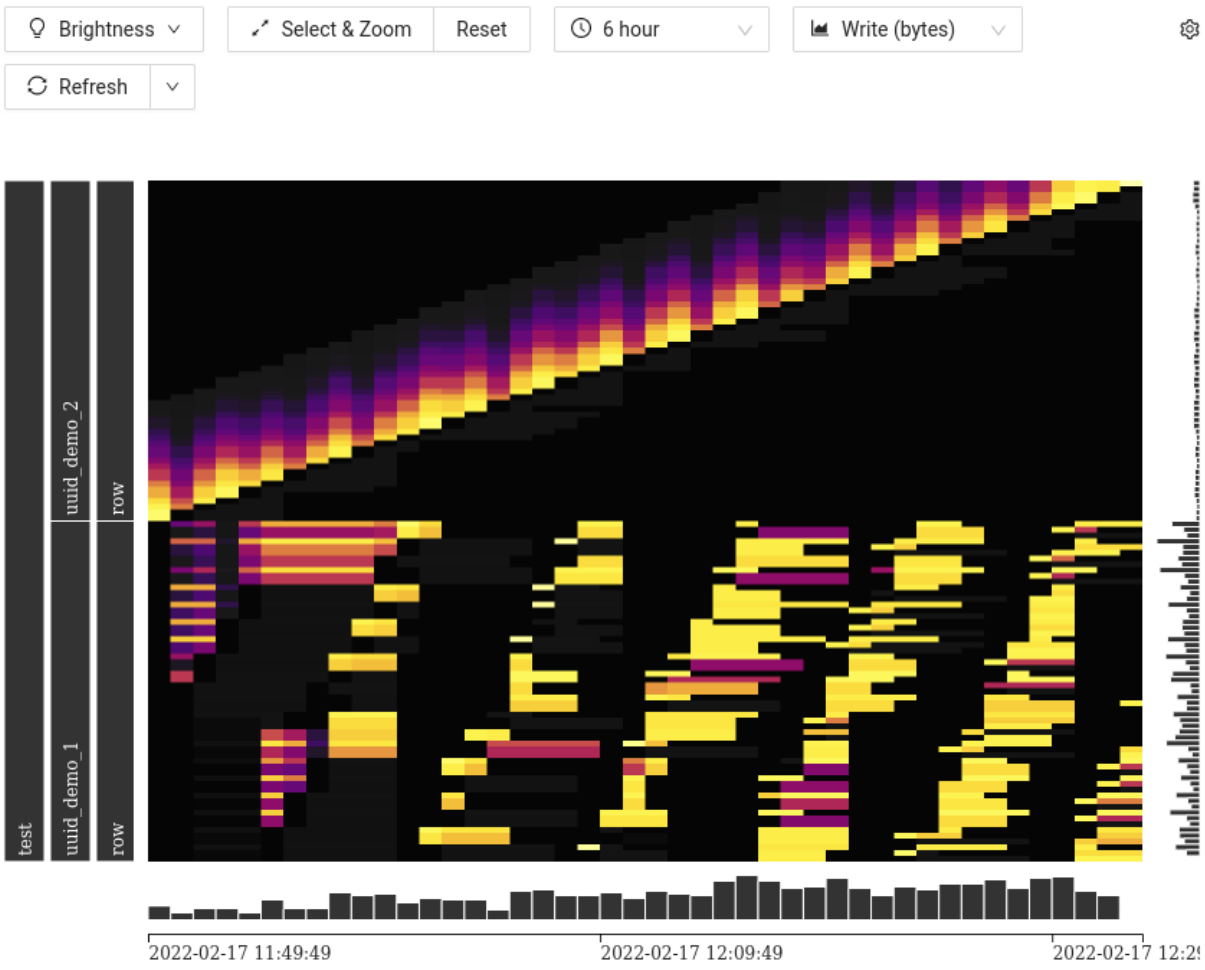


Figure 235: Key Visualizer

```
CREATE TABLE `uuid_demo_1` (
  `uuid` varbinary(16) NOT NULL,
  `c1` varchar(255) NOT NULL,
  PRIMARY KEY (`uuid`) CLUSTERED
)
```

```
CREATE TABLE `uuid_demo_2` (
  `uuid` varbinary(16) NOT NULL,
  `c1` varchar(255) NOT NULL,
  PRIMARY KEY (`uuid`) CLUSTERED
)
```

12.5.10.3 MySQL compatibility

UUIDs can be used in MySQL as well. The `BIN_TO_UUID()` and `UUID_TO_BIN()` functions were introduced in MySQL 8.0. The `UUID()` function is available in earlier MySQL versions as well.

12.6 Placement Rules

> **Note:**
>
> This document introduces how to manually specify placement rules in Placement Driver (PD). It is now recommended to use Placement Rules in SQL. This offers a more convenient way to configure the placement of tables and partitions.

Placement Rules, introduced in v5.0, is a replica rule system that guides PD to generate corresponding schedules for different types of data. By combining different scheduling rules, you can finely control the attributes of any continuous data range, such as the number of replicas, the storage location, the host type, whether to participate in Raft election, and whether to act as the Raft leader.

The Placement Rules feature is enabled by default in v5.0 and later versions of TiDB. To disable it, refer to Disable Placement Rules.

12.6.1 Rule system

The configuration of the whole rule system consists of multiple rules. Each rule can specify attributes such as the number of replicas, the Raft role, the placement location, and the key range in which this rule takes effect. When PD is performing schedule, it first finds the rule corresponding to the Region in the rule system according to the key range of the Region, and then generates the corresponding schedule to make the distribution of the Region replica comply with the rule.

The key ranges of multiple rules can have overlapping parts, which means that a Region can match multiple rules. In this case, PD decides whether the rules overwrite each other or take effect at the same time according to the attributes of rules. If multiple rules take effect at the same time, PD will generate schedules in sequence according to the stacking order of the rules for rule matching.

In addition, to meet the requirement that rules from different sources are isolated from each other, these rules can be organized in a more flexible way. Therefore, the concept of "Group" is introduced. Generally, users can place rules in different groups according to different sources.

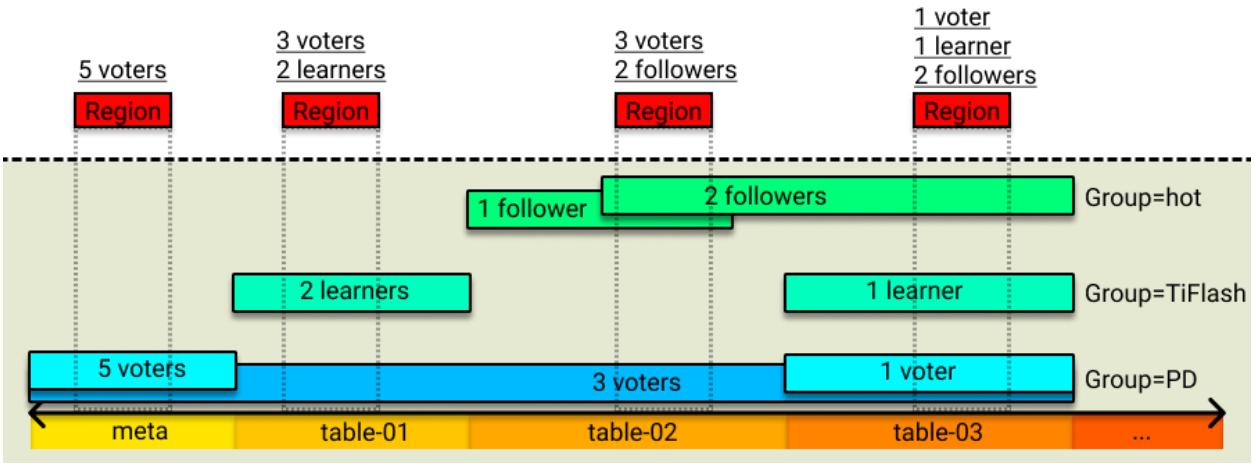


Figure 236: Placement rules overview

12.6.1.1 Rule fields

The following table shows the meaning of each field in a rule:

Field name	Type and restriction	Description
`GroupID`	`string`	The group ID that marks the source of the rule.
`ID`	`string`	The unique ID of a rule in a group.
`Index`	`int`	The stacking sequence of rules in a group.
`Override`	`true/false`	Whether to overwrite rules with smaller index (in a g
`StartKey`	`string`, in hexadecimal form	Applies to the starting key of a range.
`EndKey`	`string`, in hexadecimal form	Applies to the ending key of a range.
`Role`	`string`	Replica roles, including voter/leader/follower/learner.
`Count`	`int`, positive integer	The number of replicas.
`LabelConstraint`	`[]Constraint`	Filters nodes based on the label.
`LocationLabels`	`[]string`	Used for physical isolation.
`IsolationLevel`	`string`	Used to set the minimum physical isolation level

`LabelConstraint` is similar to the function in Kubernetes that filters labels based on these four primitives: `in`, `notIn`, `exists`, and `notExists`. The meanings of these four primitives are as follows:

- `in`: the label value of the given key is included in the given list.
- `notIn`: the label value of the given key is not included in the given list.
- `exists`: includes the given label key.
- `notExists`: does not include the given label key.

The meaning and function of `LocationLabels` are the same with those earlier than v4.0. For example, if you have deployed `[zone,rack,host]` that defines a three-layer topology:

the cluster has multiple zones (Availability Zones), each zone has multiple racks, and each rack has multiple hosts. When performing schedule, PD first tries to place the Region's peers in different zones. If this try fails (such as there are three replicas but only two zones in total), PD guarantees to place these replicas in different racks. If the number of racks is not enough to guarantee isolation, then PD tries the host-level isolation.

The meaning and function of `IsolationLevel` is elaborated in Cluster topology configuration. For example, if you have deployed `[zone,rack,host]` that defines a three-layer topology with `LocationLabels` and set `IsolationLevel` to `zone`, then PD ensures that all peers of each Region are placed in different zones during the scheduling. If the minimum isolation level restriction on `IsolationLevel` cannot be met (for example, 3 replicas are configured but there are only 2 data zones in total), PD will not try to make up to meet this restriction. The default value of `IsolationLevel` is an empty string, which means that it is disabled.

12.6.1.2 Fields of the rule group

The following table shows the description of each field in a rule group:

Field name	Type and restriction	Description
`ID`	`string`	The group ID that marks the source of the rule.
`Index`	`int`	The stacking sequence of different groups.
`Override`	`true/false`	Whether to override groups with smaller indexes.

12.6.2 Configure rules

The operations in this section are based on pd-ctl, and the commands involved in the operations also support calls via HTTP API.

12.6.2.1 Enable Placement Rules

The Placement Rules feature is enabled by default in v5.0 and later versions of TiDB. To disable it, refer to Disable Placement Rules. To enable this feature after it has been disabled, you can modify the PD configuration file as follows before initializing the cluster:

```
[replication]
enable-placement-rules = true
```

In this way, PD enables this feature after the cluster is successfully bootstrapped and generates corresponding rules according to the `max-replicas`, `location-labels`, and `isolation-level` configurations:

```
{
  "group_id": "pd",
  "id": "default",
  "start_key": "",
```

```
  "end_key": "",
  "role": "voter",
  "count": 3,
  "location_labels": ["zone", "rack", "host"],
  "isolation_level": ""
}
```

For a bootstrapped cluster, you can also enable Placement Rules dynamically through pd-ctl:

```
pd-ctl config placement-rules enable
```

PD also generates default rules based on the `max-replicas`, `location-labels`, and `isolation-level` configurations.

Note:

- When Placement Rules are enabled and multiple rules exist, the previously configured `max-replicas`, `location-labels`, and `isolation-level` no longer take effect. To adjust the replica policy, use the interface related to Placement Rules.
- When Placement Rules are enabled and only one default rule exists, TiDB will automatically update this default rule when `max-replicas`, `location-labels`, or `isolation-level` configurations are changed.

12.6.2.2 Disable Placement Rules

You can use pd-ctl to disable the Placement Rules feature and switch to the previous scheduling strategy.

```
pd-ctl config placement-rules disable
```

Note:

After disabling Placement Rules, PD uses the original `max-replicas`, `location-labels`, and `isolation-level` configurations. The modification of rules (when Placement Rules is enabled) will not update these three configurations in real time. In addition, all the rules that have been configured remain in PD and will be used the next time you enable Placement Rules.

12.6.2.3 Set rules using pd-ctl

> **Note:**
>
> The change of rules affects the PD scheduling in real time. Improper rule setting might result in fewer replicas and affect the high availability of the system.

pd-ctl supports using the following methods to view rules in the system, and the output is a JSON-format rule or a rule list.

- To view the list of all rules:

```
pd-ctl config placement-rules show
```

- To view the list of all rules in a PD Group:

```
pd-ctl config placement-rules show --group=pd
```

- To view the rule of a specific ID in a Group:

```
pd-ctl config placement-rules show --group=pd --id=default
```

- To view the rule list that matches a Region:

```
pd-ctl config placement-rules show --region=2
```

  In the above example, `2` is the Region ID.

Adding rules and editing rules are similar. You need to write the corresponding rules into a file and then use the `save` command to save the rules to PD:

```
cat > rules.json <<EOF
[
    {
        "group_id": "pd",
        "id": "rule1",
        "role": "voter",
        "count": 3,
        "location_labels": ["zone", "rack", "host"]
    },
    {
        "group_id": "pd",
```

```
        "id": "rule2",
        "role": "voter",
        "count": 2,
        "location_labels": ["zone", "rack", "host"]
    }
]
EOF
pd-ctl config placement save --in=rules.json
```

The above operation writes `rule1` and `rule2` to PD. If a rule with the same `GroupID` + `ID` already exists in the system, this rule is overwritten.

To delete a rule, you only need to set the `count` of the rule to `0`, and the rule with the same `GroupID` + `ID` will be deleted. The following command deletes the `pd / rule2` rule:

```
cat > rules.json <<EOF
[
    {
        "group_id": "pd",
        "id": "rule2"
    }
]
EOF
pd-ctl config placement save --in=rules.json
```

12.6.2.4 Use pd-ctl to configure rule groups

- To view the list of all rule groups:

```
pd-ctl config placement-rules rule-group show
```

- To view the rule group of a specific ID:

```
pd-ctl config placement-rules rule-group show pd
```

- To set the `index` and `override` attributes of the rule group:

```
pd-ctl config placement-rules rule-group set pd 100 true
```

- To delete the configuration of a rule group (use the default group configuration if there is any rule in the group):

```
pd-ctl config placement-rules rule-group delete pd
```

12.6.2.5 Use pd-ctl to batch update groups and rules in groups

To view and modify the rule groups and all rules in the groups at the same time, execute the `rule-bundle` subcommand.

In this subcommand, `get {group_id}` is used to query a group, and the output result shows the rule group and rules of the group in a nested form:

```
pd-ctl config placement-rules rule-bundle get pd
```

The output of the above command:

```
{
  "group_id": "pd",
  "group_index": 0,
  "group_override": false,
  "rules": [
    {
      "group_id": "pd",
      "id": "default",
      "start_key": "",
      "end_key": "",
      "role": "voter",
      "count": 3
    }
  ]
}
```

To write the output to a file, add the `--out` argument to the `rule-bundle get` subcommand, which is convenient for subsequent modification and saving.

```
pd-ctl config placement-rules rule-bundle get pd --out="group.json"
```

After the modification is finished, you can use the `rule-bundle set` subcommand to save the configuration in the file to the PD server. Unlike the `save` command described in Set rules using pd-ctl, this command replaces all the rules of this group on the server side.

```
pd-ctl config placement-rules rule-bundle set pd --in="group.json"
```

12.6.2.6 Use pd-ctl to view and modify all configurations

You can also view and modify all configuration using pd-ctl. To do that, save all configuration to a file, edit the configuration file, and then save the file to the PD server to overwrite the previous configuration. This operation also uses the `rule-bundle` subcommand.

For example, to save all configuration to the `rules.json` file, execute the following command:

```
pd-ctl config placement-rules rule-bundle load --out="rules.json"
```

After editing the file, execute the following command to save the configuration to the PD server:

```
pd-ctl config placement-rules rule-bundle save --in="rules.json"
```

12.6.2.7 Use tidb-ctl to query the table-related key range

If you need special configuration for metadata or a specific table, you can execute the `keyrange` command in tidb-ctl to query related keys. Remember to add `--encode` at the end of the command.

```
tidb-ctl keyrange --database test --table ttt --encode
```

```
global ranges:
  meta: (6d00000000000000f8, 6e00000000000000f8)
  table: (7400000000000000f8, 7500000000000000f8)
table ttt ranges: (NOTE: key range might be changed after DDL)
  table: (7480000000000000ff2d00000000000000f8, 7480000000000000
    ↪ ff2e00000000000000f8)
  table indexes: (7480000000000000ff2d5f690000000000fa, 7480000000000000
    ↪ ff2d5f720000000000fa)
    index c2: (7480000000000000ff2d5f698000000000ff0000010000000000fa,
      ↪ 7480000000000000ff2d5f698000000000ff0000020000000000fa)
    index c3: (7480000000000000ff2d5f698000000000ff0000020000000000fa,
      ↪ 7480000000000000ff2d5f698000000000ff0000030000000000fa)
    index c4: (7480000000000000ff2d5f698000000000ff0000030000000000fa,
      ↪ 7480000000000000ff2d5f698000000000ff0000040000000000fa)
  table rows: (7480000000000000ff2d5f720000000000fa, 7480000000000000
    ↪ ff2e00000000000000f8)
```

> **Note:**
>
> DDL and other operations can cause table ID changes, so you need to update the corresponding rules at the same time.

12.6.3 Typical usage scenarios

This section introduces the typical usage scenarios of Placement Rules.

12.6.3.1 Scenario 1: Use three replicas for normal tables and five replicas for the metadata to improve cluster disaster tolerance

You only need to add a rule that limits the key range to the range of metadata, and set the value of `count` to `5`. Here is an example of this rule:

```
{
  "group_id": "pd",
  "id": "meta",
  "index": 1,
  "override": true,
  "start_key": "6d00000000000000f8",
  "end_key": "6e00000000000000f8",
  "role": "voter",
  "count": 5,
  "location_labels": ["zone", "rack", "host"]
}
```

12.6.3.2 Scenario 2: Place five replicas in three data centers in the proportion of 2:2:1, and the Leader should not be in the third data center

Create three rules. Set the number of replicas to `2`, `2`, and `1` respectively. Limit the replicas to the corresponding data centers through `label_constraints` in each rule. In addition, change `role` to `follower` for the data center that does not need a Leader.

```
[
   {
       "group_id": "pd",
       "id": "zone1",
       "start_key": "",
       "end_key": "",
       "role": "voter",
       "count": 2,
       "label_constraints": [
          {"key": "zone", "op": "in", "values": ["zone1"]}
       ],
       "location_labels": ["rack", "host"]
   },
   {
       "group_id": "pd",
       "id": "zone2",
       "start_key": "",
       "end_key": "",
       "role": "voter",
       "count": 2,
       "label_constraints": [
          {"key": "zone", "op": "in", "values": ["zone2"]}
       ],
       "location_labels": ["rack", "host"]
```

```
    },
    {
        "group_id": "pd",
        "id": "zone3",
        "start_key": "",
        "end_key": "",
        "role": "follower",
        "count": 1,
        "label_constraints": [
            {"key": "zone", "op": "in", "values": ["zone3"]}
        ],
        "location_labels": ["rack", "host"]
    }
]
```

12.6.3.3 Scenario 3: Add two TiFlash replicas for a table

Add a separate rule for the row key of the table and limit `count` to `2`. Use `label_constraints` to ensure that the replicas are generated on the node of `engine = tiflash`. Note that a separate `group_id` is used here to ensure that this rule does not overlap or conflict with rules from other sources in the system.

```
{
  "group_id": "tiflash",
  "id": "learner-replica-table-ttt",
  "start_key": "7480000000000000ff2d5f720000000000fa",
  "end_key": "7480000000000000ff2e000000000000f8",
  "role": "learner",
  "count": 2,
  "label_constraints": [
    {"key": "engine", "op": "in", "values": ["tiflash"]}
  ],
  "location_labels": ["host"]
}
```

12.6.3.4 Scenario 4: Add two follower replicas for a table in the Beijing node with high-performance disks

The following example shows a more complicated `label_constraints` configuration. In this rule, the replicas must be placed in the `bj1` or `bj2` machine room, and the disk type must be `nvme`.

```
{
  "group_id": "follower-read",
  "id": "follower-read-table-ttt",
```

```
  "start_key": "7480000000000000ff2d00000000000000f8",
  "end_key": "7480000000000000ff2e00000000000000f8",
  "role": "follower",
  "count": 2,
  "label_constraints": [
    {"key": "zone", "op": "in", "values": ["bj1", "bj2"]},
    {"key": "disk", "op": "in", "values": ["nvme"]}
  ],
  "location_labels": ["host"]
}
```

12.6.3.5 Scenario 5: Migrate a table to the nodes with SSD disks

Different from scenario 3, this scenario is not to add new replica(s) on the basis of the existing configuration, but to forcibly override the other configuration of a data range. So you need to specify an `index` value large enough and set `override` to `true` in the rule group configuration to override the existing rule.

The rule:

```
{
  "group_id": "ssd-override",
  "id": "ssd-table-45",
  "start_key": "7480000000000000ff2d5f720000000000fa",
  "end_key": "7480000000000000ff2e00000000000000f8",
  "role": "voter",
  "count": 3,
  "label_constraints": [
    {"key": "disk", "op": "in", "values": ["ssd"]}
  ],
  "location_labels": ["rack", "host"]
}
```

The rule group:

```
{
  "id": "ssd-override",
  "index": 1024,
  "override": true,
}
```

12.7 Load Base Split

Load Base Split is a new feature introduced in TiDB 4.0. It aims to solve the hotspot issue caused by unbalanced access between Regions, such as full table scans for small tables.

12.7.1 Scenarios

In TiDB, it is easy to generate hotspots when the load is concentrated on certain nodes. PD tries to schedule the hot Regions so that they are distributed as evenly as possible across all nodes for better performance.

However, the minimum unit for PD scheduling is Region. If the number of hotspots in a cluster is smaller than the number of nodes, or if a few hotspots have far more load than other Regions, PD can only move the hotspot from one node to another, but not make the entire cluster share the load.

This scenario is especially common with workloads that are mostly read requests, such as full table scans and index lookups for small tables, or frequent access to some fields.

Previously, the solution to this problem was to manually execute a command to split one or more hotspot Regions, but this approach has two problems:

- Evenly splitting a Region is not always the best choice, because requests might be concentrated on a few keys. In such cases, hotspots might still be on one of the Regions after evenly splitting, and it might take multiple even splits to realize the goal.
- Human intervention is not timely or simple.

12.7.2 Implementation principles

Load Base Split automatically splits the Region based on statistics. It identifies the Regions whose read load consistently exceeds the threshold for 10 seconds, and splits these Regions at a proper position. When choosing the split position, Load Base Split tries to balance the access load of both Regions after the split and avoid access across Regions.

The Region split by Load Base Split will not be merged quickly. On the one hand, PD's `MergeChecker` skips the hot Regions; on the other hand, PD also determines whether to merge two Regions according to `QPS` in the heartbeat information, to avoid the merging of two Regions with high `QPS`.

12.7.3 Usage

The Load Base Split feature is currently controlled by the `split.qps-threshold` parameter (QPS threshold) and `split.byte-threshold` parameter (traffic threshold). If the sum of all types of read requests per second for a Region exceeds the QPS threshold or traffic threshold for 10 consecutive seconds, PD splits the Region.

Load Base Split is enabled by default, but the parameter is set to a rather high value. `split.qps-threshold` defaults to `3000` and `split.byte-threshold` defaults to 30MB/s. If you want to disable this feature, set the two thresholds high enough at the same time.

To modify the parameter, take either of the following two methods:

- Use a SQL statement:

```
# Set the QPS threshold to 1500
SET config tikv split.qps-threshold=1500;
# Set the byte threshold to 15 MiB (15 * 1024 * 1024)
SET config tikv split.byte-threshold=15728640;
```

- Use TiKV:

```
curl -X POST "http://ip:status_port/config" -H "accept: application/
    ↪ json" -d '{"split.qps-threshold":"1500"}'
curl -X POST "http://ip:status_port/config" -H "accept: application/
    ↪ json" -d '{"split.byte-threshold":"15728640"}'
```

Accordingly, you can view the configuration by either of the following two methods:

- Use a SQL statement:

```
show config where type='tikv' and name like '%split.qps-threshold%';
```

- Use TiKV:

```
curl "http://ip:status_port/config"
```

> **Note:**
>
> Starting from v4.0.0-rc.2, you can modify and view the configuration using SQL statements.

12.8 Store Limit

Store Limit is a feature of PD, introduced in TiDB 3.0. It is designed to control the scheduling speed in a finer manner for better performance in different scenarios.

12.8.1 Implementation principles

PD performs scheduling at the unit of operator. An operator might contain several scheduling operations. For example:

```
"replace-down-replica {mv peer: store [2] to [3]} (kind:region,replica,
    ↪ region:10(4,5), createAt:2020-05-18 06:40:25.775636418 +0000 UTC m
    ↪ =+2168762.679540369, startAt:2020-05-18 06:40:25.775684648 +0000 UTC
    ↪ m=+2168762.679588599, currentStep:0, steps:[add learner peer 20 on
    ↪ store 3, promote learner peer 20 on store 3 to voter, remove peer on
    ↪ store 2])"
```

In this above example, the `replace-down-replica` operator contains the following specific operations:

1. Add a learner peer with the ID 20 to `store 3`.
2. Promote the learner peer with the ID 20 on `store 3` to a voter.
3. Delete the peer on `store 2`.

Store Limit achieves the store-level speed limit by maintaining a mapping from store IDs to token buckets in memory. The different operations here correspond to different token buckets. Currently, Store Limit only supports limiting the speed of two operations: adding learners/peers and deleting peers. That is, each store has two types of token buckets.

Every time an operator is generated, it checks whether enough tokens exist in the token buckets for its operations. If yes, the operator is added to the scheduling queue, and the corresponding token is taken from the token bucket. Otherwise, the operator is abandoned. Because the token bucket replenishes tokens at a fixed rate, the speed limit is thus achieved.

Store Limit is different from other limit-related parameters in PD (such as `region-schedule-limit` and `leader-schedule-limit`) in that it mainly limits the consuming speed of operators, while other parameters limits the generating speed of operators. Before introducing the Store Limit feature, the speed limit of scheduling is mostly at the global scope. Therefore, even if the global speed is limited, it is still possible that the scheduling operations are concentrated on some stores, affecting the performance of the cluster. By limiting the speed at a finer level, Store Limit can better control the scheduling behavior.

12.8.2 Usage

The parameters of Store Limit can be configured using `pd-ctl`.

12.8.2.1 View setting of the current store

To view the limit setting of the current store, run the following commands:

```
store limit                     // Shows the speed limit of adding and
    ↪ deleting peers in all stores.
store limit add-peer            // Shows the speed limit of adding peers in
    ↪ all stores.
store limit remove-peer         // Shows the speed limit of deleting peers
    ↪ in all stores.
```

12.8.2.2 Set limit for all stores

To set the speed limit for all stores, run the following commands:

```
store limit all 5               // All stores can at most add and delete 5
    ↪ peers per minute.
store limit all 5 add-peer      // All stores can at most add 5 peers per
    ↪ minute.
store limit all 5 remove-peer   // All stores can at most delete 5 peers per
    ↪  minute.
```

12.8.2.3 Set limit for a single store

To set the speed limit for a single store, run the following commands:

```
store limit 1 5                 // store 1 can at most add and delete 5
    ↪ peers per minute.
store limit 1 5 add-peer        // store 1 can at most add 5 peers per
    ↪ minute.
store limit 1 5 remove-peer     // store 1 can at most delete 5 peers per
    ↪ minute.
```

13 TiDB Tools

13.1 TiDB Tools Overview

TiDB provides a rich set of tools to help you deploy and maintain TiDB, manage data (such as data migration, backup & restore, and data comparison), and run Spark SQL on TiKV. You can select the applicable tools according to your needs.

13.1.1 Deployment and operation Tools

TiDB provides TiUP and TiDB Operator to meet your deployment and operation needs in different system environments.

13.1.1.1 Deploy and operate TiDB on physical or virtual machines - TiUP

TiUP is a TiDB package manager on physical or virtual machines. TiUP can manage multiple TiDB components such as TiDB, PD, and TiKV. To start any component in the TiDB ecosystem, you just need to execute a single line of TiUP command.

TiUP provides TiUP cluster, a cluster management component written in Golang. By using TiUP cluster, you can easily perform daily database operations, including deploying, starting, stopping, destroying, scaling, and upgrading a TiDB cluster, and manage TiDB cluster parameters.

The following are the basics of TiUP:

- Terminology and Concepts
- Deploy a TiDB Cluster Using TiUP
- Manage TiUP Components with TiUP Commands
- Applicable TiDB versions: v4.0 and later versions

13.1.1.2 Deploy and operate TiDB on Kubernetes - TiDB Operator

TiDB Operator is an automatic operation system for managing TiDB clusters on Kubernetes. It provides full life-cycle management for TiDB including deployment, upgrades, scaling, backup, and configuration changes. With TiDB Operator, TiDB can run seamlessly in the Kubernetes clusters deployed on a public or private cloud.

The following are the basics of TiDB Operator:

- TiDB Operator Architecture
- Get Started with TiDB Operator on Kubernetes
- Applicable TiDB versions: v2.1 and later versions

13.1.2 Data management tools

TiDB provides multiple data management tools, such as import and export, backup and restore, incremental data replication, and data validation.

13.1.2.1 Data migration - TiDB Data Migration (DM)

TiDB Data Migration (DM) is a tool that supports full data migration and incremental data replication from MySQL/MariaDB to TiDB.

The following are the basics of DM:

- Source: MySQL/MariaDB
- Target: TiDB clusters
- Supported TiDB versions: all versions
- Kubernetes support: use TiDB Operator to deploy TiDB DM on Kubernetes.

If the data volume is less than 1 TB, it is recommended to migrate data from MySQL/-MariaDB to TiDB directly using DM. The migration process includes full data migration and incremental data replication.

If the data volume is greater than 1 TB , take the following steps:

1. Use Dumpling to export the full data from MySQL/MariaDB.
2. Use TiDB Lightning to import the data exported in Step 1 to the TiDB cluster.
3. Use TiDB DM to replicate the incremental data from MySQL/MariaDB to TiDB.

Note:

The Syncer tool is no longer maintained. For scenarios related to Syncer, it is recommended that you use DM to perform incremental replication.

13.1.2.2 Full data export - Dumpling

Dumpling supports logical full data export from MySQL or TiDB.

The following are the basics of Dumpling:

- Source: MySQL/TiDB clusters
- Output: SQL/CSV files
- Supported TiDB versions: all versions
- Kubernetes support: No

Note:

PingCAP previously maintained a fork of the mydumper project with enhancements specific to TiDB. For more information on Mydumper, refer to v4.0 Mydumper documentation. Starting from v7.5.0, Mydumper is deprecated and most of its features have been replaced by Dumpling. It is strongly recommended that you use Dumpling instead of Mydumper.

13.1.2.3 Full data import - TiDB Lightning

TiDB Lightning supports full data import of a large dataset into a TiDB cluster.

TiDB Lightning supports the following modes:

- `Physical Import Mode`: TiDB Lightning parses data into ordered key-value pairs and directly imports them into TiKV. This mode is usually for importing a large amount of data (at the TB level) to a new cluster. During the import, the cluster cannot provide services.
- `Logical Import Mode`: This mode uses TiDB/MySQL as the backend, which is slower than the `Physical Import Mode` but can be performed online. It also supports importing data to MySQL.

The following are the basics of TiDB Lightning:

- Data source:
  - The output files of Dumpling
  - Other compatible CSV files
  - Parquet files exported from Amazon Aurora or Apache Hive
- Supported TiDB versions: v2.1 and later versions
- Kubernetes support: Yes. See Quickly restore data into a TiDB cluster on Kubernetes using TiDB Lightning for details.

> **Note:**
>
> The Loader tool is no longer maintained. For scenarios related to Loader, it is recommended that you use `Logical Import Mode` instead.

13.1.2.4 Backup and restore - Backup & Restore (BR)

Backup & Restore (BR) is a command-line tool for distributed backup and restore of the TiDB cluster data. BR can effectively back up and restore TiDB clusters of huge data volume.

The following are the basics of BR:

- Input and output data source: SST + `backupmeta` file
- Supported TiDB versions: v4.0 and later versions
- Kubernetes support: Yes. See Back up Data to S3-Compatible Storage Using BR and Restore Data from S3-Compatible Storage Using BR for details.

13.1.2.5 Incremental data replication - TiCDC

TiCDC is a tool used for replicating incremental data of TiDB by pulling change logs from TiKV. It can restore data to a state consistent with any TSO in upstream. TiCDC also provides the TiCDC Open Protocol to support other systems to subscribe to data changes.

The following are the basics of TiCDC:

- Source: TiDB clusters
- Target: TiDB clusters, MySQL, Kafka, and Confluent
- Supported TiDB versions: v4.0.6 and later versions

13.1.2.6 Incremental log replication - TiDB Binlog

TiDB Binlog is a tool that collects binlog for TiDB clusters and provides nearly real-time data replication and backup. You can use it for incremental data replication between TiDB clusters, such as making a TiDB cluster the secondary cluster of the primary TiDB cluster.

The following are the basics of TiDB Binlog:

- Source: TiDB clusters
- Target: TiDB clusters, MySQL, Kafka, or incremental backup files
- Supported TiDB versions: v2.1 and later versions
- Kubernetes support: Yes. See TiDB Binlog Cluster Operations and TiDB Binlog Drainer Configurations on Kubernetes for details.

13.1.2.7 sync-diff-inspector

sync-diff-inspector is a tool that compares data stored in the MySQL or TiDB databases. In addition, you can also use sync-diff-inspector to repair data in the scenario where a small amount of data is inconsistent.

The following are the basics of sync-diff-inspector:

- Source: MySQL/TiDB clusters
- Target: MySQL/TiDB clusters
- Supported TiDB versions: all versions

13.1.3 OLAP Query tool - TiSpark

TiSpark is a product developed by PingCAP to address the complexiy of OLAP queries. It combines strengths of Spark, and the features of distributed TiKV clusters and TiDB to provide a one-stop Hybrid Transactional and Analytical Processing (HTAP) solution.

13.2 TiDB Tools Use Cases

This document introduces the common use cases of TiDB tools and how to choose the right tool for your scenario.

13.2.1 Deploy and operate TiDB on physical or virtual machines

If you need to deploy and operate TiDB on physical or virtual machines, you can install TiUP, and then use TiUP to manage TiDB components such as TiDB, PD, and TiKV.

13.2.2 Deploy and operate TiDB on Kubernetes

If you need to deploy and operate TiDB on Kubernetes, you can deploy a Kubernetes cluster, and then deploy TiDB Operator. After that, you can use TiDB Operator to deploy and operate a TiDB cluster.

13.2.3 Import data from CSV to TiDB

If you need to import the compatible CSV files exported by other tools to TiDB, use TiDB Lightning.

13.2.4 Import full data from MySQL/Aurora

If you need to import full data from MySQL/Aurora, use Dumpling first to export data as SQL dump files, and then use TiDB Lightning to import data into the TiDB cluster.

13.2.5 Migrate data from MySQL/Aurora

If you need to migrate both full data and incremental data from MySQL/Aurora, use TiDB Data Migration (DM) to perform the Migrate Data from Amazon Aurora to TiDB.

If the full data volume is large (at the TB level), you can first use Dumpling and TiDB Lightning to perform the full data migration, and then use DM to perform the incremental data migration.

13.2.6 Back up and restore TiDB cluster

If you need to back up a TiDB cluster or restore backed up data to the cluster, use BR (Backup & Restore).

In addition, BR can also be used to perform incremental backup and incremental restore of TiDB cluster data.

13.2.7 Migrate data to TiDB

If you need to migrate data from a TiDB cluster to another TiDB cluster, use Dumpling to export full data from TiDB as SQL dump files, and then use TiDB Lightning to import data to another TiDB cluster.

If you also need to migrate incremental data, you can use TiCDC.

13.2.8 TiDB incremental data subscription

If you need to subscribe to TiDB's incremental changes, you can use TiCDC.

13.3 Download TiDB Tools

This document describes how to download the TiDB Toolkit.

TiDB Toolkit contains frequently used TiDB tools, such as data export tool Dumpling, data import tool TiDB Lightning, and backup and restore tool BR.

Tip:

- If your deployment environment has internet access, you can deploy a TiDB tool using a single TiUP command, so there is no need to download the TiDB Toolkit separately.
- If you need to deploy and maintain TiDB on Kubernetes, instead of downloading the TiDB Toolkit, follow the steps in TiDB Operator offline installation.

13.3.1 Environment requirements

- Operating system: Linux
- Architecture: amd64

13.3.2 Download link

You can download TiDB Toolkit from the following link:

```
https://download.pingcap.org/tidb-community-toolkit-{version}-linux-amd64.
    ↪ tar.gz
```

`{version}` in the link indicates the version number of TiDB. For example, the download link for `v6.1.7` is `https://download.pingcap.org/tidb-community-toolkit-v6.1.7-linux-amd64.tar.gz`.

Note:

If you need to download the PD Control tool `pd-ctl`, download the TiDB installation package separately from `https://download.pingcap.org/tidb-community-server-{version}-linux-amd64.tar.gz`.

13.3.3 TiDB Toolkit description

Depending on which tools you want to use, you can install the corresponding offline packages as follows:

Tool	Offline package name
TiUP	tiup-linux-amd64.tar.gz tiup-{tiup-version}-linux-amd64.tar.gz dm-{tiup-version}-linux-amd64.tar.gz server-{version}-linux-amd64.tar.gz
Dumpling	dumpling-{version}-linux-amd64.tar.gz
TiDB Lightning	tidb-lightning-ctl tidb-lightning-{version}-linux-amd64.tar.gz

Tool	Offline package name

TiDB Data Migration (DM)	dm-worker
	↪ -{
	↪ version
	↪ }-linux
	↪ -amd64.
	↪ tar.gz
	dm-master
	↪ -{
	↪ version
	↪ }-linux
	↪ -amd64.
	↪ tar.gz
	dmctl-{
	↪ version
	↪ }-linux
	↪ -amd64.
	↪ tar.gz

TiCDC	cdc-{ ↪ version ↪ }-linux ↪ -amd64. ↪ tar.gz

TiDB Binlog	pump-{
	↪ version
	↪ }-linux
	↪ -amd64.
	↪ tar.gz
	drainer-{
	↪ version
	↪ }-linux
	↪ -amd64.
	↪ tar.gz
	binlogctl
	reparo

Backup & Restore (BR)	br-{ ↪ version ↪ }-linux ↪ -amd64. ↪ tar.gz

Tool	Offline package name
sync-diff-inspector	sync_diff_inspector ↪
TiSpark	tispark-{ ↪ tispark ↪ - ↪ version ↪ }-any- ↪ any.tar ↪ .gz
	spark-{ ↪ spark- ↪ version ↪ }-any- ↪ any.tar ↪ .gz
PD Recover	pd-recover ↪ -{ ↪ version ↪ }-linux ↪ -amd64. ↪ tar

13.4 TiUP

13.4.1 TiUP Documentation Map

13.4.1.1 User guide

- TiUP Overview: Gives an overall introduction to TiUP, for example, how to install and use TiUP, and the related terminologies.
- TiUP Terminology and Concepts: Explains the terms that you might bump into when using TiUP, and help you understand the key concepts of TiUP
- TiUP Component Management: Introduces all TiUP commands in detail, and how to use TiUP to download, update and delete components
- TiUP FAQ: Introduces common issues when you use TiUP, including FAQs of the third-party components of TiUP
- TiUP Troubleshooting Guide: Introduces the troubleshooting methods and solutions if you encounter issues when using TiUP
- TiUP Reference Guide: Introduces detailed references, including commands, components, and mirrors.

13.4.1.2 TiUP resources

- TiUP Issues: Lists TiUP GitHub issues

13.4.2 TiUP Overview

Starting with TiDB 4.0, TiUP, as the package manager, makes it far easier to manage different cluster components in the TiDB ecosystem. Now you can run any component with only a single line of TiUP commands.

13.4.2.1 Install TiUP

You can install TiUP with a single command in both Darwin and Linux operating systems:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

This command installs TiUP in the `$HOME/.tiup` folder. The installed components and the data generated by their operation are also placed in this folder. This command also automatically adds `$HOME/.tiup/bin` to the `PATH` environment variable in the Shell `.profile` file, so you can use TiUP directly.

After installation, you can check the version of TiUP:

```
tiup --version
```

> **Note:**
>
> For TiUP versions starting from v1.11.3, the telemetry is disabled by default in newly deployed TiUP, and usage information is not collected and shared with PingCAP. For details, see Telemetry.

13.4.2.2 TiUP ecosystem introduction

TiUP is not only a package manager in the TiDB ecosystem. Its ultimate mission is to enable everyone to use TiDB ecosystem tools **easier than ever before** by building its own ecosystem. This requires introducing additional packages to enrich the TiUP ecosystem.

This series of TiUP documents introduce what these packages do and how you can use them.

In the TiUP ecosystem, you can get help information by adding `--help` to any command, such as the following command to get help information for TiUP itself:

```
tiup --help
```

```
TiUP is a command-line component management tool that can help to download
   ↪ and install
TiDB platform components to the local system. You can run a specific version
   ↪  of a component via
"tiup <component>[:version]". If no version number is specified, the latest
   ↪ version installed
locally will be used. If the specified component does not have any version
   ↪ installed locally,
the latest stable version will be downloaded from the repository.

Usage:
  tiup [flags] <command> [args...]
  tiup [flags] <component> [args...]

Available Commands:
  install    Install a specific version of a component
  list       List the available TiDB components or versions
  uninstall  Uninstall components or versions of a component
  update     Update tiup components to the latest version
  status     List the status of instantiated components
  clean      Clean the data of instantiated components
  mirror     Manage a repository mirror for TiUP components
  help       Help about any command or component

Components Manifest:
  use "tiup list" to fetch the latest components manifest

Flags:
      --binary <component>[:version] Print binary path of a specific version
         ↪  of a component <component>[:version]
                                     and the latest version installed will be
                                        ↪ selected if no version specified
      --binpath string               Specify the binary path of component
         ↪ instance
  -h, --help                         help for tiup
  -T, --tag string                   Specify a tag for component instance
  -v, --version                      version for tiup

Component instances with the same "tag" will share a data directory (
   ↪ $TIUP_HOME/data/$tag):
  $ tiup --tag mycluster playground
```

```
Examples:
  $ tiup playground                # Quick start
  $ tiup playground nightly        # Start a playground with the latest
    ↪ nightly version
  $ tiup install <component>[:version] # Install a component of specific
    ↪ version
  $ tiup update --all              # Update all installed components to the
    ↪ latest version
  $ tiup update --nightly          # Update all installed components to the
    ↪ nightly version
  $ tiup update --self             # Update the "tiup" to the latest version
  $ tiup list                      # Fetch the latest supported components
    ↪ list
  $ tiup status                    # Display all running/terminated
    ↪ instances
  $ tiup clean <name>              # Clean the data of running/terminated
    ↪ instance (Kill process if it's running)
  $ tiup clean --all               # Clean the data of all running/
    ↪ terminated instances

Use "tiup [command] --help" for more information about a command.
```

The output is long but you can focus on only two parts:

- Available commands
  - install: used to install components
  - list: used to view the list of available components
  - uninstall: used to uninstall components
  - update: used to update the component version
  - status: used to view the running history of components
  - clean: used to clear the running log of components
  - mirror: used to clone a private mirror from the official mirror
  - help: used to print out help information
- Available components
  - playground: used to start a TiDB cluster locally
  - client: used to connect to a TiDB cluster in a local machine
  - cluster: used to deploy a TiDB cluster for production environments
  - bench: used to stress test the database

Note:

- The number of available components will continue to grow. To check the latest supported components, execute the `tiup list` command.
- The list of available versions of components will also continue to grow. To check the latest supported component versions, execute the `tiup list <component>` command.

TiUP commands are implemented in TiUP's internal code and used for package management operations, while TiUP components are independent component packages installed by TiUP commands.

For example, if you run the `tiup list` command, TiUP directly runs its own internal code; if you run the `tiup playground` command, TiUP first checks whether there is a local package named "playground", and if not, TiUP downloads the package from the mirror, and then run it.

13.4.3 TiUP Terminology and Concepts

This document explains important terms and concepts of TiUP.

13.4.3.1 TiUP components

The TiUP program contains only a few commands for downloading, updating, and uninstalling components. TiUP expands its functions with various components. A **component** is a program or script that can be run. When running a component through `tiup <component>`, TiUP adds a set of environment variables, creates the data directory for the program, and then runs the program.

By running the `tiup <component>` command, you can run a component supported by TiUP. The running logic is:

- If you specify a version of a component through `tiup <component>[:version]`:
  - If the component does not have any version installed locally, TiUP downloads the latest stable version from the mirror server.
  - If the component has one or more versions installed locally, but there is no version specified by you, TiUP downloads the specified version from the mirror server.
  - If the specified version of the component is installed locally, TiUP sets the environment variable to run the installed version.
- If you run a component through `tiup <component>` and specify no version:
  - If the component does not have any version installed locally, TiUP downloads the latest stable version from the mirror server.
  - If one or more versions have been installed locally, TiUP sets the environment variable to run the latest installed version.

13.4.3.2 TiUP mirrors

All components of TiUP are downloaded from the TiUP mirrors. TiUP mirrors contain the TAR package of each component and the corresponding meta information (version, entry startup file, checksum). TiUP uses PingCAP's official mirrors by default. You can customize the mirror source through the `TIUP_MIRRORS` environment variable.

TiUP mirrors can be a local file directory or an online HTTP server:

- `TIUP_MIRRORS=/path/to/local tiup list`
- `TIUP_MIRRORS=https://private-mirrors.example.com tiup list`

13.4.4 Manage TiUP Components with TiUP Commands

You can use the following TiUP commands to manage components in the TiUP ecosystem:

- list: Queries the component list. By using this TiUP command, you can see all the optional components to install and all the optional versions of each component.
- install: Installs the specific version of a component.
- update: Updates a component to the latest version.
- uninstall: Uninstalls a component.
- status: Checks the status of a running component.
- clean: Cleans up the instance on which a component is deployed.
- help: Prints the help information. If you append another TiUP command to this command, the usage of the appended command is printed.

This document introduces the common component management operations and the corresponding TiUP commands.

13.4.4.1 Query the component list

You can use the `tiup list` command to query the component list. This usage of this command is as follows:

- `tiup list`: checks which components can be installed.
- `tiup list ${component}`: checks which versions of a specific component can be installed.

You can also use the following flags in the above commands:

- `--installed`: checks which components or which version of a specific component has been installed locally. `--all`: views all components, including the hidden ones `--verbose`: views all columns (including installed versions and supported platforms)

Example 1: View all currently installed components.

```
tiup list --installed
```

Example 2: Get a list of the TiKV component of all installable versions from the server.

```
tiup list tikv
```

13.4.4.2 Install components

You can use the `tiup install` command to query the component list. This usage of this command is as follows:

- `tiup install <component>`: installs the latest stable version of a specified component.
- `tiup install <component>:[version]`: installs the specified version of a specified component.

Example 1: Use TiUP to install the latest stable version of TiDB.

```
tiup install tidb
```

Example 2: Use TiUP to install the nightly version of TiDB.

```
tiup install tidb:nightly
```

Example 3: Use TiUP to install TiKV v6.1.7.

```
tiup install tikv:v6.1.7
```

13.4.4.3 Upgrade components

After a new version of a component is published, you can use the `tiup update` command to upgrade this component. The usage of this command is basically the same as that of `tiup install`, except for the following flags:

- `--all`: Upgrades all components.
- `--nightly`: Upgrades to the nightly version.
- `--self`: Upgrades TiUP itself to the latest version.
- `--force`: Forcibly upgrades to the latest version.

Example 1: Upgrade all components to the latest versions.

```
tiup update --all
```

Example 2: Upgrade all components to the nightly version.

```
tiup update --all --nightly
```

Example 3: Upgrade TiUP to the latest version.

```
tiup update --self
```

13.4.4.4 Operate components

After the installation is complete, you can use the `tiup <component>` command to start the corresponding component:

```
tiup [flags] <component>[:version] [args...]

Flags:
  -T, --tag string                Specifies the tag for the component
      ↪ instance.
```

To use this command, you need to specify the component name and the optional version. If no version is specified, the latest stable version (installed) of this component is used.

Before the component is started, TiUP creates a directory for it, and then puts this component into the directory for operation. The component generates all the data in this directory, and the name of this directory is the tag name specified when the component operates. If no tag is specified, a tag name is randomly generated. This working directory will be *automatically deleted* when the instance is terminated.

If you want to start the same component multiple times and reuse the previous working directory, you can use `--tag` to specify the same name when the component is started. After the tag is specified, the working directory will *not be automatically deleted* when the instance is terminated, which makes it convenient to reuse the working directory.

Example 1: Operate TiDB v6.1.7.

```
tiup tidb:v6.1.7
```

Example 2: Specify the tag with which TiKV operates.

```
tiup --tag=experiment tikv
```

13.4.4.4.1 Query the operating status of a component

You can use the `tiup status` command to check the operating status of a component:

```
tiup status
```

By executing this command, you will get a list of instances, one instance per line. The list contains the following columns:

- `Name`: The tag name of the instance.
- `Component`: The component name of the instance.
- `PID`: The process ID of the operating instance.
- `Status`: The instance status. `RUNNING` means that the instance is operating. `TERM` means that the instance is terminated.
- `Created Time`: The starting time of the instance.
- `Directory`: The working directory of the instance, which can be specified using `--tag`.
- `Binary`: The executable program of the instance, which can be specified using `--binpath`.
- `Args`: The arguments of the operating instance.

13.4.4.4.2 Clean component instance

You can use the `tiup clean` command to clean up component instances and delete the working directory. If the instance is still operating before the cleaning, the related process is killed first. The command usage is as follows:

```
tiup clean [tag] [flags]
```

The following flag is supported:

- `--all`: Cleans up all instance information.

In the above command, `tag` is the instance tag to be cleaned. If `--all` is used, no tag is passed.

Example 1: Clean up the component instance with the `experiment` tag name.

```
tiup clean experiment
```

Example 2: Clean up all component instances.

```
tiup clean --all
```

13.4.4.4.3 Uninstall components

The components installed using TiUP take up local disk space. If you do not want to keep too many components of old versions, you can check which versions of a component are currently installed, and then uninstall this component.

You can use the `tiup uninstall` command to uninstall all versions or specific versions of a component. This command also supports uninstalling all components. The command usage is as follows:

```
tiup uninstall [component][:version] [flags]
```

The following flags are supported in this command:

- `--all`: Uninstalls all components or versions.
- `--self`: Uninstalls TiUP itself.

`component` is the component to be uninstalled. `version` is the version to be uninstalled. Both `component` and `version` can be ignored in the `tiup uninstall` command. If you ignore either one of these two, you need to add the `--all` flag.

- If the version is ignored, adding `--all` means to uninstall all versions of this component.
- If the version and the component are both ignored, adding `--all` means to uninstall all components of all versions.

Example 1: Uninstall TiDB v6.1.7.

```
tiup uninstall tidb:v6.1.7
```

Example 2: Uninstall TiKV of all versions.

```
tiup uninstall tikv --all
```

Example 3: Uninstall all installed components.

```
tiup uninstall --all
```

13.4.5 TiUP FAQs

This document collects the frequently asked questions (FAQs) about TiUP.

13.4.5.1 Can TiUP not use the official mirror source?

TiUP supports specifying the mirror source through the `TIUP_MIRRORS` environment variable. The address of the mirror source can be a local directory or an HTTP server address. If your environment cannot access the network, you can create your own offline mirror source to use TiUP.

After using an unofficial mirror, if you want the official mirror back and use it, take one of the following measures:

- Set the `TIUP_MIRRORS` variable to the official mirror address: `https://tiup-mirrors.pingcap.com`.
- Make sure that the `TIUP_MIRRORS` variable is not set, and then execute the `tiup mirror set https://tiup-mirrors.pingcap.com` command.

13.4.5.2 How do I put my own component into the TiUP mirrors?

TiUP does not support third-party components for the time being, but the TiUP Team has developed the TiUP component development specifications and is developing the tiup-publish component. After everything is ready, a contributor can publish their own components to TiUP's official mirrors by using the `tiup publish <comp> <version>` command.

13.4.5.3 What is the difference between the TiUP playground and TiUP cluster components?

The TiUP playground component is mainly used to build a stand-alone development environment on Linux or macOS operating systems. It helps you get started quickly and run a specified version of the TiUP cluster easily. The TiUP cluster component is mainly used to deploy and maintain a production environment cluster, which is usually a large-scale cluster.

13.4.5.4 How do I write the topology file for the TiUP cluster component?

Refer to these templates to write the topology file. The templates include:

- Multi-DC deployment topology
- Minimal deployment topology
- Complete topology file

You can edit your topology file based on the templates and your needs.

13.4.5.5 Can multiple instances be deployed on the same host?

You can use the TiUP cluster component to deploy multiple instances on the same host, but with different ports and directories configured; otherwise, directory and port conflicts might occur.

13.4.5.6 Are port and directory conflicts detected within the same cluster?

Port and directory conflicts in the same cluster are detected during deployment and scaling. If there is any directory or port conflict, the deployment or scaling process is interrupted.

13.4.5.7 Are port and directory conflicts detected among different clusters?

If multiple different clusters are deployed by the same TiUP control machine, the port and directory conflicts among these clusters are detected during deployment and scaling. If the clusters are deployed by different TiUP control machines, conflict detection is not supported currently.

13.4.5.8 During cluster deployment, TiUP received an `ssh: handshake failed: read tcp 10.10.10.34:38980 -> 10.10.10.34:3600: read: connection reset by peer` error

The error might occur because the default number of concurrent threads of TiUP exceeds the default maximum number of SSH connections. To solve the issue, you can increase the default number of SSH connections, and then restart the sshd service:

```
vi /etc/ssh/sshd_config
```

```
MaxSessions 1000
MaxStartups 1000
```

13.4.6 TiUP Troubleshooting Guide

This document introduces some common issues when you use TiUP and the troubleshooting methods. If this document does not include the issues you bump into, file a new issue in the Github TiUP repository.

13.4.6.1 Troubleshoot TiUP commands

13.4.6.1.1 Can't see the latest component list using `tiup list`

TiUP does not update the latest component list from the mirror server every time. You can forcibly refresh the component list by running `tiup list`.

13.4.6.1.2 Can't see the latest version information of a component using `tiup list <component>`

Same as the previous issue, the component version information is only obtained from the mirror server when there is no local cache. You can refresh the component list by running `tiup list <component>`.

13.4.6.1.3 Component downloading process is interrupted

Unstable network might result in an interrupted component downloading process. You can try to download the component again. If you cannot download it after trying multiple times, it might be caused by the CDN server and you can report the issue here.

13.4.6.1.4 A checksum error occurs during component downloading process

Because the CDN server has a short cache time, the new checksum file might not match the component package. Try to download again after 5 minutes. If the new checksum file still does not match the component package, report the issue here.

13.4.6.2 Troubleshoot TiUP cluster component

13.4.6.2.1 `unable to authenticate, attempted methods [none publickey]` is prompted during deployment

During deployment, component packages are uploaded to the remote host and the initialization is performed. This process requires connecting to the remote host. This error is caused by the failure to find the SSH private key to connect to the remote host.

To solve this issue, confirm whether you have specified the private key by running `tiup cluster deploy -i identity_file`:

- If the `-i` flag is not specified, it might be that TiUP does not automatically find the private key path. It is recommended to explicitly specify the private key path using `-i`.
- If the `-i` flag is specified, it might be that TiUP cannot log in to the remote host using the specified private key. You can verify it by manually executing the `ssh -i identity_file user@remote` command.
- If a password is used to log in to the remote host, make sure that you have specified the `-p` flag and entered the correct login password.

13.4.6.2.2 The process of upgrading the cluster using the TiUP cluster component is interrupted

To avoid misuse cases, the TiUP cluster component does not support the upgrade of specified nodes, so after the upgrade fails, you need to perform the upgrade operations again, including idempotent operations during the upgrade process.

The upgrade process can be divided into the following steps:

1. Back up the old version of components on all nodes
2. Distribute new components to remote
3. Perform a rolling restart to all components

If the upgrade is interrupted during a rolling restart, instead of repeating the `tiup cluster upgrade` operation, you can use `tiup cluster restart -N <node1> -N <node2>` to restart the nodes that have not completed the restart.

If the number of un-restarted nodes of the same component is relatively large, you can also restart a certain type of component by running `tiup cluster restart -R <component>`.

13.4.6.2.3 During the upgrade, you find that `node_exporter-9100.service/blackbox_expor` does not exist

If you previously migrated your cluster from TiDB Ansible and the exporter was not deployed in TiDB Ansible, this situation might happen. To solve it, you can manually copy the missing files from other nodes to the new node for the time being. The TiUP team will complete the missing components during the migration process.

13.4.7 Command Reference

13.4.7.1 TiUP Reference

TiUP serves as the package manager of the TiDB ecosystem. It manages components in the TiDB ecosystem, such as TiDB, PD, and TiKV.

13.4.7.1.1 Syntax

```
tiup [flags] <command> [args...]   # Executes a command
#### or
tiup [flags] <component> [args...] # Runs a component
```

You can use the `--help` command to get the information of a specific command. The summary of each command shows its parameters and their usage. Mandatory parameters are shown in angle brackets, and optional parameters are shown in square brackets.

`<command>` represents the command name. For the list of supported commands, see the Command list below. `<component>` represents the component name. For the list of supported components, see the Component list below.

13.4.7.1.2 Options

–binary

- If you enable this option, the specified binary file path is printed.
    - Executing `tiup --binary <component>` will have the path of the latest stable installed `<component>` component printed. If `<component>` is not installed, an error is returned.
    - Executing `tiup --binary <component>:<version>` will have the path of the installed `<component>` component's `<version>` printed. If this `<version>` is not printed, an error is returned.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

> **Note:**
>
> This option can only be used in commands of the `tiup [flags] <component > [args...]` format.

–binpath

> **Note:**
>
> This option can only be used in commands of the `tiup [flags] <component > [args...]` format.

- Specifies the path of the component to be executed. When a component is executed, if you do not want to use the binary file in the TiUP mirror, you can add this option to specify using the binary file in a custom path.
- Data type: `STRING`

-T, –tag

- Specifies a tag for the component to be started. Some components need to use disk storage during the execution, and TiUP allocates a temporary storage directory for this execution. If you want TiUP to allocate a fixed directory, you can use `-T/--tag` to specify the name of the directory, so that the same batch of files can be read and written in multiple executions with the same tag.
- Data type: `STRING`

-v, –version

Prints the TiUP version.

–help

Prints the help information.

13.4.7.1.3 Command list

TiUP has multiple commands, and these commands have multiple sub-commands. For the specific commands and their detailed descriptions, click the corresponding links in the list below:

- install: Installs a component.
- list: Shows the component list.
- uninstall: Uninstalls a component.
- update: Updates the installed component.
- status: Shows the running status of a component.
- clean: Cleans the data directory of a component.
- mirror: Manages the mirror.
- telemetry: Enables or disables the telemetry.
- completion: Completes the TiUP command.
- env: Shows the TiUP-related environment variables.
- help: Shows the help information of a command or component.

13.4.7.1.4 Component list

- cluster: Manages the TiDB cluster in a production environment.
- dm: Manages the TiDB Data Migration (DM) cluster in a production environment.

13.4.7.2 TiUP Commands

13.4.7.2.1 tiup clean

The `tiup clean` command is used to clear the data generated during component operation.

Syntax

```
tiup clean [name] [flags]
```

The value of `[name]` is the `Name` field output by the `status` command. If `[name]` is omitted, you must add the `--all` option in the `tiup clean` command.

Option

–all

- Clears all operation records
- Data type: Boolean
- Default: false

Output

```
Clean instance of `%s`, directory: %s
```

<< Back to the previous page - TiUP Reference command list

13.4.7.2.2 tiup completion

To reduce user costs, TiUP provides the `tiup completion` command to generate a configuration file for automatic command-line completion. Currently, TiUP supports completing `bash` and `zsh` commands.

If you want to complete `bash` commands, you need to install `bash-completion` first. See the following instructions:

- On macOS: If your bash version is earlier than 4.1, run `brew install bash-completion`; otherwise, run `brew install bash-completion@2`.
- On Linux: Use a package manager to install `bash-completion`. For example, run `yum install bash-completion` or `apt install bash-completion`.

Syntax

```
tiup completion <shell>
```

`<shell>` is used to set the type of shell you use. Currently, `bash` and `zsh` are supported.

Usage

bash

Write the `tiup completion bash` command into a file and source the file in `.bash_profile`. See the following example:

```
tiup completion bash > ~/.tiup.completion.bash

printf "
##### tiup shell completion
source '$HOME/.tiup.completion.bash'
" >> $HOME/.bash_profile

source $HOME/.bash_profile
```

zsh

```
tiup completion zsh > "${fpath[1]}/_tiup"
```

<< Back to the previous page - TiUP Reference command list

13.4.7.2.3 tiup env

TiUP provides users with flexible and customized interfaces, some of which are implemented using environment variables. The `tiup env` command is used to query the user-defined environment variables that TiUP supports and their values.

Syntax

```
tiup env [name1...N]
```

`[name1...N]` is used to view the specified environment variables. If it is not specified, all supported environment variables are viewed by default.

Option

None

Output

- If `[name1...N]` is not specified, a list of "{key}"="{value}" is output.
- If `[name1...N]` is specified, the "{value}" list is output in order.

In the above output, if `value` is empty, it means that the value of the environment variable is not set. In this case, TiUP uses the default value.

<< Back to the previous page - TiUP Reference command list

13.4.7.2.4 tiup help

The TiUP command-line interface provides users with a wealth of help information. You can view it via the `help` command or the `--help` option.

Syntax

```
tiup help [command]
```

`[command]` is used to specify the help information of which command that users need to view. If it is not specified, the help information of TiUP is viewed.

Option

None

Output

The help information of `[command]` or TiUP.

<< Back to the previous page - TiUP Reference command list

13.4.7.2.5 tiup install

The `tiup install` command is used for component installation. It downloads the component package of a specified version from the mirror repository and decompresses it in the local TiUP data directory for later use. In addition, when TiUP needs to run a component that does not exist in the mirror repository, it tries to download the component first and then runs it automatically. If the component does not exist in the repository, an error is reported.

Syntax

```
tiup install <component1>[:version] [component2...N] [flags]
```

`<component1>` and `<component2>` represent component names, and `[version]` represents an optional version number. If `version` is not added, the latest stable version of the specified component is installed. `[component2...N]` means that you can specify multiple components or multiple versions of the same component at the same time.

Option

None

Output

- Normally outputs the download information of the component.
- If the component does not exist, the `The component "%s" not found` error is reported.
- If the version does not exist, the `version %s not supported by component %s` error is reported.

<< Back to the previous page - TiUP Reference command list

13.4.7.2.6 tiup list

The command `tiup list` is used to get the list of available components of a mirror.

Syntax

```
tiup list [component] [flags]
```

`[component]` is an optional parameter used to specify a certain component. If `[component]` is set, TiUP lists all versions of the specified component; if not, TiUP lists all components.

Options

–all

- Displays all components. By default, TiUP does not show hidden components.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–installed

- Only displays components and versions that have been installed.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–verbose

- Displays installed component versions in the components list.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

- If `[component]` is not set:
  - If `--verbose` is specified: TiUP outputs a component information list consisting of `Name` (component name), `Installed` (installed versions), `Owner` (component owner), and `Description` (component description).

- If `--verbose` is not specified: TiUP outputs a component information list consisting of `Name` (component name), `Owner` (component owner), and `Description` (component description).

- If `[component]` is set:
  - If the specified component exists: TiUP outputs a version information list of the specified component, consisting of `Version` (version number), `Installed` (installation status), `Release` (release date), and `Platforms` (supported platforms).
  - If the specified component does not exist: TiUP reports the error `failed to fetch component: unknown component`.

<< Back to the previous page - TiUP Reference command list

13.4.7.2.7 tiup mirror

tiup mirror

In TiUP, mirror is an important concept. TiUP currently supports two forms of mirroring:

- Local mirror: the TiUP client and the mirror are on the same machine, and the client accesses the mirror through the file system.
- Remote mirror: the TiUP client and the mirror are not on the same machine, and the client accesses the mirror through network.

The `tiup mirror` command is used to manage mirrors and provides ways to create mirrors, distribute components, and manage keys.

Syntax

```
tiup mirror <command> [flags]
```

`<command>` stands for sub-commands. For the list of supported sub-commands, refer to the command list below.

Option

None

Command list

- genkey: generates the private key file
- sign: signs a specific file using a private key file
- init: initiates an empty mirror
- set: sets the current mirror
- grant: grants a new component owner for the current mirror
- publish: publishes new components to the current mirror

- modify: modifies the attributes of the components in the current mirror
- rotate: updates the root certificate in the current mirror
- clone: clones a new mirror from an existing one
- merge: merges mirrors

<< Back to the previous page - TiUP Reference command list

tiup mirror clone

The command `tiup mirror clone` is used to clone an existing mirror or clone some of its components to create a new mirror. The new mirror has the same components as the old one, but uses a different signature key.

Syntax

```
tiup mirror clone <target-dir> [global version] [flags]
```

- `<target-dir>` is used to set the local path to the cloned mirror. If the path does not exist, TiUP automatically creates one.
- If `[global version]` is specified, TiUP tries to clone all components of the specified version. If some components do not have the specified version, then TiUP clones its latest version.

Options

-f, –full

- Whether to clone the whole mirror. If this option is set, other options becomes ignored and TiUP completely clones all components of all versions from the targeted mirror.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

-a, –arch

- Only clones components that can run on the specified platform.
- Data type: `STRING`
- Default: “amd64,arm64”

-o, –os

- Only clones components that can run on the specified operating system.
- Data type: `STRING`
- Default: “linux,darwin”

–prefix

- Whether to only match the prefix of versions. By default, TiUP downloads a component version when it is strictly matched. If this option is set, TiUP also downloads component versions of which prefixes are matched.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–{component}

- Specifies the version list of the component to be cloned. Fill component names in `{component}`. You can run `tiup list --all` to view available component names.
- Data type: Strings
- Default: Null

<< Back to the previous page - TiUP Mirror command list

tiup mirror genkey

TiUP mirror, according its definition, has three roles of users:

- Mirror administrators: They have the permission to modify `root.json`, `index.json`, `snapshot.json`, and `timestamp.json`.
- Component owners: They have the permission to modify the corresponding component.
- Normal users: They can download and use the components.

Because TiUP requires the signature of the corresponding owner/administrator to modify a file, owners/administrators must have his or her own private key. The command `tiup mirror genkey` is used to generate a private key.

> **Warning:**
>
> **DO NOT** transmit private keys over the Internet.

Syntax

```
tiup mirror genkey [flags]
```

Options

-n, –name

- Specifies the name of the key, which also determines the name of the final generated file. The path of the generated private key file is `${TIUP_HOME}/keys/{name}.json`. `TIUP_HOME` refers to the home directory of TiUP, which is `$HOME/.tiup` by default. `name` refers to the private key name that `-n/--name` specifies.
- Data type: `STRING`
- Default: "private"

-p, –public

- Shows the corresponding public key of the private key specified in the option `-n/--name`.
- TiUP does not create a new private key when `-p/--public` is specified. If the private key specified in `-n/--name` does not exist, TiUP returns an error.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–save

- Saves the information of the public key as a file in the current directory. The file name is `{hash-prefix}-public.json`. `hash-prefix` is the first 16 bits of the key ID.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

- If `-p/--public` is not specified:
  - If the private key specified in `-n/--name` exists: TiUP outputs `Key already exists, skipped`.
  - If the private key specified in `-n/--name` does not exist: TiUP outputs `private key have been write to ${TIUP_HOME}/keys/{name}.json`.
- If `-p/--public` is specified:
  - If the private key specified in `-n/--name` does not exist: TiUP reports the error `Error: open ${TIUP_HOME}/keys/{name}.json: no such file or directory`.
  - If the private key specified in `-n/--name` exists: TiUP outputs the content of the corresponding public key.

<< Back to the previous page - TiUP Mirror command list

tiup mirror grant

The `tiup mirror grant` command is used to introduce a component owner to the current mirror.

Component owners can use their keys to publish new components or to modify components they previously published. Before adding a new component owner, the component owner to be added needs to send his or her own public key to the mirror administrator.

> **Note:**
>
> This command is only supported when the current mirror is a local mirror.

Syntax

```
tiup mirror grant <id> [flags]
```

`<id>` stands for the component owner's ID, which must be unique in the whole mirror. It is recommended to use an ID that matches the regular expression `^[a-z\d](?:[a-z\d]|-(?=[a-z\d])){0,38}$`.

Options

-k, –key

- Specifies the key of the introduced component owner. This key can either be public or private. If it is a private key, TiUP converts it to the corresponding public key before storing it in the mirror.
- A key can be used by only one component owner.
- Data type: `STRING`
- Default: "${TIUP_HOME}/keys/private.json"

-n, –name

- Specifies the name of the component owner. The name is displayed on the `Owner` field of the component list. If `-n/--name` is not specified, `<id>` is used as the component owner's name.
- Data type: `STRING`
- Default: `<id>`

Outputs

- If the command is executed successfully, there is no output.

- If the component owner's ID is duplicated, TiUP reports the error `Error: owner %s exists`.
- If the key is used by another component owner, TiUP reports the error `Error: key %s exists`.

<< Back to the previous page - TiUP Mirror command list

tiup mirror init

The command `tiup mirror init` is used to initialize an empty mirror. The initialized mirror does not contain any components or component owners. The command only generates the following files for the initialized mirror:

```
+ <mirror-dir>                                # Mirror's root directory
|-- root.json                                 # Mirror's root certificate
|-- 1.index.json                              # Component/user index
|-- snapshot.json                             # Mirror's latest snapshot
|-- timestamp.json                            # Mirror's latest timestamp
|--+ keys                                     # Mirror's private key (can be
   ↪ moved to other locations)
   |-- {hash1..hashN}-root.json               # Private key of the root
      ↪ certificate
   |-- {hash}-index.json                      # Private key of the indexes
   |-- {hash}-snapshot.json                   # Private key of the snapshots
   |-- {hash}-timestamp.json                  # Private key of the timestamps
```

For the specific usage and content format of the above files, refer to TiUP Mirror Reference Guide.

Syntax

```
tiup mirror init <path> [flags]
```

`<path>` is used to specify a local directory where TiUP generates and stores mirror files. The local directory can be a relative path. If the specified directory already exists, it must be empty; if it does not exist, TiUP creates it automatically.

Options

-k, –key-dir

- Specifies the directory where TiUP generates private key files. If the specified directory does not exist, TiUP automatically creates it.
- Data type: `STRING`
- If this option is not specified in the command, TiUP generates private key files in `{path}/keys` by default.

Outputs

- If the command is executed successfully, there is no output.
- If the specified `<path>` is not empty, TiUP reports the error `Error: the target path '%s' is not an empty directory`.
- If the specified `<path>` is not a directory, TiUP reports the error `Error: fdopendir: not a directory`.

<< Back to the previous page - TiUP Mirror command list

tiup mirror merge

The `tiup mirror merge` command is used to merge one or more mirrors to the current mirror.

To execute this command, the following conditions must be met:

- The owner IDs of all components of the target mirror exist in the current mirror.
- The `${TIUP_HOME}/keys` directory of the user who executes this command contains all the private keys corresponding to the above owner IDs in the current mirror (you can use the command `tiup mirror set` to switch the current mirror to the mirror that is currently authorized to modify).

Syntax

```
tiup mirror merge <mirror-dir-1> [mirror-dir-N] [flags]
```

- `<mirror-dir-1>`: the first mirror to be merged into the current mirror
- `[mirror-dir-N]`: the Nth mirror to be merged into the current mirror

Option

None

Outputs

- If the command is executed successfully, there is no output.
- If the current mirror does not have a component owner of the target mirror, or if `${TIUP_HOME}/keys` does not have the owner's private key, TiUP reports the `Error: missing owner keys for owner %s on component %s` error.

<< Back to the previous page - TiUP Mirror command list

tiup mirror modify

The `tiup mirror modify` command is used to modify published components. Only valid component owners can modify the components that they have published on their own. For the method to publish a component, refer to the `publish` command.

Syntax

```
tiup mirror modify <component>[:version] [flags]
```

Each parameter is explained as follows:

- `<component>`: the component name
- `[version]`: the component version to modify. If it is not specified, the entire component is modified.

Options

-k, –key

- Specifies the component owner's private key used for signing the component information (`{component}.json`).
- Data type: `STRING`
- If this option is not specified in the command, `"${TIUP_HOME}/keys/private.json"` is used by default to sign the component information.

–yank

Marks a specified component or version as unavailable.

- After the component is marked as unavailable, you can neither see it in the result list of `tiup list` nor install the new version of the component.
- After a component version is marked as unavailable, you can neither see it in the result list of `tiup list <component>` nor install this version.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–hide

- Specifies whether to hide the component. If a component is hidden, you cannot see it in the result list of `tiup list`. To see the hidden component, you can use `tiup list --all`.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

> **Note:**
>
> This option can only be applied to the component, not to the component version.

–standalone

- Controls whether the component can run standalone. This option is currently **NOT available**.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

> **Note:**
>
> This option can only be applied to the component, not to the component version.

Outputs

- If the command is executed successfully, there is no output.
- If the component owner is not authorized to modify the target component:
    - If the mirror is a remote mirror, TiUP reports the error `Error: The server refused, make sure you have access to this component`.
    - If the mirror is a local mirror, TiUP reports the error `Error: the signature is not correct`.

<< Back to the previous page - TiUP Mirror command list

tiup mirror publish

The command `tiup mirror publish` is used to publish a new component or a new version of an existing component. Only component owner that has the access to the target component can publish it. To add a new component owner, see the usage of the grant command.

Syntax

```
tiup mirror publish <comp-name> <version> <tarball> <entry> [flags]
```

The meaning of each parameter is as follows:

- `<comp-name>`: The name of the components, such as `tidb`. It is recommended to use a string that matches the regular expression `^[a-z\d](?:[a-z\d]|-(?=[a-z\d])){0,38}$`.
- `<version>`: The version of the component to be published. The version number needs to follow the requirements of Semantic Versioning.
- `<tarball>`: The local directory of the `.tar.gz` package. You need to put dependencies and the executable file of the component in this package. TiUP uploads this package to the mirror.
- `<entry>`: The location of the component's executable file in `<tarball>`.

Options

-k, –key

- Specifies the component owner's private key. The client uses the private key to sign `{component}.json` files.
- Data type: `STRING`
- Default: "${TIUP_HOME}/keys/private.json"

–arch

- Specifies the platform on which the binary files in `<tarball>` can run. For a single `<tarball>` package, you can only choose the platform from the following options:
  - `amd64`: Indicates that the files run on AMD64 machines.
  - `arm64`: Indicates that the files run on ARM64 machines.
  - `any`: Indicates that the files, such as scripts, run on both AMD64 and ARM64 machines.
- Data type: `STRING`
- Default: "${GOARCH}"

> **Note:**
>
> If `--arch` is set to `any`, then `--os` must be set to `any` as well.

–os

- Specifies the operating system on which the binary files in `<tarball>` can run. For a single `<tarball>` package, you can only choose the operating system from the following options:
    - `linux`: Indicates that the files run on the Linux operating system.
    - `darwin`: Indicates that the files run on the Darwin operating system.
    - `any`: Indicates that the files, such as scripts, run on both the Linux and Darwin operating systems.
- Data type: `STRING`
- Default: "${GOOS}"

> **Note:**
>
> If `--os` is set to `any`, then `--arch` must be set to `any` as well.

–desc

- Specifies the description of the component.
- Data type: `String`
- Default: NULL

–hide

- Specifies whether the component is hidden. If it is a hidden component, it can be seen in the result list of `tiup list -all`, but not in that of `tiup list`.
- Data type: `STRING`
- Default: NULL

–standalone

- Controls whether the component can run standalone. This option is currently **NOT available**.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

- If the command is executed successfully, there is no output.
- If the component owner is not authorized to modify the target component:
    - If the mirror is a remote mirror, TiUP reports the error `Error: The server refused, make sure you have access to this component`.
    - If the mirror is a local mirror, TiUP reports the error `Error: the signature is not correct`.

<< Back to the previous page - TiUP Mirror command list

tiup mirror rotate

`root.json` is an important file in a TiUP mirror. It stores the public keys needed for the entire system and is the basis of the chain of trust in TiUP. It mainly contains the following parts:

- Signatures of mirror administrators. For the official mirror, there are five signatures. For an initialized mirror, there are three signatures by default.
- The public keys used to verify the following files:
    - root.json
    - index.json
    - snapshot.json
    - timestamp.json
- Expiration date of `root.json`. For the official mirror, the expiration date is one year later than the creation date of `root.json`.

For detailed description of TiUP mirror, see TiUP Mirror Reference.

You need to update `root.json` in the following cases:

- Replace the key of the mirror.
- Update the expiration date of certificate files.

After the content of `root.json` is updated, the file must be re-signed by all administrators; otherwise, the client rejects the file. The update process is as follows:

1. The user (client) updates the content of `root.json`.
2. All administrators sign the new `root.json` file.
3. tiup-server updates `snapshot.json` to record the version of the new `root.json` file.
4. tiup-server signs the new `snapshot.json` file.
5. tiup-server updates `timestamp.json` to record the hash value of the new `snapshot.json` file.
6. tiup-server signs the new `timestamp.json` file.

TiUP uses the command `tiup mirror rotate` to automate the above process.

Note:

- For TiUP versions earlier than v1.5.0, running this command does not returns a correct new `root.json` file. See #983.
- Before using this command, make sure that all TiUP clients are upgraded to v1.5.0 or a later version.

Syntax

```
tiup mirror rotate [flags]
```

After executing this command, TiUP starts an editor for the user to modify the file content to the target value, such as changing the value of the `expires` field to a later date. Then, TiUP changes the `version` field from `N` to `N+1` and saves the file. After the file is saved, TiUP starts a temporary HTTP server and waits for all mirror administrators to sign the file.

For how mirror administrators sign files, refer to the `sign` command.

Options

–addr

- Specifies the listening address of the temporary server. You need to make sure that the address is accessible to other mirror administrators so that they can use the `sign` command to sign the file.
- Data type: `STRING`
- If this option is not specified in the command, TiUP listens on `0.0.0.0:8080` by default.

Outputs

The current signature status of each mirror administrator.

<< Back to the previous page - TiUP Mirror command list

tiup mirror set

The `tiup mirror set` command is used to switch the current mirror and supports two forms of mirrors: local file system and remote network address.

The address of the official mirror is `https://tiup-mirrors.pingcap.com`.

Syntax

```
tiup mirror set <mirror-addr> [flags]
```

`<mirror-addr>` is the mirror address, which has two forms:

- Network address: starts with `http` or `https`. For example, `http://172.16.5.5:8080`, `https://tiup-mirrors.pingcap.com`.
- Local file path: the absolute path of the mirror directory. For example, `/path/to/local-tiup-mirror`.

Option

-r, –root

This option specifies the root certificate.

As the most critical part of mirror security, the root certificate of each mirror is different from one another. When you use the network mirror, it might suffer from man-in-the-middle attacks. To avoid such attacks, it is recommended to manually download the root certificate of the root network mirror to the local:

```
wget <mirror-addr>/root.json -O /path/to/local/root.json
```

Perform a manual check to ensure that the root certificate is correct, and then switch the mirror by manually specifying the root certificate:

```
tiup mirror set <mirror-addr> -r /path/to/local/root.json
```

In the steps above, if the mirror is attacked before the `wget` command, you can find that the root certificate is incorrect. If the mirror is attacked after the `wget` command, TiUP will find that the mirror does not match the root certificate.

- Data type: `String`
- Default: `{mirror-dir}/root.json`

Output

None

<< Back to the previous page - TiUP Mirror command list

tiup mirror sign

The `tiup mirror sign` command is used to sign the metadata files (*.json) defined in TiUP mirror. These metadata files might be stored on the local file system or remotely stored using the HTTP protocol to provide a signature entry.

Syntax

```
tiup mirror sign <manifest-file> [flags]
```

`<manifest-file>` is the address of the file to be signed, which has two forms:

- Network address, which starts with HTTP or HTTPS, such as `http://172.16.5.5:8080/rotate/root.json`
- Local file path, which is a relative path or an absolute path

If it is a network address, this address must provide the following features:

- Supports the access via `http get` that returns the complete content of the signed file (including the `signatures` field).
- Supports the access via `http post`. The client adds the signature to the `signatures` field of the content that is returned by `http get` and posts to this network address.

Options

-k, –key

- Specifies the location of the private key used for signing the `{component}.json` file.
- Data type: `STRING`
- - If this option is not specified in the command, `"${TIUP_HOME}/keys/private.json"` is used by default.

–timeout

- Specifies the access timeout time for signing through the network. The unit is in seconds.
- Data type: `INT`
- Default: 10

> **Note:**
>
> This option is valid only when `<manifest-file>` is a network address.

Output

- If the command is executed successfully, there is no output.
- If the file has been signed by the specified key, TiUP reports the error `Error: this manifest file has already been signed by specified key`.
- If the file is not a valid manifest, TiUP reports the error `Error: unmarshal manifest : %s`.

<< Back to the previous page - TiUP Mirror command list

13.4.7.2.8 tiup status

The `tiup status` command is used to view the operation information of the components after you run the components using the `tiup [flags] <component> [args...]` command.

> **Note:**
>
> You can only check the information of the following components:
>
> - Components that are still in operation
> - Components that run through the tag specified by `tiup -T/--tag`

Syntax

```
tiup status [flags]
```

Option

None

Output

A table consisting of the following fields:

- `Name`: The tag name specified by `-T/--tag`. If not specified, it is a random string.
- `Component`: The operating components.
- `PID`: The corresponding process ID of the operating components.
- `Status`: The status of the operating components.
- `Created Time`: The starting time of the components.
- `Directory`: The data directory of the components.
- `Binary`: The binary file path of the components.
- `Args`: The starting arguments of the operating components.

Component status

A component can run in one of the following statuses:

- Up: The component is running normally.
- Down or Unreachable: The component is not running or a network problem exists on the corresponding host.
- Tombstone: The data on the component has been completely migrated out and the scaling-in is complete. This status exists only on TiKV or TiFlash.
- Pending Offline: The data on the component is being migrated out and the scaling-in is in process. This status exists only on TiKV or TiFlash.
- Unknown: The running status of the component is unknown.

> **Note:**
>
> `Pending Offline` in TiUP, `Offline` returned by PD API, and `Leaving` in TiDB Dashboard indicate the same status.

Component status derives from the PD scheduling information. For more details, see Information collection.

<< Back to the previous page - TiUP Reference command list

13.4.7.2.9 tiup telemetry

Starting from v1.11.3, the telemetry is disabled by default in newly deployed TiUP, and usage information is not collected and shared with PingCAP. In versions earlier than v1.11.3, the telemetry is enabled by default in TiUP, and usage information is collected and shared with PingCAP to improve the product.

When TiUP telemetry is enabled, usage information is shared with PingCAP when TiUP commands are executed, including (but not limited to):

- Randomly generated telemetry identifiers.
- The execution status of the TiUP command, such as whether the command execution is successful and the duration of command execution.
- Situations using TiUP for deployment, such as target machine hardware information, component version number, and modified deployment configuration name.

The information below is not shared:

- The accurate name of the cluster
- The cluster topology
- The cluster configuration file

TiUP uses the `tiup telemetry` command to control telemetry.

Syntax

```
tiup telemetry <command>
```

`<command>` stands for sub-commands. For the list of supported sub-commands, refer to the commands section below.

Commands

status

The `tiup telemetry status` command is used to view the current telemetry settings and output the following information:

- `status`: specifies enabling or disabling the telemetry `(enable|disable)`.
- `uuid`: specifies the randomly generated telemetry identifiers.

reset

The `tiup telemetry reset` command is used to reset the current telemetry identifier and replaces it with a new random identifier.

enable

The `tiup telemetry enable` command is used to enable the telemetry.

disable

The `tiup telemetry disable` command is used to disable the telemetry.

<< Back to the previous page - TiUP Reference command list

13.4.7.2.10 tiup uninstall

The `tiup uninstall` command is used to uninstall the installed components.

Syntax

```
tiup uninstall <component1>:<version> [component2...N] [flags]
```

- `<component1>`: the name of the component to uninstall.
- `<version>`: the version to uninstall. If this field is omitted, all installed versions of the component are uninstalled. For security reasons, the `--all` option must be added when `<version>` is omitted, indicating that all versions of a component need to be uninstalled.
- `[component2...N]`: multiple components or versions to uninstall.

Options

–all

- Uninstalls all installed versions of the specified component(s). You must use this option when `<version>` is omitted.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–self

- Uninstalls TiUP itself. When this option is used, all data downloaded from the mirror is deleted, but the data generated by TiUP and its components is retained. The data is stored in the directory specified by the `TIUP_HOME` environment variable. If `TIUP_HOME` is not set, the default value is `~/.tiup/`.

- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

- If the command exits without any error, `Uninstalled component "%s" successfully ↪ !` is output.
- If neither `<version>` nor `--all` is specified, the `Use "tiup uninstall tidbx --all ↪ " if you want to remove all versions.` error is reported.

<< Back to the previous page - TiUP Reference command list

13.4.7.2.11 tiup update

The `tiup update` command is used to update the installed components or TiUP itself.

Syntax

```
tiup update [component1][:version] [component2..N] [flags]
```

- `[component1]`: the name of the component to update.
- `[version]`: the version to update. If this field is omitted, it means updating to the latest stable version of the component.
- `[component2...N]`: specifies updating multiple components or versions. If no component is specified, which means `[component1][:version] [component2..N]` is empty, you need to use the `--all` or the `--self` option together.

The update operation does not delete the old version. You can still specify using the old version during execution.

Options

–all

- If no component is specified, this option must be specified.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–force

- If the specified version of the component is already installed, the update operation is skipped by default. Specifying this option will have the installed version forcibly updated.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–nightly

- Updates the specified components to the nightly version. The `tiup update` command with this option is equivalent to the `tiup update <component>:nightly` command.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–self

- Updates TiUP itself.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

- If the update is successful, `Updated successfully!` is output.
- If target version does not exist, the `Error: version %s not supported by component %s` error is reported.

<< Back to the previous page - TiUP Reference command list

13.4.7.3 TiUP Cluster Commands

13.4.7.3.1 TiUP Cluster

TiUP Cluster is a cluster management component of TiUP written in Golang. You can use the TiUP Cluster component to perform daily operations and maintenance, including deploying, starting, shutting down, destroying, elastic scaling, upgrading TiDB clusters, and managing TiDB cluster parameters.

Syntax

```
tiup cluster [command] [flags]
```

`[command]` is the name of the command. For the supported commands, refer to the command list below.

Options

–ssh

- Specifies the SSH client to connect to the remote end (the machine where the TiDB service is deployed) for the command execution.
- Data type: `STRING`
- Supported values:
  - `builtin`: uses the easyssh client built in tiup-cluster as the SSH client.
  - `system`: uses the default SSH client of the current operating system.
  - `none`: The SSH client is not used. The deployment is only for the current machine.
- If this option is not specified in the command, `builtin` is used as the default value.

–ssh-timeout

- Specifies the SSH connection timeout in seconds.
- Data type: `UINT`
- If this option is not specified in the command, the default timeout is `5` seconds.

–wait-timeout

- Specifies the maximum waiting time (in seconds) for each step in the operation process. The operation process consists of many steps, such as specifying systemctl to start or stop services, and waiting for ports to be online or offline. Each step may take several seconds. If the execution time of a step exceeds the specified timeout, the step exits with an error.
- Data type: `UINT`
- If this option is not specified in the command, the maximum waiting time for each step is `120` seconds.

-y, –yes

- Skips the secondary confirmation of all risky operations. Using this option is not recommended, unless you are using a script to call TiUP.

- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-v, –version

- Prints the current version of TiUP Cluster.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-h, –help

- Prints the help information of the related commands.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Command list

- import: imports a cluster deployed by Ansible
- template: outputs the topology template
- check: checks a cluster before and after the deployment
- deploy: deploys a cluster based on a specified topology
- list: queries the list of deployed clusters
- display: displays the status of a specified cluster
- start: starts a specified cluster
- stop: stops a specified cluster
- restart: restarts a specified cluster
- scale-in: scales in a specified cluster
- scale-out: scales out a specified cluster
- upgrade: upgrades a specified cluster
- prune: cleans up the instances in the Tombstone status for a specified cluster
- edit-config: modifies the configuration of a specified cluster
- reload: reloads the configuration of a specified cluster
- patch: replaces a service in a deployed cluster
- rename: renames a cluster
- clean: deletes data from the specified cluster
- destroy: destroys a specified cluster
- audit: queries the operation audit log of a specified cluster
- replay: retries the specified command
- enable: enables the auto-enabling of the cluster service after a machine is restarted
- disable: disables the auto-enabling of the cluster service after a machine is restarted
- meta backup: backs up the TiUP meta file required for the operation and maintenance of a specified cluster

- meta restore: restores the TiUP meta file of a specified cluster
- help: prints the help information

<< Back to the previous page - TiUP Reference component list

13.4.7.3.2 tiup cluster audit

The `tiup cluster audit` command is used to view commands executed on all clusters in the history and the execution log of each command.

Syntax

```
tiup cluster audit [audit-id] [flags]
```

- If you do not fill in the `[audit-id]`, the table of operation records is output in reverse chronological order. The first column is the `audit-id`.
- If you fill in the `[audit-id]`, it means checking the execution log of the specified `audit-id`.

Option

-h, –help

- Prints the help information.
- Data type: `Boolean`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

- If `[audit-id]` is specified, the corresponding execution log is output.
- If `[audit-id]` is not specified, a table with the following fields is output:
  - ID: the `audit-id` corresponding to the record
  - Time: the execution time of the command corresponding to the record
  - Command: the command corresponding to the record

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.3 tiup cluster audit cleanup

The `tiup cluster audit cleanup` command is used to clean up the logs generated in executing the `tiup cluster`command.

Syntax

```
tiup cluster audit cleanup [flags]
```

Options

–retain-days

- Specifies the days for which logs are retained.
- Data type: `INT`
- Default value: `60`, in the unit of day.
- By default, logs generated within the last 60 days are retained, which means logs generated before 60 days are removed.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default value: `false`
- To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

```
clean audit log successfully
```

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.4 tiup cluster check

For a formal production environment, before the environment goes live, you need to perform a series of checks to ensure the clusters are in their best performance. To simplify the manual check steps, TiUP Cluster provides the `check` command to check whether the hardware and software environments of the target machines of a specified cluster meet the requirements to work normally.

List of check items

Operating system version

Check the operating system distribution and version of the deployed machines. For a list of supported versions, see OS and platform requirements.

CPU EPOLLEXCLUSIVE

Check whether the CPU of the target machine supports EPOLLEXCLUSIVE.

numactl

Check whether `numactl` is installed on the target machine. If tied cores are configured on the target machine, you must install `numactl`.

System time

Check whether the system time of the target machine is synchronized. Compare the system time of the target machine with that of the central control machine, and report an error if the deviation exceeds a certain threshold (500 ms).

System time zone

Check whether the system time zone of the target machines is synchronized. Compare the time zone configuration of these machines and report an error if the time zone is inconsistent.

Time synchronization service

Check whether the time synchronization service is configured on the target machine. Namely, check whether ntpd is running.

Swap partitioning

Check whether swap partitioning is enabled on the target machine. It is recommended to disable swap partitioning.

Kernel parameters

Check the values of the following kernel parameters:

- `net.ipv4.tcp_tw_recycle`: 0
- `net.ipv4.tcp_syncookies`: 0
- `net.core.somaxconn`: 32768
- `vm.swappiness`: 0
- `vm.overcommit_memory`: 0 or 1
- `fs.file-max`: 1000000

Transparent Huge Pages (THP)

Check whether THP is enabled on the target machine. It is recommended to disable THP.

System limits

Check the limit values in the `/etc/security/limits.conf` file:

```
<deploy-user> soft nofile 1000000
<deploy-user> hard nofile 1000000
<deploy-user> soft stack 10240
```

`<deploy-user>` is the user who deploys and runs the TiDB cluster, and the last column is the minimum value required for the system.

SELinux

Check whether SELinux is enabled. It is required to disable SELinux.

Firewall

Check whether the FirewallD service is enabled. It is recommended to either disable the FirewallD service or add permission rules for each service in the TiDB cluster.

irqbalance

Check whether the irqbalance service is enabled. It is recommended to enable the irqbalance service.

Disk mount options

Check the mount options for ext4 partitions. Make sure the mount options include the nodelalloc option and the noatime option.

Port usage

Check if the ports defined in the topology (including the auto-completion default ports) are already used by the processes on the target machine.

> **Note:**
>
> The port usage check assumes that a cluster is not started yet. If a cluster is already deployed and started, the port usage check on the cluster fails because the ports must be in use in this case.

CPU core number

Check the CPU information of the target machine. For a production cluster, it is recommended that the number of the CPU logical core is greater than or equal to 16.

> **Note:**
>
> CPU core number is not checked by default. To enable the check, you need to add the `-enable-cpu` option to the command.

Memory size

Check the memory size of the target machine. For a production cluster, it is recommended that the total memory capacity is greater than or equal to 32GB.

> **Note:**
>
> Memory size is not checked by default. To enable the check, you need to add the `-enable-mem` option to the command.

Fio disk performance test

Use flexible I/O tester (fio) to test the performance of the disk where `data_dir` is located, including the following three test items:

- fio_randread_write_latency
- fio_randread_write
- fio_randread

> **Note:**
>
> The fio disk performance test is not performed by default. To perform the test, you need to add the `-enable-disk` option to the command.

Syntax

```
tiup cluster check <topology.yml | cluster-name> [flags]
```

- If a cluster is not deployed yet, you need to pass the topology.yml file that is used to deploy the cluster. According to the content in this file, tiup-cluster connects to the corresponding machine to perform the check.
- If a cluster is already deployed, you can use the `<cluster-name>` as the check object.

> **Note:**
>
> If `<cluster-name>` is used for the check, you need to add the `--cluster` option in the command.

Options

–apply

- Attempts to automatically repair the failed check items. Currently, tiup-cluster only attempts to repair the following check items:
    - SELinux
    - firewall
    - irqbalance
    - kernel parameters
    - System limits
    - THP (Transparent Huge Pages)
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

–cluster

- Indicates that the check is for the deployed clusters.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Note:

tiup-cluster supports checking both un-deployed clusters and deployed clusters with the following command format:

```
tiup cluster check <topology.yml | cluster-name> [flags]
```

If the `tiup cluster check <cluster-name>` command is used, you must add the `--cluster` option: `tiup cluster check <cluster-name> --cluster`.

-N, –node

- Specifies the nodes to be checked. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all nodes are checked by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are checked.

-R, –role

- Specifies the roles to be checked. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all roles are checked by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are checked.

–enable-cpu

- Enables the check of CPU core number.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

–enable-disk

- Enables the fio disk performance test.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

–enable-mem

- Enables the memory size check.
- Data type: `BOOLEAN`

- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

–u, –user

- Specifies the user name to connect to the target machine. The specified user needs to have the password-free sudo root privileges on the target machine.
- Data type: `STRING`
- If this option is not specified in the command, the user who executes the command is used as the default value.

> **Note:**
>
> This option is valid only if the `-cluster` option is false. Otherwise, the value of this option is fixed to the username specified in the topology file for the cluster deployment.

-i, –identity_file

- Specifies the key file to connect to the target machine.
- Data type: `STRING`
- The option is enabled by default with `~/.ssh/id_rsa` (the default value) passed in.

> **Note:**
>
> This option is valid only if the `--cluster` option is false. Otherwise, the value of this option is fixed to `${TIUP_HOME}/storage/cluster/clusters` ↪ `/<cluster-name>/ssh/id_rsa`.

-p, –password

- Logs in with a password when connecting to the target machine.
  - If the `--cluster` option is added for a cluster, the password is the password of the user specified in the topology file when the cluster was deployed.
  - If the `--cluster` option is not added for a cluster, the password is the password of the user specified in the `-u/--user` option.
- Data type: `BOOLEAN`

- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-h, –help

- Prints the help information of the related commands.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

A table containing the following fields:

- `Node`: the target node
- `Check`: the check item
- `Result`: the check result (Pass, Warn, or Fail)
- `Message`: the result description

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.5 tiup cluster clean

In the test environment, sometimes you might need to reset the cluster back to the state as it has been just deployed, which means deleting all data. You can do that easily using the `tiup cluster clean` command. After running it, the cluster is stopped and then cluster data is deleted. After restarting the cluster manually, you will get a clean cluster.

> **Warning:**
>
> This command will first stop the cluster even if you choose only to clean up logs. Therefore, do not use it in a production environment.

Syntax

```
tiup cluster clean <cluster-name> [flags]
```

`<cluster-name>` is the cluster to clean.

Options

–all

- Cleans data and the log at the same time. It is equivalent to specifying `--data` and `--log` at the same time.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.
- If it is not specified, you must specify at lease one of the following options:
    - –data: Cleans data
    - –log: Cleans the log

–data

- Cleans data. If neither of it nor `--all` is specified, data will not be cleaned.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–log

- Cleans the log. If neither of it nor `--all` is specified, the log will not be cleaned.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–ignore-node

- Specifies the node that does not need cleaning. To specify multiple nodes, you can use this option multiple times. For example, `--ignore-node <node-A> --ignore-node <node-B>`.
- Data type: `StringArray`
- Default: empty

–ignore-role

- Specifies the role that does not need cleaning. To specify multiple roles, you can use this option multiple times. For example, `--ignore-role <role-A> --ignore-role <role-B>`.
- Data type: `StringArray`
- Default: empty

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

The execution logs of tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.6 tiup cluster deploy

The `tiup cluster deploy` command is used to deploy a new cluster.

Syntax

```
tiup cluster deploy <cluster-name> <version> <topology.yaml> [flags]
```

- `<cluster-name>`: the name of the new cluster, which cannot be the same as the existing cluster names.
- `<version>`: the version number of the TiDB cluster to deploy, such as `v6.1.7`.
- `<topology.yaml>`: the prepared topology file.

Options

-u, –user

- Specifies the user name used to connect to the target machine. This user must have the secret-free sudo root permission on the target machine.
- Data type: `STRING`
- Default: the current user who executes the command.

-i, –identity_file

- Specifies the key file used to connect to the target machine.
- Data type: `STRING`
- If this option is not specified in the command, the `~/.ssh/id_rsa` file is used to connect to the target machine by default.

-p, –password

- Specifies the password used to connect to the target machine. Do not use this option with `-i/--identity_file` at the same time.

- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–ignore-config-check

- This option is used to skip the configuration check. After the binary files of components are deployed, the configurations of TiDB, TiKV, and PD components are checked using `<binary> --config-check <config-file>`. `<binary>` is the path of the deployed binary file. `<config-file>` is the configuration file generated based on the user configuration.
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.
- Default: false

–no-labels

- This option is used to skip the label check.
- When two or more TiKV nodes are deployed on the same physical machine, a risk exists: PD cannot learn the cluster topology, so PD might schedule multiple replicas of a Region to different TiKV nodes on one physical machine, which makes this physical machine a single point. To avoid this risk, you can use labels to tell PD not to schedule the same Region to the same machine. See Schedule Replicas by Topology Labels for label configuration.
- For the test environment, this risk might matter and you can use `--no-labels` to skip the check.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–skip-create-user

- During the cluster deployment, tiup-cluster checks whether the specified user name in the topology file exists or not. If not, it creates one. To skip this check, you can use the `--skip-create-user` option.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

The deployment log.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.7 tiup cluster destroy

After an application goes offline, if you want to release the machines occupied by the cluster for use by other applications, you need to clean up the data on the cluster and the deployed binary files. To destroy the cluster, the `tiup cluster destroy` command performs the following operations:

- Stops the cluster.
- For each service, delete its log directory, deployment directory, and data directory.
- If the parent directory of the data directory or deployment directory of each service is created by tiup-cluster, also delete the parent directory.

Syntax

```
tiup cluster destroy <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to destroy.

Options

–force

- In some cases, some nodes in the cluster have been down, making it impossible to connect to the node through SSH for operation. At this time, you can use the `--force` option to ignore these errors.
- Data type: `Boolean`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–retain-node-data

- Specifies the nodes that need to retain data. If you need to specify more than one node, use this option multiple times: `--retain-node-data <node-A> --retain-node-data ↪ <node-B>`.

- Data type: `StringArray`
- Default: empty

–retain-role-data

- Specifies the role that needs to retain data. If you need to specify more than one role, use this option multiple times: `--retain-role-data <role-A> --retain-role-data <role-B>`.
- Data type: `StringArray`
- Default: empty

-h, –help

- Prints the help information.
- Data type: `Boolean`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

The execution log of the tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.8 tiup cluster disable

After restarting the machine on which the cluster service is located, the cluster service will be automatically enabled. To disable the auto-enabling of cluster service, you can use the `tiup cluster disable` command. This command executes `systemctl disable <service>` on the specified node to disable the auto-enabling of the service.

Syntax

```
tiup cluster disable <cluster-name> [flags]
```

`<cluster-name>`: the cluster whose service auto-enabling is to be disabled.

Options

-N, –node

- Specifies the nodes whose service auto-enabling is to be disabled. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`

- If this option is not specified in the command, the auto-enabling of all nodes is disabled by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is disabled.

-R, –role

- Specifies the roles whose service auto-enabling is to be disabled. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the auto-enabling of all roles is disabled by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is disabled.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

The execution log of the tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.9 tiup cluster display

If you want to see the operation status of each component in the cluster, it is obviously inefficient to log in to each machine one by one. Therefore, tiup-cluster provides the `tiup cluster display` command to efficiently complete this task.

Syntax

```
tiup cluster display <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the cluster list command.

Options

–dashboard

- By default, all node information of the entire cluster is displayed. With the `--dashboard` option, only dashboard information is displayed.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

-N, –node

- Specifies the node to display. If this option is not specified, all nodes are displayed. The value of this option is a comma-separated list of node IDs. If you are not sure about the ID of a node, you can skip this option in the command to show the IDs and status of all nodes in the output.
- Data type: `STRINGS`
- If this option is not specified in the command, all nodes are checked by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are checked.

-R, –role

- Specifies the role to display. If it is not specified, all roles are displayed. The value of this option is a comma-separated list of node roles. If you are not sure about the role deployed on a node, you can skip this option in the command to show the roles and status of all nodes in the output.

- Data type: `STRINGS`
- If this option is not specified in the command, all roles are displayed by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are displayed.

–process

- Displays the CPU and memory usage information of the node when this option is enabled. This option is disabled by default.
- Data type: `BOOLEAN`
- Default value: `false`
- To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–uptime

- Displays the `uptime` information of the node when this option is enabled. This option is disabled by default.
- Data type: `BOOLEAN`
- Default value: `false`
- To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

–status-timeout

- Specifies the timeout period for obtaining the node status information.
- Data type: `INT`
- Default value: `10`, in the unit of second.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

- The cluster name
- The cluster version
- SSH Client Type
- Dashboard address
- The table with the following fields:
    - ID: the node ID, composed of `IP:PORT`
    - Role: the service role deployed on this node (such as TiDB, TiKV)
    - Host: the IP of the machine corresponding to the node
    - Ports: the port number occupied by the service
    - OS/Arch: the operating system and the machine architecture of this node
    - Status: the current status of the node service
    - Data Dir: the data directory of the service. `-` means no data directory.
    - Deploy Dir: the deployment directory of the service

Node service status

A node service can run in one of the following statuses:

- Up: The node service is running normally.
- Down or Unreachable: The node service is not running or a network problem exists on the corresponding host.
- Tombstone: The data on the node service has been completely migrated out and the scaling-in is complete. This status exists only on TiKV or TiFlash.
- Pending Offline: The data on the node service is being migrated out and the scaling-in is in process. This status exists only on TiKV or TiFlash.
- Unknown: The running status of the node service is unknown.

> **Note:**
>
> `Pending Offline` in TiUP, `Offline` returned by PD API, and `Leaving` in TiDB Dashboard indicate the same status.

Node service status derives from the PD scheduling information. For more details, see Information collection.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.10 tiup cluster edit-config

If you need to modify the cluster configuration after the cluster is deployed, you can use the `tiup cluster edit-config` command that starts an editor for you to modify the topology file of a cluster. This editor is specified in the `$EDITOR` environment variable by default. If the `$EDITOR` environment variable does not exist, the `vi` editor is used.

Note:

- When you modify the configuration, you cannot add or delete machines. For how to add machines, see Scale out a cluster. For how to delete machines, see Scale in a cluster.
- After you execute the `tiup cluster edit-config` command, the configuration is modified only on the control machine. Then you need to execute the `tiup cluster reload` command to reload the configuration.

Syntax

```
tiup cluster edit-config <cluster-name> [flags]
```

`<cluster-name>` is the cluster to operate on.

Option

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

- If the command is successfully executed, there is no output.
- If you have mistakenly modified the fields that cannot be modified, when you save the file, an error will be reported, reminding you to edit the file again. For the fields that cannot be modified, see the topology file.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.11 tiup cluster enable

The `tiup cluster enable` command is used to set the auto-enabling of the cluster service after a machine is restarted. This command enables the auto-enabling of the service by executing `systemctl enable <service>` at the specified node.

> **Note:**
>
> When all clusters are shut down and restarted, the order of service startup is determined by the node's operating system startup order. When the restart order is incorrect, in some cases, the restarted cluster still cannot provide services. For example, if TiKV is started first but PD is not started, systemd gives up if TiKV is restarted multiple times while PD is not found).

Syntax

```
tiup cluster enable <cluster-name> [flags]
```

`<cluster-name>`: the cluster whose service auto-enabling is to be enabled.

Options

-N, –node

- Specifies the nodes whose service auto-enabling is to be enabled. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the auto-enabling of all nodes is enabled by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is enabled.

-R, –role

- Specifies the roles whose service auto-enabling is to be enabled. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup cluster display` command.

- Data type: `STRINGS`
- If this option is not specified in the command, the auto-enabling of all roles is enabled by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is enabled.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

The execution log of the tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.12 tiup cluster help

tiup-cluster provides a wealth of help information for users in the command line interface. You can obtain it via the `help` command or the `--help` option. `tiup cluster help <command>` is basically equivalent to `tiup cluster <command> --help`.

Syntax

```
tiup cluster help [command] [flags]
```

`[command]` is used to specify the help information of which command that users need to view. If it is not specified, the help information of tiup-cluster is viewed.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The help information of the `[command]` or tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.13 tiup cluster import

Before TiDB v4.0, TiDB clusters were mainly deployed using TiDB Ansible. For TiDB v4.0 and later releases, TiUP Cluster provides the `import` command to transfer the clusters to the tiup-cluster component for management.

Note:

- After importing the TiDB Ansible configuration to TiUP for management, **DO NOT** use TiDB Ansible for cluster operations anymore. Otherwise, conflicts might be caused due to inconsistent meta information.
- If the clusters deployed using TiDB Ansible are in any of the following situations, do not use the `import` command.
  - Clusters with TLS encryption enabled
  - Pure KV clusters (clusters without TiDB instances)
  - Clusters with Kafka enabled
  - Clusters with Spark enabled
  - Clusters with TiDB Lightning/TiKV Importer enabled
  - Clusters still using the old `push` mode to collect monitoring metrics (if you keep the default mode `pull` unchanged, using the `import` command is supported)
  - Clusters in which the non-default ports (the ports configured in the `group_vars` directory are compatible) are separately configured in the `inventory.ini` configuration file using `node_exporter_port` / `blackbox_exporter_port`
- If some nodes in the cluster deployed using TiDB Ansible are deployed without monitoring components, you should first use TiDB Ansible to add the corresponding node information in the `monitored_servers` section of the `inventory.ini` file, and then use the `deploy.yaml` playbook to fully deploy monitoring components. Otherwise, when you perform maintenance operations after the cluster is imported into TiUP, errors might occur due to the lack of monitoring components.

Syntax

```
tiup cluster import [flags]
```

Options

-d, –dir

- Specifies the directory where TiDB Ansible is located.

- Data type: `STRING`
- The option is enabled by default with the current directory (the default value) passed in.

–ansible-config

- Specifies the path of the Ansible configuration file.
- Data type: `STRING`
- The option is enabled by default with `. /ansible.cfg` (the default value) passed in.

–inventory

- Specifies the name of the Ansible inventory file.
- Data type: `STRING`
- The option is enabled by default with `inventory.ini` (the default value) passed in.

–no-backup

- Controls whether to disable the backup of files in the directory where TiDB Ansible is located.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. After a successful import, everything in the directory specified by the `-dir` option is backed up to the `${TIUP_HOME}/.tiup/storage/cluster/clusters/{cluster-name}/ansible-backup` directory. If there are multiple inventory files (when multiple clusters are deployed) in this directory, it is recommended to enable this option. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

–rename

- Renames the imported cluster.
- Data type: `STRING`
- Default: NULL. If this option is not specified in the command, the cluster_name specified in inventory is used as the cluster name.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

Shows the logs of the import process.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.14 tiup cluster list

tiup-cluster supports deploying multiple clusters using the same control machine. The `tiup cluster list` command outputs all clusters deployed by the currently logged-in user using this control machine.

> **Note:**
>
> The deployed cluster data is stored in the `~/.tiup/storage/cluster/clusters/` directory by default, so on the same control machine, the currently logged-in user cannot view the clusters deployed by other users.

Syntax

```
tiup cluster list [flags]
```

Options

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Outputs

Outputs the table with the following fields:

- Name: the cluster name
- User: the deployment user
- Version: the cluster version
- Path: the path of the cluster deployment data on the control machine
- PrivateKey: the path of the private key that is used to connect the cluster

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.15 tiup cluster meta backup

The TiUP meta file is used for cluster operation and maintenance (OM). If this file is lost, you cannot use TiUP to manage the cluster. To avoid this situation, you can use the `tiup cluster meta backup` command to back up the TiUP meta file regularly.

Syntax

```
tiup cluster meta backup <cluster-name> [flags]
```

`<cluster-name>` is the name of the cluster to be operated on. If you forget the cluster name, you can check it using the `tiup cluster list` command.

Options

–file (string, defaults to the current directory)

Specifies the target directory to store the TiUP meta backup file.

-h, –help

- Prints the help information.
- Data type: `Boolean`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

The execution logs of tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.16 tiup cluster meta restore

To restore the TiUP meta file, you can use the `tiup cluster meta restore` command to restore from the backup file.

Syntax

```
tiup cluster meta restore <cluster-name> <backup-file> [flags]
```

- `<cluster-name>` is the name of the cluster to be operated on.
- `<backup-file>` is the path to the TiUP meta backup file.

> **Note:**
>
> The restore operation overwrites the current meta file. It is recommended to restore the meta file only when it is lost.

Options

-h, –help

- Prints the help information.
- Data type: `Boolean`
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

Output

The execution logs of tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.17 tiup cluster patch

If you need to dynamically replace the binaries of a service while the cluster is running (namely, keep the cluster available during the replacement process), you can use the `tiup cluster patch` command. After the command is executed, TiUP does the following things:

- Uploads the binary package for replacement to the target machine.
- If the target service is a storage service such as TiKV, TiFlash, or TiDB Binlog, TiUP first takes the related nodes offline via the API.
- Stops the target service.
- Unpacks the binary package and replace the service.
- Starts the target service.

Syntax

```
tiup cluster patch <cluster-name> <package-path> [flags]
```

- `<cluster-name>`: The name of the cluster to be operated.
- `<package-path>`: The path to the binary package used for replacement.

Preparation

Before running the `tiup cluster patch` command, you need to pack the binary package required. Take the following steps:

1. Determine the following variables:
    - `${component}`: the name of the component to be replaced (such as `tidb`, `tikv`, or `pd`).
    - `${version}`: the version of the component (such as `v6.1.7`).
    - `${os}`: the operating system (`linux`).
    - `${arch}`: the platform on which the component runs (`amd64`, `arm64`).

2. Download the current component package using the command:

```
wget https://tiup-mirrors.pingcap.com/${component}-${version}-${os}-${
    ↪ arch}.tar.gz -O /tmp/${component}-${version}-${os}-${arch}.tar.gz
```

3. Create a temporary directory to pack files and change to it:

```
mkdir -p /tmp/package && cd /tmp/package
```

4. Extract the original binary package:

```
tar xf /tmp/${component}-${version}-${os}-${arch}.tar.gz
```

5. Check out the file structure in the temporary directory:

```
find .
```

6. Copy the binary files or configuration files to their corresponding locations in the temporary directory.

7. Pack all files in the temporary directory:

```
tar czf /tmp/${component}-hotfix-${os}-${arch}.tar.gz *
```

After you have completed the preceding steps, you can use `/tmp/${component}-hotfix-
↪ ${os}-${arch}.tar.gz` as the `<package-path>` in the `tiup cluster patch` command.

Options

–overwrite

- After you patch a certain component (such as TiDB or TiKV), when the tiup cluster scales out the component, TiUP uses the original component version by default. To use the version that you patch when the cluster scales out in the future, you need to specified the option `--overwrite` in the command.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

–transfer-timeout

- When restarting the PD or TiKV service, TiKV/PD first transfers the leader of the node to be restarted to another node. Because the transfer process takes some time, you can use the option `--transfer-timeout` to set the maximum waiting time (in seconds). After the timeout, TiUP directly restarts the service.
- Data type: `UINT`

- If this option is not specified, TiUP directly restarts the service after waiting for `300` seconds.

> **Note:**
>
> If TiUP directly restarts the service after the timeout, the service performance might jitter.

-N, –node

- Specifies nodes to be replaced. The value of this option is a comma-separated list of node IDs. You can get the node ID from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified, TiUP does not select any nodes to replace by default.

> **Note:**
>
> If the option `-R, --role` is specified at the same time, TiUP then replaces service nodes that match both the requirements of `-N, --node` and `-R, --role`.

-R, –role

- Specifies the roles to be replaced. The value of this option is a comma-separated list of the roles of the nodes. You can get the role deployed on a node from the second column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified, TiUP does not select any roles to replace by default.

> **Note:**
>
> If the option `-N, --node` is specified at the same time, TiUP then replaces service nodes that match both the requirements of `-N, --node` and `-R, --role`.

–offline

- Declares that the current cluster is not running. When the option is specified, TiUP does not evict the service leader to another node or restart the service, but only replaces the binary files of cluster components.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Outputs

The execution log of the tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.18 tiup cluster prune

When scaling in the cluster, for some components, TiUP does not immediately stop their services or delete their data. You need to wait for the data scheduling to complete and then manually execute the `tiup cluster prune` command to clean up.

Syntax

```
tiup cluster prune <cluster-name> [flags]
```

Option

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The log of the cleanup process.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.19 tiup cluster reload

After modifying the cluster configuration, the cluster needs to be reloaded using the `tiup cluster reload` command for the configuration to take effect. This command publishes the configuration of the control machine to the remote machine where the service is running, and follows the upgrade process to restart the services in order according to the upgrade process. The cluster is still available during the restart process.

Syntax

```
tiup cluster reload <cluster-name> [flags]
```

`<cluster-name>`: the cluster name to operate on.

Options

–force

- Ignores errors in the reloading process and forces reload.
- Data type: `BOOLEAN`
- Default: false

–transfer-timeout

- When restarting PD or TiKV, the leader of the restarted node is migrated to other nodes first, and the migration process takes some time. You can set the maximum wait time (in seconds) by setting `-transfer-timeout`. After the timeout, the service can be restarted directly without waiting.
- Data type: `UINT`
- Default: 300

> **Note:**
>
> In the case of skipping the waiting and restarting directly, the service performance might jitter.

–ignore-config-check

- After the binary files of components are deployed, the configurations of TiDB, TiKV, and PD components are checked using `<binary> --config-check <config-file>`. `<binary>` is the path of the deployed binary file. `<config-file>` is the configuration file generated based on the user configuration. If you want to skip this check, you can use this option.
- Data type: `BOOLEAN`

- Default: false

-N, –node

- Specifies the nodes to be restarted. If not specified, all nodes are restarted. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all nodes are selected by default.

> **Note:**
>
> - If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are restarted.
> - If the `--skip-restart` option is specified, the `-N, --node` option is invalid.

-R, –role

- Specifies the roles to be restarted. If not specified, all roles are restarted. The value of this option is a comma-separated list of node roles. The role is the second column of the cluster status table.
- Data type: `STRINGS`
- If this option is not specified in the command, all roles are selected by default.

> **Note:**
>
> 1. If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are restarted.
> 2. If the `--skip-restart` option is specified, the `-R, --role` option is invalid.

–skip-restart

The `tiup cluster reload` command performs two operations:

- Refreshes all node configurations
- Restarts the specified node

After you specify the `--skip-restart` option, it only refreshes the configuration without restarting any nodes, so that the refreshed configuration is not applied and does not take effect until the next restart of the corresponding service.

- Data type: `BOOLEAN`
- Default: false

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The execution log of the tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.20 tiup cluster rename

The cluster name is specified when the cluster is deployed. If you want to change the cluster name after the cluster is deployed, you can use the command `tiup cluster rename`.

> **Note:**
>
> If the `dashboard_dir` field of `grafana_servers` is configured for the TiUP cluster, after you execute the command `tiup cluster rename` to rename the cluster, the following additional steps are required:
>
> - For the `*.json` files in the local dashboards directory, update the `datasource` field of each file to the new cluster name, because the value of `datasource` must be the name of the cluster.
> - Execute the command `tiup cluster reload -R grafana`.

Syntax

```
tiup cluster rename <old-cluster-name> <new-cluster-name> [flags]
```

- `<old-cluster-name>`: The old cluster name.
- `<new-cluster-name>`: The new cluster name.

Options

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Outputs

The execution log of the tiup-cluster.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.21 tiup cluster replay

When you perform a cluster operation such as upgrade or restart, the operation might fail due to cluster environment issues. If you re-perform the operation, you need to perform all the steps from the very beginning. If the cluster is large, re-performing these steps will take a long time. In this case, you can use the `tiup cluster replay` command to retry the failed commands and skip the successfully performed steps.

Syntax

```
tiup cluster replay <audit-id> [flags]
```

- `<audit-id>`: the `audit-id` of the command to be retried. You can view the historical commands and their `audit-id`s using the `tiup cluster audit` command.

Option

-h, –help

Prints the help information.

Output

The output of the command corresponding to `<audit-id>`.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.22 tiup cluster restart

The command `tiup cluster restart` is used to restart all or some of the services of the specified cluster.

> **Note:**
>
> During the restart process, the related services are unavailable for a period of time.

Syntax

```
tiup cluster restart <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the cluster list command.

Options

-N, –node

- Specifies the nodes to be restarted. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRING`
- If this option is not specified, TiUP restarts all nodes by default.

> **Note:**
>
> If the option `-R, --role` is specified at the same time, TiUP restarts service nodes that match both the requirements of `-N, --node` and `-R, --role`.

-R, –role

- Specified the roles of nodes to be restarted. The value of this option is a comma-separated list of the roles of the nodes. You can get the roles of the nodes from the second column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRING`
- If this option is not specified, TiUP restarts nodes of all roles by default.

> **Note:**
>
> If the option `-N, --node` is specified at the same time, TiUP restarts service nodes that match both the requirements of `-N, --node` and `-R, --role`.

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Outputs

The log of the service restart process.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.23 tiup cluster scale-in

The `tiup cluster scale-in` command is used to scale in the cluster, which takes the services of the specified nodes offline, removes the specified nodes from the cluster, and deletes the remaining files from those nodes.

Particular handling of components' offline process

Because the TiKV, TiFlash, and TiDB Binlog components are taken offline asynchronously (which requires TiUP to remove the node through API first) and the stopping process takes a long time (which requires TiUP to continuously check whether the node is successfully taken offline), the TiKV, TiFlash, and TiDB Binlog components are handled particularly as follows:

- For TiKV, TiFlash, and TiDB Binlog components:

    1. TiUP Cluster takes the node offline through API and directly exits without waiting for the process to be completed.
    2. To check the status of the nodes being scaled in, you need to execute the `tiup cluster display` command and wait for the status to become `Tombstone`.
    3. To clean up the nodes in the `Tombstone` status, you need to execute the `tiup cluster prune` command. The `tiup cluster prune` command performs the following operations:
        - Stops the services of the nodes that have been taken offline.
        - Cleans up the data files of the nodes that have been taken offline.
        - Updates the cluster topology and removes the nodes that have been taken offline.

For other components:

- When taking the PD components offline, TiUP Cluster quickly deletes the specified nodes from the cluster through API, stops the service of the specified PD nodes, and then deletes the related data files from the nodes.

- When taking other components down, TiUP Cluster directly stops the node services and deletes the related data files from the specified nodes.

Syntax

```
tiup cluster scale-in <cluster-name> [flags]
```

`<cluster-name>` is the name of the cluster to scale in. If you forget the cluster name, you can check it using the `tiup cluster list` command.

Options

-N, –node

- Specifies the nodes to take down. Multiple nodes are separated by commas.
- Data type: `STRING`
- There is no default value. This option is mandatory and the value must be not null.

–force

- Controls whether to forcibly remove the specified nodes from the cluster. Sometimes, the host of the node to take offline might be down, which makes it impossible to connect to the node via SSH for operations, so you can forcibly remove the node from the cluster using the `--force` option.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

> **Warning:**
>
> When you use this option to forcibly remove TiKV or TiFlash nodes that are in service or are pending offline, these nodes will be deleted immediately without waiting for data to be migrated. This imposes a very high risk of data loss. If data loss occurs in the region where the metadata is located, the entire cluster will be unavailable and unrecoverable.

–transfer-timeout

- When a PD or TiKV node is to be removed, the Region leader on the node will be transferred to another node first. Because the transferring process takes some time, you can set the maximum waiting time (in seconds) by configuring `--transfer-timeout`. After the timeout, the `tiup cluster scale-in` command skips waiting and starts the scaling-in directly.

- Data type: `UINT`
- The option is enabled by default with `300` seconds (the default value) passed in.

> **Note:**
>
> If a PD or TiKV node is taken offline directly without waiting for the leader transfer to be completed, the service performance might jitter.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

Shows the logs of the scaling-in process.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.24 tiup cluster scale-out

The `tiup cluster scale-out` command is used for scaling out the cluster. The internal logic of scaling out the cluster is similar to the cluster deployment. The tiup-cluster component first establishes an SSH connection to the new node, creates the necessary directories on the target node, then performs the deployment and starts the service.

When PD is scaled out, new PD nodes are added to the cluster by the join operation, and the configuration of the services associated with PD is updated; other services are directly started and added to the cluster.

Syntax

```
tiup cluster scale-out <cluster-name> <topology.yaml> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the `cluster list` command.

`<topology.yaml>`: the prepared topology file. This topology file should only contain the new nodes that are to be added to the current cluster.

Options

-u, –user

- Specifies the user name used to connect to the target machine. This user must have the secret-free sudo root permission on the target machine.
- Data type: `STRING`
- Default: the current user who executes the command.

-i, –identity_file

- Specifies the key file used to connect to the target machine.
- Data type: `STRING`
- If this option is not specified in the command, the `~/.ssh/id_rsa` file is used to connect to the target machine by default.

-p, –password

- Specifies the password used to connect to the target machine. Do not use this option and `-i/--identity_file` at the same time.
- Data type: `BOOLEAN`
- Default: false

–no-labels

- This option is used to skip the label check.
- When two or more TiKV nodes are deployed on the same physical machine, a risk exists: PD does not know the cluster topology, so it might schedule multiple replicas of a Region to different TiKV nodes on one physical machine, which makes this physical machine a single point of failure. To avoid this risk, you can use labels to tell PD not to schedule the same Region to the same machine. See Schedule Replicas by Topology Labels for label configuration.
- For the test environment, this risk might not matter, and you can use `--no-labels` to skip the check.
- Data type: `BOOLEAN`
- Default: false

–skip-create-user

- During the cluster deployment, tiup-cluster checks whether the specified user name in the topology file exists or not. If not, it creates one. To skip this check, you can use the `--skip-create-user` option.
- Data type: `BOOLEAN`
- Default: false

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The log of scaling out.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.25 tiup cluster start

The `tiup cluster start` command is used to start all services or some services of the specified cluster.

Syntax

```
tiup cluster start <cluster-name> [flags]
```

`<cluster-name>` is the name of the cluster to operate on. If you forget the cluster name, you can check it using the `tiup cluster list` command.

Options

–init

Starts the cluster in a safe way. It is recommended to use this option when the cluster is started for the first time. This method generates the password of the TiDB root user at startup and returns the password in the command line interface.

> **Note:**
>
> - After safe start of a TiDB cluster, you cannot log in to the database using the root user without a password. Therefore, you need to record the password returned by the command line for future logins.
> - The password is generated only once. If you do not record or forget the password, refer to Forget the `root` password to change the password.

-N, –node

- Specifies the nodes to be started. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all nodes are started by default.

Note:

If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are started.

-R, –role

- Specifies the roles of nodes to be started. The value of this option is a comma-separated list of the roles of the nodes. You can get the roles of the nodes from the second column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all roles are started by default.

Note:

If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are started.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

The log of starting the service.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.26 tiup cluster stop

The `tiup cluster stop` command is used to stop all services or some services of the specified cluster.

> **Note:**
>
> If the core services of a cluster are stopped, the cluster cannot provide services anymore.

Syntax

```
tiup cluster stop <cluster-name> [flags]
```

`<cluster-name>` is the name of the cluster to operate on. If you forget the cluster name, you can check it using the `tiup cluster list` command.

Options

-N, –node

- Specifies the nodes to be stopped. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the command stops all the nodes by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are stopped.

-R, –role

- Specifies the roles of nodes to be stopped. The value of this option is a comma-separated list of the roles of the nodes. You can get the roles of the nodes from the second column of the cluster status table returned by the `tiup cluster display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the command stops all the roles by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are stopped.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

The log of stopping the service.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.27 tiup cluster template

Before deploying the cluster, you need to prepare a topology file of the cluster. TiUP has a built-in topology file template, and you can modify this template to create the final topology file. To output the built-in template content, you can use the `tiup cluster template` command.

Syntax

```
tiup cluster template [flags]
```

If this option is not specified, the output default template contains the following instances:

- 3 PD instances
- 3 TiKV instances
- 1 TiDB instance
- 1 Prometheus instance
- 1 Grafana instance
- 1 Alertmanager instance

Options

–full

- Outputs a detailed topology template that is commented with configurable parameters. To enable this option, add it to the command.
- If this option is not specified, the simple topology template is output by default.

–local

- Outputs a simple topology template for the local cluster, which can be used directly, and the `global` parameter can be adjusted as needed.
- This template creates a PD service, a TiDB service, a TiKV service, a monitoring service, and a Grafana service.

–multi-dc

- Outputs the topology template of multiple data centers. To enable this option, add it to the command.
- If this option is not specified, the topology template of a single data center is output by default.

-h, –help

Prints the help information.

Output

Outputs the topology template according to the specified options, which can be redirected to the topology file for deployment.

<< Back to the previous page - TiUP Cluster command list

13.4.7.3.28 tiup cluster upgrade

The `tiup cluster upgrade` command is used to upgrade the specified cluster to a specific version.

Syntax

```
tiup cluster upgrade <cluster-name> <version> [flags]
```

- `<cluster-name>`: the cluster name to operate on. If you forget the cluster name, you can check it with the cluster list command.
- `<version>`: the target version to upgrade to. Currently, it is only allowed to upgrade to a version higher than the current cluster, that is, no downgrade is allowed. It is also not allowed to upgrade to the nightly version.

Options

–force

- To upgrade the cluster, you need to ensure that the cluster is currently started. In some cases, you might want to upgrade when the cluster is not started. At this time, you can use `--force` to ignore the error during the upgrade, forcibly replace the binary file and start the cluster.
- Data type: `BOOLEAN`
- Default: false

> **Note:**
>
> Forcing an upgrade of the cluster that is providing services might result in service unavailability. Unstarted clusters are started automatically after a successful upgrade.

–transfer-timeout

- When upgrading PD or TiKV, the leader of the upgraded node is migrated to other nodes first. The migration process takes some time, and you can set the maximum wait time (in seconds) by the `-transfer-timeout` option. After the timeout, the wait is skipped and the service is upgraded directly.
- Data type: `uint`
- Default: 300

> **Note:**
>
> If the wait is skipped and the service is upgraded directly, the service performance might jitter.

–ignore-config-check

- After the binary is updated, a configuration check is performed on the TiDB, TiKV and PD components using `<binary> --config-check <config-file>`. `<binary>` is the path to the newly deployed binary and `<config-file>` is the configuration file generated based on the user configuration. To skip this check, you can use the `--ignore-config-check` option.
- Data type: `BOOLEAN`
- Default: false

–offline

- Declares that the current cluster is not running. When this option is specified, TiUP does not evict the service leader to another node or restart the service, but only replaces the binary files of the cluster components.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The log of the upgrading progress.

<< Back to the previous page - TiUP Cluster command list

13.4.7.4 TiUP DM Commands

13.4.7.4.1 TiUP DM

Similar to TiUP Cluster which is used to manage TiDB clusters, TiUP DM is used to manage DM clusters. You can use the TiUP DM component to perform daily operations and maintenance tasks of DM clusters, including deploying, starting, stopping, destroying, elastic scaling, upgrading DM clusters, and managing the configuration parameters of DM clusters.

Syntax

```
tiup dm [command] [flags]
```

`[command]` is used to pass the name of the command. See the Command list for supported commands.

Options

–ssh

- Specifies the SSH client to connect to the remote end (the machine where the TiDB service is deployed) for the command execution.
- Data type: `STRING`
- Support values:
  - `builtin`: Uses the built-in easyssh client of tiup-cluster as the SSH client.

    - `system`: Uses the default SSH client of the current operating system.
    - `none`: No SSH client is used. The deployment is only for the current machine.

- If this option is not specified in the command, `builtin` is used as the default value.

–ssh-timeout

- Specifies the SSH connection timeout in seconds.
- Data type: `UINT`
- If this option is not specified in the command, the default timeout is `5` seconds.

–wait-timeout

- Specifies the maximum waiting time (in seconds) for each step in the operation process. The operation process consists of many steps, such as specifying systemctl to start or stop services, and waiting for ports to be online or offline. Each step may take several seconds. If the execution time of a step exceeds the specified timeout, the step exits with an error.
- Data type: `UINT`
- If this option is not specified in the command, the maximum waiting time for each steps is `120` seconds.

-y, –yes

- Skips the secondary confirmation of all risky operations. It is not recommended to use this option unless you use a script to call TiUP.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-v, –version

- Prints the current version of TiUP DM.
- Data type:`BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-h, –help

- Prints help information about the specified command.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Command list

- import: Imports a DM v1.0 cluster deployed by DM-Ansible.
- template: Outputs the topology template.
- deploy: Deploys a cluster based on a specified topology.
- list: Queries the list of deployed clusters.
- display: Displays the status of a specified cluster.
- start: Starts a specified cluster.
- stop: Stops a specified cluster.
- restart: Restarts a specified cluster.
- scale-in: Scales in a specified cluster.
- scale-out: Scales out a specified cluster.
- upgrade: Upgrades a specified cluster.
- prune: Cleans up instances in the Tombstone status for a specified cluster.
- edit-config: Modifies the configuration of a specified cluster.
- reload: Reloads the configuration of a specified cluster.
- patch: Replaces a specified service in a deployed cluster.
- destroy: Destroys a specified cluster.
- audit: Queries the operation audit log of a specified cluster.
- replay: Replays the specified commands
- enable: Enables the auto-enabling of the cluster service after a machine is restarted.
- disable: Disables the auto-enabling of the cluster service after a machine is restarted.
- help: Prints help information.

<< Back to the previous page - TiUP Reference component list

13.4.7.4.2 tiup dm audit

The `tiup dm audit` command is used to view historical commands executed on all clusters and the execution log of each command.

Syntax

```
tiup dm audit [audit-id] [flags]
```

- If you do not fill in the `[audit-id]`, the table of operation records is output in reverse chronological order. The first column is the `audit-id`.
- If you fill in the `[audit-id]`, the execution log of the specified `audit-id` is checked.

Option

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`

- Default: false

Output

- If `[audit-id]` is specified, the corresponding execution log is output.
- If `[audit-id]` is not specified, a table with the following fields is output:
    - ID: the `audit-id` corresponding to this record
    - Time: the execution time of the command corresponding to the record
    - Command: the command corresponding to the record

<< Back to the previous page - TiUP DM command list

13.4.7.4.3 tiup dm deploy

The `tiup dm deploy` command is used to deploy a new cluster.

Syntax

```
tiup dm deploy <cluster-name> <version> <topology.yaml> [flags]
```

- `<cluster-name>`: the name of the new cluster, which cannot be the same as the existing cluster names.
- `<version>`: the version number of the DM cluster to be deployed, such as `v2.0.0`.
- `<topology.yaml>`: the prepared topology file.

Options

-u, –user

- Specifies the user name used to connect to the target machine. This user must have the secret-free sudo root permission on the target machine.
- Data type: `STRING`
- Default: the current user who executes the command.

-i, –identity_file

- Specifies the key file used to connect to the target machine.
- Data type: `STRING`
- Default: `~/.ssh/id_rsa`

-p, –password

- Specifies the password used to connect to the target machine. Do not use this option and `-i/--identity_file` at the same time.

- Data type: `BOOLEAN`
- Default: false

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The deployment log.

<< Back to the previous page - TiUP DM command list

13.4.7.4.4 tiup dm destroy

After an application goes offline, if you want to release the machines occupied by the cluster for use by other applications, you need to clean up the data on the cluster and the deployed binary files. To destroy the cluster, the `tiup dm destroy` command performs the following operations:

- Stops the cluster.
- For each service, delete its log directory, deployment directory, and data directory.
- If the parent directory of the data directory or deployment directory of each service is created by `tiup-dm`, also delete the parent directory.

Syntax

```
tiup dm destroy <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to be destroyed.

Option

-h, –help

- Prints the help information.
- Data type: `Boolean`
- Default: false

Output

The execution log of the tiup-dm.

<< Back to the previous page - TiUP DM command list

13.4.7.4.5 tiup dm disable

After restarting the machine on which the cluster service is located, the cluster service will be automatically enabled. To disable the auto-enabling of cluster service, you can use the `tiup dm disable` command. This command executes `systemctl disable <service>` on the specified node to disable the auto-enabling of the service.

Syntax

```
tiup dm disable <cluster-name> [flags]
```

`<cluster-name>` is the cluster whose service auto-enabling is to be disabled.

Options

-N, –node

- Specifies the nodes whose service auto-enabling is to be disabled. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the auto-enabling of all nodes is disabled by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is disabled.

-R, –role

- Specifies the roles whose service auto-enabling is to be disabled. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the auto-enabling of all roles is disabled by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is disabled.

-h, –help

Prints the help information.

Output

The execution log of the tiup-dm.

<< Back to the previous page - TiUP DM command list

13.4.7.4.6 tiup dm display

If you want to check the operational status of each component in a DM cluster, it is inefficient to log in to each machine one by one. Therefore, tiup-dm provides the `tiup dm display` command to do this job efficiently.

Syntax

```
tiup dm display <cluster-name> [flags]
```

`<cluster-name>` is the name of the cluster to be operated. If you forget the cluster name, you can check it using the `tiup dm list` command.

Options

-N, –node

- Specifies the IDs of the nodes to query, splitting by commas for multiple nodes. If you are not sure about the ID of a node, you can skip this option in the command to show the IDs and status of all nodes in the output.
- Data type: `STRING`
- This option is enabled by default with `[]` (which means all nodes) passed in.

> **Note:**
>
> If `-R, --role` is also specified, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are queried.

-R, –role

- Specifies the roles to query, splitting by commas for multiple roles. If you are not sure about the role deployed on a node, you can skip this option in the command to show the roles and status of all nodes in the output.
- Data type: `STRING`
- This option is enabled by default with `[]` (which means all roles) passed in.

Note:

If `-N, --node` is also specified, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are queried.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

- Cluster name
- Cluster version
- SSH client type
- A table containing the following fields:
  - `ID`: the node ID, consisting of IP:PORT.
  - `Role`: the service role deployed on the node (for example, TiDB or TiKV).
  - `Host`: the IP address of the machine corresponding to the node.
  - `Ports`: the port number used by the service.
  - `OS/Arch`: the operating system and machine architecture of the node.
  - `Status`: the current status of the services on the node.
  - `Data Dir`: the data directory of the service. `-` means that there is no data directory.
  - `Deploy Dir`: the deployment directory of the service.

<< Back to the previous page - TiUP DM command list

13.4.7.4.7 tiup dm edit-config

If you need to modify the cluster service configuration after the cluster is deployed, you can use the `tiup dm edit-config` command that starts an editor for you to modify the topology file. of the specified cluster. This editor is specified in the `$EDITOR` environment variable by default. If the `$EDITOR` environment variable does not exist, the `vi` editor is used.

Note:

- When you modify the configuration, you cannot add or delete machines. For how to add machines, see Scale out a cluster. For how to delete machines, see Scale in a cluster.
- After you execute the `tiup dm edit-config` command, the configuration is modified only on the control machine. Then you need to execute the `tiup dm reload` command to reload the configuration.

Syntax

```
tiup dm edit-config <cluster-name> [flags]
```

`<cluster-name>`: the cluster to operate on.

Option

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

- Normally, no output.
- If you have mistakenly modified the fields that cannot be modified, when you save the file, an error is reported, reminding you to edit the file again. For the fields that cannot be modified, see the topology file.

<< Back to the previous page - TiUP DM command list

13.4.7.4.8 tiup dm enable

The `tiup dm enable` command is used to set the auto-enabling of the cluster service after a machine is restarted. This command enables the auto-enabling of the service by executing `systemctl enable <service>` at the specified node.

Syntax

```
tiup dm enable <cluster-name> [flags]
```

`<cluster-name>` is the cluster whose service auto-enabling is to be enabled.

Options

-N, –node

- Specifies the nodes whose service auto-enabling is to be enabled. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the auto-enabling of all nodes is enabled by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is enabled.

-R, –role

- Specifies the roles whose service auto-enabling is to be enabled. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, the auto-enabling of all roles is enabled by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, the auto-enabling of services that match both the specifications of `-N, --node` and `-R, --role` is enabled.

-h, –help

Prints the help information.

Output

the execution log of tiup-dm.

<< Back to the previous page - TiUP DM command list

13.4.7.4.9 tiup dm help

tiup-dm command-line interface provides users with a wealth of help information. You can view it via the `help` command or the `--help` option. Basically, `tiup dm help <command >` is equivalent to `tiup dm <command> --help`.

Syntax

```
tiup dm help [command] [flags]
```

`[command]` is used to specify the help information of which command that users need to view. If it is not specified, the help information of `tiup-dm` is viewed.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The help information of `[command]` or `tiup-dm`.

<< Back to the previous page - TiUP DM command list

13.4.7.4.10 tiup dm import Only for upgrading DM v1.0

This command is used only for upgrading DM clusters from v1.0 to v2.0 or later versions.

In DM v1.0, the cluster is basically deployed using TiDB Ansible. TiUP DM provides the `import` command to import v1.0 clusters and redeploy the clusters in DM v2.0.

Note:

- The command does not support importing DM Portal components from DM v1.0 clusters.
- Before importing the cluster, stop running the original cluster first.
- For data migration tasks that need to be upgraded to v2.0, do not execute `stop-task` on these tasks.
- The command only supports importing to DM v2.0.0-rc.2 and later versions.
- The `import` command is used to import a DM v1.0 cluster to a new DM v2.0 cluster. If you need to import data migration tasks to an existing v2.0 cluster, refer to Manually Upgrade TiDB Data Migration from v1.0.x to v2.0+

- The deployment directories of some components might be different from those in the original cluster. You can check it with the `display` command.
- Before importing the cluster, run `tiup update --self && tiup update dm` to upgrade TiUP DM components to the latest version.
- After the cluster is imported, there is only one DM-master node in the cluster. You can refer to the `scale out` command to scale out the DM-master node.

Syntax

```
tiup dm import [flags]
```

Options

-v, –cluster-version

- Specifies the version number for redeploying. You must use a version later than v2.0.0-rc.2 (including v2.0.0-rc.2).
- Data type: `STRING`
- This option is **required** to execute the command.

-d, –dir

- Specifies the directory of TiDB Ansible.
- Data type: `STRING`
- If this option is not specified in the command, the current directory is the default directory.

–inventory

- Specifies the name of the Ansible inventory file.
- Data type: `STRING`
- If this option is not specified in the command, the default file name is `"inventory.ini"`.

–rename

- Renames the imported cluster.
- Data type: `STRING`
- If this option is not specified in the command, the default cluster name is the `cluster_name` specified in the inventory file.

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Outputs

The log of the importing process.

<< Back to the previous page - TiUP DM command list

13.4.7.4.11 tiup dm list

`tiup-dm` supports deploying multiple clusters using the same control machine. You can use the `tiup dm list` command to check which clusters are deployed using the control machine by the currently logged-in user.

> **Note:**
>
> By default, the data of the deployed clusters is stored in the `~/.tiup/storage` ↪ `/dm/clusters/` directory. The currently logged-in user cannot view the clusters deployed by other users on the same control machine.

Syntax

```
tiup dm list [flags]
```

Options

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

A table consisting of the following fields:

- `Name`: the cluster name.
- `User`: the user who deployed the cluster.

- `Version`: the cluster version.
- `Path`: the path of the cluster deployment data on the control machine.
- `PrivateKey`: the path of the private key to the cluster.

<< Back to the previous page - TiUP DM command list

13.4.7.4.12 Apply Hotfix to DM Clusters Online

If you need to dynamically replace the binaries of a service while the cluster is running (that is, to keep the cluster available during the replacement), you can use the `tiup dm patch` command. The command does the following:

- Uploads the binary package for replacement to the target machine.
- Takes the related nodes offline using the API.
- Stops the target service.
- Unpacks the binary package and replaces the service.
- Starts the target service.

Syntax

```
tiup dm patch <cluster-name> <package-path> [flags]
```

- `<cluster-name>`: The name of the cluster to be operated
- `<package-path>`: The path to the binary package used for replacement

Preparation

You need to pack the binary package required for this command in advance according to the following steps:

- Determine the name `${component}` of the component to be replaced (dm-master, dm-worker ...), the `${version}` of the component (v2.0.0, v2.0.1 ...), and the operating system `${os}` and platform `${arch}` on which the component runs.
- Download the current component package using the command `wget https://tiup-mirrors.pingcap.com/${component}-${version}-${os}-${arch}.tar.gz -O /tmp/${component}-${version}-${os}-${arch}.tar.gz`.
- Run `mkdir -p /tmp/package && cd /tmp/package` to create a temporary directory to pack files.
- Run `tar xf /tmp/${component}-${version}-${os}-${arch}.tar.gz` to unpack the original binary package.
- Run `find .` to view the file structure in the temporary package directory.
- Copy the binary files or configuration files to the corresponding locations in the temporary directory.

- Run `tar czf /tmp/${component}-hotfix-${os}-${arch}.tar.gz *` to pack the files in the temporary directory.
- Finally, you can use `/tmp/${component}-hotfix-${os}-${arch}.tar.gz` as the value of `<package-path>` in the `tiup dm patch` command.

Options

–overwrite

- After you patch a certain component (such as dm-worker), when the tiup-dm scales out the component, tiup-dm uses the original component version by default. To use the version that you patch when the cluster scales out in the future, you need to specify the option `--overwrite` in the command.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

-N, –node

- Specifies the nodes to be replaced. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `[tiup dm display](#tiup-dm-display)` command.
- Data type: `STRING`
- If this option is not specified, TiUP selects all nodes to replace by default.

> **Note:**
>
> If the option `-R, --role` is specified at the same time, TiUP then replaces service nodes that match both the requirements of `-N, --node` and `-R, --role`.

-R, –role

- Specifies the roles to be replaced. The value of this option is a comma-separated list of the roles of the nodes. You can get the roles of the nodes from the second column of the cluster status table returned by the `[tiup dm display](#tiup-dm-display)` command.
- Data type: `STRING`
- If this option is not specified, TiUP selects all roles to replace by default.

> **Note:**
>
> If the option `-N, --node` is specified at the same time, TiUP then replaces service nodes that match both the requirements of `-N, --node` and `-R, --role`.

–offline

- Declares that the current cluster is offline. When this option is specified, TiUP DM only replaces the binary files of the cluster components in place without restarting the service.

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Example

The following example shows how to apply `v5.3.0-hotfix` to the `v5.3.0` cluster deployed using TiUP. The operations might vary if you deploy the cluster using other methods.

> **Note:**
>
> Hotfix is used only for emergency fixes. Its daily maintenance is complicated. It is recommend that you upgrade the DM cluster to an official version as soon as it is released.

Preparations

Before applying a hotfix, prepare the hotfix package `dm-linux-amd64.tar.gz` and confirm the current DM software version:

```
/home/tidb/dm/deploy/dm-master-8261/bin/dm-master/dm-master -V
```

Output:

```
Release Version: v5.3.0

Git Commit Hash: 20626babf21fc381d4364646c40dd84598533d66
Git Branch: heads/refs/tags/v5.3.0
UTC Build Time: 2021-11-29 08:29:49
Go Version: go version go1.16.4 linux/amd64
```

Prepare the patch package and apply it to the DM cluster

1. Prepare the DM software package that matches the current version:

```
mkdir -p /tmp/package
tar -zxvf /root/.tiup/storage/dm/packages/dm-master-v5.3.0-linux-amd64.
    ↪ tar.gz -C /tmp/package/
tar -zxvf /root/.tiup/storage/dm/packages/dm-worker-v5.3.0-linux-amd64.
    ↪ tar.gz -C /tmp/package/
```

2. Replace the binary file with the hotfix package:

```
# Decompress the hotfix package and use it to replace the binary file.
cd /root; tar -zxvf dm-linux-amd64.tar.gz
cp /root/dm-linux-amd64/bin/dm-master /tmp/package/dm-master/dm-master
cp /root/dm-linux-amd64/bin/dm-worker /tmp/package/dm-worker/dm-worker
# Re-package the modified files.
# Note that the packaging method might be different for other
    ↪ deployment methods.
cd /tmp/package/ && tar -czvf dm-master-hotfix-linux-amd64.tar.gz dm-
    ↪ master/
cd /tmp/package/ && tar -czvf dm-worker-hotfix-linux-amd64.tar.gz dm-
    ↪ worker/
```

3. Apply the hotfix:

   Query the cluster status. The following uses the cluster named `dm-test` as an example:

```
tiup dm display dm-test
```

   Output:

```
Cluster type:     dm
Cluster name:     dm-test
Cluster version:  v5.3.0
Deploy user:      tidb
SSH type:         builtin
ID                Role              Host         Ports     OS/Arch
    ↪ Status    Data Dir                        Deploy Dir
```

```
--                ----              ----         -----     -------
   ↪ ------    --------                          ----------
172.16.100.21:9093 alertmanager     172.16.100.21 9093/9094 linux/
   ↪ x86_64 Up      /home/tidb/dm/data/alertmanager-9093 /home/tidb/dm
   ↪ /deploy/alertmanager-9093
172.16.100.21:8261 dm-master        172.16.100.21 8261/8291 linux/
   ↪ x86_64 Healthy|L /home/tidb/dm/data/dm-master-8261 /home/tidb/dm/
   ↪ deploy/dm-master-8261
172.16.100.21:8262 dm-worker        172.16.100.21 8262     linux/x86_64
   ↪  Free     /home/tidb/dm/data/dm-worker-8262 /home/tidb/dm/deploy/
   ↪ dm-worker-8262
172.16.100.21:3000 grafana          172.16.100.21 3000     linux/x86_64
   ↪  Up       -                                 /home/tidb/dm/deploy/
   ↪ grafana-3000
172.16.100.21:9090 prometheus       172.16.100.21 9090     linux/x86_64
   ↪  Up       /home/tidb/dm/data/prometheus-9090 /home/tidb/dm/deploy
   ↪ /prometheus-9090
Total nodes: 5
```

Apply the hotfix to the specified node or specified role. If both `-R` and `-N` are specified, the intersection will be taken.

```
# Apply hotfix to a specified node.
tiup dm patch dm-test dm-master-hotfix-linux-amd64.tar.gz -N
   ↪ 172.16.100.21:8261
tiup dm patch dm-test dm-worker-hotfix-linux-amd64.tar.gz -N
   ↪ 172.16.100.21:8262
# Apply hotfix to a specified role.
tiup dm patch dm-test dm-master-hotfix-linux-amd64.tar.gz -R dm-master
tiup dm patch dm-test dm-worker-hotfix-linux-amd64.tar.gz -R dm-worker
```

4. Query the hotfix application result:

```
/home/tidb/dm/deploy/dm-master-8261/bin/dm-master/dm-master -V
```

Output:

```
Release Version: v5.3.0-20211230
Git Commit Hash: ca7070c45013c24d34bd9c1e936071253451d707
Git Branch: heads/refs/tags/v5.3.0-20211230
UTC Build Time: 2022-01-05 14:19:02
Go Version: go version go1.16.4 linux/amd64
```

The cluster information changes accordingly:

```
tiup dm display dm-test
```

Output:

```
Starting component `dm`: /root/.tiup/components/dm/v1.8.1/tiup-dm
   ↪ display dm-test
Cluster type:     dm
Cluster name:     dm-test
Cluster version:  v5.3.0
Deploy user:      tidb
SSH type:         builtin
ID                Role               Host          Ports     OS/Arch
   ↪ Status    Data Dir                          Deploy Dir
--                ----               ----          -----     -------
   ↪ ------    --------                          ----------
172.16.100.21:9093 alertmanager     172.16.100.21 9093/9094 linux/
   ↪ x86_64 Up      /home/tidb/dm/data/alertmanager-9093 /home/tidb/dm
   ↪ /deploy/alertmanager-9093
172.16.100.21:8261 dm-master (patched) 172.16.100.21 8261/8291 linux/
   ↪ x86_64 Healthy|L /home/tidb/dm/data/dm-master-8261 /home/tidb/dm/
   ↪ deploy/dm-master-8261
172.16.100.21:8262 dm-worker (patched) 172.16.100.21 8262  linux/x86_64
   ↪  Free     /home/tidb/dm/data/dm-worker-8262 /home/tidb/dm/deploy/
   ↪ dm-worker-8262
172.16.100.21:3000 grafana          172.16.100.21 3000     linux/x86_64
   ↪  Up       -                                 /home/tidb/dm/deploy/
   ↪ grafana-3000
172.16.100.21:9090 prometheus       172.16.100.21 9090     linux/x86_64
   ↪  Up       /home/tidb/dm/data/prometheus-9090 /home/tidb/dm/deploy
   ↪ /prometheus-9090
Total nodes: 5
```

<< Back to the previous page - TiUP DM command list

13.4.7.4.13 tiup dm prune

When you scale in the cluster(/tiup/tiup-component-dm-scale-in.md), a small amount of metadata in etcd is not cleaned up, which usually causes no problem. If you need to clean up the metadata, you can manually execute the `tiup dm prune` command.

Syntax

```
tiup dm prune <cluster-name> [flags]
```

Option

-h, –help

- Prints the help information.

- Data type: `BOOLEAN`
- Default: false

Output

The log of the cleanup process.

<< Back to the previous page - TiUP DM command list

13.4.7.4.14 tiup dm reload

After modifying the cluster configuration, the cluster needs to be reloaded using the `tiup dm reload` command for the configuration to take effect. This command publishes the configuration of the control machine to the remote machine where the service is running and restarts the service in order according to the upgrade process. The cluster remains available during the restart process.

Syntax

```
tiup dm reload <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to operate on.

Options

-N, –node

- Specifies the nodes to be restarted. If not specified, all nodes are restarted. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all nodes are selected by default.

Note:

- If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are restarted.
- If the `--skip-restart` option is specified, the `-N, --node` option is invalid.

-R, –role

- Specifies the roles to be restarted. If not specified, all roles are restarted. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all roles are selected by default.

> **Note:**
>
> - If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are restarted.
> - If the `--skip-restart` option is specified, the `-R, --role` option is invalid.

–skip-restart

The `tiup dm reload` command performs two operations:

- Refreshes all node configurations
- Restarts the specified node

After you specify the `--skip-restart` option, it only refreshes the configuration without restarting any nodes, so that the refreshed configuration is not applied and does not take effect until the next restart of the corresponding service.

- Data type: `BOOLEAN`
- Default: false

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The execution log of the tiup-dm.

<< Back to the previous page - TiUP DM command list

13.4.7.4.15 tiup dm replay

When you perform a cluster operation such as upgrade or restart, the operation might fail due to cluster environment issues. If you re-perform the operation, you need to perform all the steps from the very beginning. If the cluster is large, re-performing these steps will take a long time. In this case, you can use the `tiup dm replay` command to retry the failed commands and skip the successfully performed steps.

Syntax

```
tiup dm replay <audit-id> [flags]
```

- `<audit-id>`: the `audit-id` of the command to be retried. You can view the historical commands and their `audit-id`s using the `tiup dm audit` command.

Option

-h, –help

Prints the help information.

Output

The output of the command corresponding to `<audit-id>`.

<< Back to the previous page - TiUP DM command list

13.4.7.4.16 tiup dm restart

The command `tiup dm restart` is used to restart all or some of the services of the specified cluster.

> **Note:**
>
> During the restart process, the related services are unavailable for a period of time.

Syntax

```
tiup dm restart <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the cluster list command.

Options

-N, –node

- Specifies the nodes to be restarted. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `[tiup dm display](#tiup-dm-display)` command.
- Data type: `STRING`
- If this option is not specified, TiUP restarts all nodes by default.

> **Note:**
>
> If the option `-R, --role` is specified at the same time, TiUP restarts service nodes that match both the requirements of `-N, --node` and `-R, --role`.

-R, –role

- Specifies the roles of nodes to be restarted. The value of this option is a comma-separated list of the roles of the nodes. You can get the roles of the nodes from the second column of the cluster status table returned by the `[tiup dm display](#tiup-dm-display)` command.
- Data type: `STRING`
- If this option is not specified, TiUP restarts nodes of all roles by default.

> **Note:**
>
> If the option `-N, --node` is specified at the same time, TiUP restarts service nodes that match both the requirements of `-N, --node` and `-R, --role`.

-h, –help

- Prints help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Outputs

The log of the service restart process.

<< Back to the previous page - TiUP DM command list

13.4.7.4.17 tiup dm scale-in

The `tiup dm scale-in` command is used to scale in the cluster. Scaling in the cluster means taking the service offline, which eventually removes the specified node from the cluster and deletes the remaining related files.

Syntax

```
tiup dm scale-in <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the cluster list command.

Options

-N, –node

- Specifies the nodes to be scaled in. If you need to scale in multiple nodes, split them by commas.
- Data type: `STRINGS`
- Default: no. This option is mandatory and the value must be not null.

–force

- In some cases, some scale-in nodes in the cluster have been down, making it impossible to connect to the node through SSH for operation. At this time, you can use the `--force` option to remove these nodes from the cluster.
- Data type: `BOOLEAN`
- Default: false. If this option is not specified in the command, the specified nodes are not forcibly removed.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The log of scaling in.

<< Back to the previous page - TiUP DM command list

13.4.7.4.18 tiup dm scale-out

The `tiup dm scale-out` command is used for scaling out the cluster. The internal logic of scaling out the cluster is similar to the cluster deployment. The `tiup-dm` components first establish an SSH connection to the new node, create the necessary directories on the target node, then perform the deployment and start the service.

Syntax

```
tiup dm scale-out <cluster-name> <topology.yaml> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the cluster list command.

`<topology.yaml>`: the prepared topology file. This topology file should only contain the new nodes that are to be added to the current cluster.

Options

-u, –user

- Specifies the user name used to connect to the target machine. This user must have the secret-free sudo root permission on the target machine.
- Data type: `STRING`
- Default: the current user who executes the command.

-i, –identity_file

- Specifies the key file used to connect to the target machine.
- Data type: `STRING`
- If this option is not specified in the command, the `~/.ssh/id_rsa` file is used to connect to the target machine by default.

-p, –password

- Specifies the password used to connect to the target machine. Do not use this option and `-i/--identity_file` at the same time.
- Data type: `BOOLEAN`
- Default: false

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The log of scaling out.

<< Back to the previous page - TiUP DM command list

13.4.7.4.19 tiup dm start

The `tiup dm start` command is used to start all or part of the services of the specified cluster.

Syntax

```
tiup dm start <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the cluster list command.

Options

-N, –node

- Specifies the nodes to be started. If not specified, all nodes are started. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all nodes are started.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are started.

-R, –role

- Specifies the roles to be started. If not specified, all roles are started. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all roles are started.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are started.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The log of starting the service.

<< Back to the previous page - TiUP DM command list

13.4.7.4.20 tiup dm stop

The `tiup dm stop` command is used to stop all or part of the services of the specified cluster.

> **Note:**
>
> The cluster cannot provide services after the core service is stopped.

Syntax

```
tiup dm stop <cluster-name> [flags]
```

`<cluster-name>`: the name of the cluster to operate on. If you forget the cluster name, you can check it with the cluster list command.

Options

-N, –node

- Specifies the nodes to be stopped. If not specified, all nodes are stopped. The value of this option is a comma-separated list of node IDs. You can get the node IDs from the first column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all nodes are selected by default.

> **Note:**
>
> If the `-R, --role` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are stopped.

-R, –role

- Specifies the roles to be stopped. If not specified, all roles are stopped. The value of this option is a comma-separated list of node roles. You can get the roles of nodes from the second column of the cluster status table returned by the `tiup dm display` command.
- Data type: `STRINGS`
- If this option is not specified in the command, all roles are selected by default.

> **Note:**
>
> If the `-N, --node` option is specified at the same time, only the service nodes that match both the specifications of `-N, --node` and `-R, --role` are stopped.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- Default: false

Output

The log of stopping the service.

<< Back to the previous page - TiUP DM command list

13.4.7.4.21 tiup dm template

Before deploying the cluster, you need to prepare a topology file of the cluster. TiUP has a built-in topology file template, and you can modify this template to create the final topology file. To output the built-in template content, you can use the `tiup dm template` command.

Syntax

```
tiup dm template [flags]
```

If this option is not specified, the output default template contains the following instances:

- 3 DM-master instances
- 3 DM-worker instances

- 1 Prometheus instance
- 1 Grafana instance
- 1 Alertmanager instance

Options

–full

- Outputs a detailed topology template that is commented with configurable parameters. To enable this option, add it to the command.
- If this option is not specified, the simple topology template is output by default.

-h, –help

Prints the help information.

Output

Outputs the topology template according to the specified options, which can be redirected to the topology file for deployment.

<< Back to the previous page - TiUP DM command list

13.4.7.4.22 tiup dm upgrade

The `tiup dm upgrade` command is used to upgrade a specified cluster to a specific version.

Syntax

```
tiup dm upgrade <cluster-name> <version> [flags]
```

- `<cluster-name>` is the name of the cluster to be operated on. If you forget the cluster name, you can check it using the `tiup dm list` command.
- `<version>` is the target version to be upgraded to. Currently, only upgrading to a later version is allowed, and upgrading to an earlier version is not allowed, which means the downgrade is not allowed. Upgrading to a nightly version is not allowed either.

Options

–offline

- Declares that the current cluster is offline. When this option is specified, TiUP DM only replaces the binary files of the cluster components in place without restarting the service.

-h, –help

- Prints the help information.
- Data type: `BOOLEAN`
- This option is disabled by default with the `false` value. To enable this option, add this option to the command, and either pass the `true` value or do not pass any value.

Output

Log of the service upgrade process.

<< Back to the previous page - TiUP DM command list

13.4.8 Topology Configuration File for TiDB Deployment Using TiUP

To deploy or scale TiDB using TiUP, you need to provide a topology file (sample) to describe the cluster topology.

Similarly, to modify the cluster topology, you need to modify the topology file. The difference is that, after the cluster is deployed, you can only modify a part of the fields in the topology file. This document introduces each section of the topology file and each field in each section.

13.4.8.1 File structure

A topology configuration file for TiDB deployment using TiUP might contain the following sections:

- global: The cluster's global configuration. Some of the configuration items use the default values and you can configure them separately in each instance.
- monitored: Configuration for monitoring services, namely, the blackbox_exporter and the `node_exporter`. On each machine, a `node_exporter` and a `blackbox_exporter` are deployed.
- server_configs: Components' global configuration. You can configure each component separately. If an instance has a configuration item with the same name, the instance's configuration item will take effect.
- pd_servers: The configuration of the PD instance. This configuration specifies the machines to which the PD component is deployed.
- tidb_servers: The configuration of the TiDB instance. This configuration specifies the machines to which the TiDB component is deployed.
- tikv_servers: The configuration of the TiKV instance. This configuration specifies the machines to which the TiKV component is deployed.
- tiflash_servers: The configuration of the TiFlash instance. This configuration specifies the machines to which the TiFlash component is deployed.
- pump_servers: The configuration of the Pump instance. This configuration specifies the machines to which the Pump component is deployed.
- drainer_servers: The configuration of the Drainer instance. This configuration specifies the machines to which the Drainer component is deployed.

- cdc_servers: The configuration of the TiCDC instance. This configuration specifies the machines to which the TiCDC component is deployed.
- tispark_masters: The configuration of the TiSpark master instance. This configuration specifies the machines to which the TiSpark master component is deployed. Only one node of TiSpark master can be deployed.
- tispark_workers: The configuration of the TiSpark worker instance. This configuration specifies the machines to which the TiSpark worker component is deployed.
- monitoring_servers: Specifies the machines to which Prometheus and NGMonitoring are deployed. TiUP supports deploying multiple Prometheus instances but only the first instance is used.
- grafana_servers: The configuration of the Grafana instance. This configuration specifies the machines to which Grafana is deployed.
- alertmanager_servers: The configuration of the Alertmanager instance. This configuration specifies the machines to which Alertmanager is deployed.

13.4.8.1.1 `global`

The `global` section corresponds to the cluster's global configuration and has the following fields:

- `user`: The user used to start the deployed cluster. The default value is `"tidb"`. If the user specified in the `<user>` field does not exist on the target machine, this user is automatically created.
- `group`: The user group to which a user belongs. It is specified when the user is created. The value defaults to that of the `<user>` field. If the specified group does not exist, it is automatically created.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. The default value is 22.
- `enable_tls`: Specifies whether to enable TLS for the cluster. After TLS is enabled, the generated TLS certificate must be used for connections between components or between the client and the component. The default value is `false`.
- `deploy_dir`: The deployment directory of each component. The default value is `"deployed"`. Its application rules are as follows:
  - If the absolute path of `deploy_dir` is configured at the instance level, the actual deployment directory is `deploy_dir` configured for the instance.
  - For each instance, if you do not configure `deploy_dir`, its default value is the relative path `<component-name>-<component-port>`.
  - If `global.deploy_dir` is an absolute path, the component is deployed to the `<global.deploy_dir>/<instance.deploy_dir>` directory.

    - If `global.deploy_dir` is a relative path, the component is deployed to the `/home/<global.user>/<global.deploy_dir>/<instance.deploy_dir>` directory.

- `data_dir`: The data directory. Default value: `"data"`. Its application rules are as follows:

    - If the absolute path of `data_dir` is configured at the instance level, the actual deployment directory is `data_dir` configured for the instance.
    - For each instance, if you do not configure `data_dir`, its default value is `<global.data_dir>`.
    - If `data_dir` is a relative path, the component data is placed in `<deploy_dir>/<data_dir>`. For the calculation rules of `<deploy_dir>`, see the application rules of the `deploy_dir` field.

- `log_dir`: The log directory. Default value: `"log"`. Its application rules are as follows:

    - If the absolute path `log_dir` is configured at the instance level, the actual log directory is the `log_dir` configured for the instance.
    - For each instance, if you not configure `log_dir`, its default value is `<global.log_dir>`.
    - If `log_dir` is a relative path, the component log is placed in `<deploy_dir>/<log_dir>`. For the calculation rules of `<deploy_dir>`, see the application rules of the `deploy_dir` field.

- `os`: The operating system of the target machine. The field controls which operating system to adapt to for the components pushed to the target machine. The default value is "linux".

- `arch`: The CPU architecture of the target machine. The field controls which platform to adapt to for the binary packages pushed to the target machine. The supported values are "amd64" and "arm64". The default value is "amd64".

- `resource_control`: Runtime resource control. All configurations in this field are written into the service file of systemd. There is no limit by default. The resources that can be controlled are as follows:

    - `memory_limit`: Limits the maximum runtime memory. For example, "2G" means that the maximum memory of 2 GB can be used.
    - `cpu_quota`: Limits the maximum CPU usage at runtime. For example, "200%".
    - `io_read_bandwidth_max`: Limits the maximum I/O bandwidth for disk reads. For example, `"/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0 100M"`.
    - `io_write_bandwidth_max`: Limits maximum I/O bandwidth for disk writes. For example, `/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0 100M`.

- `limit_core`: Controls the size of core dump.

A `global` configuration example is as follows:

```
global:
  user: "tidb"
  resource_control:
    memory_limit: "2G"
```

In the above configuration, the `tidb` user is used to start the cluster. At the same time, each component is restricted to a maximum of 2 GB of memory when it is running.

13.4.8.1.2 `monitored`

`monitored` is used to configure the monitoring service on the target machine: `node_exporter` and `blackbox_exporter`. The following fields are included:

- `node_exporter_port`: The service port of `node_exporter`. The default value is `9100`.
- `blackbox_exporter_port`: The service port of `blackbox_exporter`. The default value is `9115`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.

A `monitored` configuration example is as follows:

```
monitored:
  node_exporter_port: 9100
  blackbox_exporter_port: 9115
```

The above configuration specifies that `node_exporter` uses the `9100` port and `blackbox_exporter` uses the `9115` port.

13.4.8.1.3 `server_configs`

`server_configs` is used to configure services and to generate configuration files for each component. Similar to the `global` section, the configuration of this section can be overwritten by the configurations with the same names in an instance. `server_configs` mainly includes the following fields:

- `tidb`: TiDB service-related configuration. For the complete configuration, see TiDB configuration file.
- `tikv`: TiKV service-related configuration. For the complete configuration, see TiKV configuration file.
- `pd`: PD service-related configuration. For the complete configuration, see PD configuration file.
- `tiflash`: TiFlash service-related configuration. For the complete configuration, see TiFlash configuration file.
- `tiflash_learner`: Each TiFlash node has a special built-in TiKV. This configuration item is used to configure this special TiKV. It is generally not recommended to modify the content under this configuration item.
- `pump`: Pump service-related configuration. For the complete configuration, see TiDB Binlog configuration file.
- `drainer`: Drainer service-related configuration. For the complete configuration, see TiDB Binlog configuration file.
- `cdc`: TiCDC service-related configuration. For the complete configuration, see Deploy TiCDC.

A `server_configs` configuration example is as follows:

```
server_configs:
  tidb:
    run-ddl: true
    lease: "45s"
    split-table: true
    token-limit: 1000
  tikv:
    log-level: "info"
    readpool.unified.min-thread-count: 1
```

The above configuration specifies the global configuration of TiDB and TiKV.

13.4.8.1.4 `pd_servers`

`pd_servers` specifies the machines to which PD services are deployed. It also specifies the service configuration on each machine. `pd_servers` is an array, and each element of the array contains the following fields:

- `host`: Specifies the machine to which the PD services are deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is `0.0.0.0`.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `name`: Specifies the name of the PD instance. Different instances must have unique names; otherwise, instances cannot be deployed.
- `client_port`: Specifies the port that PD uses to connect to the client. The default value is 2379.
- `peer_port`: Specifies the port for communication between PDs. The default value is 2380.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: The configuration rule of this field is the same as the `pd` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `pd` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.

- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `name`
- `client_port`
- `peer_port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `pd_servers` configuration example is as follows:

```
pd_servers:
  - host: 10.0.1.11
    config:
      schedule.max-merge-region-size: 20
      schedule.max-merge-region-keys: 200000
  - host: 10.0.1.12
```

The above configuration specifies that PD will be deployed on `10.0.1.11` and `10.0.1.12`, and makes specific configurations for the PD of `10.0.1.11`.

13.4.8.1.5 `tidb_servers`

`tidb_servers` specifies the machines to which TiDB services are deployed. It also specifies the service configuration on each machine. `tidb_servers` is an array, and each element of the array contains the following fields:

- `host`: Specifies the machine to which the TiDB services are deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is `0.0.0.0`.

- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of TiDB services, which is used to provide connection to the MySQL client. The default value is `4000`.
- `status_port`: The listening port of the TiDB status service, which is used to view the status of the TiDB services from the external via HTTP requests. The default value is `10080`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: The configuration rule of this field is the same as the `tidb` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `tidb` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `port`
- `status_port`
- `deploy_dir`

- log_dir
- arch
- os

A tidb_servers configuration example is as follows:

```
tidb_servers:
  - host: 10.0.1.14
    config:
      log.level: warn
      log.slow-query-file: tidb-slow-overwrited.log
  - host: 10.0.1.15
```

13.4.8.1.6 tikv_servers

tikv_servers specifies the machines to which TiKV services are deployed. It also specifies the service configuration on each machine. tikv_servers is an array, and each element of the array contains the following fields:

- host: Specifies the machine to which the TiKV services are deployed. The field value is an IP address and is mandatory.
- listen_host: When the machine has multiple IP addresses, listen_host specifies the listening IP address of the service. The default value is 0.0.0.0.
- ssh_port: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the ssh_port of the global section is used.
- port: The listening port of the TiKV services. The default value is 20160.
- status_port: The listening port of the TiKV status service. The default value is 20180.
- deploy_dir: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the deploy_dir directory configured in global.
- data_dir: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the data_dir directory configured in global.
- log_dir: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the log_dir directory configured in global.
- numa_node: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".

- `config`: The configuration rule of this field is the same as the `tikv` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `tikv` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `port`
- `status_port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `tikv_servers` configuration example is as follows:

```
tikv_servers:
  - host: 10.0.1.14
    config:
      server.labels: { zone: "zone1", host: "host1" }
  - host: 10.0.1.15
    config:
      server.labels: { zone: "zone1", host: "host2" }
```

13.4.8.1.7 `tiflash_servers`

`tiflash_servers` specifies the machines to which TiFlash services are deployed. It also specifies the service configuration on each machine. This section is an array, and each element of the array contains the following fields:

- `host`: Specifies the machine to which the TiFlash services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `tcp_port`: The port of the TiFlash TCP service. The default value is `9000`.
- `http_port`: The port of the TiFlash HTTP service. The default value is `8123`.
- `flash_service_port`: The port via which TiFlash provides services. TiDB reads data from TiFlash via this port. The default value is `3930`.
- `metrics_port`: TiFlash's status port, which is used to output metric data. The default value is `8234`.
- `flash_proxy_port`: The port of the built-in TiKV. The default value is `20170`.
- `flash_proxy_status_port`: The status port of the built-in TiKV. The default value is `20292`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`. TiFlash supports multiple `data_dir` directories separated by commas.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `tmp_path`: The storage path of TiFlash temporary files. The default value is [`path` or the first directory of `storage.latest.dir`] + "/tmp".
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: The configuration rule of this field is the same as the `tiflash` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `tiflash` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- `learner_config`: Each TiFlash node has a special built-in TiKV. This configuration item is used to configure this special TiKV. It is generally not recommended to modify the content under this configuration item.

- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

After the deployment, for the fields above, you can only add directories to `data_dir`; for the fields below, you cannot modified these fields:

- `host`
- `tcp_port`
- `http_port`
- `flash_service_port`
- `flash_proxy_port`
- `flash_proxy_status_port`
- `metrics_port`
- `deploy_dir`
- `log_dir`
- `tmp_path`
- `arch`
- `os`

A `tiflash_servers` configuration example is as follows:

```
tiflash_servers:
  - host: 10.0.1.21
  - host: 10.0.1.22
```

13.4.8.1.8 `pump_servers`

`pump_servers` specifies the machines to which the Pump services of TiDB Binlog are deployed. It also specifies the service configuration on each machine. `pump_servers` is an array, and each element of the array contains the following fields:

- `host`: Specifies the machine to which the Pump services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.

- `port`: The listening port of the Pump services. The default value is `8250`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: The configuration rule of this field is the same as the `pump` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `pump` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `pump_servers` configuration example is as follows:

```
pump_servers:
  - host: 10.0.1.21
    config:
      gc: 7
  - host: 10.0.1.22
```

13.4.8.1.9 `drainer_servers`

`drainer_servers` specifies the machines to which the Drainer services of TiDB Binlog are deployed. It also specifies the service configuration on each machine. `drainer_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the Drainer services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of Drainer services. The default value is `8249`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `commit_ts` (deprecated): When Drainer starts, it reads the checkpoint. If Drainer gets no checkpoint, it uses this field as the replication time point for the initial startup. This field defaults to `-1` (Drainer always gets the latest timestamp from the PD as the commit_ts).
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: The configuration rule of this field is the same as the `drainer` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `drainer` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.

- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

The `commit_ts` field is deprecated since TiUP v1.9.2 and is not recorded in the starting script of Drainer. If you still need to use this field, refer to the following example to configure the `initial-commit-ts` field in `config`.

A `drainer_servers` configuration example is as follows:

```
drainer_servers:
  - host: 10.0.1.21
    config:
      initial-commit-ts: -1
      syncer.db-type: "mysql"
      syncer.to.host: "127.0.0.1"
      syncer.to.user: "root"
      syncer.to.password: ""
      syncer.to.port: 3306
      syncer.ignore-table:
        - db-name: test
          tbl-name: log
        - db-name: test
          tbl-name: audit
```

13.4.8.1.10 `cdc_servers`

`cdc_servers` specifies the machines to which the TiCDC services are deployed. It also specifies the service configuration on each machine. `cdc_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the TiCDC services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of the TiCDC services. The default value is `8300`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `gc-ttl`: The Time To Live (TTL) duration of the service level GC safepoint set by TiCDC in PD, in seconds. The default value is `86400`, which is 24 hours.
- `tz`: The time zone that the TiCDC services use. TiCDC uses this time zone when internally converting time data types such as timestamp and when replicating data to the downstream. The default value is the local time zone where the process runs.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as “0,1”.
- `config`: The field content is merged with the `cdc` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.

- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `cdc_servers` configuration example is as follows:

```
cdc_servers:
  - host: 10.0.1.20
    gc-ttl: 86400
    data_dir: "/cdc-data"
  - host: 10.0.1.21
    gc-ttl: 86400
    data_dir: "/cdc-data"
```

13.4.8.1.11 `tispark_masters`

`tispark_masters` specifies the machines to which the master node of TiSpark is deployed. It also specifies the service configuration on each machine. `tispark_masters` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the TiSpark master is deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is `0.0.0.0`.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: Spark's listening port, used for communication before the node. The default value is 7077.
- `web_port`: Spark's web port, which provides web services and the task status. The default value is 8080.

- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `java_home`: Specifies the path of the JRE environment to be used. This parameter corresponds to the `JAVA_HOME` system environment variable.
- `spark_config`: Configures to configure the TiSpark services. Then, a configuration file is generated and sent to the machine specified in `host`.
- `spark_env`: Configures the environment variables when Spark starts.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `port`
- `web_port`
- `deploy_dir`
- `arch`
- `os`

A `tispark_masters` configuration example is as follows:

```
tispark_masters:
  - host: 10.0.1.21
    spark_config:
      spark.driver.memory: "2g"
      spark.eventLog.enabled: "False"
      spark.tispark.grpc.framesize: 2147483647
      spark.tispark.grpc.timeout_in_sec: 100
      spark.tispark.meta.reload_period_in_sec: 60
      spark.tispark.request.command.priority: "Low"
      spark.tispark.table.scan_concurrency: 256
    spark_env:
      SPARK_EXECUTOR_CORES: 5
      SPARK_EXECUTOR_MEMORY: "10g"
      SPARK_WORKER_CORES: 5
      SPARK_WORKER_MEMORY: "10g"
  - host: 10.0.1.22
```

13.4.8.1.12 `tispark_workers`

`tispark_workers` specifies the machines to which the worker nodes of TiSpark are deployed. It also specifies the service configuration on each machine. `tispark_workers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the TiSpark workers are deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is `0.0.0.0`.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: Spark's listening port, used for communication before the node. The default value is `7077`.
- `web_port`: Spark's web port, which provides web services and the task status. The default value is `8080`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `java_home`: Specifies the path in which the JRE environment to be used is located. This parameter corresponds to the `JAVA_HOME` system environment variable.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `port`
- `web_port`
- `deploy_dir`
- `arch`
- `os`

A `tispark_workers` configuration example is as follows:

```
tispark_workers:
  - host: 10.0.1.22
  - host: 10.0.1.23
```

13.4.8.1.13 monitoring_servers

monitoring_servers specifies the machines to which the Prometheus services are deployed. It also specifies the service configuration on each machine. monitoring_servers is an array. Each array element contains the following fields:

- host: Specifies the machine to which the monitoring services are deployed. The field value is an IP address and is mandatory.
- ng_port: Specifies the port that NgMonitoring listens to. Introduced in TiUP v1.7.0, this field supports Continuous Profiling and Top SQL. The default value is 12020.
- ssh_port: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the ssh_port of the global section is used.
- port: The listening port of the Prometheus services. The default value is 9090.
- deploy_dir: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the deploy_dir directory configured in global.
- data_dir: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the data_dir directory configured in global.
- log_dir: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the log_dir directory configured in global.
- numa_node: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- storage_retention: The retention time of the Prometheus monitoring data. The default value is "30d".
- rule_dir: Specifies a local directory that should contain complete *.rules.yml files. These files are transferred to the target machine during the initialization phase of the cluster configuration as the rules for Prometheus.
- remote_config: Supports writing Prometheus data to the remote, or reading data from the remote. This field has two configurations:
  - remote_write: See the Prometheus document <remote_write>.
  - remote_read: See the Prometheus document <remote_read>.
- external_alertmanagers: If the external_alertmanagers field is configured, Prometheus alerts the configuration behavior to the Alertmanager that is outside the cluster. This field is an array, each element of which is an external Alertmanager and consists of the host and web_port fields.

- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `monitoring_servers` configuration example is as follows:

```
monitoring_servers:
  - host: 10.0.1.11
    rule_dir: /local/rule/dir
    remote_config:
      remote_write:
      - queue_config:
          batch_send_deadline: 5m
          capacity: 100000
          max_samples_per_send: 10000
          max_shards: 300
        url: http://127.0.0.1:8003/write
      remote_read:
      - url: http://127.0.0.1:8003/read
      external_alertmanagers:
      - host: 10.1.1.1
        web_port: 9093
      - host: 10.1.1.2
        web_port: 9094
```

13.4.8.1.14 `grafana_servers`

`grafana_servers` specifies the machines to which the Grafana services are deployed. It also specifies the service configuration on each machine. `grafana_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the Grafana services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of the Grafana services. The default value is `3000`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `username`: The user name on the Grafana login interface.
- `password`: The password corresponding to Grafana.
- `dashboard_dir`: Specifies a local directory that should contain complete `dashboard (*.json)` files. These files are transferred to the target machine during the initialization phase of the cluster configuration as the dashboards for Grafana.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

> **Note:**
>
> If the `dashboard_dir` field of `grafana_servers` is configured, after executing the `tiup cluster rename` command to rename the cluster, you need to perform the following operations:
>
> 1. For the `*.json` files in the local dashboards directory, update the value of the `datasource` field to the new cluster name (because `datasource` is named after the cluster name).
> 2. Execute the `tiup cluster reload -R grafana` command.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `arch`
- `os`

A `grafana_servers` configuration example is as follows:

```
grafana_servers:
  - host: 10.0.1.11
    dashboard_dir: /local/dashboard/dir
```

13.4.8.1.15 `alertmanager_servers`

`alertmanager_servers` specifies the machines to which the Alertmanager services are deployed. It also specifies the service configuration on each machine. `alertmanager_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the Alertmanager services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `web_port`: Specifies the port used that Alertmanager uses to provide web services. The default value is `9093`.
- `cluster_port`: Specifies the communication port between one Alertmanger and other Alertmanager. The default value is `9094`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is the string type. The field value is the ID of the NUMA node, such as “0,1”.

- `config_file`: Specifies a local file that is transferred to the target machine during the initialization phase of the cluster configuration as the configuration of Alertmanager.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `web_port`
- `cluster_port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `alertmanager_servers` configuration example is as follows:

```
alertmanager_servers:
  - host: 10.0.1.11
    config_file: /local/config/file
  - host: 10.0.1.12
    config_file: /local/config/file
```

13.4.9 Topology Configuration File for DM Cluster Deployment Using TiUP

To deploy or scale a TiDB Data Migration (DM) cluster, you need to provide a topology file (sample) to describe the cluster topology.

Similarly, to modify the cluster topology, you need to modify the topology file. The difference is that, after the cluster is deployed, you can only modify a part of the fields in the topology file. This document introduces each section of the topology file and each field in each section.

13.4.9.1 File structure

A topology configuration file for DM cluster deployment using TiUP might contain the following sections:

- global: the cluster's global configuration. Some of the configuration items use the default values of the cluster, and you can configure them separately in each instance.
- server_configs: the components' global configuration. You can configure each component separately. If an instance has a configuration item with the same key, the instance's configuration item will take effect.
- master_servers: the configuration of the DM-master instance. The configuration specifies the machines to which the master service of the DM component is deployed.
- worker_servers: the configuration of the DM-worker instance. The configuration specifies the machines to which the worker service of the DM component is deployed.
- monitoring_servers: specifies the machines to which the Prometheus instances are deployed. TiUP supports deploying multiple Prometheus instances but only the first instance is used.
- grafana_servers: the configuration of the Grafana instances. The configuration specifies the machines to which the Grafana instances are deployed.
- alertmanager_servers: the configuration of the Alertemanager instances. The configuration specifies the machines to which the Alertmanager instances are deployed.

13.4.9.1.1 `global`

The `global` section corresponds to the cluster's global configuration and has the following fields:

- `user`: the user to start the deployed cluster. The default value is "tidb". If the user specified in the `<user>` field does not exist on the target machine, TiUP will automatically try to create the user.
- `group`: the user group to which a user belongs when the user is automatically created. The default value is the same as the `<user>` field. If the specified group does not exist, it will be created automatically.
- `ssh_port`: the SSH port to connect to the target machine for operations. The default value is "22".
- `deploy_dir`: the deployment directory for each component. The default value is "deploy". The construction rules are as follows:
  - If the absolute path `deploy_dir` is configured at the instance level, the actual deployment directory is the `deploy_dir` configured for the instance.
  - For each instance, if you do not configure `deploy_dir`, the default value is the relative path `<component-name>-<component-port>`.
  - If `global.deploy_dir` is set to an absolute path, the component is deployed to the `<global.deploy_dir>/<instance.deploy_dir>` directory.

    - If `global.deploy_dir` is set to a relative path, the component is deployed to the `/home/<global.user>/<global.deploy_dir>/<instance.deploy_dir>` directory.

- `data_dir`: the data directory. The default value is "data". The construction rules are as follows.

    - If the absolute path `data_dir` is configured at the instance level, the actual data directory is the `data_dir` configured for the instance.
    - For each instance, if `data_dir` is not configured, the default value is `<global.data_dir>`.
    - If `data_dir` is set to a relative path, the component data is stored in `<deploy_dir>/<data_dir>`. For the construction rules of `<deploy_dir>`, see the construction rules of the `deploy_dir` field.

- `log_dir`: the data directory. The default value is "log". The construction rules are as follows.

    - If the absolute path of `log_dir` is configured at the instance level, the actual log directory is the `log_dir` configured for the instance.
    - For each instance, if `log_dir` is not configured by the user, the default value is `<global.log_dir>`.
    - If `log_dir` is a relative path, the component logs will be stored in `<deploy_dir>/<log_dir>`. For the construction rules of `<deploy_dir>`, see the construction rules of the `deploy_dir` field.

- `os`: the operating system of the target machine. The field controls which operating system to adapt to for the components pushed to the target machine. The default value is "linux".
- `arch`: the CPU architecture of the target machine. The field controls which platform to adapt to for the binary packages pushed to the target machine. The supported values are "amd64" and "arm64". The default value is "amd64".
- `resource_control`: runtime resource control. All configurations in this field are written to the service file of systemd. There is no limit by default. The resources that can be controlled are as follows:

    - `memory_limit`: limits the maximum memory at runtime. For example, "2G" means that the maximum memory of 2 GB can be used.
    - `cpu_quota`: limits the maximum CPU usage at runtime. For example, "200%".
    - `io_read_bandwidth_max`: limits the maximum I/O bandwidth for disk reads. For example, `"/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0:0 100M"`.
    - `io_write_bandwidth_max`: limits the maximum I/O bandwidth for disk writes. For example, `"/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0:0 100M"`.
    - `limit_core`: controls the size of core dump.

A `global` configuration example:

```
global:
  user: "tidb"
  resource_control:
    memory_limit: "2G"
```

In the example, the configuration specifies that the `tidb` user is used to start the cluster, and that each component is limited to a maximum of 2 GB of memory when it is running.

13.4.9.1.2 `server_configs`

`server_configs` is used to configure services and to generate configuration files for each component. Similar to the `global` section, the configurations in the `server_configs` section can be overwritten by the configurations with the same keys in an instance. `server_configs` mainly contains the following fields:

- `master`: the configuration related to the DM-master service. For all the supported configuration items, see DM-master Configuration File.
- `worker`: the configuration related to the DM-worker service, For all the supported configuration items, see DM-worker Configuration File.

A `server_configs` configuration example is as follows:

```
server_configs:
  master:
    log-level: info
    rpc-timeout: "30s"
    rpc-rate-limit: 10.0
    rpc-rate-burst: 40
  worker:
    log-level: info
```

13.4.9.2 `master_servers`

`master_servers` specifies the machines to which the master node of the DM component is deployed. You can also specify the service configuration on each machine. `master_servers` is an array. Each array element contains the following fields:

- `host`: specifies the machine to deploy to. The field value is an IP address and is mandatory.
- `ssh_port`: specifies the SSH port to connect to the target machine for operations. If the field is not specified, the `ssh_port` in the `global` section is used.
- `name`: specifies the name of the DM-master instance. The name must be unique for different instances. Otherwise, the cluster cannot be deployed.

- `port`: specifies the port on which DM-master provides services. The default value is "8261".
- `peer_port`: specifies the port for communication between DM-masters. The default value is "8291".
- `deploy_dir`: specifies the deployment directory. If the field is not specified, or specified as a relative directory, the deployment directory is generated according to the `deploy_dir` configuration in the `global` section.
- `data_dir`: specifies the data directory. If the field is not specified, or specified as a relative directory, the data directory is generated according to the `data_dir` configuration in the `global` section.
- `log_dir`: specifies the log directory. If the field is not specified, or specified as a relative directory, the log directory is generated according to the `log_dir` configuration in the `global` section.
- `numa_node`: allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is a string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: the configuration rules of this field are the same as that of `master` in the `server_configs` section. If `config` is specified, the configuration of `config` will be merged with the configuration of `master` in `server_configs` (if the two fields overlap, the configuration of this field takes effect), and then the configuration file is generated and distributed to the machine specified in the `host` field.
- `os`: the operating system of the machine specified in the `host` field. If the field is not specified, the default value is the `os` value configured in the `global` section.
- `arch`: the architecture of the machine specified in the `host` field. If the field is not specified, the default value is the `arch` value configured in the `global` section.
- `resource_control`: resource control on this service. If this field is specified, the configuration of this field will be merged with the configuration of `resource_control` in the `global` section (if the two fields overlap, the configuration of this field takes effect), and then the configuration file of systemd is generated and distributed to the machine specified in the `host` field. The configuration rules of this field are the same as that of `resource_control` in the `global` section.
- `v1_source_path`: when upgrading from v1.0.x, you can specify the directory where the configuration file of the V1 source is located in this field.

In the `master_servers` section, the following fields cannot be modified after the deployment is completed:

- `host`
- `name`
- `port`
- `peer_port`
- `deploy_dir`
- `data_dir`

- log_dir
- arch
- os
- v1_source_path

A master_servers configuration example is as follows:

```
master_servers:
  - host: 10.0.1.11
    name: master1
    ssh_port: 22
    port: 8261
    peer_port: 8291
    deploy_dir: "/dm-deploy/dm-master-8261"
    data_dir: "/dm-data/dm-master-8261"
    log_dir: "/dm-deploy/dm-master-8261/log"
    numa_node: "0,1"
    # The following configs are used to overwrite the `server_configs.master
       ↪ ` values.
    config:
      log-level: info
      rpc-timeout: "30s"
      rpc-rate-limit: 10.0
      rpc-rate-burst: 40
  - host: 10.0.1.18
    name: master2
  - host: 10.0.1.19
    name: master3
```

13.4.9.3 worker_servers

worker_servers specifies the machines to which the master node of the DM component is deployed. You can also specify the service configuration on each machine. worker_servers is an array. Each array element contains the following fields:

- host: specifies the machine to deploy to. The field value is an IP address and is mandatory.
- ssh_port: specifies the SSH port to connect to the target machine for operations. If the field is not specified, the ssh_port in the global section is used.
- name: specifies the name of the DM-worker instance. The name must be unique for different instances. Otherwise, the cluster cannot be deployed.
- port: specifies the port on which DM-worker provides services. The default value is "8262".

- `deploy_dir`: specifies the deployment directory. If the field is not specified, or specified as a relative directory, the deployment directory is generated according to the `deploy_dir` configuration in the `global` section.
- `data_dir`: specifies the data directory. If the field is not specified, or specified as a relative directory, the data directory is generated according to the `data_dir` configuration in the `global` section.
- `log_dir`: specifies the log directory. If the field is not specified, or specified as a relative directory, the log directory is generated according to the `log_dir` configuration in the `global` section.
- `numa_node`: allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is a string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: the configuration rules of this field are the same as that of `worker` in the `server_configs` section. If `config` is specified, the configuration of `config` will be merged with the configuration of `worker` in `server_configs` (if the two fields overlap, the configuration of this field takes effect), and then the configuration file is generated and distributed to the machine specified in the `host` field.
- `os`: the operating system of the machine specified in the `host` field. If the field is not specified, the default value is the `os` value configured in the `global` section.
- `arch`: the architecture of the machine specified in the `host` field. If the field is not specified, the default value is the `arch` value configured in the `global` section.
- `resource_control`: resource control on this service. If this field is specified, the configuration of this field will be merged with the configuration of `resource_control` in the `global` section (if the two fields overlap, the configuration of this field takes effect), and then the configuration file of systemd is generated and distributed to the machine specified in the `host` field. The configuration rules of this field are the same as that of `resource_control` in the `global` section.

In the `worker_servers` section, the following fields cannot be modified after the deployment is completed:

- `host`
- `name`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `worker_servers` configuration example is as follows:

```
worker_servers:
  - host: 10.0.1.12
```

```
    ssh_port: 22
    port: 8262
    deploy_dir: "/dm-deploy/dm-worker-8262"
    log_dir: "/dm-deploy/dm-worker-8262/log"
    numa_node: "0,1"
    # config is used to overwrite the `server_configs.worker` values
    config:
      log-level: info
  - host: 10.0.1.19
```

13.4.9.3.1 monitoring_servers

`monitoring_servers` specifies the machines to which the Prometheus service is deployed. You can also specify the service configuration on the machine. `monitoring_servers` is an array. Each array element contains the following fields:

- `host`: specifies the machine to deploy to. The field value is an IP address and is mandatory.
- `ssh_port`: specifies the SSH port to connect to the target machine for operations. If the field is not specified, the `ssh_port` in the `global` section is used.
- `port`: specifies the port on which Prometheus provides services. The default value is "9090".
- `deploy_dir`: specifies the deployment directory. If the field is not specified, or specified as a relative directory, the deployment directory is generated according to the `deploy_dir` configuration in the `global` section.
- `data_dir`: specifies the data directory. If the field is not specified, or specified as a relative directory, the data directory is generated according to the `data_dir` configuration in the `global` section.
- `log_dir`: specifies the log directory. If the field is not specified, or specified as a relative directory, the log directory is generated according to the `log_dir` configuration in the `global` section.
- `numa_node`: allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is a string type. The field value is the ID of the NUMA node, such as "0,1"
- `storage_retention`: specifies the retention time of the Prometheus monitoring data. The default value is "15d".
- `rule_dir`: specifies a local directory where the complete `*.rules.yml` files are located. The files in the specified directory will be sent to the target machine as the Prometheus rules during the initialization phase of the cluster configuration.
- `remote_config`: Supports writing Prometheus data to the remote, or reading data from the remote. This field has two configurations:
    - `remote_write`: See the Prometheus document `<remote_write>`.
    - `remote_read`: See the Prometheus document `<remote_read>`.

- `external_alertmanagers`: If the `external_alertmanagers` field is configured, Prometheus alerts the configuration behavior to the Alertmanager that is outside the cluster. This field is an array, each element of which is an external Alertmanager and consists of the `host` and `web_port` fields.
- `os`: the operating system of the machine specified in the `host` field. If the field is not specified, the default value is the `os` value configured in the `global` section.
- `arch`: the architecture of the machine specified in the `host` field. If the field is not specified, the default value is the `arch` value configured in the `global` section.
- `resource_control`: resource control on this service. If this field is specified, the configuration of this field will be merged with the configuration of `resource_control` in the `global` section (if the two fields overlap, the configuration of this field takes effect), and then the configuration file of systemd is generated and distributed to the machine specified in the `host` field. The configuration rules of this field are the same as that of `resource_control` in the `global` section.

In the `monitoring_servers` section, the following fields cannot be modified after the deployment is completed:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `monitoring_servers` configuration example is as follows:

```
monitoring_servers:
  - host: 10.0.1.11
    rule_dir: /local/rule/dir
    remote_config:
      remote_write:
      - queue_config:
          batch_send_deadline: 5m
          capacity: 100000
          max_samples_per_send: 10000
          max_shards: 300
        url: http://127.0.0.1:8003/write
      remote_read:
      - url: http://127.0.0.1:8003/read\
      external_alertmanagers:
      - host: 10.1.1.1
      web_port: 9093
      - host: 10.1.1.2
```

```
    web_port: 9094
```

13.4.9.3.2 grafana_servers

`grafana_servers` specifies the machines to which the Grafana service is deployed. You can also specify the service configuration on the machine. `grafana_servers` is an array. Each array element contains the following fields:

- `host`: specifies the machine to deploy to. The field value is an IP address and is mandatory.
- `ssh_port`: specifies the SSH port to connect to the target machine for operations. If the field is not specified, the `ssh_port` in the `global` section is used.
- `port`: specifies the port on which Grafana provides services. The default value is "3000".
- `deploy_dir`: specifies the deployment directory. If the field is not specified, or specified as a relative directory, the deployment directory is generated according to the `deploy_dir` configuration in the `global` section.
- `os`: the operating system of the machine specified in the `host` field. If the field is not specified, the default value is the `os` value configured in the `global` section.
- `arch`: the architecture of the machine specified in the `host` field. If the field is not specified, the default value is the `arch` value configured in the `global` section.
- `username`: specifies the username of the Grafana login screen.
- `password`: specifies the corresponding password of Grafana.
- `dashboard_dir`: specifies a local directory where the complete `dashboard(*.json)` files are located. The files in the specified directory will be sent to the target machine as Grafana dashboards during the initialization phase of the cluster configuration.
- `resource_control`: resource control on this service. If this field is specified, the configuration of this field will be merged with the configuration of `resource_control` in the `global` section (if the two fields overlap, the configuration of this field takes effect), and then the configuration file of systemd is generated and distributed to the machine specified in the `host` field. The configuration rules of this field are the same as that of `resource_control` in the `global` section.

Note:

If the `dashboard_dir` field of `grafana_servers` is configured, after executing the `tiup cluster rename` command to rename the cluster, you need to perform the following operations:

1. In the local `dashboards` directory, update the value of the `datasource` field to the new cluster name (the `datasource` is named after the cluster name).
2. Execute the `tiup cluster reload -R grafana` command.

In `grafana_servers`, the following fields cannot be modified after the deployment is completed:

- `host`
- `port`
- `deploy_dir`
- `arch`
- `os`

A `grafana_servers` configuration example is as follows:

```
grafana_servers:
  - host: 10.0.1.11
    dashboard_dir: /local/dashboard/dir
```

13.4.9.3.3 `alertmanager_servers`

`alertmanager_servers` specifies the machines to which the Alertmanager service is deployed. You can also specify the service configuration on each machine. `alertmanager_servers` is an array. Each array element contains the following fields:

- `host`: specifies the machine to deploy to. The field value is an IP address and is mandatory.
- `ssh_port`: specifies the SSH port to connect to the target machine for operations. If the field is not specified, the `ssh_port` in the `global` section is used.
- `web_port`: specify the port on which Alertmanager provides web services. The default value is "9093".
- `cluster_port`: Specify the communication port between one Alertmanger and other Alertmanager. The default value is "9094".
- `deploy_dir`: specifies the deployment directory. If the field is not specified, or specified as a relative directory, the deployment directory is generated according to the `deploy_dir` configuration in the `global` section.
- `data_dir`: specifies the data directory. If the field is not specified, or specified as a relative directory, the data directory is generated according to the `data_dir` configuration in the `global` section.
- `log_dir`: specifies the log directory. If the field is not specified, or specified as a relative directory, the log directory is generated according to the `log_dir` configuration in the `global` section.
- `numa_node`: allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has numactl installed. If this field is specified, cpubind and membind policies are allocated using numactl. This field is a string type. The field value is the ID of the NUMA node, such as "0,1"
- `config_file`: specifies a local file. The specified file will be sent to the target machine as the configuration for Alertmanager during the initialization phase of the cluster configuration.

- `os`: the operating system of the machine specified in the `host` field. If the field is not specified, the default value is the `os` value configured in the `global` section.
- `arch`: the architecture of the machine specified in the `host` field. If the field is not specified, the default value is the `arch` value configured in the `global` section.
- `resource_control`: resource control on this service. If this field is specified, the configuration of this field will be merged with the configuration of `resource_control` in the `global` section (if the two fields overlap, the configuration of this field takes effect), and then the configuration file of systemd is generated and distributed to the machine specified in the `host` field. The configuration rules of this field are the same as that of `resource_control` in the `global` section.

In `alertmanager_servers`, the following fields cannot be modified after the deployment is completed:

- `host`
- `web_port`
- `cluster_port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

An `alertmanager_servers` configuration example is as follows:

```
alertmanager_servers:
  - host: 10.0.1.11
    config_file: /local/config/file
  - host: 10.0.1.12
    config_file: /local/config/file
```

13.4.10 TiUP Mirror Reference Guide

TiUP mirrors are TiUP's component warehouse, which stores components and their metadata. TiUP mirrors take the following two forms:

- Directory on the local disk: serves the local TiUP client, which is called a local mirror in this document.
- HTTP mirror started based on the remote disk directory: serves the remote TiUP client, which is called a remote mirror in this document.

13.4.10.1 Create and update mirror

You can create a TiUP mirror using one of the following two methods:

- Execute `tiup mirror init` to create a mirror from scratch.
- Execute `tiup mirror clone` to clone from an existing mirror.

After the mirror is created, you can add components to or delete components from the mirror using the `tiup mirror` commands. TiUP updates a mirror by adding files and assigning a new version number to it, rather than deleting any files from the mirror.

13.4.10.2 Mirror structure

A typical mirror structure is as follows:

```
+ <mirror-dir>                                  # Mirror's root directory
|-- root.json                                   # Mirror's root certificate
|-- {2..N}.root.json                            # Mirror's root certificate
|-- {1..N}.index.json                           # Component/user index
|-- {1..N}.{component}.json                     # Component metadata
|-- {component}-{version}-{os}-{arch}.tar.gz # Component binary package
|-- snapshot.json                               # Mirror's latest snapshot
|-- timestamp.json                              # Mirror's latest timestamp
|--+ commits                                    # Mirror's update log (deletable)
   |--+ commit-{ts1..tsN}
      |-- {N}.root.json
      |-- {N}.{component}.json
      |-- {N}.index.json
      |-- {component}-{version}-{os}-{arch}.tar.gz
      |-- snapshot.json
      |-- timestamp.json
|--+ keys                                       # Mirror's private key (can be
   ↪ moved to other locations)
   |-- {hash1..hashN}-root.json                 # Private key of the root
      ↪ certificate
   |-- {hash}-index.json                        # Private key of the indexes
   |-- {hash}-snapshot.json                     # Private key of the snapshots
   |-- {hash}-timestamp.json                    # Private key of the timestamps
```

Note:

- The `commits` directory stores the logs generated in the process of mirror update and is used to roll back the mirror. You can delete the old log directories regularly when the disk space is insufficient.

- The private key stored in the `keys` directory is sensitive. It is recommended to keep it separately.

13.4.10.2.1 Root directory

In a TiUP mirror, the root certificate is used to store the public key of other metadata files. Each time any metadata file (`*.json`) is obtained, TiUP client needs to find the corresponding public key in the installed `root.json` based on the metadata file type (root, index, snapshot, timestamp). Then TiUP client uses the public key to verify whether the signature is valid.

The root certificate's format is as follows:

```
{
    "signatures": [                                         # Each metadata file
        ↪  has some signatures which are signed by several private keys
        ↪ corresponding to the file.
        {
            "keyid": "{id-of-root-key-1}",                  # The ID of the
                ↪ first private key that participates in the signature. This
                ↪ ID is obtained by hashing the content of the public key
                ↪ that corresponds to the private key.
            "sig": "{signature-by-root-key-1}"              # The signed part of
                ↪  this file by this private key.
        },
        ...
        {
            "keyid": "{id-of-root-key-N}",                  # The ID of the Nth
                ↪ private key that participates in the signature.
            "sig": "{signature-by-root-key-N}"              # The signed part of
                ↪  this file by this private key.
        }
    ],
    "signed": {                                             # The signed part.
        "_type": "root",                                    # The type of this
            ↪ file. root.json's type is root.
        "expires": "{expiration-date-of-this-file}",        # The expiration
            ↪ time of the file. If the file expires, the client rejects the
            ↪ file.
        "roles": {                                          # Records the keys
            ↪ used to sign each metadata file.
            "{role:index,root,snapshot,timestamp}": {       # Each involved
                ↪ metadata file includes index, root, snapshot, and timestamp
                ↪ .
```

```
            "keys": {                                   # Only the key's
                ↪ signature recorded in `keys` is valid.
                "{id-of-the-key-1}": {                  # The ID of the
                    ↪ first key used to sign {role}.
                    "keytype": "rsa",                   # The key's type.
                        ↪ Currently, the key type is fixed as rsa.
                    "keyval": {                         # The key's payload.
                        "public": "{public-key-content}" # The public key's
                            ↪  content.
                    },
                    "scheme": "rsassa-pss-sha256"       # Currently, the
                        ↪ scheme is fixed as rsassa-pss-sha256.
                },
                "{id-of-the-key-N}": {                  # The ID of the Nth
                    ↪ key used to sign {role}.
                    "keytype": "rsa",
                    "keyval": {
                        "public": "{public-key-content}"
                    },
                    "scheme": "rsassa-pss-sha256"
                }
            },
            "threshold": {N},                           # Indicates that the
                ↪  metadata file needs at least N key signatures.
            "url": "/{role}.json"                       # The address from
                ↪ which the file can be obtained. For index files, prefix
                ↪  it with the version number (for example, /{N}.index.
                ↪ json).
        }
    },
    "spec_version": "0.1.0",                            # The specified
        ↪ version followed by this file. If the file structure is
        ↪ changed in the future, the version number needs to be upgraded
        ↪ . The current version number is 0.1.0.
    "version": {N}                                      # The version number
        ↪  of this file. You need to create a new {N+1}.root.json every
        ↪ time you update the file, and set its version to N + 1.
  }
}
```

13.4.10.2.2 Index

The index file records all the components in the mirror and the owner information of the components.

The index file's format is as follows:

```
{
    "signatures": [                                         # The file's
        ↪ signature.
        {
            "keyid": "{id-of-index-key-1}",                 # The ID of the
                ↪ first private key that participates in the signature.
            "sig": "{signature-by-index-key-1}",            # The signed part of
                ↪  this file by this private key.
        },
        ...
        {
            "keyid": "{id-of-root-key-N}",                  # The ID of the Nth
                ↪ private key that participates in the signature.
            "sig": "{signature-by-root-key-N}"              # The signed part of
                ↪  this file by this private key.
        }
    ],
    "signed": {
        "_type": "index",                                   # The file type.
        "components": {                                     # The component list
            ↪ .
            "{component1}": {                               # The name of the
                ↪ first component.
                "hidden": {bool},                           # Whether it is a
                    ↪ hidden component.
                "owner": "{owner-id}",                      # The component
                    ↪ owner's ID.
                "standalone": {bool},                       # Whether it is a
                    ↪ standalone component.
                "url": "/{component}.json",                 # The address from
                    ↪ which the component can be obtained. You need to prefix
                    ↪  it with the version number (for example, /{N}.{
                    ↪ component}.json).
                "yanked": {bool}                            # Indicates whether
                    ↪ the component is marked as deleted.
            },
            ...
            "{componentN}": {                               # The name of the
                ↪ Nth component.
                ...
            },
        },
        "default_components": ["{component1}".."{componentN}"], # The default
```

```
    ↪  component that a mirror must contain. Currently, this field
    ↪ defaults to empty (disabled).
"expires": "{expiration-date-of-this-file}",     # The expiration
    ↪ time of the file. If the file expires, the client rejects the
    ↪ file.
"owners": {
    "{owner1}": {                                # The ID of the
       ↪ first owner.
        "keys": {                                # Only the key's
           ↪ signature recorded in `keys` is valid.
            "{id-of-the-key-1}": {               # The first key of
               ↪ the owner.
                "keytype": "rsa",                # The key's type.
                   ↪ Currently, the key type is fixed as rsa.
                "keyval": {                      # The key's payload.
                    "public": "{public-key-content}" # The public key's
                       ↪  content.
                },
                "scheme": "rsassa-pss-sha256"    # Currently, the
                   ↪ scheme is fixed as rsassa-pss-sha256.
            },
            ...
            "{id-of-the-key-N}": {               # The Nth key of the
               ↪  owner.
                ...
            }
        },
        "name": "{owner-name}",                  # The name of the
           ↪ owner.
        "threshod": {N}                          # Indicates that the
           ↪  components owned by the owner must have at least N
           ↪ valid signatures.
    },
    ...
    "{ownerN}": {                                # The ID of the Nth
       ↪ owner.
        ...
    }
}
"spec_version": "0.1.0",                         # The specified
    ↪ version followed by this file. If the file structure is
    ↪ changed in the future, the version number needs to be upgraded
    ↪ . The current version number is 0.1.0.
"version": {N}                                   # The version number
    ↪  of this file. You need to create a new {N+1}.index.json every
```

```
            ↪  time you update the file, and set its version to N + 1.
    }
}
```

13.4.10.2.3 Component

The component's metadata file records information of the component-specific platform and the version.

The component metadata file's format is as follows:

```
{
    "signatures": [                                     # The file's
       ↪ signature.
        {
            "keyid": "{id-of-index-key-1}",             # The ID of the
               ↪ first private key that participates in the signature.
            "sig": "{signature-by-index-key-1}",        # The signed part of
               ↪  this file by this private key.
        },
        ...
        {
            "keyid": "{id-of-root-key-N}",              # The ID of the Nth
               ↪ private key that participates in the signature.
            "sig": "{signature-by-root-key-N}"          # The signed part of
               ↪  this file by this private key.
        }
    ],
    "signed": {
        "_type": "component",                           # The file type.
        "description": "{description-of-the-component}", # The description of
           ↪  the component.
        "expires": "{expiration-date-of-this-file}",    # The expiration
           ↪ time of the file. If the file expires, the client rejects the
           ↪ file.
        "id": "{component-id}",                         # The globally
           ↪ unique ID of the component.
        "nightly": "{nightly-cursor}",                  # The nightly cursor
           ↪ , and the value is the latest nightly version number (for
           ↪ example, v5.0.0-nightly-20201209).
        "platforms": {                                  # The component's
           ↪ supported platforms (such as darwin/amd64, linux/arm64).
            "{platform-pair-1}": {
                "{version-1}": {                        # The semantic
                   ↪ version number (for example, v1.0.0).
```

```
                "dependencies": null,                   # Specifies the
                    ↪ dependency relationship between components. The
                    ↪ field is not used yet and is fixed as null.
                "entry": "{entry}",                     # The relative path
                    ↪ of the entry binary file in the tar package.
                "hashs": {                              # The checksum of
                    ↪ the tar package. sha256 and sha512 are used.
                    "sha256": "{sum-of-sha256}",
                    "sha512": "{sum-of-sha512}",
                },
                "length": {length-of-tar},              # The length of the
                    ↪ tar package.
                "released": "{release-time}",           # The release date
                    ↪ of the version.
                "url": "{url-of-tar}",                  # The download
                    ↪ address of the tar package.
                "yanked": {bool}                        # Indicates whether
                    ↪ this version is disabled.
            }
        },
        ...
        "{platform-pair-N}": {
            ...
        }
    },
    "spec_version": "0.1.0",                            # The specified
        ↪ version followed by this file. If the file structure is
        ↪ changed in the future, the version number needs to be upgraded
        ↪ . The current version number is 0.1.0.
    "version": {N}                                      # The version number
        ↪  of this file. You need to create a new {N+1}.{component}.json
        ↪  every time you update the file, and set its version to N + 1.
}
```

13.4.10.2.4 Snapshot

The snapshot file records the version number of each metadata file:

The snapshot file's structure is as follows:

```
{
    "signatures": [                                     # The file's
        ↪ signature.
        {
            "keyid": "{id-of-index-key-1}",             # The ID of the
                ↪ first private key that participates in the signature.
```

```
        "sig": "{signature-by-index-key-1}",          # The signed part of
           ↪  this file by this private key.
    },
    ...
    {
        "keyid": "{id-of-root-key-N}",                # The ID of the Nth
           ↪ private key that participates in the signature.
        "sig": "{signature-by-root-key-N}"            # The signed part of
           ↪  this file by this private key.
    }
],
"signed": {
    "_type": "snapshot",                              # The file type.
    "expires": "{expiration-date-of-this-file}",      # The expiration
       ↪ time of the file. If the file expires, the client rejects the
       ↪ file.
    "meta": {                                         # Other metadata
       ↪ files' information.
        "/root.json": {
            "length": {length-of-json-file},          # The length of root
               ↪ .json
            "version": {version-of-json-file}         # The version of
               ↪ root.json
        },
        "/index.json": {
            "length": {length-of-json-file},
            "version": {version-of-json-file}
        },
        "/{component-1}.json": {
            "length": {length-of-json-file},
            "version": {version-of-json-file}
        },
        ...
        "/{component-N}.json": {
            ...
        }
    },
    "spec_version": "0.1.0",                          # The specified
       ↪ version followed by this file. If the file structure is
       ↪ changed in the future, the version number needs to be upgraded
       ↪ . The current version number is 0.1.0.
    "version": 0                                      # The version number
       ↪  of this file, which is fixed as 0.
}
```

13.4.10.2.5 Timestamp

The timestamp file records the checksum of the current snapshot.

The timestamp file's format is as follows:

```
{
    "signatures": [                                         # The file's
        ↪ signature.
        {
            "keyid": "{id-of-index-key-1}",                 # The ID of the
                ↪ first private key that participates in the signature.
            "sig": "{signature-by-index-key-1}",            # The signed part of
                ↪  this file by this private key.
        },
        ...
        {
            "keyid": "{id-of-root-key-N}",                  # The ID of the Nth
                ↪ private key that participates in the signature.
            "sig": "{signature-by-root-key-N}"              # The signed part of
                ↪  this file by this private key.
        }
    ],
    "signed": {
        "_type": "timestamp",                               # The file type.
        "expires": "{expiration-date-of-this-file}",        # The expiration
            ↪ time of the file. If the file expires, the client rejects the
            ↪ file.
        "meta": {                                           # The information of
            ↪  snapshot.json.
            "/snapshot.json": {
                "hashes": {
                    "sha256": "{sum-of-sha256}"             # snapshot.json's
                        ↪ sha256.
                },
                "length": {length-of-json-file}             # The length of
                    ↪ snapshot.json.
            }
        },
        "spec_version": "0.1.0",                            # The specified
            ↪ version followed by this file. If the file structure is
            ↪ changed in the future, the version number needs to be upgraded
            ↪ . The current version number is 0.1.0.
        "version": {N}                                      # The version number
            ↪  of this file. You need to overwrite timestamp.json every time
            ↪  you update the file, and set its version to N + 1.
```

13.4.10.3 Client workflow

The client uses the following logic to ensure that the files downloaded from the mirror are safe:

- A `root.json` file is included with the binary when the client is installed.
- The running client performs the following tasks based on the existing `root.json`:

    1. Obtain the version from `root.json` and mark it as `N`.
    2. Request `{N+1}.root.json` from the mirror. If the request is successful, use the public key recorded in `root.json` to verify whether the file is valid.
    3. Request `timestamp.json` from the mirror and use the public key recorded in `root.json` to verify whether the file is valid.
    4. Check whether the checksum of `snapshot.json` recorded in `timestamp.json` matches the checksum of the local `snapshot.json`. If the two do not match, request the latest `snapshot.json` from the mirror and use the public key recorded in `root.json` to verify whether the file is valid.
    5. Obtain the version number `N` of the `index.json` file from `snapshot.json` and request `{N}.index.json` from the mirror. Then use the public key recorded in `root.json` to verify whether the file is valid.
    6. For components such as `tidb.json` and `tikv.json`, the client obtains the version numbers `N` of the components from `snapshot.json` and requests `{N}.{component ↪ }.json` from the mirror. Then the client uses the public key recorded in `index.json` to verify whether the file is valid.
    7. For component's tar files, the client obtains the URLs and checksums of the files from `{component}.json` and request the URLs for the tar packages. Then the client verifies whether the checksum is correct.

13.4.11 TiUP Components

13.4.11.1 Quickly Deploy a Local TiDB Cluster

The TiDB cluster is a distributed system that consists of multiple components. A typical TiDB cluster consists of at least three PD nodes, three TiKV nodes, and two TiDB nodes. If you want to have a quick experience on TiDB, you might find it time-consuming and complicated to manually deploy so many components. This document introduces the playground component of TiUP and how to use it to quickly build a local TiDB test environment.

13.4.11.1.1 TiUP playground overview

The basic usage of the playground component is shown as follows:

```
tiup playground ${version} [flags]
```

If you directly execute the `tiup playground` command, TiUP uses the locally installed TiDB, TiKV, and PD components or installs the stable version of these components to start

a TiDB cluster that consists of one TiKV instance, one TiDB instance, one PD instance, and one TiFlash instance.

This command actually performs the following operations:

- Because this command does not specify the version of the playground component, TiUP first checks the latest version of the installed playground component. Assume that the latest version is v1.10.0, then this command works the same as `tiup playground:v1.10.0`.
- If you have not used TiUP playground to install the TiDB, TiKV, and PD components, the playground component installs the latest stable version of these components, and then start these instances.
- Because this command does not specify the version of the TiDB, PD, and TiKV component, TiUP playground uses the latest version of each component by default. Assume that the latest version is v6.1.7, then this command works the same as `tiup playground:v1.10.0 v6.1.7`.
- Because this command does not specify the number of each component, TiUP playground, by default, starts a smallest cluster that consists of one TiDB instance, one TiKV instance, one PD instance, and one TiFlash instance.
- After starting each TiDB component, TiUP playground reminds you that the cluster is successfully started and provides you some useful information, such as how to connect to the TiDB cluster through the MySQL client and how to access the TiDB Dashboard.

The command-line flags of the playground component are described as follows:

```
Flags:
      --db int                Specify the number of TiDB instances (default:
          ↪ 1)
      --db.host host          Specify the listening address of TiDB
      --db.port int           Specify the port of TiDB
      --db.binpath string     Specify the TiDB instance binary path (optional
          ↪ , for debugging)
      --db.config string      Specify the TiDB instance configuration file (
          ↪ optional, for debugging)
      --db.timeout int        Specify TiDB maximum wait time in seconds for
          ↪ starting. 0 means no limit
      --drainer int           Specify Drainer data of the cluster
      --drainer.binpath string Specify the location of the Drainer binary
          ↪ files (optional, for debugging)
      --drainer.config string Specify the Drainer configuration file
  -h, --help                  help for tiup
      --host string           Specify the listening address of each component
          ↪  (default: `127.0.0.1`). Set it to `0.0.0.0` if provided for
          ↪ access of other machines
      --kv int                Specify the number of TiKV instances (default:
          ↪ 1)
```

```
    --kv.binpath string     Specify the TiKV instance binary path (optional
       ↪ , for debugging)
    --kv.config string      Specify the TiKV instance configuration file (
       ↪ optional, for debugging)
    --mode string           Specify the playground mode: 'tidb' (default)
       ↪ and 'tikv-slim'
    --pd int                Specify the number of PD instances (default: 1)
    --pd.host host          Specify the listening address of PD
    --pd.binpath string     Specify the PD instance binary path (optional,
       ↪ for debugging)
    --pd.config string      Specify the PD instance configuration file (
       ↪ optional, for debugging)
    --pump int              Specify the number of Pump instances. If the
       ↪ value is not `0`, TiDB Binlog is enabled.
    --pump.binpath string   Specify the location of the Pump binary files (
       ↪ optional, for debugging)
    --pump.config string    Specify the Pump configuration file (optional,
       ↪ for debugging)
    -T, --tag string        Specify a tag for playground
    --ticdc int             Specify the number of TiCDC instances (default:
       ↪  0)
    --ticdc.binpath string  Specify the TiCDC instance binary path (
       ↪ optional, for debugging)
    --ticdc.config string   Specify the TiCDC instance configuration file (
       ↪ optional, for debugging)
    --tiflash int           Specify the number of TiFlash instances (
       ↪ default: 1)
    --tiflash.binpath string Specify the TiFlash instance binary path (
       ↪ optional, for debugging)
    --tiflash.config string Specify the TiFlash instance configuration
       ↪ file (optional, for debugging)
    --tiflash.timeout int   Specify TiFlash maximum wait time in seconds
       ↪ for starting. 0 means no limit
    -v, --version           Specify the version of playground
    --without-monitor       Disable the monitoring function of Prometheus
       ↪ and Grafana. If you do not add this flag, the monitoring
       ↪ function is enabled by default.
```

13.4.11.1.2 Examples

Check available TiDB versions

```
tiup list tidb
```

Start a TiDB cluster of a specific version

```
tiup playground ${version}
```

Replace `${version}` with the target version number.

Start a TiDB cluster of the nightly version

```
tiup playground nightly
```

In the command above, `nightly` indicates the latest development version of TiDB.

Override PD's default configuration

First, you need to copy the PD configuration template. Assume you place the copied file to `~/config/pd.toml` and make some changes according to your need, then you can execute the following command to override PD's default configuration:

```
tiup playground --pd.config ~/config/pd.toml
```

Replace the default binary files

By default, when playground is started, each component is started using the binary files from the official mirror. If you want to put a temporarily compiled local binary file into the cluster for testing, you can use the `--{comp}.binpath` flag for replacement. For example, execute the following command to replace the binary file of TiDB:

```
tiup playground --db.binpath /xx/tidb-server
```

Start multiple component instances

By default, only one instance is started for each TiDB, TiKV, and PD component. To start multiple instances for each component, add the following flag:

```
tiup playground --db 3 --pd 3 --kv 3
```

13.4.11.1.3 Quickly connect to the TiDB cluster started by playground

TiUP provides the `client` component, which is used to automatically find and connect to a local TiDB cluster started by playground. The usage is as follows:

```
tiup client
```

This command provides a list of TiDB clusters that are started by playground on the current machine on the console. Select the TiDB cluster to be connected. After clicking Enter, a built-in MySQL client is opened to connect to TiDB.

13.4.11.1.4 View information of the started cluster

```
tiup playground display
```

The command above returns the following results:

```
Pid   Role    Uptime
---   ----    ------
84518 pd      35m22.929404512s
84519 tikv    35m22.927757153s
84520 pump    35m22.92618275s
86189 tidb    exited
86526 tidb    34m28.293148663s
86190 drainer 35m19.91349249s
```

13.4.11.1.5 Scale out a cluster

The command-line parameter for scaling out a cluster is similar to that for starting a cluster. You can scale out two TiDB instances by executing the following command:

```
tiup playground scale-out --db 2
```

13.4.11.1.6 Scale in a cluster

You can specify a `pid` in the `tiup playground scale-in` command to scale in the corresponding instance. To view the `pid`, execute `tiup playground display`.

```
tiup playground scale-in --pid 86526
```

13.4.11.2 Deploy and Maintain an Online TiDB Cluster Using TiUP

This document focuses on how to use the TiUP cluster component. For the complete steps of online deployment, refer to Deploy a TiDB Cluster Using TiUP.

Similar to the TiUP playground component used for a local test deployment, the TiUP cluster component quickly deploys TiDB for production environment. Compared with playground, the cluster component provides more powerful production cluster management features, including upgrading, scaling, and even operation and auditing.

For the help information of the cluster component, run the following command:

```
tiup cluster
```

```
Starting component `cluster`: /home/tidb/.tiup/components/cluster/v1.10.0/
    ↪ cluster
Deploy a TiDB cluster for production

Usage:
  tiup cluster [command]

Available Commands:
```

```
  check       Precheck a cluster
  deploy      Deploy a cluster for production
  start       Start a TiDB cluster
  stop        Stop a TiDB cluster
  restart     Restart a TiDB cluster
  scale-in    Scale in a TiDB cluster
  scale-out   Scale out a TiDB cluster
  destroy     Destroy a specified cluster
  clean       (Experimental) Clean up a specified cluster
  upgrade     Upgrade a specified TiDB cluster
  display     Display information of a TiDB cluster
  list        List all clusters
  audit       Show audit log of cluster operation
  import      Import an existing TiDB cluster from TiDB-Ansible
  edit-config Edit TiDB cluster config
  reload      Reload a TiDB cluster's config and restart if needed
  patch       Replace the remote package with a specified package and restart
     ↪  the service
  help        Help about any command

Flags:
  -c, --concurrency int  Maximum number of concurrent tasks allowed (
     ↪ defaults to `5`)
      --format string    (EXPERIMENTAL) The format of output, available
         ↪ values are [default, json] (default "default")
  -h, --help             help for tiup
      --ssh string       (Experimental) The executor type. Optional values
         ↪ are 'builtin', 'system', and 'none'.
      --ssh-timeout uint Timeout in seconds to connect a host via SSH.
         ↪ Operations that don't need an SSH connection are ignored. (
         ↪ default 5)
  -v, --version         TiUP version
      --wait-timeout uint Timeout in seconds to wait for an operation to
         ↪ complete. Inapplicable operations are ignored. (defaults to
         ↪ `120`)
  -y, --yes              Skip all confirmations and assumes 'yes'
```

13.4.11.2.1 Deploy the cluster

To deploy the cluster, run the `tiup cluster deploy` command. The usage of the command is as follows:

```
tiup cluster deploy <cluster-name> <version> <topology.yaml> [flags]
```

This command requires you to provide the cluster name, the TiDB cluster version, and a topology file of the cluster.

To write a topology file, refer to the example. The following file is an example of the simplest topology:

> **Note:**
>
> The topology file used by the TiUP cluster component for deployment and scaling is written using yaml syntax, so make sure that the indentation is correct.

```
---

pd_servers:
  - host: 172.16.5.134
    name: pd-134
  - host: 172.16.5.139
    name: pd-139
  - host: 172.16.5.140
    name: pd-140

tidb_servers:
  - host: 172.16.5.134
  - host: 172.16.5.139
  - host: 172.16.5.140

tikv_servers:
  - host: 172.16.5.134
  - host: 172.16.5.139
  - host: 172.16.5.140

tiflash_servers:
  - host: 172.16.5.141
  - host: 172.16.5.142
  - host: 172.16.5.143

grafana_servers:
  - host: 172.16.5.134

monitoring_servers:
  - host: 172.16.5.134
```

By default, TiUP is deployed as the binary files running on the amd64 architecture. If the target machine is the arm64 architecture, you can configure it in the topology file:

```
global:
  arch: "arm64"        # Configures all machines to use the binary files of
     ↪ the arm64 architecture by default

tidb_servers:
  - host: 172.16.5.134
    arch: "amd64"      # Configures this machine to use the binary files of
       ↪ the amd64 architecture
  - host: 172.16.5.139
    arch: "arm64"      # Configures this machine to use the binary files of
       ↪ the arm64 architecture
  - host: 172.16.5.140 # Machines that are not configured with the arch
     ↪ field use the default value in the global field, which is arm64 in
     ↪ this case.

...
```

Save the file as `/tmp/topology.yaml`. If you want to use TiDB v6.1.7 and your cluster name is `prod-cluster`, run the following command:

```
tiup cluster deploy -p prod-cluster v6.1.7 /tmp/topology.yaml
```

During the execution, TiUP asks you to confirm your topology again and requires the root password of the target machine (the `-p` flag means inputting password):

```
Please confirm your topology:
TiDB Cluster: prod-cluster
TiDB Version: v6.1.7
Type       Host         Ports       Directories
----       ----         -----       -----------
pd         172.16.5.134 2379/2380  deploy/pd-2379,data/pd-2379
pd         172.16.5.139 2379/2380  deploy/pd-2379,data/pd-2379
pd         172.16.5.140 2379/2380  deploy/pd-2379,data/pd-2379
tikv       172.16.5.134 20160/20180 deploy/tikv-20160,data/tikv-20160
tikv       172.16.5.139 20160/20180 deploy/tikv-20160,data/tikv-20160
tikv       172.16.5.140 20160/20180 deploy/tikv-20160,data/tikv-20160
tidb       172.16.5.134 4000/10080 deploy/tidb-4000
tidb       172.16.5.139 4000/10080 deploy/tidb-4000
tidb       172.16.5.140 4000/10080 deploy/tidb-4000
prometheus 172.16.5.134 9090      deploy/prometheus-9090,data/prometheus
   ↪ -9090
grafana    172.16.5.134 3000      deploy/grafana-3000
Attention:
```

```
   1. If the topology is not what you expected, check your yaml file.
   2. Please confirm there is no port/directory conflicts in same host.
Do you want to continue? [y/N]:
```

After you enter the password, TiUP cluster downloads the required components and deploy them on the corresponding machines. When you see the following message, the deployment is successful:

```
Deployed cluster `prod-cluster` successfully
```

13.4.11.2.2 View the cluster list

After the cluster is successfully deployed, view the cluster list by running the following command:

```
tiup cluster list
```

```
Starting /root/.tiup/components/cluster/v1.10.0/cluster list
Name         User Version   Path
   ↪ PrivateKey
----         ---- -------   ----
   ↪ ----------
prod-cluster tidb v6.1.7  /root/.tiup/storage/cluster/clusters/prod-cluster
   ↪  /root/.tiup/storage/cluster/clusters/prod-cluster/ssh/id_rsa
```

13.4.11.2.3 Start the cluster

After the cluster is successfully deployed, start the cluster by running the following command:

```
tiup cluster start prod-cluster
```

If you forget the name of your cluster, view the cluster list by running `tiup cluster list`.

TiUP uses `systemd` to start a daemon process. If the process terminates unexpectedly, it will be pulled up after 15 seconds.

13.4.11.2.4 Check the cluster status

TiUP provides the `tiup cluster display` command to view the status of each component in the cluster. With this command, you don't have to log in to each machine to see the component status. The usage of the command is as follows:

```
tiup cluster display prod-cluster
```

```
Starting /root/.tiup/components/cluster/v1.10.0/cluster display prod-cluster
TiDB Cluster: prod-cluster
TiDB Version: v6.1.7
ID                  Role       Host         Ports       Status    Data Dir
   ↪            Deploy Dir
--                  ----       ----         -----       ------    --------
   ↪            ----------
172.16.5.134:3000 grafana   172.16.5.134 3000       Up        -
   ↪                    deploy/grafana-3000
172.16.5.134:2379 pd        172.16.5.134 2379/2380  Up|L      data/pd-2379
   ↪        deploy/pd-2379
172.16.5.139:2379 pd        172.16.5.139 2379/2380  Up|UI     data/pd-2379
   ↪        deploy/pd-2379
172.16.5.140:2379 pd        172.16.5.140 2379/2380  Up        data/pd-2379
   ↪        deploy/pd-2379
172.16.5.134:9090 prometheus 172.16.5.134 9090      Up        data/
   ↪ prometheus-9090 deploy/prometheus-9090
172.16.5.134:4000 tidb      172.16.5.134 4000/10080 Up        -
   ↪                    deploy/tidb-4000
172.16.5.139:4000 tidb      172.16.5.139 4000/10080 Up        -
   ↪                    deploy/tidb-4000
172.16.5.140:4000 tidb      172.16.5.140 4000/10080 Up        -
   ↪                    deploy/tidb-4000
172.16.5.134:20160 tikv     172.16.5.134 20160/20180 Up       data/tikv
   ↪ -20160     deploy/tikv-20160
172.16.5.139:20160 tikv     172.16.5.139 20160/20180 Up       data/tikv
   ↪ -20160     deploy/tikv-20160
172.16.5.140:20160 tikv     172.16.5.140 20160/20180 Up       data/tikv
   ↪ -20160     deploy/tikv-20160
```

The `Status` column uses `Up` or `Down` to indicate whether the service is running normally.

For the PD component, `|L` or `|UI` might be appended to `Up` or `Down`. `|L` indicates that the PD node is a Leader, and `|UI` indicates that TiDB Dashboard is running on the PD node.

13.4.11.2.5 Scale in a cluster

> **Note:**
>
> This section describes only the syntax of the scale-in command. For detailed steps of online scaling, refer to Scale a TiDB Cluster Using TiUP.

Scaling in a cluster means making some node(s) offline. This operation removes the specific node(s) from the cluster and deletes the remaining files.

Because the offline process of the TiKV, TiFlash, and TiDB Binlog components is asynchronous (which requires removing the node through API), and the process takes a long time (which requires continuous observation on whether the node is successfully taken offline), special treatment is given to the TiKV, TiFlash, and TiDB Binlog components.

- For TiKV, TiFlash, and Binlog:
    - TiUP cluster takes the node offline through API and directly exits without waiting for the process to be completed.
    - Afterwards, when a command related to the cluster operation is executed, TiUP cluster examines whether there is a TiKV, TiFlash, or Binlog node that has been taken offline. If not, TiUP cluster continues with the specified operation; If there is, TiUP cluster takes the following steps:
        1. Stop the service of the node that has been taken offline.
        2. Clean up the data files related to the node.
        3. Remove the node from the cluster topology.
- For other components:
    - When taking the PD component down, TiUP cluster quickly deletes the specified node from the cluster through API, stops the service of the specified PD node, and deletes the related data files.
    - When taking other components down, TiUP cluster directly stops the node service and deletes the related data files.

The basic usage of the scale-in command:

```
tiup cluster scale-in <cluster-name> -N <node-id>
```

To use this command, you need to specify at least two flags: the cluster name and the node ID. The node ID can be obtained by using the `tiup cluster display` command in the previous section.

For example, to make the TiKV node on `172.16.5.140` offline, run the following command:

```
tiup cluster scale-in prod-cluster -N 172.16.5.140:20160
```

By running `tiup cluster display`, you can see that the TiKV node is marked `Offline`:

```
tiup cluster display prod-cluster
```

```
Starting /root/.tiup/components/cluster/v1.10.0/cluster display prod-cluster
TiDB Cluster: prod-cluster
TiDB Version: v6.1.7
ID                  Role       Host         Ports       Status    Data Dir
   ↪            Deploy Dir
--                  ----       ----         -----       ------    --------
   ↪            ----------
172.16.5.134:3000 grafana   172.16.5.134 3000       Up        -
   ↪                    deploy/grafana-3000
172.16.5.134:2379 pd        172.16.5.134 2379/2380  Up|L      data/pd-2379
   ↪        deploy/pd-2379
172.16.5.139:2379 pd        172.16.5.139 2379/2380  Up|UI     data/pd-2379
   ↪        deploy/pd-2379
172.16.5.140:2379 pd        172.16.5.140 2379/2380  Up        data/pd-2379
   ↪        deploy/pd-2379
172.16.5.134:9090 prometheus 172.16.5.134 9090      Up        data/
   ↪ prometheus-9090 deploy/prometheus-9090
172.16.5.134:4000 tidb      172.16.5.134 4000/10080 Up        -
   ↪                    deploy/tidb-4000
172.16.5.139:4000 tidb      172.16.5.139 4000/10080 Up        -
   ↪                    deploy/tidb-4000
172.16.5.140:4000 tidb      172.16.5.140 4000/10080 Up        -
   ↪                    deploy/tidb-4000
172.16.5.134:20160 tikv     172.16.5.134 20160/20180 Up       data/tikv
   ↪ -20160    deploy/tikv-20160
172.16.5.139:20160 tikv     172.16.5.139 20160/20180 Up       data/tikv
   ↪ -20160    deploy/tikv-20160
172.16.5.140:20160 tikv     172.16.5.140 20160/20180 Offline data/tikv
   ↪ -20160    deploy/tikv-20160
```

After PD schedules the data on the node to other TiKV nodes, this node will be deleted automatically.

13.4.11.2.6 Scale out a cluster

> **Note:**
>
> This section describes only the syntax of the scale-out command. For detailed steps of online scaling, refer to Scale a TiDB Cluster Using TiUP.

The scale-out operation has an inner logic similar to that of deployment: the TiUP cluster component firstly ensures the SSH connection of the node, creates the required directories

on the target node, then executes the deployment operation, and starts the node service.

When you scale out PD, the node is added to the cluster by `join`, and the configurations of services associated with PD are updated. When you scale out other services, the service is started directly and added to the cluster.

All services conduct correctness validation when they are scaled out. The validation results show whether the scaling-out is successful.

To add a TiKV node and a PD node in the `tidb-test` cluster, take the following steps:

1. Create a `scale.yaml` file, and add IPs of the new TiKV and PD nodes:

> **Note:**
> You need to create a topology file, which includes only the description of the new nodes, not the existing nodes.

```
---

pd_servers:
  - host: 172.16.5.140

tikv_servers:
  - host: 172.16.5.140
```

2. Perform the scale-out operation. TiUP cluster adds the corresponding nodes to the cluster according to the port, directory, and other information described in `scale.yaml`.

```
tiup cluster scale-out tidb-test scale.yaml
```

After the command is executed, you can check the status of the scaled-out cluster by running `tiup cluster display tidb-test`.

13.4.11.2.7 Rolling upgrade

> **Note:**
> This section describes only the syntax of the upgrade command. For detailed steps of online upgrade, refer to Upgrade TiDB Using TiUP.

The rolling upgrade feature leverages the distributed capabilities of TiDB. The upgrade process is made as transparent as possible to the application, and does not affect the business.

Before the upgrade, TiUP cluster checks whether the configuration file of each component is rational. If so, the components are upgraded node by node; if not, TiUP reports an error and exits. The operations vary with different nodes.

Operations for different nodes

- Upgrade the PD node
    - First, upgrade non-Leader nodes.
    - After all the non-Leader nodes are upgraded, upgrade the Leader node.
        - The upgrade tool sends a command to PD that migrates Leader to an already upgraded node.
        - After the Leader role is switched to another node, upgrade the previous Leader node.
    - During the upgrade, if any unhealthy node is detected, the tool stops this upgrade operation and exits. You need to manually analyze the cause, fix the issue and run the upgrade again.
- Upgrade the TiKV node
    - First, add a scheduling operation in PD that migrates the Region Leader of this TiKV node. This ensures that the upgrade process does not affect the business.
    - After the Leader is migrated, upgrade this TiKV node.
    - After the upgraded TiKV is started normally, remove the scheduling of the Leader.
- Upgrade other services
    - Stop the service normally and update the node.

Upgrade command

The flags for the upgrade command is as follows:

```
Usage:
  cluster upgrade <cluster-name> <version> [flags]

Flags:
      --force               Force upgrade won't transfer leader
  -h, --help                help for upgrade
      --transfer-timeout int Timeout in seconds when transferring PD and
          ↪ TiKV store leaders (default 300)

Global Flags:
```

```
      --ssh string       (Experimental) The executor type. Optional values
         ↪ are 'builtin', 'system', and 'none'.
      --wait-timeout int Timeout of waiting the operation
      --ssh-timeout int Timeout in seconds to connect host via SSH, ignored
         ↪ for operations that don't need an SSH connection. (default 5)
  -y, --yes             Skip all confirmations and assumes 'yes'
```

For example, the following command upgrades the cluster to v6.1.7:

```
tiup cluster upgrade tidb-test v6.1.7
```

13.4.11.2.8 Update configuration

If you want to dynamically update the component configurations, the TiUP cluster component saves a current configuration for each cluster. To edit this configuration, execute the `tiup cluster edit-config <cluster-name>` command. For example:

```
tiup cluster edit-config prod-cluster
```

TiUP cluster opens the configuration file in the vi editor. If you want to use other editors, use the `EDITOR` environment variable to customize the editor, such as `export EDITOR=nano`.

After editing the file, save the changes. To apply the new configuration to the cluster, execute the following command:

```
tiup cluster reload prod-cluster
```

The command sends the configuration to the target machine and restarts the cluster to make the configuration take effect.

> **Note:**
>
> For monitoring components, customize the configuration by executing the `tiup cluster edit-config` command to add a custom configuration path on the corresponding instance. For example:

```
---

grafana_servers:
  - host: 172.16.5.134
    dashboard_dir: /path/to/local/dashboards/dir

monitoring_servers:
  - host: 172.16.5.134
```

```
    rule_dir: /path/to/local/rules/dir

alertmanager_servers:
  - host: 172.16.5.134
    config_file: /path/to/local/alertmanager.yml
```

The content and format requirements for files under the specified path are as follows:

- The folder specified in the `dashboard_dir` field of `grafana_servers` must contain full `*.json` files.
- The folder specified in the `rule_dir` field of `monitoring_servers` must contain full `*.rules.yml` files.
- For the format of files specified in the `config_file` field of `alertmanager_servers`, refer to the Alertmanager configuration template.

When you execute `tiup reload`, TiUP first deletes all old configuration files in the target machine and then uploads the corresponding configuration from the control machine to the corresponding configuration directory of the target machine. Therefore, if you want to modify a particular configuration file, make sure that all configuration files (including the unmodified ones) are in the same directory. For example, to modify Grafana's `tidb.json` file, you need to first copy all the `*.json` files from Grafana's `dashboards` directory to your local directory. Otherwise, other JSON files will be missing from the target machine.

> **Note:**
>
> If you have configured the `dashboard_dir` field of `grafana_servers`, after executing the `tiup cluster rename` command to rename the cluster, you need to complete the following operations:
>
> 1. In the local `dashboards` directory, change the cluster name to the new cluster name.
> 2. In the local `dashboards` directory, change `datasource` to the new cluster name, because `datasource` is named after the cluster name.
> 3. Execute the `tiup cluster reload -R grafana` command.

13.4.11.2.9 Update component

For normal upgrade, you can use the `upgrade` command. But in some scenarios, such as debugging, you might need to replace the currently running component with a temporary package. To achieve this, use the `patch` command:

```
tiup cluster patch --help
```

```
Replace the remote package with a specified package and restart the service

Usage:
  cluster patch <cluster-name> <package-path> [flags]

Flags:
  -h, --help                help for patch
  -N, --node strings        Specify the nodes
      --overwrite           Use this package in the future scale-out
          ↪ operations
  -R, --role strings        Specify the role
      --transfer-timeout int Timeout in seconds when transferring PD and
          ↪ TiKV store leaders (default 300)

Global Flags:
      --ssh string       (Experimental) The executor type. Optional values
          ↪ are 'builtin', 'system', and 'none'.
      --wait-timeout int Timeout of waiting the operation
      --ssh-timeout int Timeout in seconds to connect host via SSH, ignored
          ↪ for operations that don't need an SSH connection. (default 5)
  -y, --yes             Skip all confirmations and assumes 'yes'
```

If a TiDB hotfix package is in `/tmp/tidb-hotfix.tar.gz` and you want to replace all the TiDB packages in the cluster, run the following command:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -R tidb
```

You can also replace only one TiDB package in the cluster:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -N 172.16.4.5:4000
```

13.4.11.2.10 Import TiDB Ansible cluster

> **Note:**
>
> Currently, TiUP cluster's support for TiSpark is still **experimental**. It is not supported to import a TiDB cluster with TiSpark enabled.

Before TiUP is released, TiDB Ansible is often used to deploy TiDB clusters. To enable TiUP to take over the cluster deployed by TiDB Ansible, use the `import` command.

The usage of the `import` command is as follows:

```
tiup cluster import --help
```

```
Import an exist TiDB cluster from TiDB-Ansible

Usage:
  cluster import [flags]

Flags:
  -d, --dir string      The path to TiDB-Ansible directory
  -h, --help            help for import
      --inventory string The name of inventory file (default "inventory.ini
         ↪ ")
      --no-backup       Don't backup ansible dir, useful when there're
         ↪ multiple inventory files
  -r, --rename NAME     Rename the imported cluster to NAME

Global Flags:
      --ssh string      (Experimental) The executor type. Optional values are
         ↪  'builtin', 'system', and 'none'.
      --wait-timeout int Timeout of waiting the operation
      --ssh-timeout int Timeout in seconds to connect host via SSH, ignored
         ↪ for operations that don't need an SSH connection. (default 5)
  -y, --yes             Skip all confirmations and assumes 'yes'
```

You can use either of the following commands to import a TiDB Ansible cluster:

```
cd tidb-ansible
tiup cluster import
```

```
tiup cluster import --dir=/path/to/tidb-ansible
```

13.4.11.2.11 View the operation log

To view the operation log, use the `audit` command. The usage of the `audit` command is as follows:

```
Usage:
  tiup cluster audit [audit-id] [flags]

Flags:
  -h, --help help for audit
```

If the `[audit-id]` flag is not specified, the command shows a list of commands that have been executed. For example:

```
tiup cluster audit
```

```
Starting component `cluster`: /home/tidb/.tiup/components/cluster/v1.10.0/
    ↪ cluster audit
ID     Time                       Command
--     ----                       -------
4BLhr0 2023-07-12T23:55:09+08:00 /home/tidb/.tiup/components/cluster/v1
    ↪ .11.3/cluster deploy test v6.1.7 /tmp/topology.yaml
4BKWjF 2023-07-12T23:36:57+08:00 /home/tidb/.tiup/components/cluster/v1
    ↪ .11.3/cluster deploy test v6.1.7 /tmp/topology.yaml
4BKVwH 2023-07-12T23:02:08+08:00 /home/tidb/.tiup/components/cluster/v1
    ↪ .11.3/cluster deploy test v6.1.7 /tmp/topology.yaml
4BKKH1 2023-07-12T16:39:04+08:00 /home/tidb/.tiup/components/cluster/v1
    ↪ .11.3/cluster destroy test
4BKKDx 2023-07-12T16:36:57+08:00 /home/tidb/.tiup/components/cluster/v1
    ↪ .11.3/cluster deploy test v6.1.7 /tmp/topology.yaml
```

The first column is `audit-id`. To view the execution log of a certain command, pass the `audit-id` of a command as the flag as follows:

```
tiup cluster audit 4BLhr0
```

13.4.11.2.12 Run commands on a host in the TiDB cluster

To run command on a host in the TiDB cluster, use the `exec` command. The usage of the `exec` command is as follows:

```
Usage:
  cluster exec <cluster-name> [flags]

Flags:
      --command string the command run on cluster host (default "ls")
  -h, --help           help for exec
  -N, --node strings   Only exec on host with specified nodes
  -R, --role strings   Only exec on host with specified roles
      --sudo           use root permissions (default false)

Global Flags:
      --ssh-timeout int Timeout in seconds to connect host via SSH, ignored
          ↪ for operations that don't need an SSH connection. (default 5)
  -y, --yes             Skip all confirmations and assumes 'yes'
```

For example, to execute `ls /tmp` on all TiDB nodes, run the following command:

```
tiup cluster exec test-cluster --command='ls /tmp'
```

13.4.11.2.13 Cluster controllers

Before TiUP is released, you can control the cluster using `tidb-ctl`, `tikv-ctl`, `pd-ctl`, and other tools. To make the tools easier to download and use, TiUP integrates them into an all-in-one component, `ctl`.

```
Usage:
  tiup ctl:<cluster-version> {tidb/pd/tikv/binlog/etcd} [flags]

Flags:
  -h, --help help for tiup
```

This command has a corresponding relationship with those of the previous tools:

```
tidb-ctl [args] = tiup ctl tidb [args]
pd-ctl [args] = tiup ctl pd [args]
tikv-ctl [args] = tiup ctl tikv [args]
binlogctl [args] = tiup ctl bindlog [args]
etcdctl [args] = tiup ctl etcd [args]
```

For example, if you previously view the store by running `pd-ctl -u http://127.0.0.1:2379 store`, now you can run the following command in TiUP:

```
tiup ctl:<cluster-version> pd -u http://127.0.0.1:2379 store
```

13.4.11.2.14 Environment checks for target machines

You can use the `check` command to perform a series of checks on the environment of the target machine and output the check results. By executing the `check` command, you can find common unreasonable configurations or unsupported situations. The command flag list is as follows:

```
Usage:
  tiup cluster check <topology.yml | cluster-name> [flags]
Flags:
      --apply               Try to fix failed checks
      --cluster             Check existing cluster, the input is a cluster
        ↪ name.
      --enable-cpu          Enable CPU thread count check
      --enable-disk         Enable disk IO (fio) check
      --enable-mem          Enable memory size check
  -h, --help                help for check
  -i, --identity_file string The path of the SSH identity file. If specified
     ↪ , public key authentication will be used.
  -p, --password            Use password of target hosts. If specified,
     ↪ password authentication will be used.
      --user string         The user name to login via SSH. The user must
        ↪ has root (or sudo) privilege.
```

By default, this command is used to check the environment before deployment. By specifying the `--cluster` flag to switch the mode, you can also check the target machines of an existing cluster, for example:

```
#### check deployed servers before deployment
tiup cluster check topology.yml --user tidb -p
#### check deployed servers of an existing cluster
tiup cluster check <cluster-name> --cluster
```

The CPU thread count check, memory size check, and disk performance check are disabled by default. For the production environment, it is recommended that you enable the three checks and make sure they pass to obtain the best performance.

- CPU: If the number of threads is greater than or equal to 16, the check is passed.
- Memory: If the total size of physical memory is greater than or equal to 32 GB, the check is passed.
- Disk: Execute `fio` test on the partitions of `data_dir` and record the results.

When running the checks, if the `--apply` flag is specified, the program automatically repairs the failed items. Automatic repair is limited to some items that can be adjusted by modifying the configuration or system parameters. Other unrepaired items need to be handled manually according to the actual situation.

Environment checks are not necessary for deploying a cluster. For the production environment, it is recommended to perform environment checks and pass all check items before deployment. If not all the check items are passed, the cluster might be deployed and run normally, but the best performance might not be obtained.

13.4.11.2.15 Use the system's native SSH client to connect to cluster

All operations above performed on the cluster machine use the SSH client embedded in TiUP to connect to the cluster and execute commands. However, in some scenarios, you might also need to use the SSH client native to the control machine system to perform such cluster operations. For example:

- To use a SSH plug-in for authentication
- To use a customized SSH client

Then you can use the `--ssh=system` command-line flag to enable the system-native command-line tool:

- Deploy a cluster: `tiup cluster deploy <cluster-name> <version> <topo> --ssh=system`
- Start a cluster: `tiup cluster start <cluster-name> --ssh=system`
- Upgrade a cluster: `tiup cluster upgrade ... --ssh=system`

You can add `--ssh=system` in all cluster operation commands above to use the system's native SSH client.

To avoid adding such a flag in every command, you can use the `TIUP_NATIVE_SSH` system variable to specify whether to use the local SSH client:

```
export TIUP_NATIVE_SSH=true
#### or
export TIUP_NATIVE_SSH=1
#### or
export TIUP_NATIVE_SSH=enable
```

If you specify this environment variable and `--ssh` at the same time, `--ssh` has higher priority.

> **Note:**
>
> During the process of cluster deployment, if you need to use a password for connection (`-p`) or `passphrase` is configured in the key file, you must ensure that `sshpass` is installed on the control machine; otherwise, a timeout error is reported.

13.4.11.2.16 Migrate control machine and back up TiUP data

The TiUP data is stored in the `.tiup` directory in the user's home directory. To migrate the control machine, you can take the following steps to copy the `.tiup` directory to the corresponding target machine:

1. Execute `tar czvf tiup.tar.gz .tiup` in the home directory of the original machine.
2. Copy `tiup.tar.gz` to the home directory of the target machine.
3. Execute `tar xzvf tiup.tar.gz` in the home directory of the target machine.
4. Add the `.tiup` directory to the `PATH` environment variable.

   If you use `bash` and you are a `tidb` user, you can add `export PATH=/home/tidb` ↪ `/.tiup/bin:$PATH` in `~/.bashrc` and execute `source ~/.bashrc`. Then make corresponding adjustments according to the shell and the user you use.

> **Note:**
>
> It is recommended that you back up the `.tiup` directory regularly to avoid the loss of TiUP data caused by abnormal conditions, such as disk damage of the control machine.

13.4.11.2.17 Back up and restore meta files for cluster deployment and O&M

If the meta files used for operation and maintenance (O&M) are lost, managing the cluster using TiUP will fail. It is recommended that you back up the meta files regularly by running the following command:

```
tiup cluster meta backup ${cluster_name}
```

If the meta files are lost, you can restore them by running the following command:

```
tiup cluster meta restore ${cluster_name} ${backup_file}
```

> **Note:**
>
> The restore operation overwrites the current meta files. Therefore, it is recommended to restore the meta files only when they are lost.

13.4.11.3 Create a Private Mirror

When creating a private cloud, usually, you need to use an isolated network environment, where the official mirror of TiUP is not accessible. Therefore, you can create a private mirror, which is mainly implemented by the `mirror` command. You can also use the `mirror` command for offline deployment. A private mirror also allows you to use components that you build and package by yourself.

13.4.11.3.1 TiUP `mirror` overview

Execute the following command to get the help information of the `mirror` command:

```
tiup mirror --help
```

```
The `mirror` command is used to manage a component repository for TiUP, you
   ↪  can use
it to create a private repository, or to add new component to an existing
   ↪ repository.
The repository can be used either online or offline.
It also provides some useful utilities to help manage keys, users, and
   ↪ versions
of components or the repository itself.

Usage:
  tiup mirror <command> [flags]
```

```
Available Commands:
  init       Initialize an empty repository
  sign       Add signatures to a manifest file
  genkey     Generate a new key pair
  clone      Clone a local mirror from a remote mirror and download all
     ↪ selected components
  merge      Merge two or more offline mirrors
  publish    Publish a component
  show       Show the mirror address
  set        Set mirror address
  modify     Modify published component
  renew      Renew the manifest of a published component.
  grant      grant a new owner
  rotate     Rotate root.json

Global Flags:
      --help              Help for this command

Use "tiup mirror [command] --help" for more information about a command.
```

13.4.11.3.2 Clone a mirror

You can run the `tiup mirror clone` command to build a local mirror:

```
tiup mirror clone <target-dir> [global-version] [flags]
```

- `target-dir`: used to specify the directory in which cloned data is stored.
- `global-version`: used to quickly set a global version for all components.

The `tiup mirror clone` command provides many optional flags (might provide more in the future). These flags can be divided into the following categories according to their intended usages:

- Determines whether to use prefix matching to match the version when cloning

  If the `--prefix` flag is specified, the version number is matched by prefix for the clone. For example, if you specify `--prefix` as "v5.0.0", then "v5.0.0-rc", and "v5.0.0" are matched.

- Determines whether to use the full clone

  If you specify the `--full` flag, you can clone the official mirror fully.

> **Note:**
>
> If `--full`, `global-version` flags, and the component versions are not specified, only some meta information is cloned.

- Determines whether to clone packages from the specific platform

  If you want to clone packages only for a specific platform, use `-os` and `-arch` to specify the platform. For example:

  - Execute the `tiup mirror clone <target-dir> [global-version] --os=linux` command to clone for linux.
  - Execute the `tiup mirror clone <target-dir> [global-version] --arch=amd64` command to clone for amd64.
  - Execute the `tiup mirror clone <target-dir> [global-version] --os=linux --arch=amd64` command to clone for linux/amd64.

- Determines whether to clone a specific version of a package

  If you want to clone only one version (not all versions) of a component, use `--<component>=<version>` to specify this version. For example:

  - Execute the `tiup mirror clone <target-dir> --tidb v6.1.7` command to clone the v6.1.7 version of the TiDB component.
  - Run the `tiup mirror clone <target-dir> --tidb v6.1.7 --tikv all` command to clone the v6.1.7 version of the TiDB component and all versions of the TiKV component.
  - Run the `tiup mirror clone <target-dir> v6.1.7` command to clone the v6.1.7 version of all components in a cluster.

After cloning, signing keys are set up automatically.

Manage the private repository

You can share the repository cloned using `tiup mirror clone` among hosts either by sharing files via SCP, NFS, or by making the repository available over the HTTP or HTTPS protocol. Use `tiup mirror set <location>` to specify the location of the repository.

```
tiup mirror set /shared_data/tiup
```

```
tiup mirror set https://tiup-mirror.example.com/
```

Note:

If you run `tiup mirror set...` on the machine where you run `tiup mirror clone`, the next time you run `tiup mirror clone...`, the machine clones from the local mirror, not the remote one. Therefore, you need to reset the mirror by running `tiup mirror set --reset` before updating the private mirror.

Another way of using a mirror is to use the `TIUP_MIRRORS` environment variable. Here is an example for running `tiup list` with a private repository.

```
export TIUP_MIRRORS=/shared_data/tiup
tiup list
```

`TIUP_MIRRORS` setting can permanently change the mirror configuration, for example, `tiup mirror set`. For details, see tiup issue #651.

Update the private repository

If you run the `tiup mirror clone` command again with the same `target-dir`, the machine will create new manifests and download the latest versions of components available.

Note:

Before recreating the manifest, ensure that all components and versions (including earlier ones downloaded previously) are included.

13.4.11.3.3 Custom repository

You can create a custom repository to work with TiDB components like TiDB, TiKV, or PD that you build by yourself. It is also possible to create your own tiup components.

To create your own components, run the `tiup package` command and perform as instructed in Component packaging.

Create a custom repository

To create an empty repository in `/data/mirror`:

```
tiup mirror init /data/mirror
```

As part of creating the repository, keys will be written to `/data/mirror/keys`.

To create a new private key in `~/.tiup/keys/private.json`:

```
tiup mirror genkey
```

Grant `jdoe` with private key `~/.tiup/keys/private.json` ownership of `/data/mirror`:

```
tiup mirror set /data/mirror
tiup mirror grant jdoe
```

Work with custom components

1. Create a custom component called hello.

```
$ cat > hello.c << END
> #include <stdio.h>
int main() {
  printf("hello\n");
  return (0);
}
END
$ gcc hello.c -o hello
$ tiup package hello --entry hello --name hello --release v0.0.1
```

`package/hello-v0.0.1-linux-amd64.tar.gz` is created.

2. Create a repository and a private key, and grant ownership to the repository.

```
$ tiup mirror init /tmp/m
$ tiup mirror genkey
$ tiup mirror set /tmp/m
$ tiup mirror grant $USER
```

```
tiup mirror publish hello v0.0.1 package/hello-v0.0.1-linux-amd64.tar.
   ↪ gz hello
```

3. Run the component. If it is not installed yet, it will be downloaded first.

```
$ tiup hello
```

```
The component `hello` version is not installed; downloading from
   ↪ repository.
Starting component `hello`: /home/dvaneeden/.tiup/components/hello/v0
   ↪ .0.1/hello
hello
```

With `tiup mirror merge`, you can merge a repository with custom components into another one. This assumes that all components in `/data/my_custom_components` are signed by the current `$USER`.

```
$ tiup mirror set /data/my_mirror
$ tiup mirror grant $USER
$ tiup mirror merge /data/my_custom_components
```

13.4.11.4 Stress Test TiDB Using TiUP Bench Component

When you test the performance of a database, it is often required to stress test the database. To facilitate this, TiUP has integrated the bench component, which provides two workloads for stress testing: TPC-C and TPC-H. The commands and flags are as follows. For more information, see the TPC official website.

```
tiup bench
```

```
Benchmark database with different workloads

Usage:
  tiup bench [command]

Available Commands:
  help       Help about any command
  tpcc
  tpch

Flags:
      --count int         Total execution count, 0 means infinite
  -D, --db string         Database name (default "test")
  -d, --driver string     Database driver: mysql
      --dropdata          Cleanup data before prepare
  -h, --help              help for /Users/joshua/.tiup/components/bench/v0
     ↪ .0.1/bench
  -H, --host string       Database host (default "127.0.0.1")
      --ignore-error      Ignore error when running workload
      --interval duration Output interval time (default 10s)
      --isolation int     Isolation Level 0: Default, 1: ReadUncommitted,
                          2: ReadCommitted, 3: WriteCommitted, 4:
                             ↪ RepeatableRead,
                          5: Snapshot, 6: Serializable, 7: Linerizable
      --max-procs int     runtime.GOMAXPROCS
  -p, --password string   Database password
  -P, --port int          Database port (default 4000)
      --pprof string      Address of pprof endpoint
      --silence           Don't print error when running workload
      --summary           Print summary TPM only, or also print current TPM
         ↪ when running workload
  -T, --threads int       Thread concurrency (default 16)
      --time duration     Total execution time (default 2562047h47m16
         ↪ .854775807s)
  -U, --user string       Database user (default "root")
```

The following sections describe how to run TPC-C and TPC-H tests using TiUP.

13.4.11.4.1 Run TPC-C test using TiUP

The TiUP bench component supports the following commands and flags to run the TPC-C test:

```
Available Commands:
  check      Check data consistency for the workload
  cleanup    Cleanup data for the workload
  prepare    Prepare data for the workload
  run        Run workload

Flags:
      --check-all      Run all consistency checks
  -h, --help           help for tpcc
      --parts int      Number to partition warehouses (default 1)
      --warehouses int Number of warehouses (default 10)
```

Test procedures

1. Create 4 warehouses using 4 partitions via hash:

```
tiup bench tpcc --warehouses 4 --parts 4 prepare
```

2. Run the TPC-C test:

```
tiup bench tpcc --warehouses 4 run
```

3. Clean up data:

```
tiup bench tpcc --warehouses 4 cleanup
```

4. Check the consistency:

```
tiup bench tpcc --warehouses 4 check
```

5. Generate the CSV file:

```
tiup bench tpcc --warehouses 4 prepare --output-dir data --output-type=
    ↪ csv
```

6. Generate the CSV file for the specified table:

```
tiup bench tpcc --warehouses 4 prepare --output-dir data --output-type=
    ↪ csv --tables history,orders
```

13.4.11.4.2 Run TPC-H test using TiUP

The TiUP bench component supports the following commands and parameters to run the TPC-H test:

```
Available Commands:
  cleanup    Cleanup data for the workload
  prepare    Prepare data for the workload
  run        Run workload

Flags:
      --check          Check output data, only when the scale factor equals 1
  -h, --help           help for tpch
      --queries string All queries (default "q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,
          ↪ q11,q12,q13,q14,q15,q16,q17,q18,q19,q20,q21,q22")
      --sf int         scale factor
```

Test procedures

1. Prepare data:

   ```
   tiup bench tpch --sf=1 prepare
   ```

2. Run the TPC-H test by executing one of the following commands:

   - If you check the result, run this command:

     ```
     tiup bench tpch --sf=1 --check=true run
     ```

   - If you do not check the result, run this command:

     ```
     tiup bench tpch --sf=1 run
     ```

3. Clean up data:

   ```
   tiup bench tpch cleanup
   ```

13.5 TiDB Operator

TiDB Operator is an automatic operation system for TiDB clusters on Kubernetes. It provides full life-cycle management for TiDB including deployment, upgrades, scaling, backup, fail-over, and configuration changes. With TiDB Operator, TiDB can run seamlessly in the Kubernetes clusters deployed on a public or private cloud.

Currently, the TiDB Operator documentation (also named as TiDB on Kubernetes documentation) is independent of the TiDB documentation. To access the documentation, click the following link:

- TiDB on Kubernetes documentation

13.6 TiDB Data Migration

13.6.1 TiDB Data Migration Overview

TiDB Data Migration (DM) is an integrated data migration task management platform, which supports the full data migration and the incremental data replication from MySQL-compatible databases (such as MySQL, MariaDB, and Aurora MySQL) into TiDB. It can help to reduce the operation cost of data migration and simplify the troubleshooting process.

13.6.1.1 Basic features

- **Compatibility with MySQL.** DM is compatible with MySQL 5.7 protocols and most of the features and syntax of MySQL 5.7.
- **Replicating DML and DDL events.** It supports parsing and replicating DML and DDL events in MySQL binlog.
- **Migrating and merging MySQL shards.** DM supports migrating and merging multiple MySQL database instances upstream to one TiDB database downstream. It supports customizing replication rules for different migration scenarios. It can automatically detect and handle DDL changes of upstream MySQL shards, which greatly reduces the operational cost.
- **Various types of filters.** You can predefine event types, regular expressions, and SQL expressions to filter out MySQL binlog events during the data migration process.
- **Centralized management.** DM supports thousands of nodes in a cluster. It can run and manage a large number of data migration tasks concurrently.
- **Optimization of the third-party Online Schema Change process.** In the MySQL ecosystem, tools such as gh-ost and pt-osc are widely used. DM optimizes its change process to avoid unnecessary migration of intermediate data. For details, see online-ddl.
- **High availability.** DM supports data migration tasks to be scheduled freely on different nodes. The running tasks are not affected when a small number of nodes crash.

13.6.1.2 Quick installation

Run the following command to install DM:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
tiup install dm dmctl
```

13.6.1.3 Usage restrictions

Before using the DM tool, note the following restrictions:

- Database version requirements

- MySQL version 5.5 ~ 5.7
- MySQL version 8.0 (experimental features)
- MariaDB version >= 10.1.2 (experimental features)

> **Note:**
>
> If there is a primary-secondary migration structure between the upstream MySQL/MariaDB servers, then choose the following version.
>
> - MySQL version > 5.7.1
> - MariaDB version >= 10.1.3

- DDL syntax compatibility
  - Currently, TiDB is not compatible with all the DDL statements that MySQL supports. Because DM uses the TiDB parser to process DDL statements, it only supports the DDL syntax supported by the TiDB parser. For details, see MySQL Compatibility.
  - DM reports an error when it encounters an incompatible DDL statement. To solve this error, you need to manually handle it using dmctl, either skipping this DDL statement or replacing it with specified DDL statements. For details, see Skip or replace abnormal SQL statements.
  - DM does not replicate view-related DDL statements and DML statements to the downstream TiDB cluster. It is recommended that you create the view in the downstream TiDB cluster manually.
- GBK character set compatibility
  - DM does not support migrating `charset=GBK` tables to TiDB clusters earlier than v5.4.0.

13.6.1.4 Contributing

You are welcome to participate in the DM open sourcing project. Your contribution would be highly appreciated. For more details, see CONTRIBUTING.md.

13.6.1.5 Community support

You can learn about DM through the online documentation. If you have any questions, contact us on GitHub.

13.6.1.6 License

DM complies with the Apache 2.0 license. For more details, see LICENSE.

13.6.1.7 DM versions

Before v5.4, the DM documentation is independent of the TiDB documentation. To access these earlier versions of the DM documentation, click one of the following links:

- DM v5.3 documentation
- DM v2.0 documentation
- DM v1.0 documentation

Note:

- Since October 2021, DM's GitHub repository has been moved to pingcap/tiflow. If you see any issues with DM, submit your issue to the `pingcap/tiflow` repository for feedback.
- In earlier versions (v1.0 and v2.0), DM uses version numbers that are independent of TiDB. Since v5.3, DM uses the same version number as TiDB. The next version of DM v2.0 is DM v5.3. There are no compatibility changes from DM v2.0 to v5.3, and the upgrade process is the same as a normal upgrade, only an increase in version number.

13.6.2 Data Migration Architecture

This document introduces the architecture of Data Migration (DM).

DM consists of three components: DM-master, DM-worker, and dmctl.

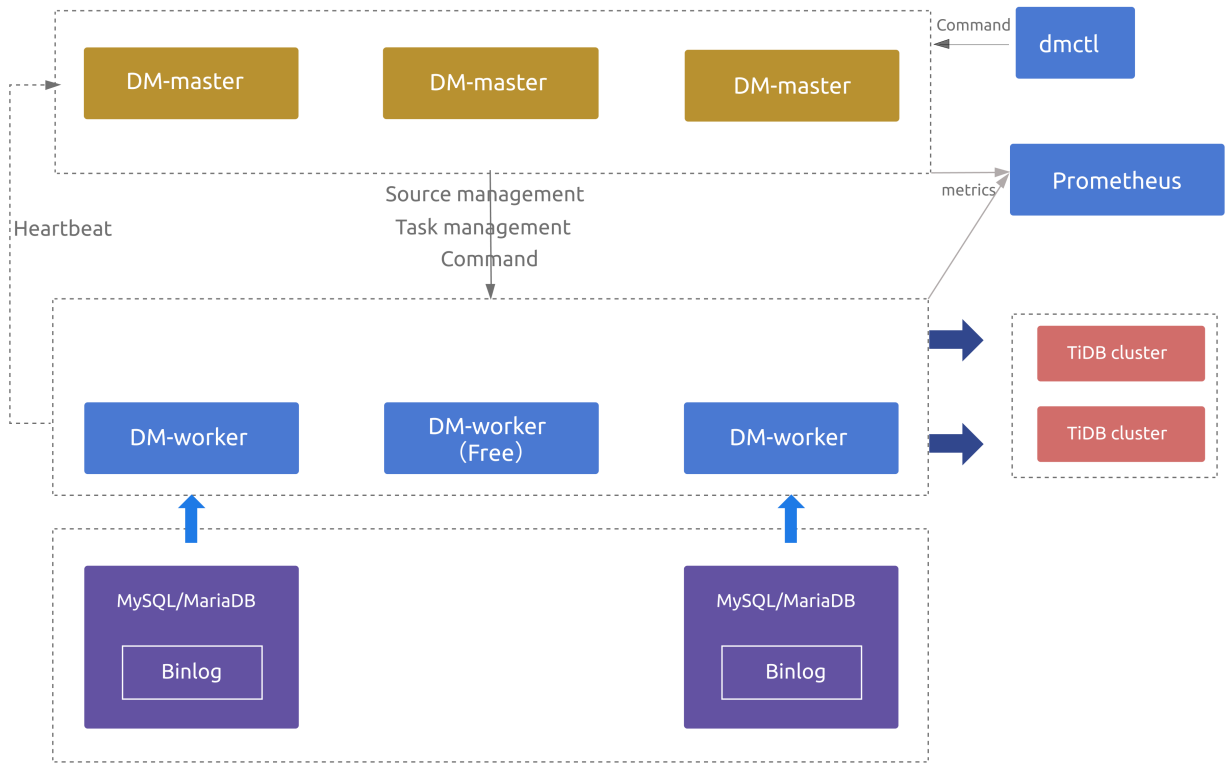


Figure 237: Data Migration architecture

13.6.2.1 Architecture components

13.6.2.1.1 DM-master

DM-master manages and schedules the operations of data migration tasks.

- Storing the topology information of the DM cluster
- Monitoring the running state of DM-worker processes
- Monitoring the running state of data migration tasks
- Providing a unified portal for the management of data migration tasks
- Coordinating the DDL migration of sharded tables in each instance under the sharding scenario

13.6.2.1.2 DM-worker

DM-worker executes specific data migration tasks.

- Persisting the binlog data to the local storage
- Storing the configuration information of the data migration subtasks
- Orchestrating the operation of the data migration subtasks

- Monitoring the running state of the data migration subtasks

For more details of DM-worker, see DM-worker Introduction.

13.6.2.1.3 dmctl

dmctl is a command line tool used to control the DM cluster.

- Creating, updating, or dropping data migration tasks
- Checking the state of data migration tasks
- Handling errors of data migration tasks
- Verifying the configuration correctness of data migration tasks

13.6.2.2 Architecture features

13.6.2.2.1 High availability

When you deploy multiple DM-master nodes, all DM-master nodes use the embedded etcd to form a cluster. The DM-master cluster is used to store metadata such as cluster node information and task configuration. The leader node elected through etcd is used to provide services such as cluster management and data migration task management. Therefore, if the number of available DM-master nodes exceeds half of the deployed nodes, the DM cluster can normally provide services.

When the number of deployed DM-worker nodes exceeds the number of upstream MySQL/MariaDB nodes, the extra DM-worker nodes are idle by default. If a DM-worker node goes offline or is isolated from the DM-master leader, DM-master automatically schedules data migration tasks of the original DM-worker node to other idle DM-worker nodes. (If a DM-worker node is isolated, it automatically stops the data migration tasks on it); if there are no available idle DM-worker nodes, the data migration tasks of the original DM-worker are temporarily hung until one DM-worker node becomes idle, and then the tasks are automatically resumed.

Note:

When the data migration task is in the process of full export or import, the migration task does not support high availability. Here are the main reasons:

- For the full export, MySQL does not support exporting from a specific snapshot point yet. This means that after the data migration task is rescheduled or restarted, the export cannot resume from the previous interruption point.

- For the full import, DM-worker does not support reading exported full data across the nodes yet. This means that after the data migration task is scheduled to a new DM-worker node, you cannot read the exported full data on the original DM-worker node before the scheduling happens.

13.6.3 Quick Start Guide for TiDB Data Migration

This document describes how to migrate data from MySQL to TiDB using TiDB Data Migration (DM). This guide is a quick demo of DM features and is not recommended for any production environment.

13.6.3.1 Step 1: Deploy a DM cluster

1. Install TiUP, and install `dmctl` using TiUP:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
tiup install dm dmctl
```

2. Generate the minimal deployment topology file of a DM cluster:

```
tiup dm template
```

3. Copy the configuration information in the output, and save it as the `topology.yaml` file with the modified IP address. Deploy the DM cluster with the `topology.yaml` file using TiUP:

```
tiup dm deploy dm-test 6.1.7 topology.yaml -p
```

13.6.3.2 Step 2: Prepare the data source

You can use one or multiple MySQL instances as an upstream data source.

1. Create a configuration file for each data source as follows:

```
source-id: "mysql-01"
from:
  host: "127.0.0.1"
  user: "root"
  password: "fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg=" # encrypt with `tiup
      ↪ dmctl --encrypt "123456"`
  port: 3306
```

2. Add the source to the DM cluster by running the following command. `mysql-01.yaml` is the configuration file created in the previous step.

```
tiup dmctl --master-addr=127.0.0.1:8261 operate-source create mysql-01.
    ↪ yaml # use one of master_servers as the argument of --master-
    ↪ addr
```

If you do not have a MySQL instance for testing, you can create a MySQL instance in Docker by taking the following steps:

1. Create a MySQL configuration file:

```
mkdir -p /tmp/mysqltest && cd /tmp/mysqltest

cat > my.cnf <<EOF
[mysqld]
bind-address   = 0.0.0.0
character-set-server=utf8
collation-server=utf8_bin
default-storage-engine=INNODB
transaction-isolation=READ-COMMITTED
server-id      = 100
binlog_format  = row
log_bin        = /var/lib/mysql/mysql-bin.log
show_compatibility_56 = ON
EOF
```

2. Start the MySQL instance using Docker:

```
docker run --name mysql-01 -v /tmp/mysqltest:/etc/mysql/conf.d -e
    ↪ MYSQL_ROOT_PASSWORD=my-secret-pw -d -p 3306:3306 mysql:5.7
```

3. After the MySQL instance is started, access the instance:

> **Note:**
> This command is only suitable for trying out data migration, and cannot be used in production environments or stress tests.

```
mysql -uroot -p -h 127.0.0.1 -P 3306
```

13.6.3.3 Step 3: Prepare a downstream database

You can choose an existing TiDB cluster as a target for data migration.

If you do not have a TiDB cluster for testing, you can quickly build a demonstration environment by running the following command:

```
tiup playground
```

13.6.3.4 Step 4: Prepare test data

Create a test table and data in one or multiple data sources. If you use an existing MySQL database, and the database contains available data, you can skip this step.

```
drop database if exists `testdm`;
create database `testdm`;
use `testdm`;
create table t1 (id bigint, uid int, name varchar(80), info varchar(100),
    ↪ primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=utf8mb4
    ↪ COLLATE=utf8mb4_bin;
create table t2 (id bigint, uid int, name varchar(80), info varchar(100),
    ↪ primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=utf8mb4
    ↪ COLLATE=utf8mb4_bin;
insert into t1 (id, uid, name) values (1, 10001, 'Gabriel García Márquez'),
    ↪  (2, 10002, 'Cien años de soledad');
insert into t2 (id, uid, name) values (3, 20001, 'José Arcadio Buendía'),
    ↪ (4, 20002, 'Úrsula Iguarán'), (5, 20003, 'José Arcadio');
```

13.6.3.5 Step 5: Create a data migration task

1. Create a task configuration file `testdm-task.yaml`:

```
name: testdm
task-mode: all

target-database:
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: "" # If the password is not empty, it is recommended to use
      ↪  a password encrypted with dmctl.

# Configure the information of one or multiple data sources
mysql-instances:
  - source-id: "mysql-01"
    block-allow-list: "ba-rule1"
```

```
block-allow-list:
  ba-rule1:
    do-dbs: ["testdm"]
```

2. Create the task using dmctl:

```
tiup dmctl --master-addr 127.0.0.1:8261 start-task testdm-task.yaml
```

You have successfully created a task that migrates data from a `mysql-01` database to TiDB.

13.6.3.6 Step 6: Check the status of the task

After the task is created, you can use the `dmctl query-status` command to check the status of the task:

```
tiup dmctl --master-addr 127.0.0.1:8261 query-status testdm
```

13.6.4 Deploy a DM cluster

13.6.4.1 Software and Hardware Requirements for TiDB Data Migration

TiDB Data Migration (DM) supports mainstream Linux operating systems. See the following table for specific version requirements:

Linux OS	Version
Red Hat Enterprise Linux	7.3 or later
CentOS	7.3 or later
Oracle Enterprise Linux	7.3 or later
Ubuntu LTS	16.04 or later

DM can be deployed and run on Intel architecture servers and mainstream virtualization environments.

13.6.4.1.1 Recommended server requirements

DM can be deployed and run on a 64-bit generic hardware server platform (Intel x86-64 architecture). For servers used in the development, testing, and production environments, this section illustrates recommended hardware configurations (these do not include the resources used by the operating system).

Development and test environments

Component	CPU	Memory	Local Storage	Network	Number of Instances (Minimum Requirement)
DM-master	4 core+	8 GB+	SAS, 200 GB+	Gigabit network card	1
DM-worker	8 core+	16 GB+	SAS, 200 GB+ (Greater than the size of the migrated data)	Gigabit network card	The number of upstream MySQL instances

Note:

- In the test environment, DM-master and DM-worker used for functional verification can be deployed on the same server.
- To prevent interference with the accuracy of the performance test results, it is **not recommended** to use low-performance storage and network hardware configurations.
- If you need to verify the function only, you can deploy a DM-master on a single machine. The number of DM-worker deployed must be greater than or equal to the number of upstream MySQL instances. To ensure high availability, it is recommended to deploy more DM-workers.
- DM-worker stores full data in the `dump` and `load` phases. Therefore, the disk space for DM-worker needs to be greater than the total amount of data to be migrated. If the relay log is enabled for the migration task, DM-worker needs additional disk space to store upstream binlog data.

Production environment

Component	CPU	Memory	Hard Disk Type	Network	Number of Instances (Minimum Requirement)
DM-master	4 core+	8 GB+	SAS, 200 GB+	Gigabit network card	3
DM-worker	16 core+	32 GB+	SSD, 200 GB+ (Greater than the size of the migrated data)	10 Gigabit network card	Greater than the number of upstream MySQL instances
Monitor	8 core+	16 GB+	SAS, 200 GB+	Gigabit network card	1

Note:

- In the production environment, it is not recommended to deploy and run DM-master and DM-worker on the same server, because when DM-worker writes data to disks, it might interfere with the use of disks by DM-master's high availability component.
- If a performance issue occurs, you are recommended to modify the task configuration file according to the Optimize Configuration of DM document. If the performance is not effectively optimized by tuning the configuration file, you can try to upgrade the hardware of your server.

13.6.4.1.2 Downstream storage space requirements

The target TiKV cluster must have enough disk space to store the imported data. In addition to the standard hardware requirements, the storage space of the target TiKV cluster must be larger than **the size of the data source x the number of replicas x 2**. For example, if the cluster uses 3 replicas by default, the target TiKV cluster must have a storage space larger than 6 times the size of the data source. The formula has `x 2` because:

- Indexes might take extra space.
- RocksDB has a space amplification effect.

You can estimate the data volume by using the following SQL statements to summarize the `data-length` field:

- Calculate the size of all schemas, in MiB. Replace `${schema_name}` with your schema name.

```
select table_schema,sum(data_length)/1024/1024 as data_length,sum(
    ↪ index_length)/1024/1024 as index_length,sum(data_length+
    ↪ index_length)/1024/1024 as sum from information_schema.tables
    ↪ where table_schema = "${schema_name}" group by table_schema;
```

- Calculate the size of the largest table, in MiB. Replace ${schema_name} with your schema name.

```
select table_name,table_schema,sum(data_length)/1024/1024 as
    ↪ data_length,sum(index_length)/1024/1024 as index_length,sum(
    ↪ data_length+index_length)/1024/1024 as sum from
    ↪ information_schema.tables where table_schema = "${schema_name}"
    ↪ group by table_name,table_schema order by sum desc limit 5;
```

13.6.4.2 Deploy a DM Cluster Using TiUP

TiUP is a cluster operation and maintenance tool introduced in TiDB 4.0. TiUP provides TiUP DM, a cluster management component written in Golang. By using TiUP DM, you can easily perform daily TiDB Data Migration (DM) operations, including deploying, starting, stopping, destroying, scaling, and upgrading a DM cluster, and manage DM cluster parameters.

TiUP supports deploying DM v2.0 or later DM versions. This document introduces how to deploy DM clusters of different topologies.

> **Note:**
>
> If your target machine's operating system supports SELinux, make sure that SELinux is **disabled**.

13.6.4.2.1 Prerequisites

When DM performs a full data replication task, the DM-worker is bound with only one upstream database. The DM-worker first exports the full amount of data locally, and then imports the data into the downstream database. Therefore, the worker's host space must be large enough to store all upstream tables to be exported. The storage path is specified later when you create the task.

In addition, you need to meet the hardware and software requirements when deploying a DM cluster.

13.6.4.2.2 Step 1: Install TiUP on the control machine

Log in to the control machine using a regular user account (take the `tidb` user as an example). All the following TiUP installation and cluster management operations can be performed by the `tidb` user.

1. Install TiUP by executing the following command:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

   After the installing, `~/.bashrc` has been modified to add TiUP to PATH, so you need to open a new terminal or redeclare the global environment variables `source ~/.bashrc` to use it.

2. Install the TiUP DM component:

```
tiup install dm dmctl
```

13.6.4.2.3 Step 2: Edit the initialization configuration file

According to the intended cluster topology, you need to manually create and edit the cluster initialization configuration file.

You need to create a YAML configuration file (named `topology.yaml` for example) according to the configuration file template. For other scenarios, edit the configuration accordingly.

You can use the command `tiup dm template > topology.yaml` to generate a configuration file template quickly.

The configuration of deploying three DM-masters, three DM-workers, and one monitoring component instance is as follows:

```
#### The global variables apply to all other components in the configuration
   ↪ . If one specific value is missing in the component instance, the
   ↪ corresponding global variable serves as the default value.
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/dm-deploy"
  data_dir: "/dm-data"

server_configs:
  master:
    log-level: info
    # rpc-timeout: "30s"
    # rpc-rate-limit: 10.0
    # rpc-rate-burst: 40
  worker:
    log-level: info

master_servers:
  - host: 10.0.1.11
    name: master1
    ssh_port: 22
    port: 8261
    # peer_port: 8291
    # deploy_dir: "/dm-deploy/dm-master-8261"
    # data_dir: "/dm-data/dm-master-8261"
    # log_dir: "/dm-deploy/dm-master-8261/log"
    # numa_node: "0,1"
    # The following configs are used to overwrite the `server_configs.master
       ↪ ` values.
    config:
      log-level: info
      # rpc-timeout: "30s"
      # rpc-rate-limit: 10.0
      # rpc-rate-burst: 40
  - host: 10.0.1.18
    name: master2
    ssh_port: 22
    port: 8261
  - host: 10.0.1.19
```

```
    name: master3
    ssh_port: 22
    port: 8261
#### If you do not need to ensure high availability of the DM cluster,
    ↪ deploy only one DM-master node, and the number of deployed DM-worker
    ↪ nodes must be no less than the number of upstream MySQL/MariaDB
    ↪ instances to be migrated.
#### To ensure high availability of the DM cluster, it is recommended to
    ↪ deploy three DM-master nodes, and the number of deployed DM-worker
    ↪ nodes must exceed the number of upstream MySQL/MariaDB instances to
    ↪ be migrated (for example, the number of DM-worker nodes is two more
    ↪ than the number of upstream instances).
worker_servers:
  - host: 10.0.1.12
    ssh_port: 22
    port: 8262
    # deploy_dir: "/dm-deploy/dm-worker-8262"
    # log_dir: "/dm-deploy/dm-worker-8262/log"
    # numa_node: "0,1"
    # The following configs are used to overwrite the `server_configs.worker
       ↪ ` values.
    config:
      log-level: info
  - host: 10.0.1.19
    ssh_port: 22
    port: 8262

monitoring_servers:
  - host: 10.0.1.13
    ssh_port: 22
    port: 9090
    # deploy_dir: "/tidb-deploy/prometheus-8249"
    # data_dir: "/tidb-data/prometheus-8249"
    # log_dir: "/tidb-deploy/prometheus-8249/log"

grafana_servers:
  - host: 10.0.1.14
    port: 3000
    # deploy_dir: /tidb-deploy/grafana-3000

alertmanager_servers:
  - host: 10.0.1.15
    ssh_port: 22
    web_port: 9093
    # cluster_port: 9094
```

```
# deploy_dir: "/tidb-deploy/alertmanager-9093"
# data_dir: "/tidb-data/alertmanager-9093"
# log_dir: "/tidb-deploy/alertmanager-9093/log"
```

Note:

- It is not recommended to run too many DM-workers on one host. Each DM-worker should be allocated at least 2 core CPU and 4 GiB memory.
- Make sure that the ports among the following components are interconnected:
  - The `peer_port` (`8291` by default) among the DM-master nodes are interconnected.
  - Each DM-master node can connect to the `port` of all DM-worker nodes (`8262` by default).
  - Each DM-worker node can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-worker nodes (`8262` by default).

For more `master_servers.host.config` parameter description, refer to master parameter. For more `worker_servers.host.config` parameter description, refer to worker parameter.

13.6.4.2.4 Step 3: Execute the deployment command

Note:

You can use secret keys or interactive passwords for security authentication when you deploy TiDB using TiUP:

- If you use secret keys, you can specify the path of the keys through `-i` or `--identity_file`;
- If you use passwords, add the `-p` flag to enter the password interaction window;
- If password-free login to the target machine has been configured, no authentication is required.

```
tiup dm deploy ${name} ${version} ./topology.yaml -u ${ssh_user} [-p] [-i /
    ↪ home/root/.ssh/gcp_rsa]
```

The parameters used in this step are as follows.

Parameter	Description
`${name}`	The name of the DM cluster, eg: dm-test
`${version}`	The version of the DM cluster. You can see other supported versions by running `tiup list dm-master`.

Parameter	Description
`./topology.yaml`	The topology of the topology configuration file.

Parameter	Description
`-u` or `--user`	Log in to the target machine as the root user or other user account with ssh and sudo privileges to complete the cluster deployment.

Parameter	Description
`-p` or `--password`	The password of target hosts. If specified, password authentication is used.

Parameter	Description
-i or --identity_file	The path of the SSH identity file. If specified, public key authentication is used (default "/root/.ssh/id_rsa").

At the end of the output log, you will see `Deployed cluster `dm-test` successfully`. This indicates that the deployment is successful.

13.6.4.2.5 Step 4: Check the clusters managed by TiUP

```
tiup dm list
```

TiUP supports managing multiple DM clusters. The command above outputs information of all the clusters currently managed by TiUP, including the name, deployment user, version, and secret key information:

```
Name User Version Path                                       PrivateKey
---- ---- ------- ----                                       ----------
dm-test tidb ${version} /root/.tiup/storage/dm/clusters/dm-test /root/.tiup
   ↪ /storage/dm/clusters/dm-test/ssh/id_rsa
```

13.6.4.2.6 Step 5: Check the status of the deployed DM cluster

To check the status of the `dm-test` cluster, execute the following command:

```
tiup dm display dm-test
```

Expected output includes the instance ID, role, host, listening port, and status (because the cluster is not started yet, so the status is `Down`/`inactive`), and directory information.

13.6.4.2.7 Step 6: Start the DM cluster

```
tiup dm start dm-test
```

If the output log includes `Started cluster` `` `dm-test` `` `successfully`, the start is successful.

13.6.4.2.8 Step 7: Verify the running status of the DM cluster

Check the DM cluster status using TiUP:

```
tiup dm display dm-test
```

If the `Status` is `Up` in the output, the cluster status is normal.

13.6.4.2.9 Step 8: Managing migration tasks using dmctl

dmctl is a command-line tool used to control DM clusters. You are recommended to use dmctl via TiUP.

dmctl supports both the command mode and the interactive mode. For details, see Maintain DM Clusters Using dmctl.

13.6.4.3 Deploy a DM Cluster Offline Using TiUP

This document describes how to deploy a DM cluster offline using TiUP.

13.6.4.3.1 Step 1: Prepare the TiUP offline component package

- Install the TiUP package manager online.

  1. Install the TiUP tool:

     ```
     curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.
        ↪ com/install.sh | sh
     ```

  2. Redeclare the global environment variables:

     ```
     source .bash_profile
     ```

3. Confirm whether TiUP is installed:

```
which tiup
```

- Pull the mirror using TiUP

1. Pull the needed components on a machine that has access to the Internet:

```
# You can modify ${version} to the needed version.
tiup mirror clone tidb-dm-${version}-linux-amd64 --os=linux --arch=
    ↪ amd64 \
    --dm-master=${version} --dm-worker=${version} --dmctl=${version
        ↪ } \
    --alertmanager=v0.17.0 --grafana=v4.0.3 --prometheus=v4.0.3 \
    --tiup=v$(tiup --version|grep 'tiup'|awk -F ' ' '{print $1}')
        ↪ --dm=v$(tiup --version|grep 'tiup'|awk -F ' ' '{print $1
        ↪ }')
```

The command above creates a directory named `tidb-dm-${version}-linux-amd64` in the current directory, which contains the component package managed by TiUP.

2. Pack the component package by using the `tar` command and send the package to the control machine in the isolated environment:

```
tar czvf tidb-dm-${version}-linux-amd64.tar.gz tidb-dm-${version}-
    ↪ linux-amd64
```

`tidb-dm-${version}-linux-amd64.tar.gz` is an independent offline environment package.

13.6.4.3.2 Step 2: Deploy the offline TiUP component

After sending the package to the control machine of the target cluster, install the TiUP component by running the following command:

```
#### You can modify ${version} to the needed version.
tar xzvf tidb-dm-${version}-linux-amd64.tar.gz
sh tidb-dm-${version}-linux-amd64/local_install.sh
source /home/tidb/.bash_profile
```

The `local_install.sh` script automatically executes the `tiup mirror set tidb-dm-${version}-linux-amd64` command to set the current mirror address to `tidb-dm-${version}-linux-amd64`.

To switch the mirror to another directory, manually execute the `tiup mirror set <mirror-dir>` command. If you want to switch back to the official mirror, execute `tiup mirror set https://tiup-mirrors.pingcap.com`.

13.6.4.3.3 Step 3: Edit the initialization configuration file

You need to edit the cluster initialization configuration file according to different cluster topologies.

For the full configuration template, refer to the TiUP configuration parameter template. Create a configuration file `topology.yaml`. In other combined scenarios, edit the configuration file as needed according to the templates.

The configuration of deploying three DM-masters, three DM-workers, and one monitoring component instance is as follows:

```
---
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/home/tidb/dm/deploy"
  data_dir: "/home/tidb/dm/data"
  # arch: "amd64"

master_servers:
  - host: 172.19.0.101
  - host: 172.19.0.102
  - host: 172.19.0.103

worker_servers:
  - host: 172.19.0.101
  - host: 172.19.0.102
  - host: 172.19.0.103

monitoring_servers:
  - host: 172.19.0.101

grafana_servers:
  - host: 172.19.0.101

alertmanager_servers:
  - host: 172.19.0.101
```

Note:

- If you do not need to ensure high availability of the DM cluster, deploy only one DM-master node, and the number of deployed DM-worker nodes must be no less than the number of upstream MySQL/MariaDB instances to be migrated.

- To ensure high availability of the DM cluster, it is recommended to deploy three DM-master nodes, and the number of deployed DM-worker nodes must be greater than the number of upstream MySQL/MariaDB instances to be migrated (for example, the number of DM-worker nodes is two more than the number of upstream instances).
- For parameters that should be globally effective, configure these parameters of corresponding components in the `server_configs` section of the configuration file.
- For parameters that should be effective on a specific node, configure these parameters in `config` of this node.
- Use `.` to indicate the subcategory of the configuration, such as `log.slow-threshold`. For more formats, see TiUP configuration template.
- For more parameter description, see master `config.toml.example` and worker `config.toml.example`.
- Make sure that the ports among the following components are interconnected:
  - The `peer_port` (`8291` by default) among the DM-master nodes are interconnected.
  - Each DM-master node can connect to the `port` of all DM-worker nodes (`8262` by default).
  - Each DM-worker node can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-master nodes (`8261` by default).
  - The TiUP nodes can connect to the `port` of all DM-worker nodes (`8262` by default).

13.6.4.3.4 Step 4: Execute the deployment command

Note:

You can use secret keys or interactive passwords for security authentication when you deploy DM using TiUP:

- If you use secret keys, you can specify the path of the keys through `-i` or `--identity_file`;
- If you use passwords, add the `-p` flag to enter the password interaction window;

- If password-free login to the target machine has been configured, no authentication is required.

```
tiup dm deploy dm-test ${version} ./topology.yaml --user root [-p] [-i /home
    ↪ /root/.ssh/gcp_rsa]
```

In the above command:

- The name of the deployed DM cluster is `dm-test`.
- The version of the DM cluster is `${version}`. You can view the latest versions supported by TiUP by running `tiup list dm-master`.
- The initialization configuration file is `topology.yaml`.
- `--user root`: Log in to the target machine through the `root` key to complete the cluster deployment, or you can use other users with `ssh` and `sudo` privileges to complete the deployment.
- `[-i]` and `[-p]`: optional. If you have configured login to the target machine without password, these parameters are not required. If not, choose one of the two parameters. `[-i]` is the private key of the `root` user (or other users specified by `--user`) that has access to the target machine. `[-p]` is used to input the user password interactively.
- TiUP DM uses the embedded SSH client. If you want to use the SSH client native to the control machine system, edit the configuration according to using the system's native SSH client to connect to the cluster.

At the end of the output log, you will see `Deployed cluster `dm-test` successfully`. This indicates that the deployment is successful.

13.6.4.3.5 Step 5: Check the clusters managed by TiUP

```
tiup dm list
```

TiUP supports managing multiple DM clusters. The command above outputs information of all the clusters currently managed by TiUP, including the name, deployment user, version, and secret key information:

```
Name  User  Version Path                                       PrivateKey
----  ----  ------- ----                                       ----------
dm-test tidb ${version} /root/.tiup/storage/dm/clusters/dm-test /root/.tiup
    ↪ /storage/dm/clusters/dm-test/ssh/id_rsa
```

13.6.4.3.6 Step 6: Check the status of the deployed DM cluster

To check the status of the `dm-test` cluster, execute the following command:

```
tiup dm display dm-test
```

Expected output includes the instance ID, role, host, listening port, and status (because the cluster is not started yet, so the status is `Down/inactive`), and directory information of the `dm-test` cluster.

13.6.4.3.7 Step 7: Start the cluster

```
tiup dm start dm-test
```

If the output log includes `Started cluster` `` `dm-test` `` `successfully`, the start is successful.

13.6.4.3.8 Step 8: Verify the running status of the cluster

Check the DM cluster status using TiUP:

```
tiup dm display dm-test
```

If the `Status` is `Up` in the output, the cluster status is normal.

13.6.4.4 Deploy Data Migration Using DM Binary

This document introduces how to quickly deploy the Data Migration (DM) cluster using DM binary.

> **Note:**
>
> In the production environment, it is recommended to use TiUP to deploy a DM cluster.

13.6.4.4.1 Download DM binary

The DM binary is included in the TiDB Toolkit. To download the TiDB Toolkit, see Download TiDB Tools.

13.6.4.4.2 Sample scenario

Suppose that you deploy a DM cluster based on this sample scenario:

Two DM-worker nodes and three DM-master nodes are deployed on five servers.

Here is the address of each node:

Instance	Server address	Port
DM-master1	192.168.0.4	8261
DM-master2	192.168.0.5	8261
DM-master3	192.168.0.6	8261
DM-worker1	192.168.0.7	8262
DM-worker2	192.168.0.8	8262

Based on this scenario, the following sections describe how to deploy the DM cluster.

Note:

- If you deploy multiple DM-master or DM-worker instances in a single server, the port and working directory of each instance must be unique.
- If you do not need to ensure high availability of the DM cluster, deploy only one DM-master node, and the number of deployed DM-worker nodes must be no less than the number of upstream MySQL/MariaDB instances to be migrated.
- To ensure high availability of the DM cluster, it is recommended to deploy three DM-master nodes, and the number of deployed DM-worker nodes must be greater than the number of upstream MySQL/MariaDB instances to be migrated (for example, the number of DM-worker nodes is two more than the number of upstream instances).
- Make sure that the ports among the following components are interconnected:
  - The `8291` ports among the DM-master nodes are interconnected.
  - Each DM-master node can connect to the `8262` ports of all DM-worker nodes.
  - Each DM-worker node can connect to the `8261` port of all DM-master nodes.

Deploy DM-master

You can configure DM-master by using command-line parameters or the configuration file.

DM-master command-line parameters

The following is the description of DM-master command-line parameters:

```
./dm-master --help
```

```
Usage of dm-master:
  -L string
        log level: debug, info, warn, error, fatal (default "info")
  -V    prints version and exit
  -advertise-addr string
        advertise address for client traffic (default "${master-addr}")
  -advertise-peer-urls string
        advertise URLs for peer traffic (default "${peer-urls}")
  -config string
        path to config file
  -data-dir string
        path to the data directory (default "default.${name}")
  -initial-cluster string
        initial cluster configuration for bootstrapping, e.g. dm-master=http
           ↪ ://127.0.0.1:8291
  -join string
        join to an existing cluster (usage: cluster's "${master-addr}" list,
           ↪ e.g. "127.0.0.1:8261,127.0.0.1:18261"
  -log-file string
        log file path
  -master-addr string
        master API server and status addr
  -name string
        human-readable name for this DM-master member
  -peer-urls string
        URLs for peer traffic (default "http://127.0.0.1:8291")
  -print-sample-config
        print sample config file of dm-worker
```

> **Note:**
>
> In some situations, you cannot use the above method to configure DM-master because some configurations are not exposed to the command line. In such cases, use the configuration file instead.

DM-master configuration file

The following is the configuration file of DM-master. It is recommended that you configure DM-master by using this method.

1. Write the following configuration to `conf/dm-master1.toml`:

“‘toml # Master Configuration. name = “master1”

Log configurations. log-level = “info” log-file = “dm-master.log”

The listening address of DM-master. master-addr = “192.168.0.4:8261”

The peer URLs of DM-master. peer-urls = “192.168.0.4:8291”

The value of `initial-cluster` is the combination of the `advertise-peer-urls` value of all DM-master nodes in the initial cluster. initial-cluster = “master1=http://192.168.0.4:8291,master2=http://192.168.0.5:8291,master3=http://192.168.0.6:8291”
“‘

2. Execute the following command in the terminal to run DM-master:

   ```
   bash  ./dm-master -config conf/dm-master1.toml
   ```

> **Note:**
> The console does not output logs after this command is executed. If you want to view the runtime log, you can execute `tail -f dm-master.log`.

3. For DM-master2 and DM-master3, change `name` in the configuration file to `master2` and `master3` respectively, and change `peer-urls` to `192.168.0.5:8291` and `192.168.0.6:8291` respectively. Then repeat Step 2.

Deploy DM-worker

You can configure DM-worker by using command-line parameters or the configuration file.

DM-worker command-line parameters

The following is the description of the DM-worker command-line parameters:

```
./dm-worker --help
```

```
Usage of worker:
  -L string
        log level: debug, info, warn, error, fatal (default "info")
  -V    prints version and exit
  -advertise-addr string
        advertise address for client traffic (default "${worker-addr}")
  -config string
        path to config file
  -join string
        join to an existing cluster (usage: dm-master cluster's "${master-addr}")
  -keepalive-ttl int
```

```
      dm-worker's TTL for keepalive with etcd (in seconds) (default 10)
-log-file string
      log file path
-name string
      human-readable name for DM-worker member
-print-sample-config
      print sample config file of dm-worker
-worker-addr string
      listen address for client traffic
```

> **Note:**
>
> In some situations, you cannot use the above method to configure DM-worker because some configurations are not exposed to the command line. In such cases, use the configuration file instead.

DM-worker configuration file

The following is the DM-worker configuration file. It is recommended that you configure DM-worker by using this method.

1. Write the following configuration to `conf/dm-worker1.toml`:

   “‘toml # Worker Configuration. name = “worker1”

   # Log configuration. log-level = “info” log-file = “dm-worker.log”

   # DM-worker address. worker-addr = “:8262”

   # The master-addr configuration of the DM-master nodes in the cluster. join = “192.168.0.4:8261,192.168.0.5:8261,192.168.0.6:8261” “‘

2. Execute the following command in the terminal to run DM-worker:

   ```
   bash  ./dm-worker -config conf/dm-worker1.toml
   ```

3. For DM-worker2, change `name` in the configuration file to `worker2`. Then repeat Step 2.

Now, a DM cluster is successfully deployed.

13.6.4.5 Use Kubernetes

13.6.5 Tutorials

13.6.5.1 Create a Data Source for TiDB Data Migration

> **Note:**
>
> Before creating a data source, you need to Deploy a DM Cluster Using TiUP.

The document describes how to create a data source for the data migration task of TiDB Data Migration (DM).

A data source contains the information for accessing the upstream migration task. Because a data migration task requires referring its corresponding data source to obtain the configuration information of access, you need to create the data source of a task before creating a data migration task. For specific data source management commands, refer to Manage Data Source Configurations.

13.6.5.1.1 Step 1: Configure the data source

1. (optional) Encrypt the data source password

   In DM configuration files, it is recommended to use the password encrypted with dmctl. You can follow the example below to obtain the encrypted password of the data source, which can be used to write the configuration file later.

```
tiup dmctl encrypt 'abc!@#123'
```

```
MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
```

2. Write the configuration file of the data source

   For each data source, you need an individual configuration file to create it. You can follow the example below to create a data source whose ID is "mysql-01". First create the configuration file `./source-mysql-01.yaml`:

```
source-id: "mysql-01" # The ID of the data source, you can refer this
    ↪ source-id in the task configuration and dmctl command to
    ↪ associate the corresponding data source.

from:
  host: "127.0.0.1"
  port: 3306
  user: "root"
```

```
  password: "MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=" # The user password
     ↪ of the upstream data source. It is recommended to use the
     ↪ password encrypted with dmctl.
  security:                                    # The TLS configuration of
     ↪  the upstream data source. If not necessary, it can be deleted.
    ssl-ca: "/path/to/ca.pem"
    ssl-cert: "/path/to/cert.pem"
    ssl-key: "/path/to/key.pem"
```

13.6.5.1.2 Step 2: Create a data source

You can use the following command to create a data source:

```
tiup dmctl --master-addr <master-addr> operate-source create ./source-mysql
   ↪ -01.yaml
```

For other configuration parameters, refer to Upstream Database Configuration File.

The returned results are as follows:

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-01",
            "worker": "dm-worker-1"
        }
    ]
}
```

13.6.5.1.3 Step 3: Query the data source you created

After creating a data source, you can use the following command to query the data source:

- If you konw the `source-id` of the data source, you can use the `dmctl config source <source-id>` command to directly check the configuration of the data source:

```
tiup dmctl --master-addr <master-addr> config source mysql-01
```

```
{
  "result": true,
  "msg": "",
```

```
  "cfg": "enable-gtid: false
    flavor: mysql
    source-id: mysql-01
    from:
      host: 127.0.0.1
      port: 3306
      user: root
      password: '******'
}
```

- If you do not know the `source-id`, you can use the `dmctl operate-source show` command to check the source database list, from which you can find the corresponding data source.

```
tiup dmctl --master-addr <master-addr> operate-source show
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "source is added but there is no free worker to bound
                ↪ ",
            "source": "mysql-02",
            "worker": ""
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-01",
            "worker": "dm-worker-1"
        }
    ]
}
```

13.6.5.2 Manage Data Source Configurations in TiDB Data Migration

This document introduces how to manage data source configurations, including encrypting the MySQL password, operating the data source, and changing the bindings between upstream MySQL instances and DM-workers using dmctl.

13.6.5.2.1 Encrypt the database password

In DM configuration files, it is recommended to use the password encrypted with dmctl. For one original password, the encrypted password is different after each encryption.

```
./dmctl -encrypt 'abc!@#123'
```

```
MKxn0Qo3m3XOyjCnhEMtsUCm83EhGQDZ/T4=
```

13.6.5.2.2 Operate data source

You can use the `operate-source` command to load, list or remove the data source configurations to the DM cluster.

```
help operate-source
```

```
`create`/`stop`/`show` upstream MySQL/MariaDB source.

Usage:
  dmctl operate-source <operate-type> [config-file ...] [--print-sample-
    ↪ config] [flags]

Flags:
  -h, --help                help for operate-source
  -p, --print-sample-config print sample config file of source

Global Flags:
  -s, --source strings MySQL Source ID
```

Flags description

- `create`: Creates one or more upstream database sources. When creating multiple data sources fails, DM rolls back to the state where the command was not executed.
- `stop`: Stops one or more upstream database sources. When stopping multiple data sources fails, some data sources might be stopped.
- `show`: Shows the added data source and the corresponding DM-worker.
- `config-file`: Specifies the file path of `source.yaml` and can pass multiple file paths.
- `--print-sample-config`: Prints the sample configuration file. This parameter ignores other parameters.

Usage example

Use the following `operate-source` command to create a source configuration file:

```
operate-source create ./source.yaml
```

For the configuration of `source.yaml`, refer to Upstream Database Configuration File Introduction.

The following is an example of the returned result:

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "dm-worker-1"
        }
    ]
}
```

Check data source configurations

> **Note:**
>
> The `config` command is only supported in DM v6.0 and later versions. For earlier versions, you must use the `get-config` command.

If you know the `source-id`, you can run `dmctl --master-addr <master-addr> config source <source-id>` to get the data source configuration.

```
config source mysql-replica-01
```

```
{
  "result": true,
    "msg": "",
    "cfg": "enable-gtid: false
      flavor: mysql
      source-id: mysql-replica-01
      from:
        host: 127.0.0.1
        port: 8407
        user: root
        password: '******'
}
```

If you don't know the `source-id`, you can run `dmctl --master-addr <master-addr> operate-source show` to list all data sources first.

```
operate-source show
```


```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "source is added but there is no free worker to bound",
            "source": "mysql-replica-02",
            "worker": ""
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "dm-worker-1"
        }
    ]
}
```


13.6.5.2.3 Change the bindings between upstream MySQL instances and DM-workers

You can use the `transfer-source` command to change the bindings between upstream MySQL instances and DM-workers.

```
help transfer-source
```

```
Transfers an upstream MySQL/MariaDB source to a free worker.
Usage:
  dmctl transfer-source <source-id> <worker-id> [flags]
Flags:
  -h, --help help for transfer-source
Global Flags:
  -s, --source strings MySQL Source ID.
```

Before transferring, DM checks whether the worker to be unbound still has running tasks. If the worker has any running tasks, you need to pause the tasks first, change the binding, and then resume the tasks.

Usage example

If you do not know the bindings of DM-workers, you can run `dmctl --master-addr <master-addr> list-member --worker` to list the current bindings of all workers.

```
list-member --worker
```

```
{
    "result": true,
    "msg": "",
    "members": [
        {
            "worker": {
                "msg": "",
                "workers": [
                    {
                        "name": "dm-worker-1",
                        "addr": "127.0.0.1:8262",
                        "stage": "bound",
                        "source": "mysql-replica-01"
                    },
                    {
                        "name": "dm-worker-2",
                        "addr": "127.0.0.1:8263",
                        "stage": "free",
                        "source": ""
                    }
                ]
            }
        }
    ]
}
```

In the above example, `mysql-replica-01` is bound to `dm-worker-1`. The below command transfers the binding worker of `mysql-replica-01` to `dm-worker-2`.

```
transfer-source mysql-replica-01 dm-worker-2
```

```
{
    "result": true,
    "msg": ""
}
```

Check whether the command takes effect by running `dmctl --master-addr <master-addr> list-member --worker`.

```
list-member --worker
```

```
{
    "result": true,
    "msg": "",
    "members": [
        {
            "worker": {
                "msg": "",
                "workers": [
                    {
                        "name": "dm-worker-1",
                        "addr": "127.0.0.1:8262",
                        "stage": "free",
                        "source": ""
                    },
                    {
                        "name": "dm-worker-2",
                        "addr": "127.0.0.1:8263",
                        "stage": "bound",
                        "source": "mysql-replica-01"
                    }
                ]
            }
        }
    ]
}
```


13.6.5.3 Data Migration Task Configuration Guide

This document introduces how to configure a data migration task in Data Migration (DM).

13.6.5.3.1 Configure data sources to be migrated

Before configuring the data sources to be migrated for the task, you need to first make sure that DM has loaded the configuration files of the corresponding data sources. The following are some operation references:

- To view the data source, you can refer to Check the data source configuration.
- To create a data source, you can refer to Create data source.
- To generate a data source configuration file, you can refer to Source configuration file introduction.

The following example of `mysql-instances` shows how to configure data sources that need to be migrated for the data migration task:

```
---

##### ********* Basic configuration *********
name: test          # The name of the task. Should be globally unique.

##### ******** Data source configuration **********
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source whose
    ↪ `source-id` is `mysql-replica-01`.
  - source-id: "mysql-replica-02" # Migrate data from the data source whose
    ↪ `source-id` is `mysql-replica-02`.
```

13.6.5.3.2 Configure the downstream TiDB cluster

The following example of `target-database` shows how to configure the target TiDB cluster to be migrated to for the data migration task:

```
---

##### ********* Basic configuration *********
name: test          # The name of the task. Should be globally unique.

##### ******** Data source configuration **********
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source whose
    ↪ `source-id` is `mysql-replica-01`.
  - source-id: "mysql-replica-02" # Migrate data from the data source whose
    ↪ `source-id` is `mysql-replica-02`.

##### ******** Downstream TiDB database configuration **********
target-database:    # Configuration of target TiDB database.
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""      # If the password is not null, it is recommended to use
    ↪  a password encrypted with dmctl.
```

13.6.5.3.3 Configure tables to be migrated

Note:

> If you do not need to filter specific tables or migrate specific tables, skip this configuration.

To configure the block and allow list of data source tables for the data migration task, perform the following steps:

1. Configure a global filter rule set of the block and allow list in the task configuration file.

```
block-allow-list:
  bw-rule-1:                          # The name of the block and allow
    ↪ list rule.
    do-dbs: ["test.*", "user"]     # The allow list of upstream schemas
      ↪ to be migrated. Wildcard characters (*?) are supported. You
      ↪ only need to configure either `do-dbs` or `ignore-dbs`. If
      ↪ both fields are configured, only `do-dbs` takes effect.
    # ignore-dbs: ["mysql", "account"] # The block list of upstream
      ↪ schemas to be migrated. Wildcard characters (*?) are
      ↪ supported.
    do-tables:                        # The allow list of upstream tables
      ↪ to be migrated. You only need to configure either `do-tables`
      ↪  or `ignore-tables`. If both fields are configured, only `do-
      ↪ tables` takes effect.
    - db-name: "test.*"
      tbl-name: "t.*"
    - db-name: "user"
      tbl-name: "information"
  bw-rule-2:                        # The name of the block allow list
    ↪ rule.
    ignore-tables:                # The block list of data source tables
      ↪  needs to be migrated.
    - db-name: "user"
      tbl-name: "log"
```

For detailed configuration rules, see Block and allow table lists.

2. Reference the block and allow list rules in the data source configuration to filter tables to be migrated.

```
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source
    ↪ whose `source-id` is `mysql-replica-01`.
```

```
  block-allow-list: "bw-rule-1" # The name of the block and allow
     ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
     ↪ use `black-white-list` instead.
- source-id: "mysql-replica-02" # Migrate data from the data source
   ↪ whose `source-id` is `mysql-replica-02`.
  block-allow-list: "bw-rule-2" # The name of the block and allow
     ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
     ↪ use `black-white-list` instead.
```

13.6.5.3.4 Configure binlog events to be migrated

> **Note:**
>
> If you do not need to filter specific binlog events of certain schemas or tables, skip this configuration.

To configure the filters of binlog events for the data migration task, perform the following steps:

1. Configure a global filter rule set of binlog events in the task configuration file.

```
filters:                                  # The filter rule set of data
   ↪  source binlog events. You can set multiple rules at the same
   ↪ time.
  filter-rule-1:                          # The name of the filtering
     ↪ rule.
    schema-pattern: "test_*"              # The pattern of the data
       ↪ source schema name. Wildcard characters (*?) are supported.
    table-pattern: "t_*"                  # The pattern of the data
       ↪ source table name. Wildcard characters (*?) are supported.
    events: ["truncate table", "drop table"] # The event types to be
       ↪ filtered out in schemas or tables that match the `schema-
       ↪ pattern` or the `table-pattern`.
    action: Ignore                        # Whether to migrate (Do) or
       ↪ ignore (Ignore) the binlog that matches the filtering rule.
  filter-rule-2:
    schema-pattern: "test"
    events: ["all dml"]
    action: Do
```

For detailed configuration rules, see Binlog event filter.

2. Reference the binlog event filtering rules in the data source configuration to filter specified binlog events of specified tables or schemas in the data source.

```
mysql-instances:
  - source-id: "mysql-replica-01" # Migrate data from the data source
      ↪ whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-1" # The name of the block and allow
       ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
       ↪ use `black-white-list` instead.
    filter-rules: ["filter-rule-1"] # The name of the rule that filters
       ↪  specific binlog events of the data source. You can configure
       ↪  multiple rules here.
  - source-id: "mysql-replica-02" # Migrate data from the data source
      ↪ whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-2" # The name of the block and allow
       ↪ list rule. If the DM version is earlier than v2.0.0-beta.2,
       ↪ use `black-white-list` instead.
    filter-rules: ["filter-rule-2"] # The name of the rule that filters
       ↪  specific binlog events of the data source. You can configure
       ↪  multiple rules here.
```

13.6.5.3.5 Configure the mapping of data source tables to downstream TiDB tables

> **Note:**
>
> - If you do not need to migrate a certain table of the data source to the table with a different name in the downstream TiDB instance, skip this configuration.
> - If it is a shard merge task, you **must** set mapping rules in the task configuration file.

To configure the routing mapping rules for migrating data source tables to specified downstream TiDB tables, perform the following steps:

1. Configure a global routing mapping rule set in the task configuration file.

```
routes:                          # The routing mapping rule set between
   ↪ the data source tables and downstream TiDB tables. You can set
   ↪ multiple rules at the same time.
  route-rule-1:                  # The name of the routing mapping rule.
```

```
    schema-pattern: "test_*"  # The pattern of the upstream schema name
      ↪ . Wildcard characters (*?) are supported.
    table-pattern: "t_*"      # The pattern of the upstream table name.
      ↪  Wildcard characters (*?) are supported.
    target-schema: "test"     # The name of the downstream TiDB schema.
    target-table: "t"         # The name of the downstream TiDB table.
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

For detailed configuration rules, see Table Routing.

2. Reference the routing mapping rules in the data source configuration to filter tables to be migrated.

```
mysql-instances:
  - source-id: "mysql-replica-01"               # Migrate data from the
    ↪ data source whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-1"               # The name of the block
      ↪ and allow list rule. If the DM version is earlier than v2
      ↪ .0.0-beta.2, use `black-white-list` instead.
    filter-rules: ["filter-rule-1"]             # The name of the rule
      ↪ that filters specific binlog events of the data source. You
      ↪ can configure multiple rules here.
    route-rules: ["route-rule-1", "route-rule-2"] # The name of the
      ↪ routing mapping rule. You can configure multiple rules here.
  - source-id: "mysql-replica-02"               # Migrate data from the
    ↪ data source whose `source-id` is `mysql-replica-02`.
    block-allow-list: "bw-rule-2"               # The name of the block
      ↪ and allow list rule. If the DM version is earlier than v2
      ↪ .0.0-beta.2, use `black-white-list` instead.
    filter-rules: ["filter-rule-2"]             # The name of the rule
      ↪ that filters specific binlog events of the data source. You
      ↪ can configure multiple rules here.
```

13.6.5.3.6 Configure a shard merge task

Note:

- If you need to migrate sharding DDL statements in a shard merge scenario, you **must** explicitly configure the `shard-mode` field. Otherwise, **DO NOT** configure `shard-mode` at all.

- Migrating sharding DDL statements is likely to cause many issues. Make sure you understand the principles and restrictions of DM migrating DDL statements before using this feature, and you **must** use this feature with caution.

The following example shows how to configure the task as a shard merge task:

```
---

##### ********* Basic information *********
name: test                  # The name of the task. Should be globally
    ↪ unique.
shard-mode: "pessimistic"   # The shard merge mode. Optional modes are ""/"
    ↪ pessimistic"/"optimistic". The "" mode is used by default which means
    ↪  sharding DDL merge is disabled. If the task is a shard merge task,
    ↪ set it to the "pessimistic" mode. After getting a deep understanding
    ↪ of the principles and restrictions of the "optimistic" mode, you can
    ↪ set it to the "optimistic" mode.
```

13.6.5.3.7 Other configurations

The following is an overall task configuration example of this document. The complete task configuration template can be found in DM task configuration file full introduction.

```
---

##### ********* Basic configuration *********
name: test                  # The name of the task. Should be globally
    ↪ unique.
shard-mode: "pessimistic"   # The shard merge mode. Optional modes are ""/"
    ↪ pessimistic"/"optimistic". The "" mode is used by default which means
    ↪  sharding DDL merge is disabled. If the task is a shard merge task,
    ↪ set it to the "pessimistic" mode. After getting a deep understanding
    ↪ of the principles and restrictions of the "optimistic" mode, you can
    ↪ set it to the "optimistic" mode.
task-mode: all              # The task mode. Can be set to `full`(only
    ↪ migrates full data)/`incremental`(replicates binlog synchronously)/`
    ↪ all` (replicates both full and incremental binlogs).
timezone: "UTC"           # The timezone used in SQL Session. By default,
    ↪ DM uses the global timezone setting in the target cluster, which
    ↪ ensures the correctness automatically. A customized timezone does not
    ↪  affect data migration but is unnecessary.
```

```
##### ******** Data source configuration **********
mysql-instances:
  - source-id: "mysql-replica-01"            # Migrate data from the data
    ↪ source whose `source-id` is `mysql-replica-01`.
    block-allow-list: "bw-rule-1"            # The name of the block and
       ↪ allow list rule. If the DM version is earlier than v2.0.0-beta.2,
       ↪ use `black-white-list` instead.
    filter-rules: ["filter-rule-1"]          # The name of the rule that
       ↪ filters specific binlog events of the data source. You can
       ↪ configure multiple rules here.
    route-rules: ["route-rule-1", "route-rule-2"] # The name of the routing
       ↪ mapping rule. You can configure multiple rules here.
  - source-id: "mysql-replica-02"            # Migrate data from the data
    ↪ source whose `source-id` is `mysql-replica-02`.
    block-allow-list: "bw-rule-2"            # The name of the block and
       ↪ allow list rule. If the DM version is earlier than v2.0.0-beta.2,
       ↪ use `black-white-list` instead.
    filter-rules: ["filter-rule-2"]          # The name of the rule that
       ↪ filters specific binlog events of the data source. You can
       ↪ configure multiple rules here.
    route-rules: ["route-rule-2"]            # The name of the routing
       ↪ mapping rule. You can configure multiple rules here.

##### ******** Downstream TiDB instance configuration **********
target-database:    # Configuration of the downstream database instance.
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: ""      # If the password is not null, it is recommended to use
     ↪  a password encrypted with dmctl.

##### ******** Feature configuration set **********
#### The filter rule set of tables to be migrated from the upstream database
   ↪  instance. You can set multiple rules at the same time.
block-allow-list:                  # Use black-white-list if the DM version
   ↪ is earlier than v2.0.0-beta.2.
  bw-rule-1:                       # The name of the block and allow list
     ↪ rule.
    do-dbs: ["test.*", "user"]     # The allow list of upstream schemas to
       ↪ be migrated. Wildcard characters (*?) are supported. You only need
       ↪  to configure either `do-dbs` or `ignore-dbs`. If both fields are
       ↪ configured, only `do-dbs` takes effect.
    # ignore-dbs: ["mysql", "account"] # The block list of upstream schemas
       ↪ to be migrated. Wildcard characters (*?) are supported.
    do-tables:                       # The allow list of upstream tables to be
```

```
      ↪  migrated. You only need to configure either `do-tables` or `
      ↪ ignore-tables`. If both fields are configured, only `do-tables`
      ↪ takes effect.
    - db-name: "test.*"
      tbl-name: "t.*"
    - db-name: "user"
      tbl-name: "information"
  bw-rule-2:                    # The name of the block allow list rule.
    ignore-tables:              # The block list of data source tables
      ↪ needs to be migrated.
    - db-name: "user"
      tbl-name: "log"

#### The filter rule set of data source binlog events.
filters:                                # You can set multiple rules at
   ↪ the same time.
  filter-rule-1:                        # The name of the filtering rule.
    schema-pattern: "test_*"            # The pattern of the data source
      ↪ schema name. Wildcard characters (*?) are supported.
    table-pattern: "t_*"                # The pattern of the data source
      ↪ table name. Wildcard characters (*?) are supported.
    events: ["truncate table", "drop table"] # The event types to be
      ↪ filtered out in schemas or tables that match the `schema-pattern`
      ↪ or the `table-pattern`.
    action: Ignore                      # Whether to migrate (Do) or
      ↪ ignore (Ignore) the binlog that matches the filtering rule.
  filter-rule-2:
    schema-pattern: "test"
    events: ["all dml"]
    action: Do

#### The routing mapping rule set between the data source and target TiDB
   ↪ instance tables.
routes:                       # You can set multiple rules at the same time.
  route-rule-1:               # The name of the routing mapping rule.
    schema-pattern: "test_*"  # The pattern of the data source schema name.
      ↪ Wildcard characters (*?) are supported.
    table-pattern: "t_*"      # The pattern of the data source table name.
      ↪ Wildcard characters (*?) are supported.
    target-schema: "test"     # The name of the downstream TiDB schema.
    target-table: "t"         # The name of the downstream TiDB table.
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

13.6.5.4 TiDB Data Migration Shard Merge

TiDB Data Migration (DM) supports merging the DML and DDL data in the upstream MySQL/MariaDB sharded tables and migrating the merged data to the downstream TiDB tables.

If you need to migrate and merge MySQL shards of small datasets to TiDB, refer to this tutorial.

13.6.5.4.1 Restrictions

Currently, the shard merge feature is supported only in limited scenarios. For details, refer to Sharding DDL usage Restrictions in the pessimistic mode and Sharding DDL usage Restrictions in the optimistic mode.

13.6.5.4.2 Configure parameters

In the task configuration file, set `shard-mode` to `pessimistic`:

```
shard-mode: "pessimistic"
#### The shard merge mode. Optional modes are ""/"pessimistic"/"optimistic".
   ↪  The "" mode is used by default
#### which means sharding DDL merge is disabled. If the task is a shard
   ↪ merge task, set it to the "pessimistic"
#### mode. After getting a deep understanding of the principles and
   ↪ restrictions of the "optimistic" mode, you
#### can set it to the "optimistic" mode.
```

13.6.5.4.3 Handle sharding DDL locks manually

In some abnormal scenarios, you need to handle sharding DDL Locks manually.

13.6.5.5 TiDB Data Migration Table Routing

When you migrate data using TiDB Data Migration (DM), you can configure the table routing to migrate a certain table of the upstream MySQL or MariaDB instance to the specified table in the downstream.

Note:

- Configuring multiple different routing rules for a single table is not supported.
- The match rule of schema needs to be configured separately, which is used to migrate `CREATE/DROP SCHEMA xx`, as shown in `rule-2` of the Configure table routing section.

13.6.5.5.1 Configure table routing

```
routes:
  rule-1:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    target-schema: "test"
    target-table: "t"
    # extract-table, extract-schema, and extract-source are optional and
    # are required only when you need to extract information about sharded
    # tables, sharded schemas, and source datatabase information.
    extract-table:
      table-regexp: "t_(.*)"
      target-column: "c_table"
    extract-schema:
      schema-regexp: "test_(.*)"
      target-column: "c_schema"
    extract-source:
      source-regexp: "(.*)"
      target-column: "c_source"
  rule-2:
    schema-pattern: "test_*"
    target-schema: "test"
```

In simple scenarios, it is recommended that you use the wildcard for matching schemas and tables. However, note the following version differences:

- For DM v1.0.5 or later versions, the table routing supports the wildcard match, but there can be **only one** `*` in the wildcard expression, and `*` **must be placed at the end**.
- For DM versions earlier than v1.0.5, the table routing supports the wildcard but does not support the `[...]` and `[!...]` expressions.

13.6.5.5.2 Parameter descriptions

- DM migrates the upstream MySQL or MariaDB instance tables that match the `schema-pattern/table-pattern` rule provided by Table selector to the downstream `target-schema/target-table`.
- For sharded tables that match the `schema-pattern/table-pattern` rules, DM extracts the table name by using the `extract-table.table-regexp` regular expression, the schema name by using the `extract-schema.schema-regexp` regular expression, and source information by using the `extract-source.source-regexp` regular expression. Then DM writes the extracted information to the corresponding `target-column` in the merged table in the downstream.

13.6.5.5.3 Usage examples

This section shows the usage examples in four different scenarios.

If you need to migrate and merge MySQL shards of small datasets to TiDB, refer to this tutorial.

Merge sharded schemas and tables

Assuming in the scenario of sharded schemas and tables, you want to migrate the `test_{1,2,3...}.t_{1,2,3...}` tables in two upstream MySQL instances to the `test.t` table in the downstream TiDB instance.

To migrate the upstream instances to the downstream `test.t`, you must create the following routing rules:

- `rule-1` is used to migrate DML or DDL statements of the table that matches `schema-pattern: "test_*"` and `table-pattern: "t_*"` to the downstream `test.t`.
- `rule-2` is used to migrate DDL statements of the schema that matches `schema-pattern: "test_*"`, such as `CREATE/DROP SCHEMA xx`.

Note:

- If the downstream `schema: test` already exists and is not to be deleted, you can omit `rule-2`.
- If the downstream `schema: test` does not exist and only `rule-1` is configured, then it reports the `schema test doesn't exist` error during migration.

```
rule-1:
  schema-pattern: "test_*"
  table-pattern: "t_*"
  target-schema: "test"
  target-table: "t"
rule-2:
  schema-pattern: "test_*"
  target-schema: "test"
```

Extract table, schema, and source information and write into the merged table

Assuming in the scenario of sharded schemas and tables, you want to migrate the `test_{1,2,3...}.t_{1,2,3...}` tables in two upstream MySQL instances to the `test.t` table in the downstream TiDB instance. At the same time, you want to extract the source information of the sharded tables and write it to the downstream merged table.

To migrate the upstream instances to the downstream `test.t`, you must create routing rules similar to the previous section Merge sharded schemas and tables. In addtion, you need to add the `extract-table`, `extract-schema`, and `extract-source` configurations:

- `extract-table`: For a sharded table matching `schema-pattern` and `table-pattern`, DM extracts the sharded table name by using `table-regexp` and writes the name suffix without the `t_` part to `target-column` of the merged table, that is, the `c_table` column.
- `extract-schema`: For a sharded schema matching `schema-pattern` and `table-pattern`, DM extracts the sharded schema name by using `schema-regexp` and writes the name suffix without the `test_` part to `target-column` of the merged table, that is, the `c_schema` column.
- `extract-source`: For a sharded table matching `schema-pattern` and `table-pattern`, DM writes the source instance information to the `target-column` of the merged table, that is, the `c_source` column.

```
rule-1:
  schema-pattern: "test_*"
  table-pattern: "t_*"
  target-schema: "test"
  target-table: "t"
  extract-table:
    table-regexp: "t_(.*)"
    target-column: "c_table"
  extract-schema:
    schema-regexp: "test_(.*)"
    target-column: "c_schema"
  extract-source:
    source-regexp: "(.*)"
    target-column: "c_source"
rule-2:
  schema-pattern: "test_*"
  target-schema: "test"
```

To extract the source information of upstream sharded tables to the merged table in the downstream, you **must manually create a merged table in the downstream before starting the migration**. The merged table must contain the three `target-columns` (`c_table`, `c_schema`, and `c_source`) used for specifying the source information. In addition, these columns **must be the last columns and be string types**.

```
CREATE TABLE `test`.`t` (
    a int(11) PRIMARY KEY,
    c_table varchar(10) DEFAULT NULL,
    c_schema varchar(10) DEFAULT NULL,
    c_source varchar(10) DEFAULT NULL
```

```
);
```

Assume that the upstream has the following two data sources:

Data source `mysql-01`:

```
mysql> select * from test_11.t_1;
+---+
| a |
+---+
| 1 |
+---+
mysql> select * from test_11.t_2;
+---+
| a |
+---+
| 2 |
+---+
mysql> select * from test_12.t_1;
+---+
| a |
+---+
| 3 |
+---+
```

Data source `mysql-02`:

```
mysql> select * from test_13.t_3;
+---+
| a |
+---+
| 4 |
+---+
```

After migration using DM, data in the merged table will be as follows:

```
mysql> select * from test.t;
+---+---------+----------+----------+
| a | c_table | c_schema | c_source |
+---+---------+----------+----------+
| 1 | 1       | 11       | mysql-01 |
| 2 | 2       | 11       | mysql-01 |
| 3 | 1       | 12       | mysql-01 |
| 4 | 3       | 13       | mysql-02 |
+---+---------+----------+----------+
```

Incorrect examples of creating merged tables

> **Note:**
>
> If any of the following errors occur, source information of sharded tables and schemas might fail to be written to the merged table.

- `c-table` is not in the last three columns:

```
CREATE TABLE `test`.`t` (
    c_table varchar(10) DEFAULT NULL,
    a int(11) PRIMARY KEY,
    c_schema varchar(10) DEFAULT NULL,
    c_source varchar(10) DEFAULT NULL
);
```

- `c-source` is absent:

```
CREATE TABLE `test`.`t` (
    a int(11) PRIMARY KEY,
    c_table varchar(10) DEFAULT NULL,
    c_schema varchar(10) DEFAULT NULL,
);
```

- `c_schema` is not a string type:

```
CREATE TABLE `test`.`t` (
    a int(11) PRIMARY KEY,
    c_table varchar(10) DEFAULT NULL,
    c_schema int(11) DEFAULT NULL,
    c_source varchar(10) DEFAULT NULL,
);
```

Merge sharded schemas

Assuming in the scenario of sharded schemas, you want to migrate the `test_{1,2,3...}` ↪ `.t_{1,2,3...}` tables in the two upstream MySQL instances to the `test.t_{1,2,3...}` tables in the downstream TiDB instance.

To migrate the upstream schemas to the downstream `test.t_[1,2,3]`, you only need to create one routing rule.

```
  rule-1:
    schema-pattern: "test_*"
    target-schema: "test"
```

Incorrect table routing

Assuming that the following two routing rules are configured and `test_1_bak.t_1_bak` matches both `rule-1` and `rule-2`, an error is reported because the table routing configuration violates the number limitation.

```
  rule-1:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    target-schema: "test"
    target-table: "t"
  rule-2:
    schema-pattern: "test_1_bak"
    table-pattern: "t_1_bak"
    target-schema: "test"
    target-table: "t_bak"
```

13.6.5.6 TiDB Data Migration Block and Allow Lists

When you migrate data using TiDB Data Migration (DM), you can configure the block and allow lists to filter or only migrate all operations of some databases or some tables.

13.6.5.6.1 Configure the block and allow lists

In the task configuration file, add the following configuration:

```
block-allow-list:          # Use black-white-list if the DM version is
   ↪ earlier than or equal to v2.0.0-beta.2.
  rule-1:
    do-dbs: ["test*"]      # Starting with characters other than "~"
       ↪ indicates that it is a wildcard;
                           # v1.0.5 or later versions support the regular
                              ↪ expression rules.
    do-tables:
    - db-name: "test[123]" # Matches test1, test2, and test3.
      tbl-name: "t[1-5]"   # Matches t1, t2, t3, t4, and t5.
    - db-name: "test"
      tbl-name: "t"
  rule-2:
    do-dbs: ["~^test.*"]   # Starting with "~" indicates that it is a
       ↪ regular expression.
    ignore-dbs: ["mysql"]
```

```
do-tables:
- db-name: "~^test.*"
  tbl-name: "~^t.*"
- db-name: "test"
  tbl-name: "t"
ignore-tables:
- db-name: "test"
  tbl-name: "log"
```

In simple scenarios, it is recommended that you use the wildcard for matching schemas and tables. However, note the following version differences:

- For DM v1.0.5 or later versions, the block and allow lists support the wildcard match, but there can be **only one** `*` in the wildcard expression, and `*` **must be placed at the end**.
- For DM versions earlier than v1.0.5, the block and allow lists only support regular expression matching.

13.6.5.6.2 Parameter descriptions

- `do-dbs`: allow lists of the schemas to be migrated, similar to `replicate-do-db` in MySQL.
- `ignore-dbs`: block lists of the schemas to be migrated, similar to `replicate-ignore-db` in MySQL.
- `do-tables`: allow lists of the tables to be migrated, similar to `replicate-do-table` in MySQL. Both `db-name` and `tbl-name` must be specified.
- `ignore-tables`: block lists of the tables to be migrated, similar to `replicate-ignore-table` in MySQL. Both `db-name` and `tbl-name` must be specified.

If a value of the above parameters starts with the `~` character, the subsequent characters of this value are treated as a regular expression. You can use this parameter to match schema or table names.

13.6.5.6.3 Filtering process

- The filtering rules corresponding to `do-dbs` and `ignore-dbs` are similar to the Evaluation of Database-Level Replication and Binary Logging Options in MySQL.
- The filtering rules corresponding to `do-tables` and `ignore-tables` are similar to the Evaluation of Table-Level Replication Options in MySQL.

Note:

In DM and in MySQL, the block and allow lists filtering rules are different in the following ways:

- In MySQL, `replicate-wild-do-table` and `replicate-wild-ignore-table` support wildcard characters. In DM, some parameter values directly supports regular expressions that start with the `~` character.
- DM currently only supports binlogs in the `ROW` format, and does not support those in the `STATEMENT` or `MIXED` format. Therefore, the filtering rules in DM correspond to those in the `ROW` format in MySQL.
- MySQL determines a DDL statement only by the database name explicitly specified in the `USE` section of the statement. DM determines a statement first based on the database name section in the DDL statement. If the DDL statement does not contain such a section, DM determines the statement by the `USE` section. Suppose that the SQL statement to be determined is `USE test_db_2; CREATE TABLE test_db_1.test_table (c1 INT PRIMARY KEY)`; that `replicate-do-db=test_db_1` is configured in MySQL and `do-dbs: ["test_db_1"]` is configured in DM. Then this rule only applies to DM and not to MySQL.

The filtering process of a `test.t` table is as follows:

1. Filter at the **schema** level:
    - If `do-dbs` is not empty, check whether a matched schema exists in `do-dbs`.
        - If yes, continue to filter at the **table** level.
        - If not, filter `test.t`.
    - If `do-dbs` is empty and `ignore-dbs` is not empty, check whether a matched schema exits in `ignore-dbs`.
        - If yes, filter `test.t`.
        - If not, continue to filter at the **table** level.
    - If both `do-dbs` and `ignore-dbs` are empty, continue to filter at the **table** level.
2. Filter at the **table** level:
    1. If `do-tables` is not empty, check whether a matched table exists in `do-tables`.
        - If yes, migrate `test.t`.
        - If not, filter `test.t`.

2. If `ignore-tables` is not empty, check whether a matched table exists in `ignore-tables`.
    - If yes, filter `test.t`.
    - If not, migrate `test.t`.
3. If both `do-tables` and `ignore-tables` are empty, migrate `test.t`.

> **Note:**
>
> To check whether the schema `test` should be filtered, you only need to filter at the schema level.

13.6.5.6.4 Usage examples

Assume that the upstream MySQL instances include the following tables:

```
`logs`.`messages_2016`
`logs`.`messages_2017`
`logs`.`messages_2018`
`forum`.`users`
`forum`.`messages`
`forum_backup_2016`.`messages`
`forum_backup_2017`.`messages`
`forum_backup_2018`.`messages`
```

The configuration is as follows:

```
block-allow-list: # Use black-white-list if the DM version is earlier than
    ↪ or equal to v2.0.0-beta.2.
  bw-rule:
    do-dbs: ["forum_backup_2018", "forum"]
    ignore-dbs: ["~^forum_backup_"]
    do-tables:
    - db-name: "logs"
      tbl-name: "~_2018$"
    - db-name: "~^forum.*"
       tbl-name: "messages"
    ignore-tables:
    - db-name: "~.*"
       tbl-name: "^messages.*"
```

After applying the `bw-rule` rule:

Table	Whether to filter	Why filter
`logs` ↪ `.messages_2016`	Yes	The schema `logs` fails to match any `do-dbs`.
`logs` ↪ `.messages_2017`	Yes	The schema `logs` fails to match any `do-dbs`.
`logs` ↪ `.messages_2018`	Yes	The schema `logs` fails to match any `do-dbs`.
`forum_backup_2016` ↪ `.messages`	Yes	The schema `forum_backup_2016` ↪ fails to match any `do-dbs`.
`forum_backup_2017` ↪ `.messages`	Yes	The schema `forum_backup_2017` ↪ fails to match any `do-dbs`.
`forum` ↪ `.users`	Yes	1. The schema `forum` matches `do-dbs` and continues to filter at the table level. 2. The schema and table fail to match any of `do-tables` and `ignore-tables` and `do-tables` is not empty.

Table	Whether to filter	Why filter
`forum` ↪ `.messages` ↪	No	1. The schema `forum` matches `do-dbs` and continues to filter at the table level. 2. The table `messages` is in the `db-name: "~^ ↪ forum.*", ↪ tbl-name: ↪ "messages"` of `do-tables`.
`forum_backup_2018` ↪ `.messages` ↪	No	The schema `forum_backup_2018` ↪ matches `do-dbs` and continues to filter at the table level. 2. The schema and table match the `db-name: "~^ ↪ forum.*", ↪ tbl-name: ↪ "messages"` of `do-tables`.

13.6.5.7 TiDB Data Migration Binlog Event Filter

TiDB Data Migration (DM) provides the binlog event filter feature to filter out or only receive specified types of binlog events for some schemas or tables. For example, you can filter out all `TRUNCATE TABLE` or `INSERT` events. The binlog event filter feature is more fine-grained than the block and allow lists feature.

13.6.5.7.1 Configure the binlog event filter

In the task configuration file, add the following configuration:

```
filters:
  rule-1:
```

```
  schema-pattern: "test_*"
   table-pattern: "t_*"
   events: ["truncate table", "drop table"]
  sql-pattern: ["^DROP\\s+PROCEDURE", "^CREATE\\s+PROCEDURE"]
   action: Ignore
```

Starting from DM v2.0.2, you can configure the binlog event filter in the source configuration file. For details, see Upstream Database Configuration File.

In simple scenarios, it is recommended that you use the wildcard for matching schemas and tables. However, note the following version differences:

- For DM v1.0.5 or later versions, the binlog event filter supports the wildcard match, but there can be **only one** `*` in the wildcard expression, and `*` **must be placed at the end**.
- For DM versions earlier than v1.0.5, the binlog event filter supports the wildcard but does not support the `[...]` and `[!...]` expressions.

13.6.5.7.2 Parameter descriptions

- `schema-pattern/table-pattern`: the binlog events or DDL SQL statements of upstream MySQL or MariaDB instance tables that match `schema-pattern/table-pattern` are filtered by the rules below.
- `events`: the binlog event array. You can only select one or more `Events` from the following table:

Events	Type	Description
`all`		Includes all the events below
`all dml`		Includes all DML events below
`all ddl`		Includes all DDL events below
`none`		Includes none of the events below
`none ddl`		Includes none of the DDL events below
`none dml`		Includes none of the DML events below
`insert`	DML	The `INSERT` DML event
`update`	DML	The `UPDATE` DML event
`delete`	DML	The `DELETE` DML event
`create database`	DDL	The `CREATE DATABASE` DDL event
`drop database`	DDL	The `DROP DATABASE` DDL event
`create table`	DDL	The `CREATE TABLE` DDL event
`create index`	DDL	The `CREATE INDEX` DDL event
`drop table`	DDL	The `DROP TABLE` DDL event
`truncate table`	DDL	The `TRUNCATE TABLE` DDL event
`rename table`	DDL	The `RENAME TABLE` DDL event

Events	Type	Description
`drop index`	DDL	The `DROP INDEX` DDL event
`alter table`	DDL	The `ALTER TABLE` DDL event

- `sql-pattern`: it is used to filter specified DDL SQL statements. The matching rule supports using a regular expression. For example, `"^DROP\\s+PROCEDURE"`.
- `action`: the string (`Do`/`Ignore`). Based on the following rules, it judges whether to filter. If either of the two rules is satisfied, the binlog is filtered; otherwise, the binlog is not filtered.
    - `Do`: the allow list. The binlog is filtered in either of the following two conditions:
        - The type of the event is not in the `event` list of the rule.
        - The SQL statement of the event cannot be matched by `sql-pattern` of the rule.
    - `Ignore`: the block list. The binlog is filtered in either of the following two conditions:
        - The type of the event is in the `event` list of the rule.
        - The SQL statement of the event can be matched by `sql-pattern` of the rule.
    - When multiple rules match the same table, the rules are applied sequentially. The block list has a higher priority than the allow list. For example, if both the `Ignore` and `Do` rules are applied to the same table, the `Ignore` rule takes effect.

13.6.5.7.3 Usage examples

This section shows the usage examples in the scenario of sharding (sharded schemas and tables).

Filter all sharding deletion operations

To filter out all deletion operations, configure the following two filtering rules:

- `filter-table-rule` filters out the `TRUNCATE TABLE`, `DROP TABLE` and `DELETE STATEMENT` operations of all tables that match the `test_*.t_*` pattern.
- `filter-schema-rule` filters out the `DROP DATABASE` operation of all schemas that match the `test_*` pattern.

```
filters:
  filter-table-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    events: ["truncate table", "drop table", "delete"]
    action: Ignore
  filter-schema-rule:
```

```
    schema-pattern: "test_*"
    events: ["drop database"]
    action: Ignore
```

Only migrate sharding DML statements

To only migrate sharding DML statements, configure the following two filtering rules:

- `do-table-rule` only migrates the `CREATE TABLE`, `INSERT`, `UPDATE` and `DELETE` statements of all tables that match the `test_*.t_*` pattern.
- `do-schema-rule` only migrates the `CREATE DATABASE` statement of all schemas that match the `test_*` pattern.

> **Note:**
>
> The reason why the `CREATE DATABASE/TABLE` statement is migrated is that you can migrate DML statements only after the schema and table are created.

```
filters:
  do-table-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    events: ["create table", "all dml"]
    action: Do
  do-schema-rule:
    schema-pattern: "test_*"
    events: ["create database"]
    action: Do
```

Filter out the SQL statements that TiDB does not support

To filter out the `PROCEDURE` statements that TiDB does not support, configure the following `filter-procedure-rule`:

```
filters:
  filter-procedure-rule:
    schema-pattern: "test_*"
    table-pattern: "t_*"
    sql-pattern: ["^DROP\\s+PROCEDURE", "^CREATE\\s+PROCEDURE"]
    action: Ignore
```

`filter-procedure-rule` filters out the `^CREATE\\s+PROCEDURE` and `^DROP\\s+PROCEDURE` statements of all tables that match the `test_*.t_*` pattern.

Filter out the SQL statements that the TiDB parser does not support

For the SQL statements that the TiDB parser does not support, DM cannot parse them and get the `schema/table` information. So you must use the global filtering rule: `schema-pattern: "*"`.

> **Note:**
>
> To avoid filtering out data that need to be migrated, you must configure the global filtering rule as strictly as possible.

To filter out the `PARTITION` statements that the TiDB parser (of some version) does not support, configure the following filtering rule:

```
filters:
  filter-partition-rule:
    schema-pattern: "*"
    sql-pattern: ["ALTER\\s+TABLE[\\s\\S]*ADD\\s+PARTITION", "ALTER\\s+TABLE
       ↪ [\\s\\S]*DROP\\s+PARTITION"]
    action: Ignore
```

13.6.5.8 Filter DMLs Using SQL Expressions

13.6.5.8.1 Overview

In the process of incremental data migration, you can filter certain types of binlog events using the Filter Binlog Events feature. For example, for archiving or auditing purposes, you can filter out `DELETE` events when migrating data to the downstream. However, Binlog Event Filter cannot judge with a greater granularity on whether to filter out a specific row of `DELETE` events.

To solve the above issue, DM supports filtering data during incremental migration using `binlog value filter` since v2.0.5. The binlog in the `ROW` format supported by DM has the values of all columns in binlog events. You can configure SQL expressions according to these values. If the SQL expressions evaluate a row change as `TRUE`, DM will not migrate the row change downstream.

For detailed operation and implementation, see Filter DML Events Using SQL Expressions.

13.6.5.9 TiDB Data Migration Support for Online DDL Tools

In the MySQL ecosystem, tools such as gh-ost and pt-osc are widely used. TiDB Data Migration (DM) provides supports for these tools to avoid migrating unnecessary intermediate data.

This document introduces the support for common online DDL tools, usage, and precautions in DM.

For the working principles and implementation methods of DM for online DDL tools, refer to online-ddl.

13.6.5.9.1 Restrictions

- DM only supports gh-ost and pt-osc.
- When `online-ddl` is enabled, the checkpoint corresponding to incremental replication should not be in the process of online DDL execution. For example, if an upstream online DDL operation starts at `position-A` and ends at `position-B` of the binlog, the starting point of incremental replication should be earlier than `position-A` or later than `position-B`; otherwise, an error occurs. For details, refer to FAQ.

13.6.5.9.2 Configure parameters

In v2.0.5 and later versions, you need to use the `online-ddl` configuration item in the `task` configuration file.

- If the upstream MySQL/MariaDB (at the same time) uses the gh-ost or pt-osc tool, set `online-ddl` to `true` in the task configuration file:

```
online-ddl: true
```

> **Note:**
>
> Since v2.0.5, `online-ddl-scheme` has been deprecated, so you need to use `online-ddl` instead of `online-ddl-scheme`. That means that setting `online-ddl: true` overwrites `online-ddl-scheme`, and setting `online-ddl-scheme: "pt"` or `online-ddl-scheme: "gh-ost"` is converted to `online-ddl: true`.

Before v2.0.5 (not including v2.0.5), you need to use the `online-ddl-scheme` configuration item in the `task` configuration file.

- If the upstream MySQL/MariaDB uses the gh-ost tool, set it in the task configuration file:

```
online-ddl-scheme: "gh-ost"
```

- If the upstream MySQL/MariaDB uses the pt tool, set it in the task configuration file:

```
online-ddl-scheme: "pt"
```

13.6.5.10 Manage a Data Migration Task

13.6.5.10.1 Migration Task Precheck

Before using DM to migrate data from upstream to downstream, a precheck helps detect errors in the upstream database configurations and ensures that the migration goes smoothly. This document introduces the DM precheck feature, including its usage scenario, check items, and arguments.

Usage scenario

To run a data migration task smoothly, DM triggers a precheck automatically at the start of the task and returns the check results. DM starts the migration only after the precheck is passed.

To trigger a precheck manually, run the `check-task` command.

For example:

```
tiup dmctl check-task ./task.yaml
```

Descriptions of check items

After a precheck is triggered for a task, DM checks the corresponding items according to your migration mode configuration.

This section lists all the precheck items.

> **Note:**
>
> In this document, check items that must be passed are labeled "(Mandatory)".

- If a mandatory check item does not pass, DM returns an error after the check and does not proceed with the migration task. In this case, modify the configurations according to the error message and retry the task after meeting the precheck requirements.
- If a non-mandatory check item does not pass, DM returns a warning after the check. DM automatically starts a migration task if the check result contains only warnings but no errors.

Common check items

Regardless of the migration mode you choose, the precheck always includes the following common check items:

- Database version
    - MySQL version > 5.5
    - MariaDB version >= 10.1.2

> **Warning:**
>
> - Migrating data from MySQL 8.0 to TiDB using DM is an experimental feature (introduced since DM v2.0). It is NOT recommended that you use it in a production environment.
> - Migrating data from MariaDB to TiDB using DM is an experimental feature. It is NOT recommended that you use it in a production environment.

- Compatibility of the upstream MySQL table schema
    - Check whether the upstream tables have foreign keys, which are not supported by TiDB. A warning is returned if a foreign key is found in the precheck.
    - (Mandatory) Check whether there are compatibility differences in character sets. For more information, see TiDB Supported Character Sets.
    - (Mandatory) Check whether the upstream tables have primary key constraints or unique key constraints (introduced from v1.0.7)

Check items for full data migration

For the full data migration mode (`task-mode: full`), in addition to the common check items, the precheck also includes the following check items:

- (Mandatory) dump permission of the upstream database
    - SELECT permission on INFORMATION_SCHEMA and dump tables
    - RELOAD permission if `consistency=flush`
    - LOCK TABLES permission on the dump tables if `consistency=flush/lock`
- (Mandatory) Consistency of upstream MySQL multi-instance sharding tables
    - In the pessimistic mode, check whether the table schemas of all sharded tables are consistent in the following items:
        - Number of columns
        - Column name
        - Column order
        - Column type
        - Primary key
        - Unique index

    - In the optimistic mode, check whether the schemas of all sharded tables meet the optimistic compatibility.
    - If a migration task was started successfully by the `start-task` command, the precheck of this task skips the consistency check.

- Auto-increment primary key in sharded tables

    - If sharded tables have auto-increment primary keys, the precheck returns a warning. If there are conflicts in auto-increment primary keys, see Handle conflicts of auto-increment primary key for solutions.

Check items for incremental data migration

For the incremental data migration mode (`task-mode: incremental`), in addition to the common check items, the precheck also includes the following check items:

- (Mandatory) Upstream database REPLICATION permission

    - REPLICATION CLIENT permission
    - REPLICATION SLAVE permission

- (Mandatory) Database primary-secondary configuration

    - The database ID `server_id` of the upstream database must be specified (GTID is recommended for non-AWS Aurora environments).

- (Mandatory) MySQL binlog configuration

    - Check whether binlog is enabled (required by DM).
    - Check whether `binlog_format=ROW` is configured (DM only supports the migration of binlog in the ROW format).
    - Check whether `binlog_row_image=FULL` is configured (DM only supports `binlog_row_image=FULL`).
    - If `binlog_do_db` or `binlog_ignore_db` is configured, check whether the database tables to be migrated meet the conditions of `binlog_do_db` and `binlog_ignore_db`.

- (Mandatory) Check if the upstream database is in an Online-DDL process (in which the `ghost` table is created but the `rename` phase is not executed yet). If the upstream is in the online-DDL process, the precheck returns an error. In this case, wait until the DDL to complete and retry.

Check items for full and incremental data migration

For the full and incremental data migration mode (`task-mode: all`), in addition to the common check items, the precheck also includes the full data migration check items and the incremental data migration check items.

Ignorable check items

Prechecks can find potential risks in your environments. It is not recommended to ignore check items. If your data migration task has special needs, you can use the `ignore-checking-items` configuration item to skip some check items.

Check item	Description
`dump_privilege`	Checks the dump privilege of the user in the upstream MySQL instance.
`replication_privilege`	Checks the replication privilege of the user in the upstream MySQL instance.
`version`	Checks the version of the upstream database.
`server_id`	Checks whether server_id is configured in the upstream database.
`binlog_enable`	Checks whether binlog is enabled in the upstream database.
`table_schema`	Checks the compatibility of the table schemas in the upstream MySQL tables.

Check item	Description
`schema_of_shard_tables`	Checks the consistency of the table schemas in the upstream MySQL multi-instance shards.
`auto_increment_ID`	Checks whether the auto-increment primary key conflicts in the upstream MySQL multi-instance shards.
`online_ddl`	Checks whether the upstream is in the process of online-DDL.

Note:

More ignorable check items are supported in versions earlier than v6.0. Since v6.0, DM does not allow ignoring some check items related to data safety. For example, if you configure the `binlog_row_image` parameter incorrectly, data might be lost during the replication.

Configure precheck arguments

The migration task precheck supports processing in parallel. Even if the number of rows in sharded tables reaches a million level, the precheck can be completed in minutes.

To specify the number of threads for the precheck, you can configure the `threads` argument of the `mydumpers` field in the migration task configuration file.

```
mydumpers:                          # Configuration arguments of the dump
   ↪ processing unit
  global:                           # Configuration name
```

```
  threads: 4                    # The number of threads that access the
    ↪ upstream when the dump processing unit performs the precheck and
    ↪ exports data from the upstream database (4 by default)
  chunk-filesize: 64            # The size of the files generated by the
    ↪ dump processing unit (64 MB by default)
  extra-args: "--consistency none" # Other arguments of the dump
    ↪ processing unit. You do not need to manually configure table-list
    ↪ in `extra-args`, because it is automatically generated by DM.
```

Note:

The value of `threads` determines the number of physical connections between the upstream database and DM. An excessively large `threads` value might increase the load of the upstream. Therefore, you need to set `threads` to a proper value.

13.6.5.10.2 Create a Data Migration Task

You can use the `start-task` command to create a data migration task. When the data migration task is started, DM prechecks privileges and configurations.

```
help start-task
```

```
Starts a task as defined in the configuration file

Usage:
  dmctl start-task [-s source ...] [--remove-meta] <config-file> [flags]

Flags:
  -h, --help             help for start-task
      --remove-meta      whether to remove task's meta data
      --start-time string specify the start time of binlog replication, e.g.
        ↪  '2021-10-21 00:01:00' or 2021-10-21T00:01:00

Global Flags:
      --config string     Path to config file.
      --master-addr string Master API server address, this parameter is
        ↪ required when interacting with the dm-master
      --rpc-timeout string RPC timeout, default is 10m. (default "10m")
  -s, --source strings    MySQL Source ID.
      --ssl-ca string     Path of file that contains list of trusted SSL CAs
        ↪  for connection.
```

```
    --ssl-cert string   Path of file that contains X509 certificate in PEM
       ↪  format for connection.
    --ssl-key string    Path of file that contains X509 key in PEM format
       ↪ for connection.
 -V, --version          Prints version and exit.
```

Usage example

```
start-task [ -s "mysql-replica-01"] ./task.yaml
```

Flags description

- `-s`: (Optional) Specifies the MySQL source to execute `task.yaml`. If it is set, the command only starts the subtasks of the specified task on the MySQL source.
- `config-file`: (Required) Specifies the file path of `task.yaml`.
- `remove-meta`: (Optional) Specifies whether to remove the task's previous metadata when starting the task.
- `start-time`: (Optional) Specifies the start time of binlog replication.
    - Format: `'2021-10-21 00:01:00'` or `2021-10-21T00:01:00`.
    - For incremental tasks, you can specify a rough starting point for the task using this flag. This flag takes precedence over the binlog position in the task configuration file and the binlog position in the downstream checkpoint.
    - When the task already has a checkpoint, if you start the task using this flag, DM automatically enables safe mode until the replication passes the checkpoint. This is to avoid the data duplication error caused by resetting the task to an earlier position.
        - When you reset the task to an earlier position, if the table schema at that time point is different from the downstream at the current time point, the task might report an error.
        - When you reset the task to a later position, note that the skipped binlog might have dirty data left in the downstream.
    - When you specify an earlier start time, DM starts migration from the earliest binlog position available.
    - When you specify a later start time, DM reports an error: `start-time {input-time} is too late, no binlog location matches it`.

Returned results

```
start-task task.yaml
```

```
{
    "result": true,
    "msg": "",
    "sources": [
```

```
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

13.6.5.10.3 Query Task Status in TiDB Data Migration

This document introduces how to use the `query-status` command to query the task status, and the subtask status of DM.

Query result

```
» query-status
```

```
{
    "result": true,   # Whether the query is successful.
    "msg": "",        # Describes the reason for the unsuccessful query.
    "tasks": [        # Migration task list.
        {
            "taskName": "test",      # The task name.
            "taskStatus": "Running", # The status of the task.
            "sources": [             # The upstream MySQL list.
                "mysql-replica-01",
                "mysql-replica-02"
            ]
        },
        {
            "taskName": "test2",
            "taskStatus": "Paused",
            "sources": [
                "mysql-replica-01",
                "mysql-replica-02"
            ]
        }
    ]
}
```

For detailed descriptions of `taskStatus` under the `tasks` section, refer to Task status.

It is recommended that you use `query-status` by the following steps:

1. Use `query-status` to check whether each on-going task is in the normal state.

2. If any error occurs in a task, use the `query-status <taskName>` command to see detailed error information. `<taskName>` in this command indicates the name of the task that encounters the error.

Task status

The status of a DM migration task depends on the status of each subtask assigned to DM-worker. For detailed descriptions of subtask status, see Subtask status. The table below shows how the subtask status is related to task status.

Subtask status in a task	Task status
One subtask is in the paused state and error information is returned.	`Error`
	`-`
`Some error occurred in subtask`	

Subtask status in a task	Task status
One subtask in the Sync phase is in the Running state but its Relay processing unit is not running (in the Error state).	`Error` `-` `Relay status is Error / Paused / Stopped`

Subtask status in a task	Task status
One subtask is in the `Paused` state and no error information is returned.	`Paused`
All subtasks are in the `New` state.	`New`
All subtasks are in the `Finished` state.	`Finished`

Subtask status in a task	Task status
All subtasks are in the Stopped state.	Stopped
Other situations	Running

Detailed query result

```
» query-status test
```

```
» query-status
{
    "result": true,   # Whether the query is successful.
    "msg": "",        # Describes the cause for the unsuccessful query.
    "sources": [                        # The upstream MySQL list.
        {
            "result": true,
            "msg": "",
            "sourceStatus": {               # The information of the upstream
                ↪ MySQL databases.
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [          # The information of all subtasks of
                ↪ upstream MySQL databases.
                {
                    "name": "test",       # The name of the subtask.
                    "stage": "Running",  # The running status of the subtask,
                        ↪ including "New", "Running", "Paused", "Stopped", and
                        ↪  "Finished".
```

```
"unit": "Sync",      # The processing unit of DM,
    ↪ including "Check", "Dump", "Load", and "Sync".
"result": null,      # Displays the error information if a
    ↪  subtask fails.
"unresolvedDDLLockID": "test-`test`.`t_target`", # The
    ↪ sharding DDL lock ID, used for manually handling the
    ↪  sharding DDL
                                                  # lock in the
                                                     ↪
                                                     ↪ abnormal
                                                     ↪
                                                     ↪ condition
                                                     ↪ .
"sync": {               # The replication information of
    ↪ the `Sync` processing unit. This information is
    ↪ about the
                        # same component with the current
                           ↪ processing unit.
    "totalEvents": "12", # The total number of binlog
       ↪ events that are replicated in this subtask.
    "totalTps": "1",     # The number of binlog events that
       ↪  are replicated in this subtask per second.
    "recentTps": "1",    # The number of binlog events that
       ↪  are replicated in this subtask in the last one
       ↪ second.
    "masterBinlog": "(bin.000001, 3234)",
       ↪                        # The binlog position in
       ↪ the upstream database.
    "masterBinlogGtid": "c0149e17-dff1-11e8-b6a8-0242
       ↪ ac110004:1-14", # The GTID information in the
       ↪ upstream database.
    "syncerBinlog": "(bin.000001, 2525)",
       ↪                        # The position of the
       ↪ binlog that has been replicated
                                                                #
                                                                   ↪
                                                                   ↪ i
                                                                   ↪
                                                                   ↪ t
                                                                   ↪
                                                                   ↪ `
                                                                   ↪ S
                                                                   ↪ `
                                                                   ↪
                                                                   ↪ p
```

```
                                                                    ↪
                                                                    ↪ u
                                                                    ↪ .
                                                                    ↪
    "syncerBinlogGtid": "",
       ↪                                          # The binlog
       ↪ position replicated using GTID.
    "blockingDDLs": [    # The DDL list that is blocked
       ↪ currently. It is not empty only when all the
       ↪ upstream tables of this
                         # DM-worker are in the "synced"
                            ↪ status. In this case, it
                            ↪ indicates the sharding DDL
                            ↪ statements to be executed
                            ↪ or that are skipped.
        "USE `test`; ALTER TABLE `test`.`t_target` DROP
           ↪ COLUMN `age`;"
    ],
    "unresolvedGroups": [ # The sharding group that is not
       ↪ resolved.
        {
            "target": "`test`.`t_target`",            # The
               ↪ downstream database table to be
               ↪ replicated.
            "DDLs": [
                "USE `test`; ALTER TABLE `test`.`t_target`
                   ↪ DROP COLUMN `age`;"
            ],
            "firstPos": "(bin|000001.000001, 3130)",  # The
               ↪ starting position of the sharding DDL
               ↪ statement.
            "synced": [                               # The
               ↪ upstream sharded table whose executed
               ↪ sharding DDL statement has been read by
               ↪ the `Sync` unit.
                "`test`.`t2`"
                "`test`.`t3`"
                "`test`.`t1`"
            ],
            "unsynced": [                             # The
               ↪ upstream table that has not executed this
               ↪  sharding DDL
                                                      #
                                                         ↪ statement
                                                         ↪ .
```

```
                                                                    ↪
                                                                    ↪ If
                                                                    ↪
                                                                    ↪ any
                                                                    ↪
                                                                    ↪ upstream
                                                                    ↪
                                                                    ↪ tables
                                                                    ↪
                                                                    ↪ have
                                                                    ↪
                                                                    ↪ not
                                                                    ↪
                                                                    ↪ finished
                                                                    ↪
                                                                    ↪ replication
                                                                    ↪ ,
                                                                    ↪
                                                                # `
                                                                    ↪ blockingDDLs
                                                                    ↪ `
                                                                    ↪
                                                                    ↪ is
                                                                    ↪
                                                                    ↪ empty
                                                                    ↪ .
                                                                    ↪
                            ]
                        }
                    ],
                    "synced": false      # Whether the incremental
                        ↪ replication catches up with the upstream and has
                        ↪  the same binlog position as that in the
                                         # upstream. The save point is not
                                             ↪ refreshed in real time in
                                             ↪ the `Sync` background, so "
                                             ↪ false" of "synced"
                                         # does not always mean a
                                             ↪ replication delay exits.
                }
            }
        ]
    },
    {
        "result": true,
```

```
        "msg": "",
        "sourceStatus": {
            "source": "mysql-replica-02",
            "worker": "worker2",
            "result": null,
            "relayStatus": null
        },
        "subTaskStatus": [
            {
                "name": "test",
                "stage": "Running",
                "unit": "Load",
                "result": null,
                "unresolvedDDLLockID": "",
                "load": {               # The replication information of
                    ↪ the `Load` processing unit.
                    "finishedBytes": "115", # The number of bytes that have
                        ↪  been loaded.
                    "totalBytes": "452", # The total number of bytes that
                        ↪ need to be loaded.
                    "progress": "25.44 %" # The progress of the loading
                        ↪ process.
                }
            }
        ]
    },
    {
        "result": true,
        "sourceStatus": {
            "source": "mysql-replica-03",
            "worker": "worker3",
            "result": null,
            "relayStatus": null
        },
        "subTaskStatus": [
            {
                "name": "test",
                "stage": "Paused",
                "unit": "Load",
                "result": {             # The error example.
                    "isCanceled": false,
                    "errors": [
                        {
                            "Type": "ExecSQL",
                            "msg": "Error 1062: Duplicate entry
```

```
                                ↪ '1155173304420532225' for key 'PRIMARY'\n
                                ↪ /home/jenkins/workspace/build_dm/go/src/
                                ↪ github.com/pingcap/tidb-enterprise-tools/
                                ↪ loader/db.go:160: \n/home/jenkins/
                                ↪ workspace/build_dm/go/src/github.com/
                                ↪ pingcap/tidb-enterprise-tools/loader/db.
                                ↪ go:105: \n/home/jenkins/workspace/
                                ↪ build_dm/go/src/github.com/pingcap/tidb-
                                ↪ enterprise-tools/loader/loader.go:138:
                                ↪ file test.t1.sql"
                        }
                    ],
                    "detail": null
                },
                "unresolvedDDLLockID": "",
                "load": {
                    "finishedBytes": "0",
                    "totalBytes": "156",
                    "progress": "0.00 %"
                }
            }
        ]
    }
]
}
```

For the status description and status switch relationship of "stage" of "subTaskStatus" of "sources", see the subtask status.

For operation details of "unresolvedDDLLockID" of "subTaskStatus" of "sources", see Handle Sharding DDL Locks Manually.

Subtask status

Status description

- `New`:
    - The initial status.
    - If the subtask does not encounter an error, it is switched to `Running`; otherwise it is switched to `Paused`.
- `Running`: The normal running status.
- `Paused`:
    - The paused status.

- If the subtask encounters an error, it is switched to `Paused`.
- If you run `pause-task` when the subtask is in the `Running` status, the task is switched to `Paused`.
- When the subtask is in this status, you can run the `resume-task` command to resume the task.

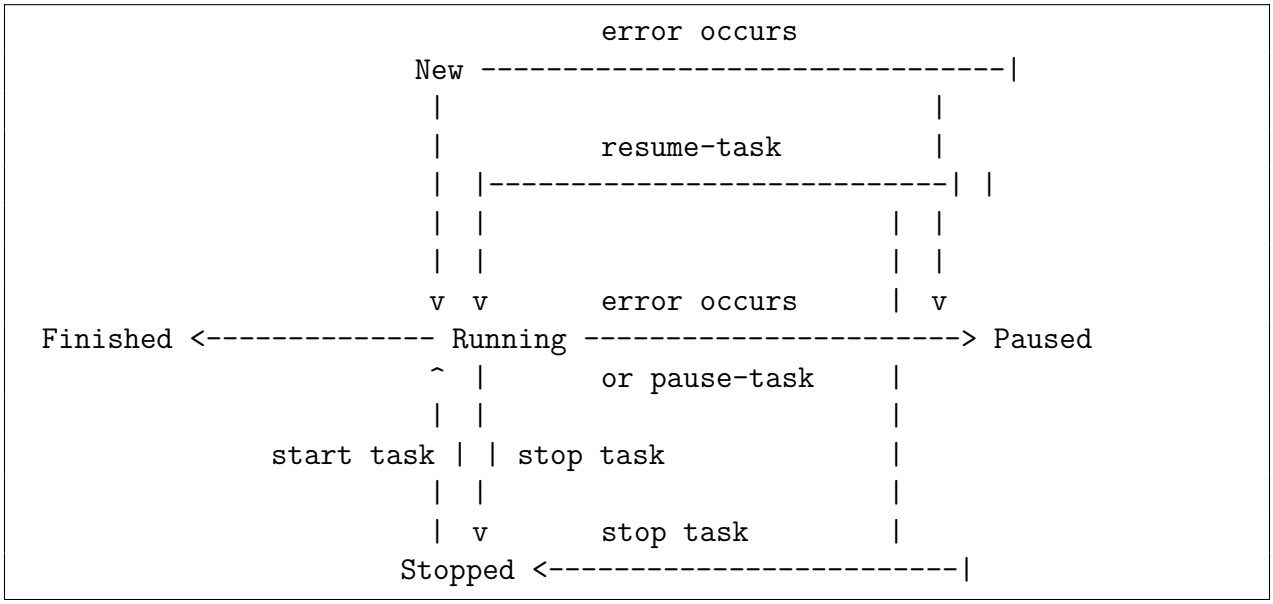
- `Stopped`:

  - The stopped status.
  - If you run `stop-task` when the subtask is in the `Running` or `Paused` status, the task is switched to `Stopped`.
  - When the subtask is in this status, you cannot use `resume-task` to resume the task.

- `Finished`:

  - The finished subtask status.
  - Only when the full replication subtask is finished normally, the task is switched to this status.

Status switch diagram



13.6.5.10.4 Pause a Data Migration Task

You can use the `pause-task` command to pause a data migration task.

`pause-task` differs from `stop-task` in that:

- `pause-task` only pauses a migration task. You can query the status information (retained in the memory) of the task using `query-status`. `stop-task` terminates a

migration task and removes all information related to this task from the memory. This means you cannot use `query-status` to query the status information. `dm_meta` like "checkpoint" and data that have been migrated to the downstream are not removed.

- If `pause-task` is executed to pause the migration task, you cannot start a new task with the same name, neither can you get the relay log of the paused task removed, since this task does exist. If `stop-task` is executed to stop a task, you can start a new task with the same name, and you can get the relay log of the stopped task removed, since this task no longer exists.
- `pause-task` is usually used to pause a task for troubleshooting, while `stop-task` is to permanently remove a migration task, or to co-work with `start-task` to update the configuration information.

```
help pause-task
```

```
pause a specified running task

Usage:
 dmctl pause-task [-s source ...] <task-name | task-file> [flags]

Flags:
 -h, --help  help for pause-task

Global Flags:
 -s, --source strings MySQL Source ID
```

Usage example

```
pause-task [-s "mysql-replica-01"] task-name
```

Flags description

- `-s`: (Optional) Specifies the MySQL source where you want to pause the subtasks of the migration task. If it is set, this command pauses only the subtasks on the specified MySQL source.
- `task-name| task-file`: (Required) Specifies the task name or task file path.

Returned results

```
pause-task test
```

```
{
    "op": "Pause",
    "result": true,
    "msg": "",
    "sources": [
```

```
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

13.6.5.10.5 Resume a Data Migration Task

You can use the `resume-task` command to resume a data migration task in the `Paused` state. This is generally used in scenarios where you want to manually resume a data migration task after handling the error that get the task paused.

```
help resume-task
```

```
resume a specified paused task

Usage:
 dmctl resume-task [-s source ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for resume-task

Global Flags:
 -s, --source strings MySQL Source ID
```

Usage example

```
resume-task [-s "mysql-replica-01"] task-name
```

Flags description

- `-s`: (Optional) Specifies the MySQL source where you want to resume the subtask of the migration task. If it is set, the command resumes only the subtasks on the specified MySQL source.
- `task-name | task-file`: (Required) Specifies the task name or task file path.

Returned results

```
resume-task test
```

```
{
    "op": "Resume",
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

13.6.5.10.6 Stop a Data Migration Task

You can use the `stop-task` command to stop a data migration task. For differences between `stop-task` and `pause-task`, refer to Pause a Data Migration Task.

```
help stop-task
```

```
stop a specified task

Usage:
 dmctl stop-task [-s source ...] <task-name | task-file> [flags]

Flags:
 -h, --help help for stop-task

Global Flags:
 -s, --source strings MySQL Source ID
```

Usage example

```
stop-task [-s "mysql-replica-01"] task-name
```

Flags description

- `-s`: (Optional) Specifies the MySQL source where the subtasks of the migration task (that you want to stop) run. If it is set, only subtasks on the specified MySQL source are stopped.
- `task-name | task-file`: (Required) Specifies the task name or task file path.

Returned results

```
stop-task test
```

```
{
    "op": "Stop",
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

13.6.6 Advanced Tutorials

13.6.6.1 Merge and Migrate Data from Sharded Tables

13.6.6.1.1 Merge and Migrate Data from Sharded Tables

This document introduces the sharding support feature provided by Data Migration (DM). This feature allows you to merge and migrate the data of tables with the same or different table schemas in the upstream MySQL or MariaDB instances into one same table in the downstream TiDB. It supports not only migrating the upstream DML statements, but also coordinating to migrate the table schema change using DDL statements in multiple upstream sharded tables.

Overview

DM supports merging and migrating the data of multiple upstream sharded tables into one table in TiDB. During the migration, the DDL of each sharded table, and the DML before and after the DDL need to be coordinated. For the usage scenarios, DM supports two different modes: pessimistic mode and optimistic mode.

Note:

- To merge and migrate data from sharded tables, you must set `shard-mode` in the task configuration file.
- DM uses the pessimistic mode by default for the merge of the sharding support feature. (If there is no special description in the document, use the pessimistic mode by default.)

- It is not recommended to use this mode if you do not understand the principles and restrictions of the optimistic mode. Otherwise, it may cause serious consequences such as migration interruption and even data inconsistency.

The pessimistic mode

When an upstream sharded table executes a DDL statement, the migration of this sharded table will be suspended. After all other sharded tables execute the same DDL, the DDL will be executed in the downstream and the data migration task will restart. The advantage of this mode is that it can ensure that the data migrated to the downstream will not go wrong. For details, refer to shard merge in pessimistic mode. ####### The optimistic mode

DM will automatically modify the DDL executed on a sharded table into a statement compatible with other sharded tables, and then migrate to the downstream. This will not block the DML migration of any sharded tables. The advantage of this mode is that it will not block data migration when processing DDL. However, improper use will cause migration interruption or even data inconsistency. For details, refer to shard merge in optimistic mode.

Contrast

Pessimistic mode	Optimistic mode
Sharded tables that executes DDL suspend DML migration	Sharded tables that executes DDL continue DML migration
The DDL execution order and statements of each sharded table must be the same	Each sharded table only needs to keep the table schema compatible with each other

Pessimistic mode	Optimistic mode
The DDL is migrated to the downstream after the entire shard group is consistent	The DDL of each sharded table immediately affects the downstream
Wrong DDL operations can be intercepted after the detection	Wrong DDL operations will be migrated to the downstream, which may cause inconsistency between the upstream and downstream data before the detection

13.6.6.1.2 Merge and Migrate Data from Sharded Tables in the Pessimistic Mode

This document introduces the sharding support feature provided by Data Migration (DM) in the pessimistic mode (the default mode). This feature allows you to merge and migrate the data of tables with the same table schema in the upstream MySQL or MariaDB instances into one same table in the downstream TiDB.

Restrictions

DM has the following sharding DDL usage restrictions in the pessimistic mode:

- For a logical **sharding group** (composed of all sharded tables that need to be merged and migrated into one same downstream table), it is limited to use one task containing exactly the sources of sharded tables to perform the migration.
- In a logical **sharding group**, the same DDL statements must be executed in the same order in all upstream sharded tables (the schema name and the table name can be different), and the next DDL statement cannot be executed unless the current DDL operation is completely finished.

    - For example, if you add `column A` to `table_1` before you add `column B`, then you cannot add `column B` to `table_2` before you add `column A`. Executing the DDL statements in a different order is not supported.

- In a sharding group, the corresponding DDL statements should be executed in all upstream sharded tables.
    - For example, if DDL statements are not executed on one or more upstream sharded tables corresponding to `DM-worker-2`, then other DM-workers that have executed the DDL statements pause their migration task and wait for `DM-worker -2` to receive the upstream DDL statements.
- The sharding group migration task does not support `DROP DATABASE`/`DROP TABLE`.
    - The sync unit in DM-worker automatically ignores the `DROP DATABASE`/`DROP TABLE` statement of upstream sharded tables.
- The sharding group migration task does not support `TRUNCATE TABLE`.
    - The sync unit in DM-worker automatically ignores the `TRUNCATE TABLE` statement of upstream sharded tables.
- The sharding group migration task supports `RENAME TABLE`, but with the following limitations (online DDL is supported in another solution):
    - A table can only be renamed to a new name that is not used by any other table.
    - A single `RENAME TABLE` statement can only involve a single `RENAME` operation.
- The sharding group migration task requires each DDL statement to involve operations on only one table.
- The table schema of each sharded table must be the same at the starting point of the incremental replication task, so as to make sure the DML statements of different sharded tables can be migrated into the downstream with a definite table schema, and the subsequent sharding DDL statements can be correctly matched and migrated.
- If you need to change the table routing rule, you have to wait for the migration of all sharding DDL statements to complete.
    - During the migration of sharding DDL statements, an error is reported if you use `dmctl` to change `router-rules`.
- If you need to `CREATE` a new table to a sharding group where DDL statements are being executed, you have to make sure that the table schema is the same as the newly modified table schema.
    - For example, both the original `table_1` and `table_2` have two columns (a, b) initially, and have three columns (a, b, c) after the sharding DDL operation, so after the migration the newly created table should also have three columns (a, b, c).
- Because the DM-worker that has received the DDL statements will pause the task to wait for other DM-workers to receive their DDL statements, the delay of data migration will be increased.

Background

Currently, DM uses the binlog in the `ROW` format to perform the migration task. The binlog does not contain the table schema information. When you use the `ROW` binlog to migrate data, if you have not migrated multiple upstream tables into the same downstream table, then there only exist DDL operations of one upstream table that can update the table schema of the downstream table. The `ROW` binlog can be considered to have the nature of self-description. During the migration process, the DML statements can be constructed accordingly with the column values and the downstream table schema.

However, in the process of merging and migrating sharded tables, if DDL statements are executed on the upstream tables to modify the table schema, then you need to perform extra operations to migrate the DDL statements so as to avoid the inconsistency between the DML statements produced by the column values and the actual downstream table schema.

Here is a simple example:

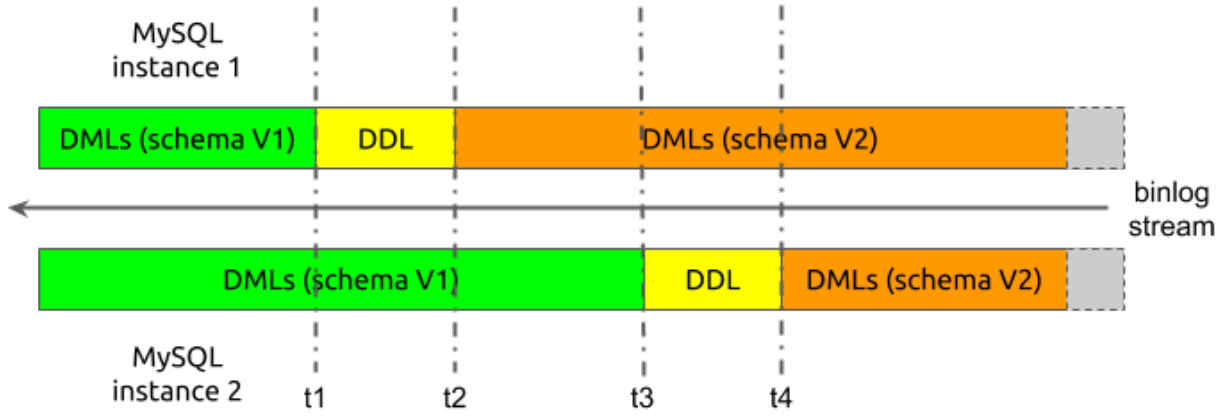


Figure 238: shard-ddl-example-1

In the above example, the merging process is simplified, where only two MySQL instances exist in the upstream and each instance has only one table. When the migration begins, the table schema version of two sharded tables is marked as `schema V1`, and the table schema version after executing DDL statements is marked as `schema V2`.

Now assume that in the migration process, the binlog data received from the two upstream sharded tables has the following time sequence:

1. When the migration begins, the sync unit in DM-worker receives the DML events of `schema V1` from the two sharded tables.
2. At `t1`, the sharding DDL events from instance 1 are received.
3. From `t2` on, the sync unit receives the DML events of `schema V2` from instance 1; but from instance 2, it still receives the DML events of `schema V1`.

4. At `t3`, the sharding DDL events from instance 2 are received.
5. From `t4` on, the sync unit receives the DML events of `schema V2` from instance 2 as well.

Assume that the DDL statements of sharded tables are not processed during the migration process. After DDL statements of instance 1 are migrated to the downstream, the downstream table schema is changed to `schema V2`. But for instance 2, the sync unit in DM-worker is still receiving DML events of `schema V1` from `t2` to `t3`. Therefore, when the DML statements of `schema V1` are migrated to the downstream, the inconsistency between the DML statements and the table schema can cause errors and the data cannot be migrated successfully.

Principles

This section shows how DM migrates DDL statements in the process of merging sharded tables based on the above example in the pessimistic mode.

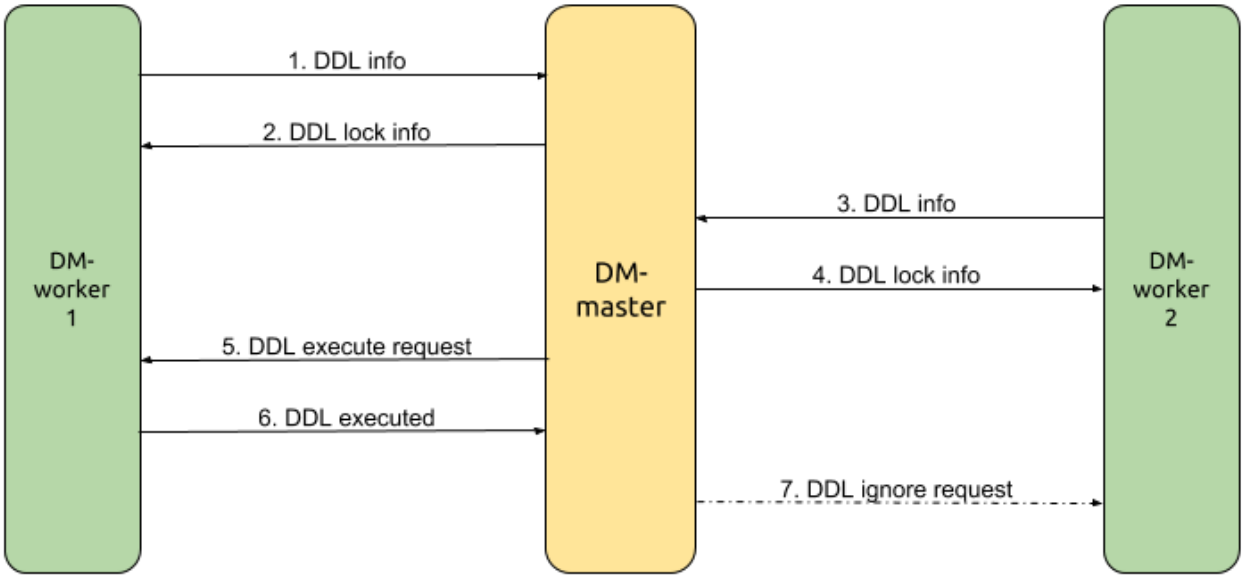


Figure 239: shard-ddl-flow

In this example, `DM-worker-1` migrates the data from MySQL instance 1 and `DM-worker-2` migrates the data from MySQL instance 2. `DM-master` coordinates the DDL migration among multiple DM-workers. Starting from `DM-worker-1` receiving the DDL statements, the DDL migration process is simplified as follows:

1. `DM-worker-1` receives the DDL statement from MySQL instance 1 at `t1`, pauses the data migration of the corresponding DDL and DML statements, and sends the DDL information to `DM-master`.

2. `DM-master` decides that the migration of this DDL statement needs to be coordinated based on the received DDL information, creates a lock for this DDL statement, sends the DDL lock information back to `DM-worker-1` and marks `DM-worker-1` as the owner of this lock at the same time.
3. `DM-worker-2` continues migrating the DML statement until it receives the DDL statement from MySQL instance 2 at `t3`, pauses the data migration of this DDL statement, and sends the DDL information to `DM-master`.
4. `DM-master` decides that the lock of this DDL statement already exists based on the received DDL information, and sends the lock information directly to `DM-worker-2`.
5. Based on the configuration information when the task is started, the sharded table information in the upstream MySQL instances, and the deployment topology information, `DM-master` decides that it has received this DDL statement of all upstream sharded tables to be merged, and requests the owner of the DDL lock (`DM-worker-1`) to migrate this DDL statement to the downstream.
6. `DM-worker-1` verifies the DDL statement execution request based on the DDL lock information received at Step #2, migrates this DDL statement to the downstream, and sends the results to `DM-master`. If this operation is successful, `DM-worker-1` continues migrating the subsequent (starting from the binlog at `t2`) DML statements.
7. `DM-master` receives the response from the lock owner that the DDL is successfully executed, and requests all other DM-workers (`DM-worker-2`) that are waiting for the DDL lock to ignore this DDL statement and then continue to migrate the subsequent (starting from the binlog at `t4`) DML statements.

The characteristics of DM handling the sharding DDL migration among multiple DM-workers can be concluded as follows:

- Based on the task configuration and DM cluster deployment topology information, a logical sharding group is built in `DM-master` to coordinate DDL migration. The group members are DM-workers that handle each sub-task divided from the migration task).
- After receiving the DDL statement from the binlog event, each DM-worker sends the DDL information to `DM-master`.
- `DM-master` creates or updates the DDL lock based on the DDL information received from each DM-worker and the sharding group information.
- If all members of the sharding group receive a same specific DDL statement, this indicates that all DML statements before the DDL execution on the upstream sharded tables have been completely migrated, and this DDL statement can be executed. Then DM can continue to migrate the subsequent DML statements.
- After being converted by the table router, the DDL statement of the upstream sharded tables must be consistent with the DDL statement to be executed in the downstream. Therefore, this DDL statement only needs to be executed once by the DDL owner and all other DM-workers can ignore this DDL statement.

In the above example, only one sharded table needs to be merged in the upstream MySQL instance corresponding to each DM-worker. But in actual scenarios, there might be multiple

sharded tables in multiple sharded schemas to be merged in one MySQL instance. And when this happens, it becomes more complex to coordinate the sharding DDL migration.

Assume that there are two sharded tables, namely `table_1` and `table_2`, to be merged in one MySQL instance:

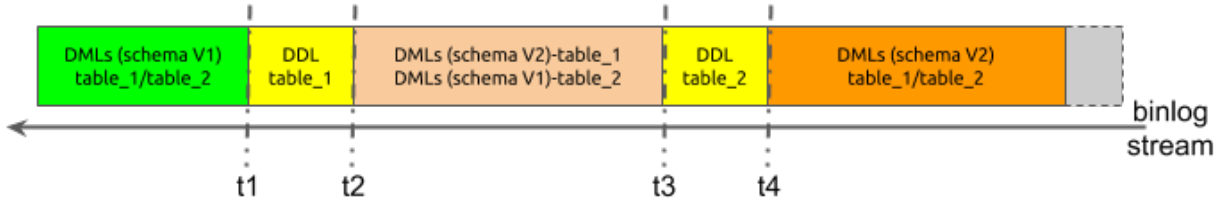


Figure 240: shard-ddl-example-2

Because data comes from the same MySQL instance, all the data is obtained from the same binlog stream. In this case, the time sequence is as follows:

1. The sync unit in DM-worker receives the DML statements of `schema V1` from both sharded tables when the migration begins.
2. At `t1`, the sync unit in DM-worker receives the DDL statements of `table_1`.
3. From `t2` to `t3`, the received data includes the DML statements of `schema V2` from `table_1` and the DML statements of `schema V1` from `table_2`.
4. At `t3`, the sync unit in DM-worker receives the DDL statements of `table_2`.
5. From `t4` on, the sync unit in DM-worker receives the DML statements of `schema V2` from both tables.

If the DDL statements are not processed particularly during the data migration, when the DDL statement of `table_1` is migrated to the downstream and changes the downstream table schema, the DML statement of `schema V1` from `table_2` cannot be migrated successfully. Therefore, within a single DM-worker, a logical sharding group similar to that within `DM-master` is created, except that members of this group are different sharded tables in the same upstream MySQL instance.

But when a DM-worker coordinates the migration of the sharding group within itself, it is not totally the same as that performed by `DM-master`. The reasons are as follows:

- When the DM-worker receives the DDL statement of `table_1`, it cannot pause the migration and needs to continue parsing the binlog to get the subsequent DDL statements of `table_2`. This means it needs to continue parsing between `t2` and `t3`.
- During the binlog parsing process between `t2` and `t3`, the DML statements of `schema V2` from `table_1` cannot be migrated to the downstream until the sharding DDL statement is migrated and successfully executed.

In DM, the simplified migration process of sharding DDL statements within the DM worker is as follows:

1. When receiving the DDL statement of `table_1` at `t1`, the DM-worker records the DDL information and the current position of the binlog.
2. DM-worker continues parsing the binlog between `t2` and `t3`.
3. DM-worker ignores the DML statement with the `schema V2` schema that belongs to `table_1`, and migrates the DML statement with the `schema V1` schema that belongs to `table_2` to the downstream.
4. When receiving the DDL statement of `table_2` at `t3`, the DM-worker records the DDL information and the current position of the binlog.
5. Based on the information of the migration task configuration and the upstream schemas and tables, the DM-worker decides that the DDL statements of all sharded tables in the MySQL instance have been received and migrates them to the downstream to modify the downstream table schema.
6. DM-worker sets the starting point of parsing the new binlog stream to be the position saved at Step #1.
7. DM-worker resumes parsing the binlog between `t2` and `t3`.
8. DM-worker migrates the DML statement with the `schema V2` schema that belongs to `table_1` to the downstream, and ignores the DML statement with the `schema V1` schema that belongs to `table_2`.
9. After parsing the binlog position saved at Step #4, the DM-worker decides that all DML statements that have been ignored in Step #3 have been migrated to the downstream again.
10. DM-worker resumes the migration starting from the binlog position at `t4`.

You can conclude from the above analysis that DM mainly uses two-level sharding groups for coordination and control when handling migration of the sharding DDL. Here is the simplified process:

1. Each DM-worker independently coordinates the DDL statements migration for the corresponding sharding group composed of multiple sharded tables within the upstream MySQL instance.
2. After the DM-worker receives the DDL statements of all sharded tables, it sends the DDL information to `DM-master`.
3. `DM-master` coordinates the DDL migration of the sharding group composed of the DM-workers based on the received DDL information.
4. After receiving the DDL information from all DM-workers, `DM-master` requests the DDL lock owner (a specific DM-worker) to execute the DDL statement.
5. The DDL lock owner executes the DDL statement and returns the result to `DM-master` ↪ . Then the owner restarts the migration of the previously ignored DML statements during the internal coordination of DDL migration.
6. After `DM-master` confirms that the owner has successfully executed the DDL statement, it asks all other DM-workers to continue the migration.

7. All other DM-workers separately restart the migration of the previously ignored DML statements during the internal coordination of DDL migration.
8. After finishing migrating the ignored DML statements again, all DM-workers resume the normal migration process.

13.6.6.1.3 Merge and Migrate Data from Sharded Tables in Optimistic Mode

This document introduces the sharding support feature provided by Data Migration (DM) in the optimistic mode. This feature allows you to merge and migrate the data of tables with the same or different table schema(s) in the upstream MySQL or MariaDB instances into one same table in the downstream TiDB.

> **Note:**
>
> If you do not have an in-depth understanding of the optimistic mode and its restrictions, it is **NOT** recommended to use this mode. Otherwise, migration interruption or even data inconsistency might occur.

Background

DM supports executing DDL statements on sharded tables online, which is called sharding DDL, and uses the "pessimistic mode" by default. In this mode, when a DDL statement is executed in an upstream sharded table, data migration of this table is paused until the same DDL statement is executed in all other sharded tables. Only by then this DDL statement is executed in the downstream and data migration resumes.

The pessimistic mode guarantees that the data migrated to the downstream is always correct, but it pauses the data migration, which is bad for making A/B changes in the upstream. In some cases, users might spend a long time executing DDL statements in a single sharded table and change the schemas of other sharded tables only after a period of validation. In the pessimistic mode, these DDL statements block data migration and cause many binlog events to pile up.

Therefore, an "optimistic mode" is needed. In this mode, a DDL statement executed on a sharded table is automatically converted to a statement that is compatible with other sharded tables, and then immediately migrated to the downstream. In this way, the DDL statement does not block any sharded table from executing DML migration.

Configuration of the optimistic mode

To use the optimistic mode, specify the `shard-mode` item in the task configuration file as `optimistic`. For the detailed sample configuration file, see DM Advanced Task Configuration File.

Restrictions

It takes some risks to use the optimistic mode. Follow these rules when you use it:

- Ensure that the schema of every sharded table is consistent with each other before and after you execute a batch of DDL statements.
- If you perform an A/B test, perform the test **ONLY** on one sharded table.
- After the A/B test is finished, migrate only the most direct DDL statement(s) to the final schema. Do not re-execute every right or wrong step of the test.

  For example, if you have executed `ADD COLUMN A INT; DROP COLUMN A; ADD COLUMN A FLOAT;` in a sharded table, you only need to execute `ADD COLUMN A FLOAT` in other sharded tables. You do not need to execute all of the three DDL statements again.
- Observe the status of the DM migration when executing the DDL statement. When an error is reported, you need to determine whether this batch of DDL statements will cause data inconsistency.

In the optimistic mode, most of the DDL statements executed in the upstream are automatically migrated to the downstream with no extra effort required. These DDL statements are called "Type 1 DDL".

DDL statements that change the column name, the column type, or the column default value are called "Type 2 DDL". When you execute Type 2 DDL statements in the upstream, make sure that you execute the DDL statements in all sharded tables in the same order.

Some examples of Type 2 DDL statements are as follows:

- Alter the type of a column: `ALTER TABLE table_name MODIFY COLUMN column_name VARCHAR(20).`
- Rename a column: `ALTER TABLE table_name RENAME COLUMN column_1 TO column_2;.`
- Add a `NOT NULL` column without a default value: `ALTER TABLE table_name ADD COLUMN column_1 NOT NULL;.`
- Rename an index: `ALTER TABLE table_name RENAME INDEX index_1 TO index_2;.`

When the sharded tables execute the DDL statements above, if the execution order is different, the migration is interrupted. For example:

- Shard 1 renames a column and then alters the column type:
  1. Rename a column: `ALTER TABLE table_name RENAME COLUMN column_1 TO column_2;.`
  2. Alter the column type: `ALTER TABLE table_name MODIFY COLUMN column_3 VARCHAR(20);.`

- Shard 2 alters a column type and then renames the column:

    1. Alter a column type: `ALTER TABLE table_name MODIFY COLUMN column_3 VARCHAR(20)`.
    2. Rename a column: `ALTER TABLE table_name RENAME COLUMN column_1 TO column_2;`.

In addition, the following restrictions apply to both the optimistic mode and the pessimistic mode:

- `DROP TABLE` or `DROP DATABASE` is not supported.
- `TRUNCATE TABLE` is not supported.
- Each DDL statement must involve operations on only one table.
- The DDL statement that is not supported in TiDB is also not supported in DM.
- The default value of a newly added column must not contain `current_timestamp` , `rand()`, `uuid()`; otherwise, data inconsistency between the upstream and the downstream might occur.

Risks

When you use the optimistic mode for a migration task, a DDL statement is migrated to the downstream immediately. If this mode is misused, data inconsistency between the upstream and the downstream might occur.

Operations that cause data inconsistency

- The schema of each sharded table is incompatible with each other. For example:

    - Two columns of the same name are added to two sharded tables respectively, but the columns are of different types.
    - Two columns of the same name are added to two sharded tables respectively, but the columns have different default values.
    - Two generated columns of the same name are added to two sharded tables respectively, but the columns are generated using different expressions.
    - Two indexes of the same name are added to two sharded tables respectively, but the keys are different.
    - Other different table schemas with the same name.

- Execute the DDL statement that can corrupt data in the sharded table and then try to roll back.

    For example, drop a column `X` and then add this column back.

Example

Merge and migrate the following three sharded tables to TiDB:

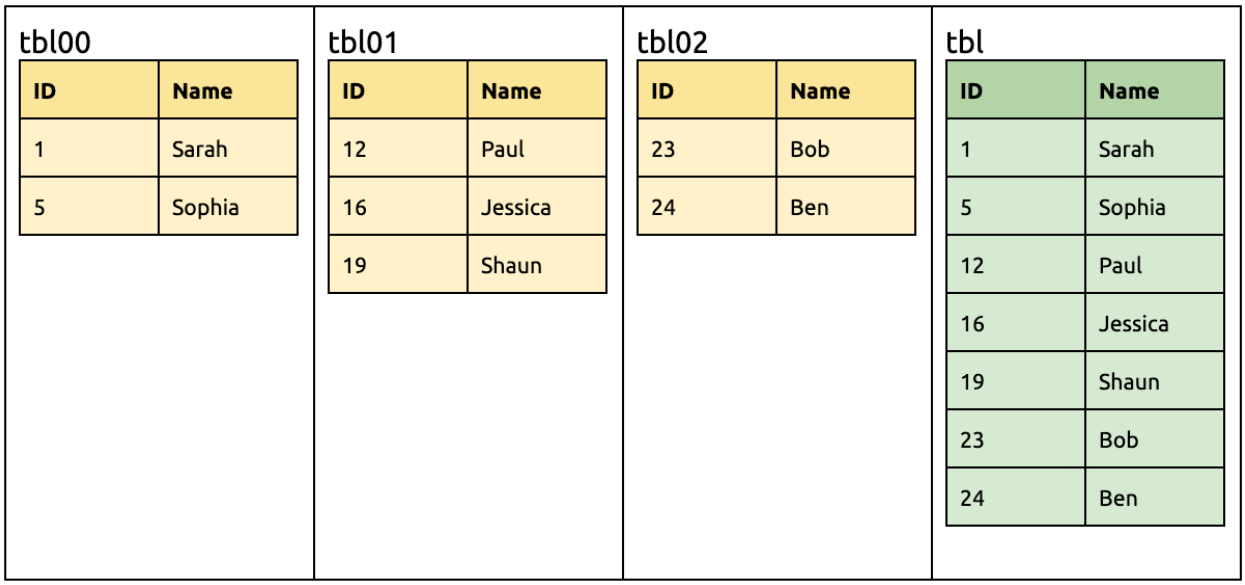


Figure 241: optimistic-ddl-fail-example-1

Add a new column `Age` in `tbl01` and set the default value of the column to `0`:

```
ALTER TABLE `tbl01` ADD COLUMN `Age` INT DEFAULT 0;
```

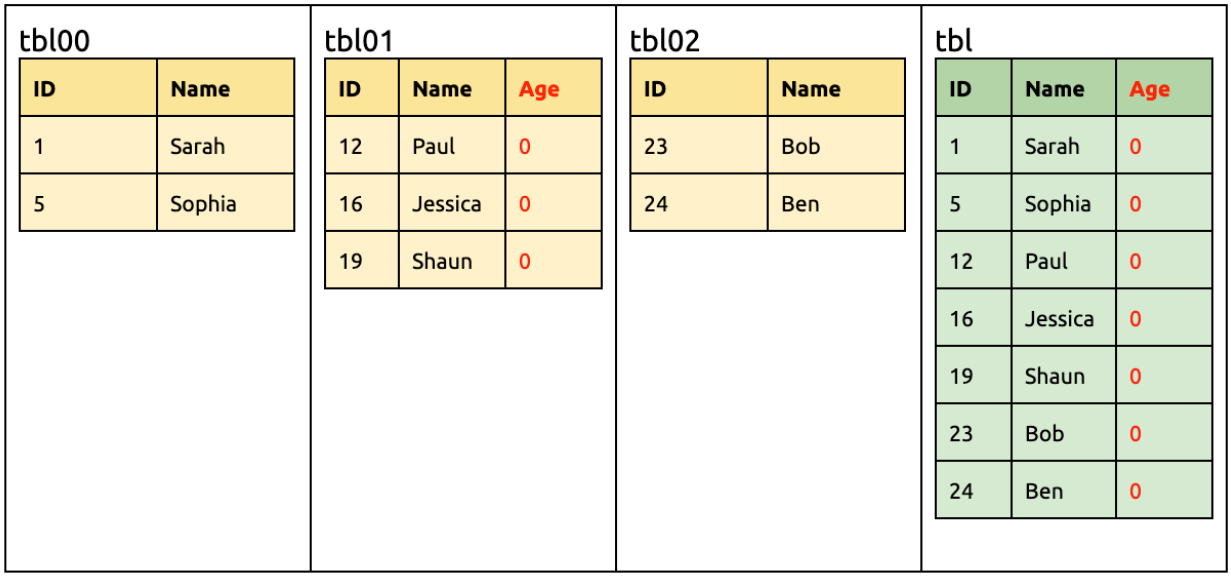


Figure 242: optimistic-ddl-fail-example-2

Add a new column `Age` in `tbl00` and set the default value of the column to `-1`:

```
ALTER TABLE `tbl00` ADD COLUMN `Age` INT DEFAULT -1;
```

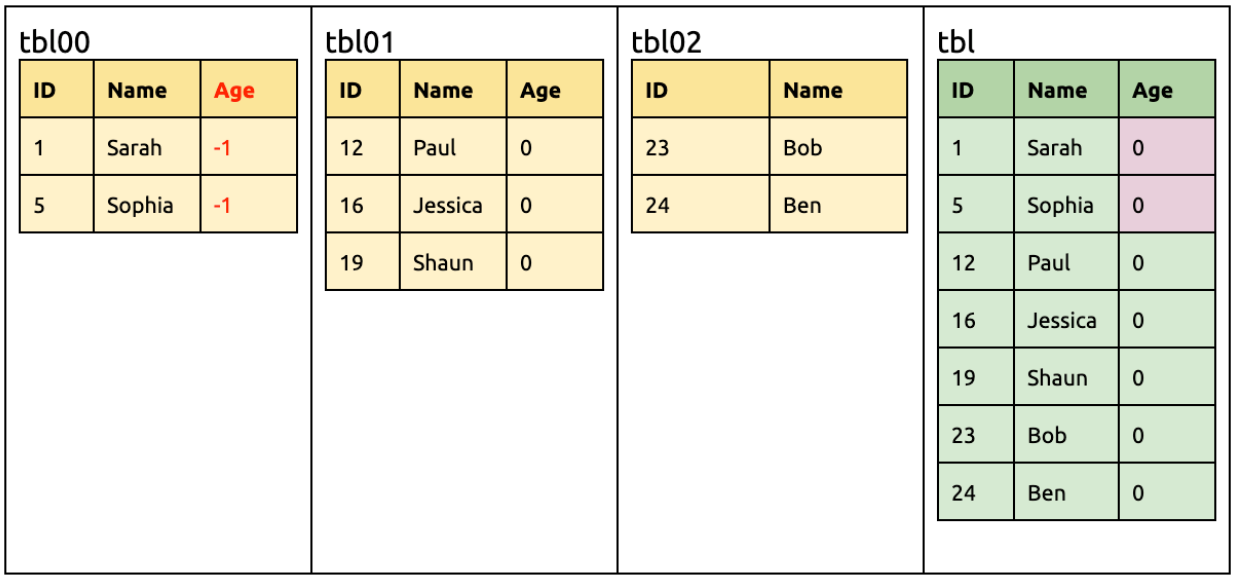


Figure 243: optimistic-ddl-fail-example-3

By then, the `Age` column of `tbl00` is inconsistent because `DEFAULT 0` and `DEFAULT -1` are incompatible with each other. In this situation, DM will report the error, but you have to manually fix the data inconsistency.

Implementation principle

In the optimistic mode, after DM-worker receives the DDL statement from the upstream, it forwards the updated table schema to DM-master. DM-worker tracks the current schema of each sharded table, and DM-master merges these schemas into a composite schema that is compatible with DML statements of every sharded table. Then DM-master migrates the corresponding DDL statement to the downstream. DML statements are directly migrated to the downstream.

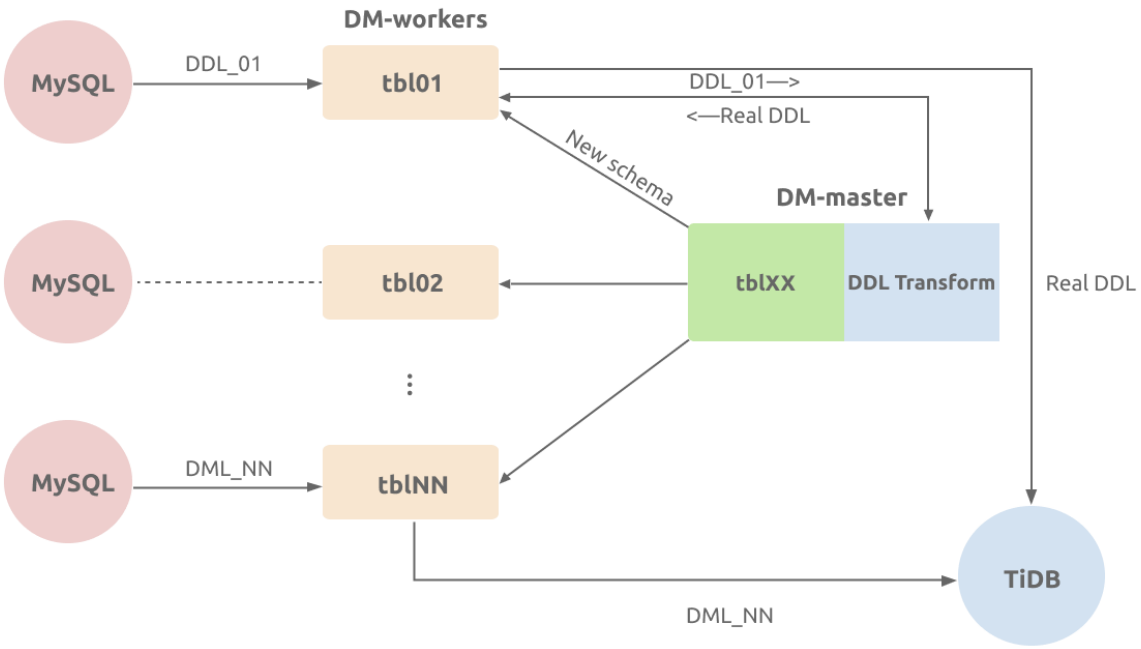


Figure 244: optimistic-ddl-flow

Examples

Assume the upstream MySQL has three sharded tables (`tbl00`, `tbl01`, and `tbl02`). Merge and migrate these sharded tables to the `tbl` table in the downstream TiDB. See the following image:

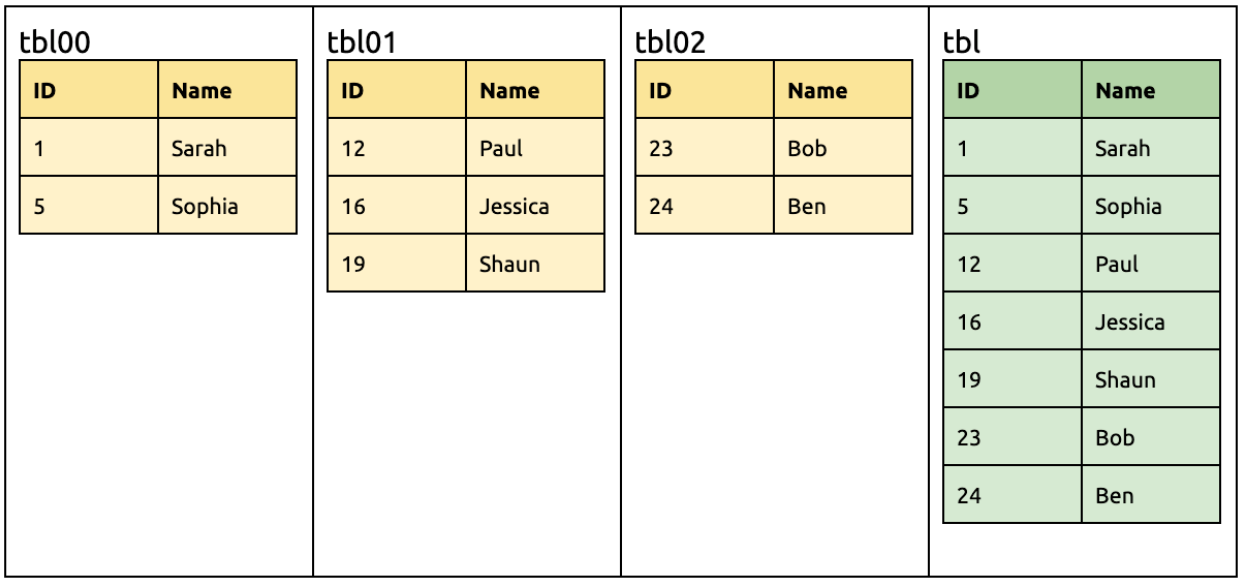


Figure 245: optimistic-ddl-example-1

Add a `Level` column in the upstream:

```
ALTER TABLE `tbl00` ADD COLUMN `Level` INT;
```

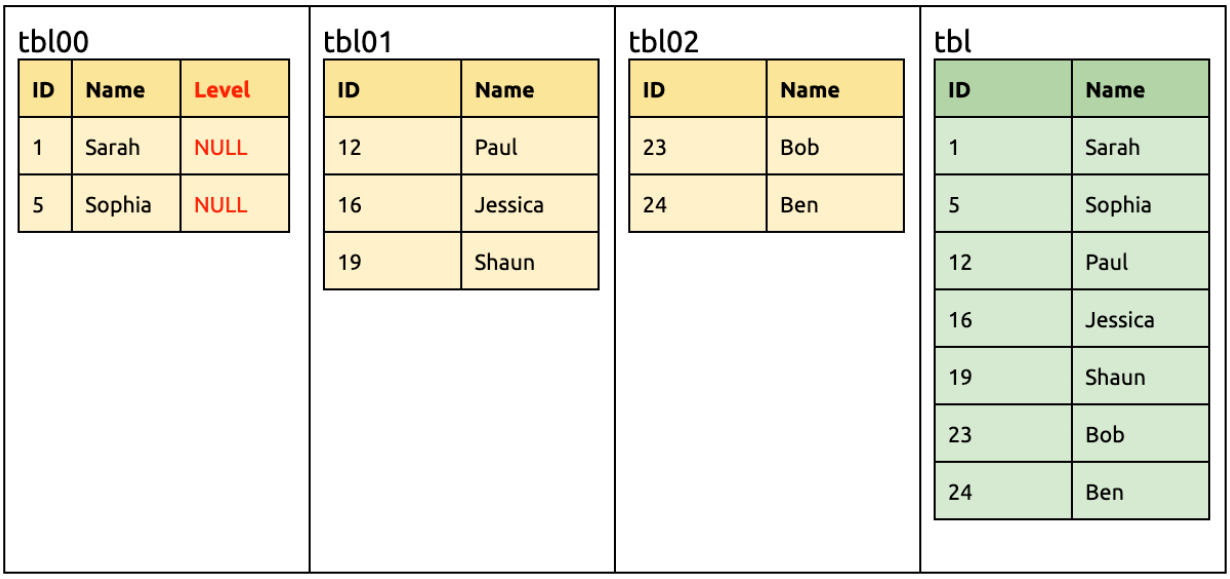


Figure 246: optimistic-ddl-example-2

Then TiDB will receive the DML statement from `tbl00` (with the `Level` column) and the DML statement from the `tbl01` and `tbl02` tables (without the `Level` column).

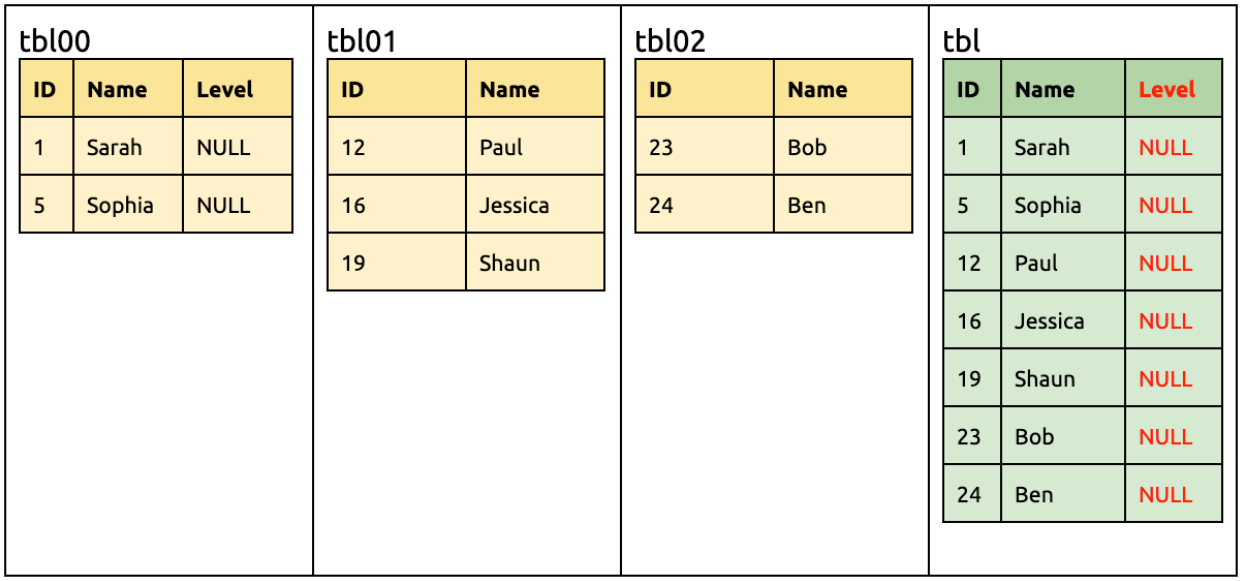


Figure 247: optimistic-ddl-example-3

The following DML statements can be migrated to the downstream without any modification:

```
UPDATE `tbl00` SET `Level` = 9 WHERE `ID` = 1;
INSERT INTO `tbl02` (`ID`, `Name`) VALUES (27, 'Tony');
```

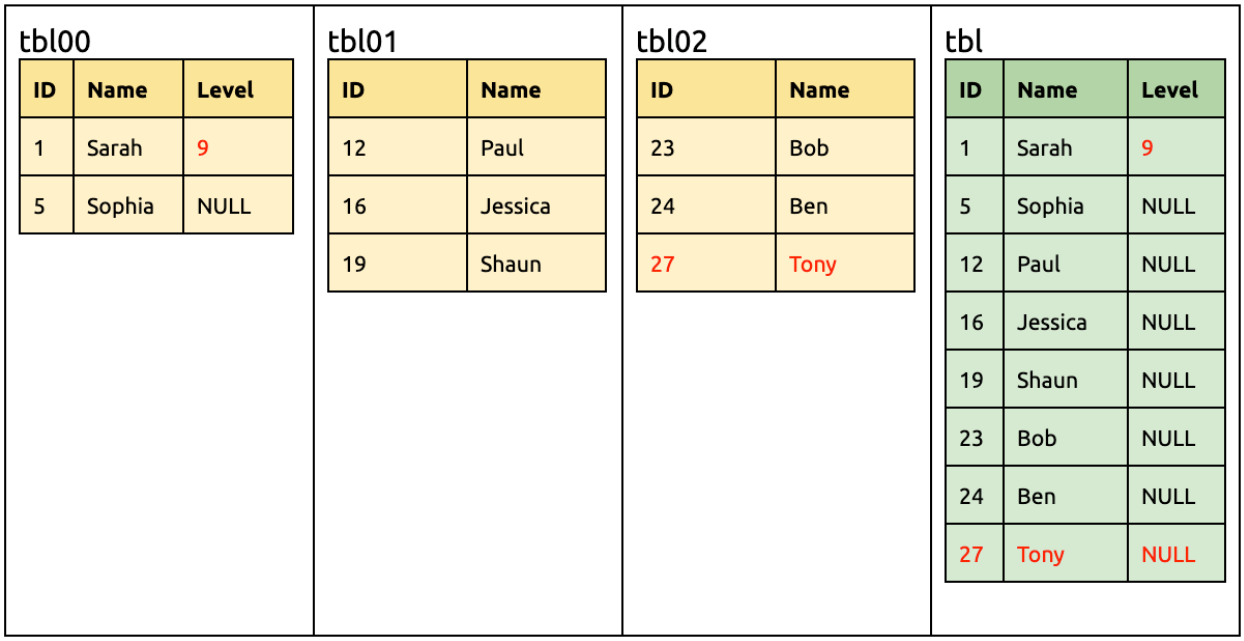


Figure 248: optimistic-ddl-example-4

Also add a `Level` column in `tbl01`:

```
ALTER TABLE `tbl01` ADD COLUMN `Level` INT;
```

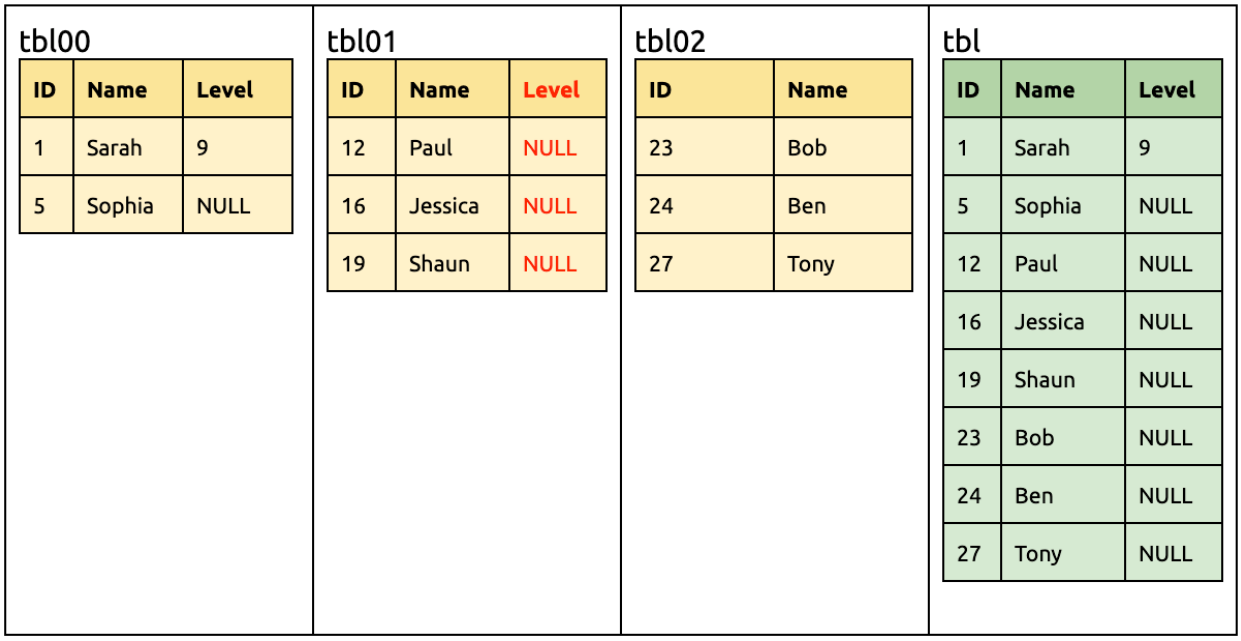


Figure 249: optimistic-ddl-example-5

At this time, the downstream already have had the same `Level` column, so DM-master performs no operation after comparing the table schemas.

Drop a `Name` column in `tbl01`:

```
ALTER TABLE `tbl01` DROP COLUMN `Name`;
```

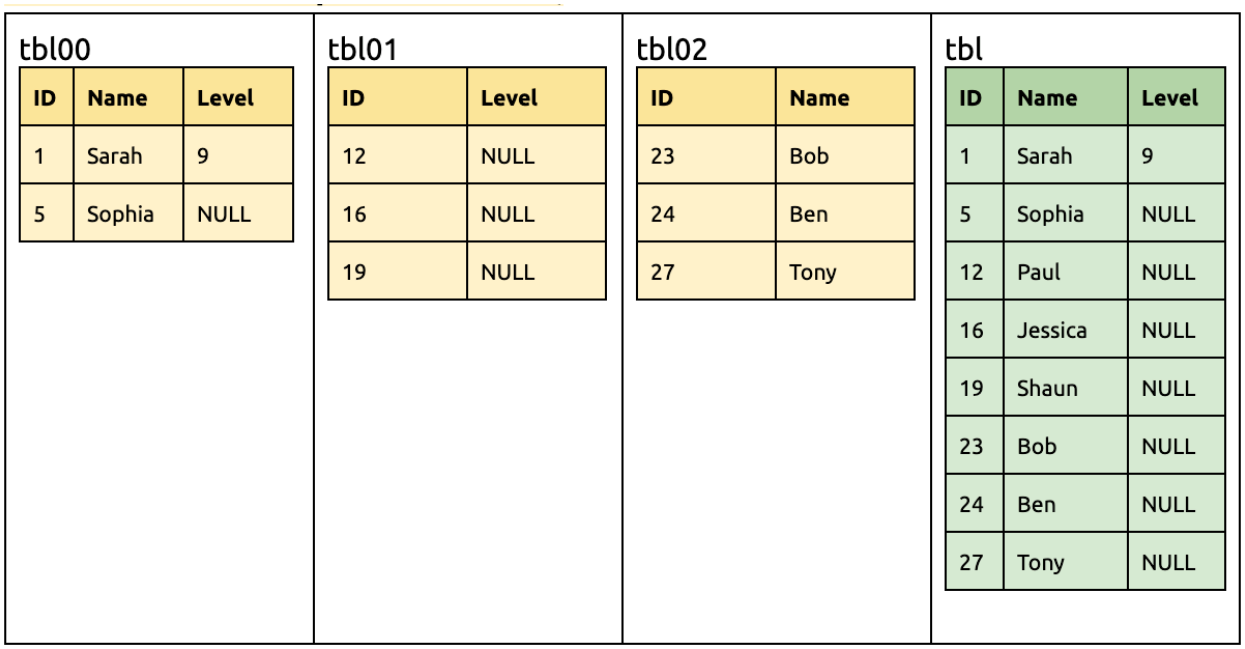
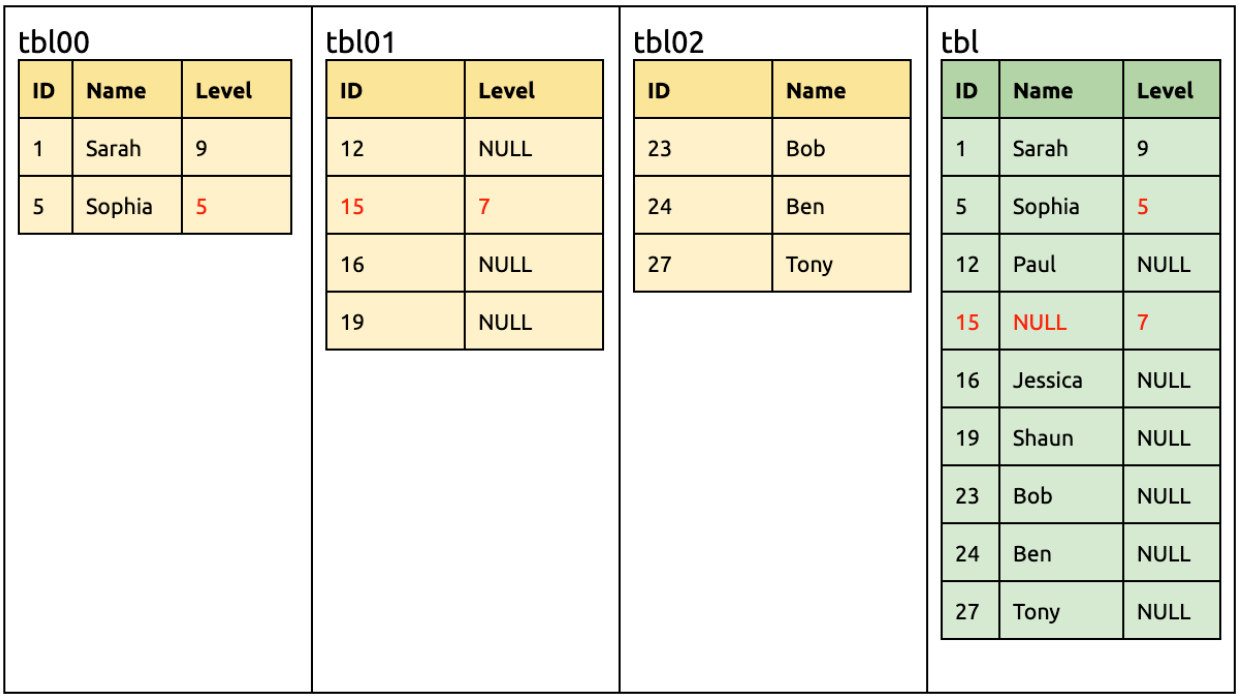


Figure 250: optimistic-ddl-example-6

Then the downstream will receive the DML statements from `tbl00` and `tbl02` with the `Name` column, so this column is not immediately dropped.

In the same way, all DML statements can still be migrated to the downstream:

```
INSERT INTO `tbl01` (`ID`, `Level`) VALUES (15, 7);
UPDATE `tbl00` SET `Level` = 5 WHERE `ID` = 5;
```



tbl00

ID	Name	Level
1	Sarah	9
5	Sophia	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Name
23	Bob
24	Ben
27	Tony

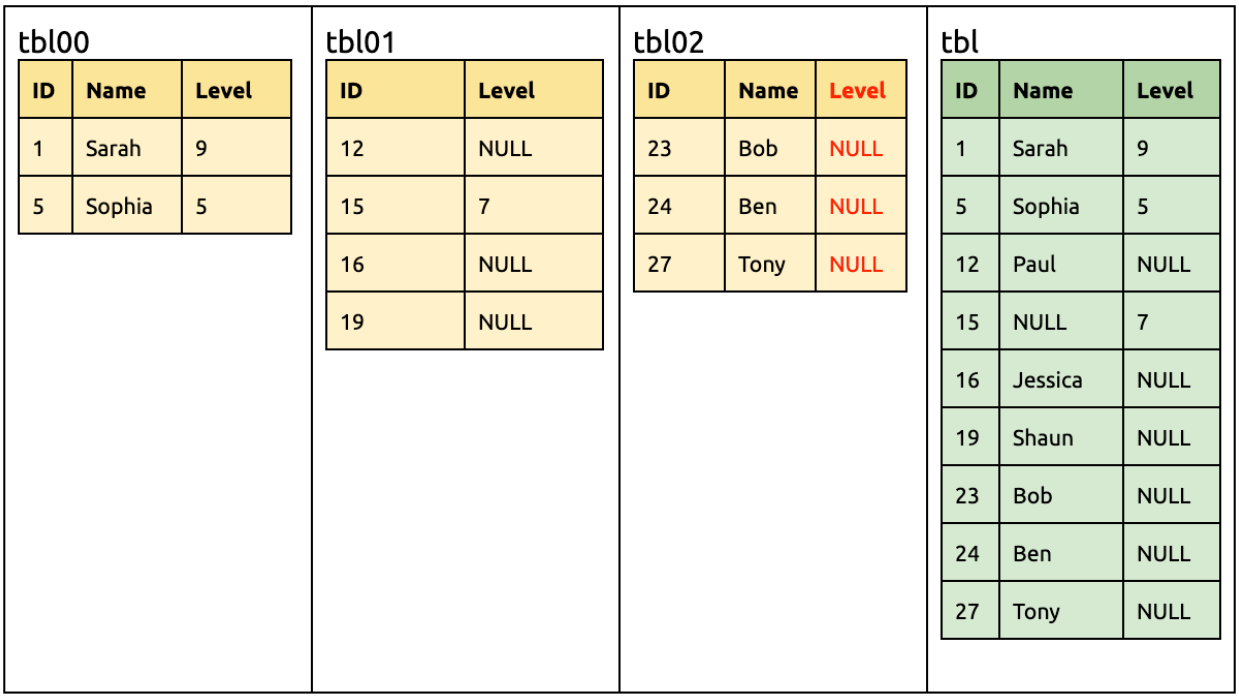
tbl

ID	Name	Level
1	Sarah	9
5	Sophia	5
12	Paul	NULL
15	NULL	7
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 251: optimistic-ddl-example-7

Add a `Level` column in `tbl02`:

```
ALTER TABLE `tbl02` ADD COLUMN `Level` INT;
```



tbl00

ID	Name	Level
1	Sarah	9
5	Sophia	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Name	Level
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

tbl

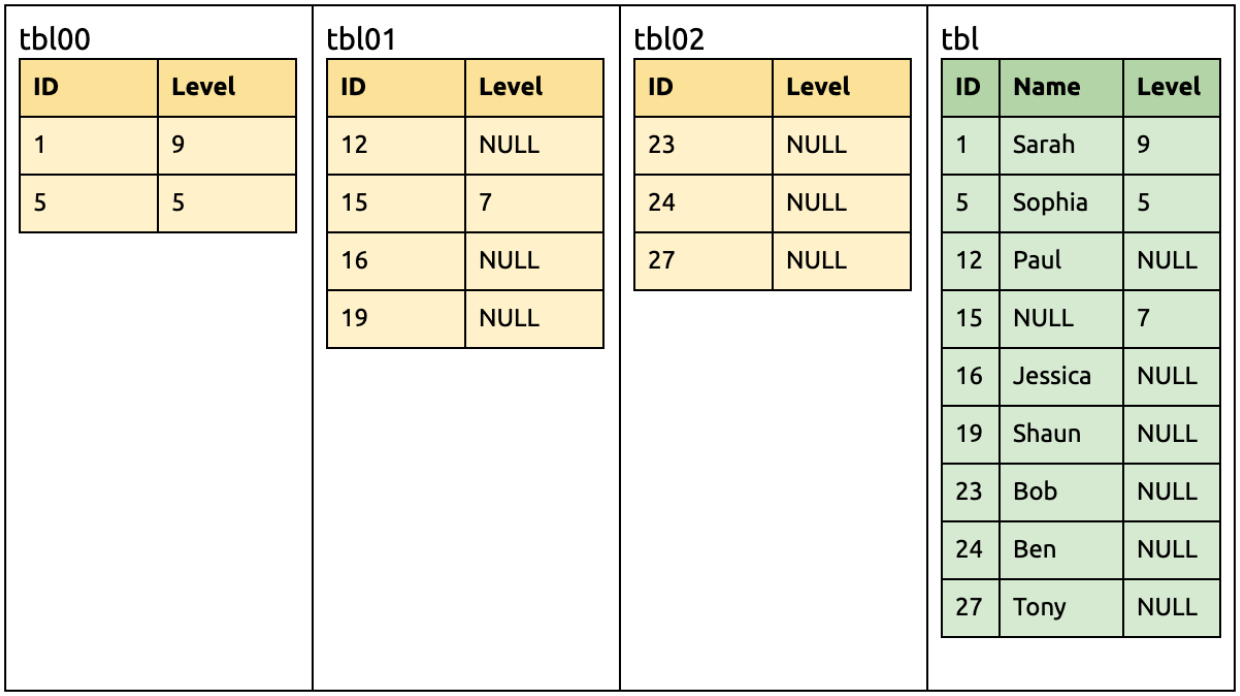
ID	Name	Level
1	Sarah	9
5	Sophia	5
12	Paul	NULL
15	NULL	7
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 252: optimistic-ddl-example-8

By then, all sharded tables have the `Level` column.

Drop the `Name` columns in `tbl00` and `tbl02` respectively:

```
ALTER TABLE `tbl00` DROP COLUMN `Name`;
ALTER TABLE `tbl02` DROP COLUMN `Name`;
```



tbl00

ID	Level
1	9
5	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Level
23	NULL
24	NULL
27	NULL

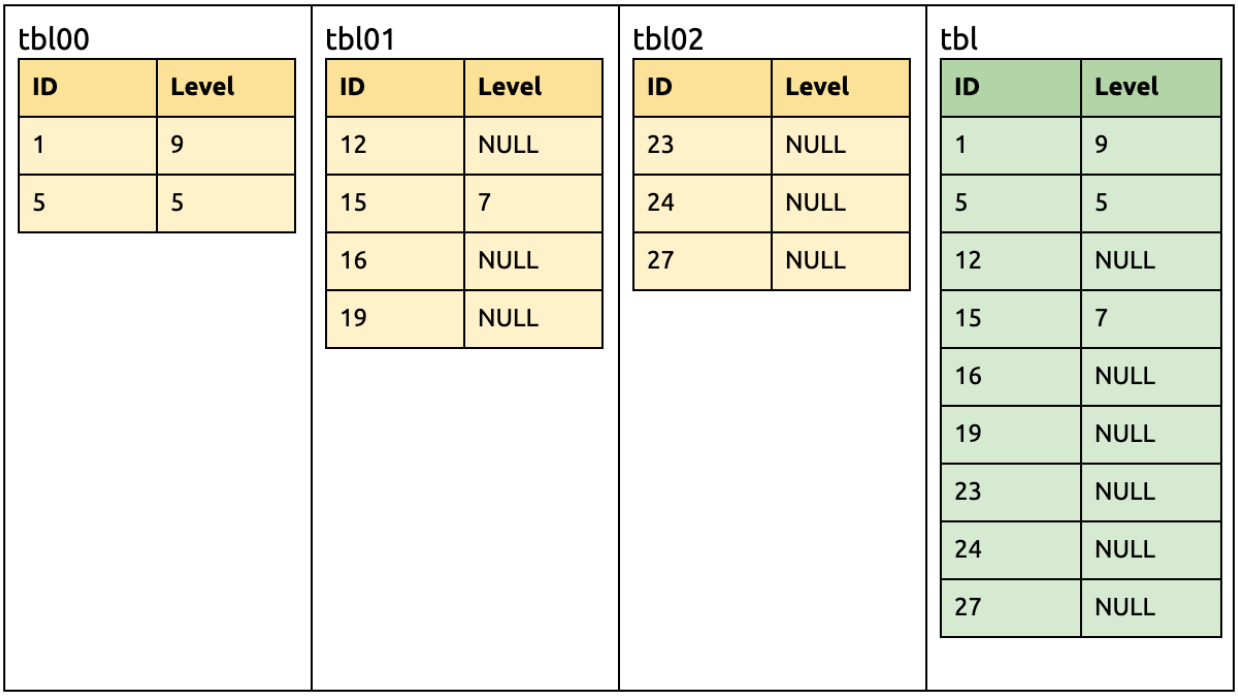
tbl

ID	Name	Level
1	Sarah	9
5	Sophia	5
12	Paul	NULL
15	NULL	7
16	Jessica	NULL
19	Shaun	NULL
23	Bob	NULL
24	Ben	NULL
27	Tony	NULL

Figure 253: optimistic-ddl-example-9

By then, the `Name` columns are dropped from all sharded tables and can be safely dropped in the downstream:

```
ALTER TABLE `tbl` DROP COLUMN `Name`;
```



tbl00

ID	Level
1	9
5	5

tbl01

ID	Level
12	NULL
15	7
16	NULL
19	NULL

tbl02

ID	Level
23	NULL
24	NULL
27	NULL

tbl

ID	Level
1	9
5	5
12	NULL
15	7
16	NULL
19	NULL
23	NULL
24	NULL
27	NULL

Figure 254: optimistic-ddl-example-10

13.6.6.1.4 Handle Sharding DDL Locks Manually in DM

DM uses the sharding DDL lock to ensure operations are performed in the correct order. This locking mechanism resolves sharding DDL locks automatically in most cases, but you need to use the `shard-ddl-lock` command to manually handle the abnormal DDL locks in some abnormal scenarios.

Note:

- This document only applies to the processing of sharding DDL lock in pessimistic coordination mode.
- The commands in the Command usage sections in this document are in interactive mode. In command-line mode, you need to add the escape characters to avoid an error report.
- Do not use `shard-ddl-lock unlock` unless you are totally aware of the possible impacts brought by the command and you can accept them.
- Before manually handling the abnormal DDL locks, make sure that you have already read the DM shard merge principles.

Command

`shard-ddl-lock`

You can use this command to view the DDL lock and request DM-master to release the specified DDL lock. This command is only supported in DM v6.0 and later. For earlier versions, you must use the `show-ddl-locks` and `unlock-ddl-locks` commands.

```
shard-ddl-lock -h
```

```
maintain or show shard-ddl locks information
Usage:
  dmctl shard-ddl-lock [task] [flags]
  dmctl shard-ddl-lock [command]
Available Commands:
  unlock    Unlock un-resolved DDL locks forcely
Flags:
  -h, --help help for shard-ddl-lock
Global Flags:
  -s, --source strings MySQL Source ID.
Use "dmctl shard-ddl-lock [command] --help" for more information about a
    ↪ command.
```

Arguments description

- `shard-ddl-lock [task] [flags]`: view the DDL lock information on the current DM-master.
- `shard-ddl-lock [command]`: request DM-master to release the specified DDL lock. `[command]` only accepts `unlock` as a value.

Usage examples

`shard-ddl-lock [task] [flags]`

You can use `shard-ddl-lock [task] [flags]` to view the DDL lock information on the current DM-master. For example:

```
shard-ddl-lock test
```

Expected output

```
{
    "result": true,                                 # The result of the
       ↪ query for the lock information.
    "msg": "",                                      # The additional message
       ↪  for the failure to query the lock information or other
       ↪ descriptive information (for example, the lock task does not exist
       ↪ ).
```

```
    "locks": [                                  # The existing lock
        ↪ information list.
        {
            "ID": "test-`shard_db`.`shard_table`",   # The lock ID, which is
                ↪ made up of the current task name and the schema/table
                ↪ information corresponding to the DDL.
            "task": "test",                     # The name of the task
                ↪ to which the lock belongs.
            "mode": "pessimistic"               # The shard DDL mode.
                ↪ Can be set to "pessimistic" or "optimistic".
            "owner": "mysql-replica-01",        # The owner of the lock
                ↪ (the ID of the first source that encounters this DDL
                ↪ operation in the pessimistic mode), which is always empty
                ↪ in the optimistic mode.
            "DDLs": [                           # The list of DDL
                ↪ operations corresponding to the lock in the pessimistic
                ↪ mode, which is always empty in the optimistic mode.
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` DROP
                    ↪ COLUMN `c2`;"
            ],
            "synced": [                         # The list of sources
                ↪ that have received all sharding DDL events in the
                ↪ corresponding MySQL instance.
                "mysql-replica-01"
            ],
            "unsynced": [                       # The list of sources
                ↪ that have not yet received all sharding DDL events in the
                ↪ corresponding MySQL instance.
                "mysql-replica-02"
            ]
        }
    ]
}
```

`shard-ddl-lock unlock`

This command actively requests `DM-master` to unlock the specified DDL lock, including requesting the owner to execute the DDL statement, requesting all other DM-workers that are not the owner to skip the DDL statement, and removing the lock information on `DM-master`.

Note:

> Currently, `shard-ddl-lock unlock` takes effect only for the lock in the `pessimistic` mode.

```
shard-ddl-lock unlock -h
```

```
Unlock un-resolved DDL locks forcely

Usage:
  dmctl shard-ddl-lock unlock <lock-id> [flags]

Flags:
  -a, --action string  accept skip/exec values which means whether to skip
      ↪ or execute ddls (default "skip")
  -d, --database string database name of the table
  -f, --force-remove   force to remove DDL lock
  -h, --help           help for unlock
  -o, --owner string   source to replace the default owner
  -t, --table string   table name

Global Flags:
  -s, --source strings MySQL Source ID.
```

`shard-ddl-lock unlock` accepts the following arguments:

- `-o, --owner`:
  - Flag; string; optional
  - If it is not specified, this command requests for the default owner (the owner in the result of `shard-ddl-lock`) to execute the DDL statement; if it is specified, this command requests for the MySQL source (the alternative of the default owner) to execute the DDL statement.
  - The new owner should not be specified unless the original owner is already removed from the cluster.
- `-f, --force-remove`:
  - Flag; boolean; optional
  - If it is not specified, this command removes the lock information only when the owner succeeds to execute the DDL statement; if it is specified, this command forcefully removes the lock information even though the owner fails to execute the DDL statement (after doing this you cannot query or operate on the lock again).
- `lock-id`:

- Non-flag; string; required
- It specifies the ID of the DDL lock that needs to be unlocked (the `ID` in the result of `shard-ddl-lock`).

The following is an example of the `shard-ddl-lock unlock` command:

```
shard-ddl-lock unlock test-`shard_db`.`shard_table`
```

```
{
    "result": true,                                     # The result of the
        ↪ unlocking operation.
    "msg": "",                                          # The additional message
        ↪  for the failure to unlock the lock.
}
```

Supported scenarios

Currently, the `shard-ddl-lock unlock` command only supports handling sharding DDL locks in the following two abnormal scenarios.

Scenario 1: Some MySQL sources are removed

The reason for the abnormal lock

Before `DM-master` tries to automatically unlock the sharding DDL lock, all the MySQL sources need to receive the sharding DDL events (for details, see shard merge principles). If the sharding DDL event is already in the migration process, and some MySQL sources have been removed and are not to be reloaded (these MySQL sources have been removed according to the application demand), then the sharding DDL lock cannot be automatically migrated and unlocked because not all the DM-workers can receive the DDL event.

> **Note:**
>
> If you need to make some DM-workers offline when not in the process of migrating sharding DDL events, a better solution is to use `stop-task` to stop the running tasks first, make the DM-workers go offline, remove the corresponding configuration information from the task configuration file, and finally use `start-task` and the new task configuration to restart the migration task.

Manual solution

Suppose that there are two instances `MySQL-1` (`mysql-replica-01`) and `MySQL-2` (`mysql-replica-02`) in the upstream, and there are two tables `shard_db_1.shard_table_1` and `shard_db_1.shard_table_2` in `MySQL-1` and two tables `shard_db_2.shard_table_1`

and `shard_db_2.shard_table_2` in `MySQL-2`. Now we need to merge the four tables and migrate them into the table `shard_db.shard_table` in the downstream TiDB.

The initial table structure is:

```
SHOW CREATE TABLE shard_db_1.shard_table_1;
+---------------+-----------------------------------------+
| Table         | Create Table                            |
+---------------+-----------------------------------------+
| shard_table_1 | CREATE TABLE `shard_table_1` (
  `c1` int(11) NOT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+---------------+-----------------------------------------+
```

The following DDL operation will be executed on the upstream sharded tables to alter the table structure:

```
ALTER TABLE shard_db_*.shard_table_* ADD COLUMN c2 INT;
```

The operation processes of MySQL and DM are as follows:

1. The corresponding DDL operations are executed on the two sharded tables of `mysql-replica-01` to alter the table structures.

   ```
   ALTER TABLE shard_db_1.shard_table_1 ADD COLUMN c2 INT;
   ```

   ```
   ALTER TABLE shard_db_1.shard_table_2 ADD COLUMN c2 INT;
   ```

2. DM-worker sends the received DDL information of the two sharded tables of `mysql-replica-01` to DM-master, and DM-master creates the corresponding DDL lock.

3. Use `shard-ddl-lock` to check the information of the current DDL lock.

   ```
   » shard-ddl-lock test
   {
       "result": true,
       "msg": "",
       "locks": [
           {
               "ID": "test-`shard_db`.`shard_table`",
               "task": "test",
               "mode": "pessimistic"
               "owner": "mysql-replica-01",
               "DDLs": [
                   "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` ADD
                       ↪  COLUMN `c2` int(11);"
   ```

```
            ],
            "synced": [
                "mysql-replica-01"
            ],
            "unsynced": [
                "mysql-replica-02"
            ]
        }
    ]
}
```

4. Due to the application demand, the data corresponding to `mysql-replica-02` is no longer needed to be migrated to the downstream TiDB, and `mysql-replica-02` is removed.

5. The lock whose ID is `test-`shard_db`.`shard_table`` on `DM-master` cannot receive the DDL information of `mysql-replica-02`.

    - The returned result `unsynced` by `shard-ddl-lock` has always included the information of `mysql-replica-02`.

6. Use `shard-ddl-lock unlock` to request `DM-master` to actively unlock the DDL lock.

    - If the owner of the DDL lock has gone offline, you can use the parameter `--owner` to specify another DM-worker as the new owner to execute the DDL.
    - If any MySQL source reports an error, `result` will be set to `false`, and at this point you should check carefully if the errors of each MySQL source is acceptable and within expectations.

    ```
    shard-ddl-lock unlock test-`shard_db`.`shard_table`
    ```

    ```
    {
        "result": true,
        "msg": ""
    ```

7. Use `shard-ddl-lock` to confirm if the DDL lock is unlocked successfully.

```
» shard-ddl-lock test
{
    "result": true,
    "msg": "no DDL lock exists",
    "locks": [
    ]
}
```

8. Check whether the table structure is altered successfully in the downstream TiDB.

```
mysql> SHOW CREATE TABLE shard_db.shard_table;
+-------------+-----------------------------------------------------+
| Table       | Create Table                                        |
+-------------+-----------------------------------------------------+
| shard_table | CREATE TABLE `shard_table` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+-------------+-----------------------------------------------------+
```

9. Use `query-status` to confirm if the migration task is normal.

Impact

After you have manually unlocked the lock by using `shard-ddl-lock unlock`, if you don't deal with the offline MySQL sources included in the task configuration information, the lock might still be unable to be migrated automatically when the next sharding DDL event is received.

Therefore, after you have manually unlocked the DDL lock, you should perform the following operations:

1. Use `stop-task` to stop the running tasks.
2. Update the task configuration file, and remove the related information of the offline MySQL source from the configuration file.
3. Use `start-task` and the new task configuration file to restart the task.

> **Note:**
>
> After you run `shard-ddl-lock unlock`, if the MySQL source that went offline is reloaded and the DM-worker tries to migrate the data of the sharded tables, a match error between the data and the downstream table structure might occur.

Scenario 2: Some DM-workers stop abnormally or the network failure occurs during the DDL unlocking process

The reason for the abnormal lock

After `DM-master` receives the DDL events of all DM-workers, automatically running `unlock DDL lock` mainly include the following steps:

1. Ask the owner of the lock to execute the DDL and update the checkpoints of corresponding sharded tables.
2. Remove the DDL lock information stored on `DM-master` after the owner successfully executes the DDL.
3. Ask all other non-owners to skip the DDL and update the checkpoints of corresponding sharded tables after the owner successfully executes the DDL.
4. DM-master removes the corresponding DDL lock information after all the owners or non-owners' operations are successful.

Currently, the above unlocking process is not atomic. If the non-owner skips the DDL operation successfully, the DM-worker where the non-owner is located stops abnormally or a network anomaly occurs with the downstream TiDB, which can cause the checkpoint updating to fail.

When the MySQL source corresponding to the non-owner restores data migration, the non-owner tries to request the DM-master to re-coordinate the DDL operation that has been coordinated before the exception occurs and will never receives the corresponding DDL operation from other MySQL sources. This can cause the DDL operation to automatically unlock the corresponding lock.

Manual solution

Suppose that now we have the same upstream and downstream table structures and the same demand for merging tables and migration as in the manual solution of Some MySQL sources are removed.

When `DM-master` automatically executes the unlocking process, the owner (`mysql-replica-01`) successfully executes the DDL and continues the migration process. However, in the process of requesting the non-owner (`mysql-replica-02`) to skip the DDL operation, the checkpoint fails to update after the DM-worker skips the DDL operation because the corresponding DM-worker was restarted.

After the data migration subtask corresponding to `mysql-replica-02` restores, a new lock is created on the DM-master, but other MySQL sources have executed or skipped DDL operations and are performing subsequent migration.

The operation processes are:

1. Use `shard-ddl-lock` to confirm if the corresponding lock of the DDL exists on `DM-master`.

   Only `mysql-replica-02` is at the `synced` state.

```
» shard-ddl-lock
{
    "result": true,
    "msg": "",
    "locks": [
        {
```

```
            "ID": "test-`shard_db`.`shard_table`",
            "task": "test",
            "mode": "pessimistic"
            "owner": "mysql-replica-02",
            "DDLs": [
                "USE `shard_db`; ALTER TABLE `shard_db`.`shard_table` ADD
                    ↪  COLUMN `c2` int(11);"
            ],
            "synced": [
                "mysql-replica-02"
            ],
            "unsynced": [
                "mysql-replica-01"
            ]
        }
    ]
}
```

2. Use `shard-ddl-lock` to ask `DM-master` to unlock the lock.

    - During the unlocking process, the owner tries to execute the DDL operation to the downstream again (the original owner before restarting has executed the DDL operation to the downstream once). Make sure that the DDL operation can be executed multiple times.

    ```
    shard-ddl-lock unlock test-`shard_db`.`shard_table`
    {
        "result": true,
        "msg": "",
    }
    ```

3. Use `shard-ddl-lock` to confirm if the DDL lock has been successfully unlocked.

4. Use `query-status` to confirm if the migration task is normal.

Impact

After manually unlocking the lock, the following sharding DDL can be migrated automatically and normally.

13.6.6.2 Migrate from Databases that Use GH-ost/PT-osc

In production scenarios, table locking during DDL execution can block the reads from or writes to the database to a certain extent. Therefore, online DDL tools are often used to execute DDLs to minimize the impact on reads and writes. Common DDL tools are gh-ost and pt-osc.

When using DM to migrate data from MySQL to TiDB, you can enbale online-ddl to allow collaboration of DM and gh-ost or pt-osc. For details about how to enable online-ddl and the workflow after enabling this option, see Continuous Replication with gh-ost or pt-osc. This document focuses on the collaboration details of DM and online DDL tools.

13.6.6.2.1 Working details for DM with online DDL tools

This section describes the working details for DM with the online DDL tools gh-ost and pt-osc when implementing online-schema-change.

online-schema-change: gh-ost

When gh-ost implements online-schema-change, 3 types of tables are created:

- gho: used to apply DDLs. When the data is fully replicated and the gho table is consistent with the origin table, the origin table is replaced by renaming.
- ghc: used to store information that is related to online-schema-change.
- del: created by renaming the origin table.

In the process of migration, DM divides the above tables into 3 categories:

- ghostTable: `_*_gho`
- trashTable: `_*_ghc`, `_*_del`
- realTable: the origin table that executes online-ddl.

The SQL statements mostly used by gh-ost and the corresponding operation of DM are as follows:

1. Create the `_ghc` table:

```
Create /* gh-ost */ table `test`.`_test4_ghc` (
                    id bigint auto_increment,
                    last_update timestamp not null DEFAULT
                        ↪ CURRENT_TIMESTAMP ON UPDATE
                        ↪ CURRENT_TIMESTAMP,
                    hint varchar(64) charset ascii not null,
                    value varchar(4096) charset ascii not null,
                    primary key(id),
                    unique key hint_uidx(hint)
            ) auto_increment=256 ;
```

   DM does not create the `_test4_ghc` table.

2. Create the `_gho` table:

```
Create /* gh-ost */ table `test`.`_test4_gho` like `test`.`test4` ;
```

DM does not create the `_test4_gho` table. DM deletes the `dm_meta.{task_name}_onlineddl` record in the downstream according to `ghost_schema`, `ghost_table`, and the `server_id` of `dm_worker`, and clears the related information in memory.

```
DELETE FROM dm_meta.{task_name}_onlineddl WHERE id = {server_id} and
    ↪ ghost_schema = {ghost_schema} and ghost_table = {ghost_table};
```

3. Apply the DDL that needs to be executed in the `_gho` table:

```
Alter /* gh-ost */ table `test`.`_test4_gho` add column cl1 varchar
    ↪ (20) not null ;
```

DM does not perform the DDL operation of `_test4_gho`. It records this DDL in `dm_meta.{task_name}_onlineddl` and memory.

```
REPLACE INTO dm_meta.{task_name}_onlineddl (id, ghost_schema ,
    ↪ ghost_table , ddls) VALUES (......);
```

4. Write data to the `_ghc` table, and replicate the origin table data to the `_gho` table:

```
INSERT /* gh-ost */ INTO `test`.`_test4_ghc` VALUES (......);
INSERT /* gh-ost `test`.`test4` */ ignore INTO `test`.`_test4_gho` (`
    ↪ id`, `date`, `account_id`, `conversion_price`, `
    ↪ ocpc_matched_conversions`, `ad_cost`, `cl2`)
  (SELECT `id`, `date`, `account_id`, `conversion_price`, `
      ↪ ocpc_matched_conversions`, `ad_cost`, `cl2` FROM `test`.`test4`
      ↪  FORCE INDEX (`PRIMARY`)
    WHERE (((`id` > _binary'1') OR ((`id` = _binary'1'))) AND ((`id` <
        ↪ _binary'2') OR ((`id` = _binary'2')))) lock IN share mode
  )   ;
```

DM does not execute DML statements that are not for **realtable**.

5. After the migration is completed, both the origin table and `_gho` table are renamed, and the online DDL operation is completed:

```
Rename /* gh-ost */ table `test`.`test4` to `test`.`_test4_del`, `test
    ↪ `.`_test4_gho` to `test`.`test4`;
```

DM performs the following two operations:

- DM splits the above `rename` operation into two SQL statements.

```
rename test.test4 to test._test4_del;
rename test._test4_gho to test.test4;
```

- DM does not execute `rename to _test4_del`. When executing `rename ↪ ghost_table to origin table`, DM takes the following steps:

- Read the DDL recorded in memory in Step 3
- Replace `ghost_table` and `ghost_schema` with `origin_table` and its corresponding schema
- Execute the DDL that has been replaced

```
alter table test._test4_gho add column cl1 varchar(20) not null;
-- Replaced with:
alter table test.test4 add column cl1 varchar(20) not null;
```

Note:

The specific SQL statements of gh-ost vary with the parameters used in the execution. This document only lists the major SQL statements. For more details, refer to the gh-ost documentation.

13.6.6.2.2 online-schema-change: pt

When pt-osc implements online-schema-change, 2 types of tables are created:

- `new`: used to apply DDL. When the data is fully replicated and the `new` table is consistent with the origin table, the origin table is replaced by renaming.
- `old`: created by renaming the origin table.
- 3 kinds of Trigger: `pt_osc_*_ins`, `pt_osc_*_upd`, `pt_osc_*_del`. In the process of pt_osc, the new data generated by the origin table is replicated to `new` by the Trigger.

In the process of migration, DM divides the above tables into 3 categories:

- ghostTable: `_*_new`
- trashTable: `_*_old`
- realTable: the origin table that executes online-ddl.

The SQL statements mostly used by pt-osc and the corresponding operation of DM are as follows:

1. Create the `_new` table:

```
CREATE TABLE `test`.`_test4_new` ( id int(11) NOT NULL AUTO_INCREMENT,
date date DEFAULT NULL, account_id bigint(20) DEFAULT NULL,
    ↪ conversion_price decimal(20,3) DEFAULT NULL,
    ↪ ocpc_matched_conversions bigint(20) DEFAULT NULL, ad_cost
    ↪ decimal(20,3) DEFAULT NULL,cl2 varchar(20) COLLATE utf8mb4_bin
    ↪ NOT NULL,cl1 varchar(20) COLLATE utf8mb4_bin NOT NULL,PRIMARY
    ↪ KEY (id) ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=
    ↪ utf8mb4 COLLATE=utf8mb4_bin ;
```

DM does not create the `_test4_new` table. DM deletes the `dm_meta.{task_name}_onlineddl` record in the downstream according to `ghost_schema`, `ghost_table`, and the `server_id` of `dm_worker`, and clears the related information in memory.

```
DELETE FROM dm_meta.{task_name}_onlineddl WHERE id = {server_id} and
    ↪ ghost_schema = {ghost_schema} and ghost_table = {ghost_table};
```

2. Execute DDL in the `_new` table:

```
ALTER TABLE `test`.`_test4_new` add column c3 int;
```

DM does not perform the DDL operation of `_test4_new`. Instead, it records this DDL in `dm_meta.{task_name}_onlineddl` and memory.

```
REPLACE INTO dm_meta.{task_name}_onlineddl (id, ghost_schema ,
    ↪ ghost_table , ddls) VALUES (......);
```

3. Create 3 Triggers used for data migration:

```
CREATE TRIGGER `pt_osc_test_test4_del` AFTER DELETE ON `test`.`test4`
    ↪ ...... ;
CREATE TRIGGER `pt_osc_test_test4_upd` AFTER UPDATE ON `test`.`test4`
    ↪ ...... ;
CREATE TRIGGER `pt_osc_test_test4_ins` AFTER INSERT ON `test`.`test4`
    ↪ ...... ;
```

DM does not execute Trigger operations that are not supported in TiDB.

4. Replicate the origin table data to the `_new` table:

```
INSERT LOW_PRIORITY IGNORE INTO `test`.`_test4_new` (`id`, `date`, `
    ↪ account_id`, `conversion_price`, `ocpc_matched_conversions`, `
    ↪ ad_cost`, `cl2`, `cl1`) SELECT `id`, `date`, `account_id`, `
    ↪ conversion_price`, `ocpc_matched_conversions`, `ad_cost`, `cl2`,
    ↪ `cl1` FROM `test`.`test4` LOCK IN SHARE MODE /*pt-online-schema-
    ↪ change 3227 copy table*/
```

DM does not execute the DML statements that are not for **realtable**.

5. After the data migration is completed, the origin table and `_new` table are renamed, and the online DDL operation is completed:

```
RENAME TABLE `test`.`test4` TO `test`.`_test4_old`, `test`.`_test4_new`
    ↪  TO `test`.`test4`
```

DM performs the following two operations:

- DM splits the above `rename` operation into two SQL statements:

  ```
  sql rename test.test4 to test._test4_old; rename test._test4_new
  ↪ to test.test4;
  ```

- DM does not execute `rename to _test4_old`. When executing `rename ↪ ghost_table to origin table`, DM takes the following steps:
  - Read the DDL recorded in memory in Step 2
  - Replace `ghost_table` and `ghost_schema` with `origin_table` and its corresponding schema
  - Execute the DDL that has been replaced

```
ALTER TABLE `test`.`_test4_new` add column c3 int;
-- Replaced with:
ALTER TABLE `test`.`test4` add column c3 int;
```

6. Delete the `_old` table and 3 Triggers of the online DDL operation:

```
DROP TABLE IF EXISTS `test`.`_test4_old`;
DROP TRIGGER IF EXISTS `pt_osc_test_test4_del` AFTER DELETE ON `test
   ↪ `.`test4` ..... ;
DROP TRIGGER IF EXISTS `pt_osc_test_test4_upd` AFTER UPDATE ON `test
   ↪ `.`test4` ..... ;
DROP TRIGGER IF EXISTS `pt_osc_test_test4_ins` AFTER INSERT ON `test
   ↪ `.`test4` ..... ;
```

DM does not delete `_test4_old` and Triggers.

Note:

The specific SQL statements of pt-osc vary with the parameters used in the execution. This document only lists the major SQL statements. For more details, refer to the pt-osc documentation.

13.6.6.2.3 Other online schema change tools

In some cases, you might need to change the default behavior of your online schema change tool. For example, you might use customized names for `ghost table` and `trash ↪ table`. In other cases, you might want to use other tools instead of gh-ost or pt-osc, with the same working principles and change processes.

To achieve such customized needs, you need to write regular expressions to match the names of the `ghost table` and `trash table`.

Starting from v2.0.7, DM experimentally supports the modified online schema change tools. By setting `online-ddl=true` in the DM task configuration and configuring `shadow-table-rules` and `trash-table-rules`, you can match the modified temporary tables with regular expressions.

For example, if you use a customized pt-osc with the name of `ghost table` being `_{origin_table}_pcnew` and the name of `trash table` being `_{origin_table}_pcold`, you can set the custom rules as follows:

```
online-ddl: true
shadow-table-rules: ["^_(.+)_(?:pcnew)$"]
trash-table-rules: ["^_(.+)_(?:pcold)$"]
```

13.6.6.3 Migrate Data to a Downstream TiDB Table with More Columns

This document provides the additional steps to be taken when you migrate data to a downstream TiDB table with more columns than the corresponding upstream table. For regular migration steps, see the following migration scenarios:

- Migrate MySQL of Small Datasets to TiDB
- Migrate MySQL of Large Datasets to TiDB
- Migrate and Merge MySQL Shards of Small Datasets to TiDB
- Migrate and Merge MySQL Shards of Large Datasets to TiDB

13.6.6.3.1 Use DM to migrate data to a downstream TiDB table with more columns

When replicating the upstream binlog, DM tries to use the current table schema of the downstream to parse the binlog and generate the corresponding DML statements. If the column number of the table in the upstream binlog does not match the column number in the downstream table schema, the following error occurs:

```
"errors": [
    {
        "ErrCode": 36027,
        "ErrClass": "sync-unit",
        "ErrScope": "internal",
        "ErrLevel": "high",
        "Message": "startLocation: [position: (mysql-bin.000001, 2022), gtid-
            ↪ set:09bec856-ba95-11ea-850a-58f2b4af5188:1-9 ], endLocation: [
            ↪  position: (mysql-bin.000001, 2022), gtid-set: 09bec856-ba95
            ↪ -11ea-850a-58f2b4af5188:1-9]: gen insert sqls failed, schema:
            ↪ log, table: messages: Column count doesn't match value count:
            ↪ 3 (columns) vs 2 (values)",
        "RawCause": "",
        "Workaround": ""
```

```
    }
]
```

The following is an example upstream table schema:

```
#### Upstream table schema
CREATE TABLE `messages` (
  `id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
)
```

The following is an example downstream table schema:

```
#### Downstream table schema
CREATE TABLE `messages` (
  `id` int(11) NOT NULL,
  `message` varchar(255) DEFAULT NULL, # This is the additional column that
    ↪ only exists in the downstream table.
  PRIMARY KEY (`id`)
)
```

When DM tries to use the downstream table schema to parse the binlog event generated by the upstream, DM reports the above `Column count doesn't match` error.

In such cases, you can use the `binlog-schema` command to set a table schema for the table to be migrated from the data source. The specified table schema needs to correspond to the binlog event data to be replicated by DM. If you are migrating sharded tables, for each sharded table, you need to set a table schema in DM to parse binlog event data. The steps are as follows:

1. Create a SQL file in DM and add the `CREATE TABLE` statement that corresponds to the upstream table schema to the file. For example, save the following table schema to `log.messages.sql`. For DM v6.0 or later versions, you can update the table schema by adding the `--from-source` or `--from-target` flag without creating a SQL file. For details, see Manage Table Schemas of Tables to be Migrated.

   ```
   # Upstream table schema
   CREATE TABLE `messages` (
   `id` int(11) NOT NULL,
   PRIMARY KEY (`id`)
   )
   ```

2. Use the `binlog-schema` command to set the table schema for the table to be migrated from the data source. At this time, the data migration task should be in the Paused state due to the above `Column count doesn't match` error.

```
tiup dmctl --master-addr ${advertise-addr} binlog-schema update -s ${
    ↪ source-id} ${task-name} ${database-name} ${table-name} ${schema-
    ↪ file}
```

The descriptions of parameters in this command are as follows:

Parameter	Description
`--master-addr`	Specifies the `${advertise-addr}` of any DM-master node in the cluster where dmctl is to be connected. `${advertise-addr}` indicates the address that DM-master advertises to the outside world.

Parameter	Description
`binlog-schema-set`	Manually set the schema information.
`-s`	Specifies the source. `${source-id}` indicates the source ID of MySQL data.

Parameter	Description
`${task-name}`	Specifies the name of the migration task defined in the `task.yaml` configuration file of the data migration task.

Parameter	Description
`${database-name}`	Specifies the database.
`${table-name}`	Specifies the name of the upstream table.
`${schema-file}`	Specifies the schema file to be set.

For example:

```
tiup dmctl --master-addr 172.16.10.71:8261 binlog-schema update -s
   ↪ mysql-01 task-test -d log -t message log.message.sql
```

3. Use the `resume-task` command to resume the migration task in the Paused state.

```
tiup dmctl --master-addr ${advertise-addr} resume-task ${task-name}
```

4. Use the `query-status` command to confirm that the data migration task is running correctly.

```
tiup dmctl --master-addr ${advertise-addr} query-status ${task-name}
```

13.6.7 Maintain

13.6.7.1 Cluster Upgrade

13.6.7.1.1 Maintain a DM Cluster Using TiUP

This document introduces how to maintain a DM cluster using the TiUP DM component.

If you have not deployed a DM cluster yet, you can refer to Deploy a DM Cluster Using TiUP for instructions.

> **Note:**
>
> - Make sure that the ports among the following components are interconnected
>   - The `peer_port` (`8291` by default) among the DM-master nodes are interconnected.
>   - Each DM-master node can connect to the `port` of all DM-worker nodes (`8262` by default).
>   - Each DM-worker node can connect to the `port` of all DM-master nodes (`8261` by default).
>   - The TiUP nodes can connect to the `port` of all DM-master nodes (`8261` by default).
>   - The TiUP nodes can connect to the `port` of all DM-worker nodes (`8262` by default).

For the help information of the TiUP DM component, run the following command:

```
tiup dm --help
```

```
Deploy a DM cluster for production

Usage:
  tiup dm [flags]
  tiup dm [command]

Available Commands:
```

```
  deploy     Deploy a DM cluster for production
  start      Start a DM cluster
  stop       Stop a DM cluster
  restart    Restart a DM cluster
  list       List all clusters
  destroy    Destroy a specified DM cluster
  audit      Show audit log of cluster operation
  exec       Run shell command on host in the dm cluster
  edit-config Edit DM cluster config
  display    Display information of a DM cluster
  reload     Reload a DM cluster's config and restart if needed
  upgrade    Upgrade a specified DM cluster
  patch      Replace the remote package with a specified package and restart
     ↪  the service
  scale-out  Scale out a DM cluster
  scale-in   Scale in a DM cluster
  import     Import an exist DM 1.0 cluster from dm-ansible and re-deploy
     ↪ 2.0 version
  help       Help about any command

Flags:
  -h, --help            help for tiup-dm
      --native-ssh      Use the native SSH client installed on local system
         ↪ instead of the build-in one.
      --ssh-timeout int Timeout in seconds to connect host via SSH, ignored
         ↪ for operations that don't need an SSH connection. (default 5)
  -v, --version         version for tiup-dm
      --wait-timeout int Timeout in seconds to wait for an operation to
         ↪ complete, ignored for operations that don't fit. (default 60)
  -y, --yes             Skip all confirmations and assumes 'yes'
```

View the cluster list

After the cluster is successfully deployed, view the cluster list by running the following command:

```
tiup dm list
```

```
Name  User  Version Path                                       PrivateKey
----  ----  ------- ----                                       ----------
prod-cluster tidb ${version} /root/.tiup/storage/dm/clusters/test /root/.
   ↪ tiup/storage/dm/clusters/test/ssh/id_rsa
```

Start the cluster

After the cluster is successfully deployed, start the cluster by running the following command:

```
tiup dm start prod-cluster
```

If you forget the name of your cluster, view the cluster list by running `tiup dm list`.

Check the cluster status

TiUP provides the `tiup dm display` command to view the status of each component in the cluster. With this command, you do not have to log in to each machine to see the component status. The usage of the command is as follows:

```
tiup dm display prod-cluster
```

```
dm Cluster: prod-cluster
dm Version: ${version}
ID               Role        Host        Ports     OS/Arch      Status
   ↪ Data Dir                          Deploy Dir
--               ----        ----        -----     -------      ------
   ↪ --------                          ----------
172.19.0.101:9093 alertmanager 172.19.0.101 9093/9094 linux/x86_64 Up    /
   ↪ home/tidb/data/alertmanager-9093 /home/tidb/deploy/alertmanager-9093
172.19.0.101:8261 dm-master  172.19.0.101 8261/8291 linux/x86_64 Healthy|L /
   ↪ home/tidb/data/dm-master-8261 /home/tidb/deploy/dm-master-8261
172.19.0.102:8261 dm-master  172.19.0.102 8261/8291 linux/x86_64 Healthy /
   ↪ home/tidb/data/dm-master-8261 /home/tidb/deploy/dm-master-8261
172.19.0.103:8261 dm-master  172.19.0.103 8261/8291 linux/x86_64 Healthy /
   ↪ home/tidb/data/dm-master-8261 /home/tidb/deploy/dm-master-8261
172.19.0.101:8262 dm-worker  172.19.0.101 8262     linux/x86_64 Free     /
   ↪ home/tidb/data/dm-worker-8262 /home/tidb/deploy/dm-worker-8262
172.19.0.102:8262 dm-worker  172.19.0.102 8262     linux/x86_64 Free     /
   ↪ home/tidb/data/dm-worker-8262 /home/tidb/deploy/dm-worker-8262
172.19.0.103:8262 dm-worker  172.19.0.103 8262     linux/x86_64 Free     /
   ↪ home/tidb/data/dm-worker-8262 /home/tidb/deploy/dm-worker-8262
172.19.0.101:3000 grafana    172.19.0.101 3000     linux/x86_64 Up       -
   ↪                                 /home/tidb/deploy/grafana-3000
172.19.0.101:9090 prometheus 172.19.0.101 9090     linux/x86_64 Up       /
   ↪ home/tidb/data/prometheus-9090 /home/tidb/deploy/prometheus-9090
```

The `Status` column uses `Up` or `Down` to indicate whether the service is running normally.

For the DM-master component, `|L` might be appended to a status, which indicates that the DM-master node is a Leader. For the DM-worker component, `Free` indicates that the current DM-worker node is not bound to an upstream.

Scale in a cluster

Scaling in a cluster means making some node(s) offline. This operation removes the specified node(s) from the cluster and deletes the remaining data files.

When you scale in a cluster, DM operations on DM-master and DM-worker components are performed in the following order:

1. Stop component processes.
2. Call the API for DM-master to delete the `member`.
3. Clean up the data files related to the node.

The basic usage of the scale-in command:

```
tiup dm scale-in <cluster-name> -N <node-id>
```

To use this command, you need to specify at least two arguments: the cluster name and the node ID. The node ID can be obtained by using the `tiup dm display` command in the previous section.

For example, to scale in the DM-worker node on `172.16.5.140` (similar to scaling in DM-master), run the following command:

```
tiup dm scale-in prod-cluster -N 172.16.5.140:8262
```

Scale out a cluster

The scale-out operation has an inner logic similar to that of deployment: the TiUP DM component first ensures the SSH connection of the node, creates the required directories on the target node, then executes the deployment operation, and starts the node service.

For example, to scale out a DM-worker node in the `prod-cluster` cluster, take the following steps (scaling out DM-master has similar steps):

1. Create a `scale.yaml` file and add information of the new worker node:

> **Note:**
> You need to create a topology file, which includes only the description of the new nodes, not the existing nodes. For more configuration items (such as the deployment directory), refer to this TiUP configuration parameter example.

```
---

worker_servers:
  - host: 172.16.5.140
```

2. Perform the scale-out operation. TiUP DM adds the corresponding nodes to the cluster according to the port, directory, and other information described in `scale.yaml`.

```
tiup dm scale-out prod-cluster scale.yaml
```

After the command is executed, you can check the status of the scaled-out cluster by running `tiup dm display prod-cluster`.

Rolling upgrade

Note:

Since v2.0.5, dmctl support Export and Import Data Sources and Task Configuration of Clusters.

Before upgrading, you can use `config export` to export the configuration files of clusters. After upgrading, if you need to downgrade to an earlier version, you can first redeploy the earlier cluster and then use `config import` to import the previous configuration files.

For clusters earlier than v2.0.5, you can use dmctl v2.0.5 or later to export and import the data source and task configuration files.

For clusters later than v2.0.2, currently, it is not supported to automatically import the configuration related to relay worker. You can use `start-relay` command to manually start relay log.

The rolling upgrade process is made as transparent as possible to the application, and does not affect the business. The operations vary with different nodes.

Upgrade command

You can run the `tiup dm upgrade` command to upgrade a DM cluster. For example, the following command upgrades the cluster to `${version}`. Modify `${version}` to your needed version before running this command:

```
tiup dm upgrade prod-cluster ${version}
```

Update configuration

If you want to dynamically update the component configurations, the TiUP DM component saves a current configuration for each cluster. To edit this configuration, execute the `tiup dm edit-config <cluster-name>` command. For example:

```
tiup dm edit-config prod-cluster
```

TiUP DM opens the configuration file in the vi editor. If you want to use other editors, use the `EDITOR` environment variable to customize the editor, such as `export EDITOR=nano` ↪ . After editing the file, save the changes. To apply the new configuration to the cluster, execute the following command:

```
tiup dm reload prod-cluster
```

The command sends the configuration to the target machine and restarts the cluster to make the configuration take effect.

Update component

For normal upgrade, you can use the `upgrade` command. But in some scenarios, such as debugging, you might need to replace the currently running component with a temporary package. To achieve this, use the `patch` command:

```
tiup dm patch --help
```

```
Replace the remote package with a specified package and restart the service

Usage:
  tiup dm patch <cluster-name> <package-path> [flags]

Flags:
  -h, --help                help for patch
  -N, --node strings        Specify the nodes
      --overwrite           Use this package in the future scale-out
          ↪ operations
  -R, --role strings        Specify the role
      --transfer-timeout int Timeout in seconds when transferring dm-master
          ↪ leaders (default 300)

Global Flags:
      --native-ssh      Use the native SSH client installed on local system
          ↪ instead of the build-in one.
      --ssh-timeout int  Timeout in seconds to connect host via SSH, ignored
          ↪ for operations that don't need an SSH connection. (default 5)
      --wait-timeout int Timeout in seconds to wait for an operation to
          ↪ complete, ignored for operations that don't fit. (default 60)
  -y, --yes             Skip all confirmations and assumes 'yes'
```

If a DM-master hotfix package is in `/tmp/dm-master-hotfix.tar.gz` and you want to replace all the DM-master packages in the cluster, run the following command:

```
tiup dm patch prod-cluster /tmp/dm-master-hotfix.tar.gz -R dm-master
```

You can also replace only one DM-master package in the cluster:

```
tiup dm patch prod-cluster /tmp/dm--hotfix.tar.gz -N 172.16.4.5:8261
```

Import and upgrade a DM 1.0 cluster deployed using DM-Ansible

Note:

- TiUP does not support importing the DM Portal component in a DM 1.0 cluster.
- You need to stop the original cluster before importing.
- Don't run `stop-task` for tasks that need to be upgraded to 2.0.
- TiUP only supports importing to a DM cluster of v2.0.0-rc.2 or a later version.
- The `import` command is used to import data from a DM 1.0 cluster to a new DM 2.0 cluster. If you need to import DM migration tasks to an existing DM 2.0 cluster, refer to Manually Upgrade TiDB Data Migration from v1.0.x to v2.0+.
- The deployment directories of some components are different from those of the original cluster. You can execute the `display` command to view the details.
- Run `tiup update --self && tiup update dm` before importing to make sure that the TiUP DM component is the latest version.
- Only one DM-master node exists in the cluster after importing. Refer to Scale out a cluster to scale out the DM-master.

Before TiUP is released, DM-Ansible is often used to deploy DM clusters. To enable TiUP to take over the DM 1.0 cluster deployed by DM-Ansible, use the `import` command.

For example, to import a cluster deployed using DM Ansible:

```
tiup dm import --dir=/path/to/dm-ansible --cluster-version ${version}
```

Execute `tiup list dm-master` to view the latest cluster version supported by TiUP.

The process of using the `import` command is as follows:

1. TiUP generates a topology file `topology.yml` based on the DM cluster previously deployed using DM-Ansible.
2. After confirming that the topology file has been generated, you can use it to deploy the DM cluster of v2.0 or later versions.

After the deployment is completed, you can execute the `tiup dm start` command to start the cluster and begin the process of upgrading the DM kernel.

View the operation log

To view the operation log, use the `audit` command. The usage of the `audit` command is as follows:

```
Usage:
  tiup dm audit [audit-id] [flags]

Flags:
  -h, --help help for audit
```

If the `[audit-id]` argument is not specified, the command shows a list of commands that have been executed. For example:

```
tiup dm audit
```

```
ID     Time                 Command
--     ----                 -------
4D5kQY 2020-08-13T05:38:19Z tiup dm display test
4D5kNv 2020-08-13T05:36:13Z tiup dm list
4D5kNr 2020-08-13T05:36:10Z tiup dm deploy -p prod-cluster ${version} ./
    ↪ examples/dm/minimal.yaml
```

The first column is `audit-id`. To view the execution log of a certain command, pass the `audit-id` argument as follows:

```
tiup dm audit 4D5kQY
```

Run commands on a host in the DM cluster

To run commands on a host in the DM cluster, use the `exec` command. The usage of the `exec` command is as follows:

```
Usage:
  tiup dm exec <cluster-name> [flags]

Flags:
      --command string the command run on cluster host (default "ls")
  -h, --help           help for exec
  -N, --node strings   Only exec on host with specified nodes
  -R, --role strings   Only exec on host with specified roles
      --sudo           use root permissions (default false)
```

For example, to execute `ls /tmp` on all DM nodes, run the following command:

```
tiup dm exec prod-cluster --command='ls /tmp'
```

dmctl

TiUP integrates the DM cluster controller `dmctl`.

Run the following command to use dmctl:

```
tiup dmctl [args]
```

Specify the version of dmctl. Modify `${version}` to your needed version before running this command:

```
tiup dmctl:${version} [args]
```

The previous dmctl command to add a source is `dmctl --master-addr master1:8261 ↪ operate-source create /tmp/source1.yml`. After dmctl is integrated into TiUP, the command is:

```
tiup dmctl --master-addr master1:8261 operate-source create /tmp/source1.
    ↪ yml
```

Use the system's native SSH client to connect to cluster

All operations above performed on the cluster machine use the SSH client embedded in TiUP to connect to the cluster and execute commands. However, in some scenarios, you might also need to use the SSH client native to the control machine system to perform such cluster operations. For example:

- To use a SSH plug-in for authentication
- To use a customized SSH client

Then you can use the `--native-ssh` command-line flag to enable the system-native command-line tool:

- Deploy a cluster: `tiup dm deploy <cluster-name> <version> <topo> --native- ↪ ssh`
- Start a cluster: `tiup dm start <cluster-name> --native-ssh`
- Upgrade a cluster: `tiup dm upgrade ... --native-ssh`

You can add `--native-ssh` in all cluster operation commands above to use the system's native SSH client.

To avoid adding such a flag in every command, you can use the `TIUP_NATIVE_SSH` system variable to specify whether to use the local SSH client:

```
export TIUP_NATIVE_SSH=true
##### or
export TIUP_NATIVE_SSH=1
##### or
export TIUP_NATIVE_SSH=enable
```

If you specify this environment variable and `--native-ssh` at the same time, `--native ↪ -ssh` has higher priority.

Note:

During the process of cluster deployment, if you need to use a password for connection or `passphrase` is configured in the key file, you must ensure that `sshpass` is installed on the control machine; otherwise, a timeout error is reported.

13.6.7.1.2 Manually Upgrade TiDB Data Migration from v1.0.x to v2.0+

This document introduces how to manually upgrade the TiDB DM tool from v1.0.x to v2.0+. The main idea is to use the global checkpoint information in v1.0.x to start a new data migration task in the v2.0+ cluster.

For how to automatically upgrade the TiDB DM tool from v1.0.x to v2.0+, refer to Using TiUP to automatically import the 1.0 cluster deployed by DM-Ansible.

Note:

- Currently, upgrading DM from v1.0.x to v2.0+ is not supported when the data migration task is in the process of full export or full import.
- As the gRPC protocol used for interaction between the components of the DM cluster is updated greatly, you need to make sure that the DM components (including dmctl) use the same version before and after the upgrade.
- Because the metadata storage of the DM cluster (such as checkpoint, shard DDL lock status, and online DDL metadata) is updated greatly, the metadata of v1.0.x cannot be reused automatically in v2.0+. So you need to make sure the following requirements are satisfied before performing the upgrade operation:
  - All data migration tasks are not in the process of shard DDL coordination.
  - All data migration tasks are not in the process of online DDL coordination.

The steps for manual upgrade are as follows.

Step 1: Prepare v2.0+ configuration file

The prepared configuration files of v2.0+ include the configuration files of the upstream database and the configuration files of the data migration task.

Upstream database configuration file

In v2.0+, the upstream database configuration file is separated from the process configuration of the DM-worker, so you need to obtain the source configuration based on the v1.0.x DM-worker configuration.

> **Note:**
>
> If `enable-gtid` in the source configuration is enabled during the upgrade from v1.0.x to v2.0+, you need to parse the binlog or relay log file to obtain the GTID sets corresponding to the binlog position.

Upgrade a v1.0.x cluster deployed by DM-Ansible

Assume that the v1.0.x DM cluster is deployed by DM-Ansible, and the following `dm_worker_servers` configuration is in the `inventory.ini` file:

```
[dm_master_servers]
dm_worker1 ansible_host=172.16.10.72 server_id=101 source_id="mysql-replica
    ↪ -01" mysql_host=172.16.10.81 mysql_user=root mysql_password='
    ↪ VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
dm_worker2 ansible_host=172.16.10.73 server_id=102 source_id="mysql-replica
    ↪ -02" mysql_host=172.16.10.82 mysql_user=root mysql_password='
    ↪ VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=' mysql_port=3306
```

Then you can convert it to the following two source configuration files:

```
##### The source configuration corresponding to the original dm_worker1. For
    ↪  example, it is named as source1.yaml.
server-id: 101                                # Corresponds to the original `
    ↪ server_id`.
source-id: "mysql-replica-01"                 # Corresponds to the original `
    ↪ source_id`.
from:
  host: "172.16.10.81"                        # Corresponds to the original `
     ↪ mysql_host`.
  port: 3306                                  # Corresponds to the original `
     ↪ mysql_port`.
  user: "root"                                # Corresponds to the original `
     ↪ mysql_user`.
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # Corresponds to the original
     ↪  `mysql_password`.
```

```
##### The source configuration corresponding to the original dm_worker2. For
    ↪  example, it is named as source2.yaml.
server-id: 102                                # Corresponds to the original `
    ↪ server_id`.
source-id: "mysql-replica-02"                 # Corresponds to the original `
    ↪ source_id`.
from:
  host: "172.16.10.82"                        # Corresponds to the original `
      ↪ mysql_host`.
  port: 3306                                  # Corresponds to the original `
      ↪ mysql_port`.
  user: "root"                                # Corresponds to the original `
      ↪ mysql_user`.
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # Corresponds to the original
      ↪  `mysql_password`.
```

Upgrade a v1.0.x cluster deployed by binary

Assume that the v1.0.x DM cluster is deployed by binary, and the corresponding DM-worker configuration is as follows:

```
log-level = "info"
log-file = "dm-worker.log"
worker-addr = ":8262"
server-id = 101
source-id = "mysql-replica-01"
flavor = "mysql"
[from]
host = "172.16.10.81"
user = "root"
password = "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU="
port = 3306
```

Then you can convert it to the following source configuration file:

```
server-id: 101                                # Corresponds to the original `
    ↪ server-id`.
source-id: "mysql-replica-01"                 # Corresponds to the original `
    ↪ source-id`.
flavor: "mysql"                               # Corresponds to the original `
    ↪ flavor`.
from:
  host: "172.16.10.81"                        # Corresponds to the original `
      ↪ from.host`.
  port: 3306                                  # Corresponds to the original `
      ↪ from.port`.
```

```
user: "root"                                    # Corresponds to the original `
    ↪ from.user`.
password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=" # Corresponds to the original
    ↪  `from.password`.
```

Data migration task configuration file

For data migration task configuration guide, v2.0+ is basically compatible with v1.0.x. You can directly copy the configuration of v1.0.x.

Step 2: Deploy the v2.0+ cluster

> **Note:**
>
> Skip this step if you have other v2.0+ clusters available.

Use TiUP to deploy a new v2.0+ cluster according to the required number of nodes.

Step 3: Stop the v1.0.x cluster

If the original v1.0.x cluster is deployed by DM-Ansible, you need to use DM-Ansible to stop the v1.0.x cluster.

If the original v1.0.x cluster is deployed by binary, you can stop the DM-worker and DM-master processes directly.

Step 4: Upgrade data migration task

1. Use the `operate-source` command to load the upstream database source configuration from step 1 into the v2.0+ cluster.
2. In the downstream TiDB cluster, obtain the corresponding global checkpoint information from the incremental checkpoint table of the v1.0.x data migration task.

    - Assume that the v1.0.x data migration configuration does not specify `meta-schema` (or specify its value as the default `dm_meta`), and the corresponding task name is `task_v1`, the corresponding checkpoint information is in the `` `dm_meta`.`task_v1_syncer_checkpoint` `` table of the downstream TiDB.
    - Use the following SQL statements to obtain the global checkpoint information of all upstream database sources corresponding to the data migration task.

```
> SELECT `id`, `binlog_name`, `binlog_pos` FROM `dm_meta`.`
    ↪ task_v1_syncer_checkpoint` WHERE `is_global`=1;
+-----------------+------------------------+------------+
| id              | binlog_name            | binlog_pos |
+-----------------+------------------------+------------+
```

```
| mysql-replica-01 | mysql-bin|000001.000123 | 15847 |
| mysql-replica-02 | mysql-bin|000001.000456 | 10485 |
+------------------+--------------------------+-----------+
```

3. Update the v1.0.x data migration task configuration file to start a new v2.0+ data migration task.

    - If the data migration task configuration file of v1.0.x is `task_v1.yaml`, copy it and rename it to `task_v2.yaml`.
    - Make the following changes to `task_v2.yaml`:
        - Modify `name` to a new name, such as `task_v2`.
        - Change `task-mode` to `incremental`.
        - Set the starting point of incremental replication for each source according to the global checkpoint information obtained in step 2. For example:

```
mysql-instances:
  - source-id: "mysql-replica-01"   # Corresponds to the `id`
      ↪ of the checkpoint information.
    meta:
      binlog-name: "mysql-bin.000123" # Corresponds to the `
          ↪ binlog_name` in the checkpoint information,
          ↪ excluding the part of `|000001`.
      binlog-pos: 15847             # Corresponds to `
          ↪ binlog_pos` in the checkpoint information.

  - source-id: "mysql-replica-02"
    meta:
      binlog-name: "mysql-bin.000456"
      binlog-pos: 10485
```

> **Note:**
> If `enable-gtid` is enabled in the source configuration, currently you need to parse the binlog or relay log file to obtain the GTID sets corresponding to the binlog position, and set it to `binlog-gtid` in the `meta`.

4. Use the `start-task` command to start the upgraded data migration task through the v2.0+ data migration task configuration file.

5. Use the `query-status` command to confirm whether the data migration task is running normally.

If the data migration task runs normally, it indicates that the DM upgrade to v2.0+ is successful.

13.6.7.2 Tools

13.6.7.2.1 Use WebUI to Manage DM migration tasks

DM WebUI is a web-based GUI platform for managing TiDB Data Migration (DM) tasks. This platform provides a simple and intuitive way to manage a large number of migration tasks, which frees you from using the dmctl command-line tool.

This document introduces how to access DM WebUI, the prerequisites, the use cases of each page on the interface, and the attention points.

> **Warning:**
>
> - DM WebUI is currently an experimental feature. It is not recommended to use it in the production environment.
> - The lifecycle of `task` in DM WebUI has been changed, and it is not recommended to use DM WebUI and dmctl at the same time.

DM WebUI has the following pages:

- **Migration**
  - **Task**: Provides an entry to task creation, and displays the detailed information of each migration task. This page helps you monitor, create, delete, and configure migration tasks.
  - **Source**: Configures the information of upstream data source for a migration task. On this page, you can manage the upstream configuration in a data migration environment, including creating and deleting upstream configuration, monitoring the task status corresponding to the upstream configuration, and modifying upstream configuration.
  - **Replication Detail**: Displays the detailed status information of migration tasks. On this page, you can view the detailed configuration and status information based on a specified filter, including the configuration information and database names of the upstream and downstream, the relation of source tables and target tables.
- **Cluster**
  - **Members**: Displays the list of all master and worker nodes in the DM cluster, and the binding relationship between worker nodes and the source. On this page, you can view the configuration information of the current DM cluster and the status information of each worker. In addition, basic management is also provided on this page.

The interface is as follows:

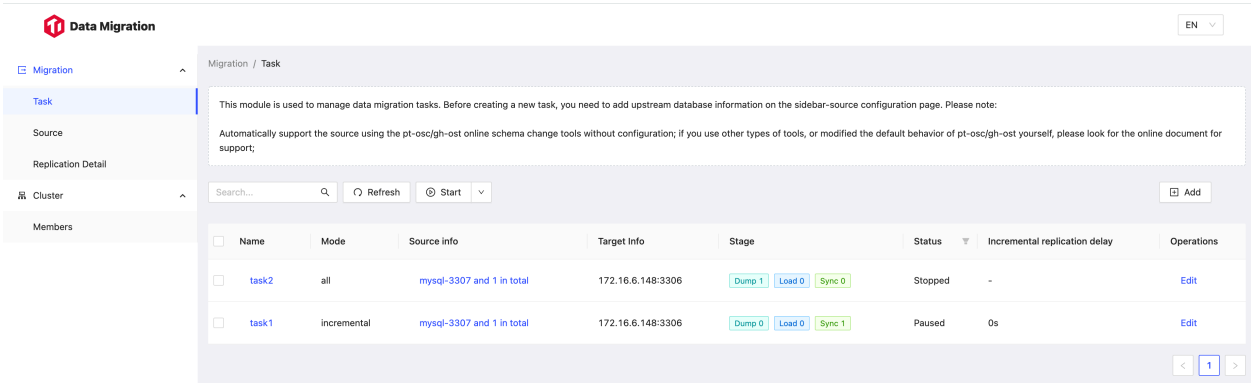


Figure 255: webui

Access method

When OpenAPI is enabled, you can access the DM WebUI from any master node of the DM cluster. The access port is `8261` by default and is the same as that of DM OpenAPI. Here is an example of an access address: `http://{master_ip}:{master_port}/dashboard/`.

Migration

Migration includes **Source**, **Task**, and **Replication Detail** pages.

Source

Before creating a migration task, you need to create the data source information of the upstream for the replication task. You can create the upstream configuration in the **Source** page. When creating sources, pay attention to the following items:

- If there is a auto failover between primary and secondary instance, enable GTID in the upstream MySQL and set GTID to `True` when creating the upstream configuration; otherwise, the migration task will be interrupted during the failover (except for AWS Aurora).
- If a MySQL instance needs to be temporarily offline, you can disable the instance. However, when the MySQL instance is being disabled, other MySQL instances running migration tasks should not execute DDL operations; otherwise, the disabled instance cannot properly migrate data after it is enabled.
- When multiple migration tasks use the same upstream, it might cause additional stress. Enabling relay log can reduce the impact on the upstream, so it is recommended to enable relay log.

Task

You can view the migration task details on the **Task** page, and create migration tasks.

View migration task details

In the task list, click the task name to view the Details page from the right. The Details page displays more detailed task status information. On this page, you can view the status of each sub-task and the current configuration information of the migration task.

In DM, each sub-task of a migration task might be at different stages, namely full dump -> full import (load) -> incremental replication (sync). Therefore, the current stage of a task is displayed with the statistics of the sub-task statuses, which can help you better understand the running status of the task.

Create migration tasks

To create a migration task on this page, click the **Add** button on the top right corner. You can use one of the following methods to create a migration task:

- By following the WebUI instruction. Fill in the required information step by step on the WebUI. This method is suitable for beginners and for daily use.
- By using a configuration file. Paste or write a JSON-formatted configuration file to create a migration task. This method supports adjusting more parameters and is suitable for advanced users.

Replication detail

You can view the status of the migration rules configured for a migration task on the **Replication Detail** page. This page supports querying by task, source, and database name.

The query result contains the corresponding information of the upstream table and the downstream table, so be careful using `.*` in case that too many query results slow down the page response.

Cluster

Members

The **Members** page displays all the master and worker nodes in the DM cluster, and the binding relationship between worker nodes and the source.

13.6.7.2.2 Maintain DM Clusters Using dmctl

> **Note:**
>
> For DM clusters deployed using TiUP, you are recommended to directly use `tiup dmctl` to maintain the clusters.

dmctl is a command line tool used to maintain DM clusters. It supports both the interactive mode and the command mode.

Interactive mode

Enter the interactive mode to interact with DM-master:

> **Note:**
>
> The interactive mode does not support Bash features. For example, you need to directly pass string flags instead of passing them in quotes.

```
./dmctl --master-addr 172.16.30.14:8261
```

```
Welcome to dmctl
Release Version: ${version}
Git Commit Hash: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Git Branch: release-x.x
UTC Build Time: yyyy-mm-dd hh:mm:ss
Go Version: go version gox.xx linux/amd64

» help
DM control

Usage:
  dmctl [command]

Available Commands:
  binlog         manage or show binlog operations
  binlog-schema manage or show table schema in schema tracker
  check-task     Checks the configuration file of the task
  config         manage config operations
  decrypt        Decrypts cipher text to plain text
  encrypt        Encrypts plain text to cipher text
  help           Gets help about any command
  list-member    Lists member information
  offline-member Offlines member which has been closed
  operate-leader `evict`/`cancel-evict` the leader
  operate-source `create`/`stop`/`show` upstream MySQL/MariaDB source
  pause-relay    Pauses DM-worker's relay unit
  pause-task     Pauses a specified running task or all (sub)tasks bound to a
     ↪ source
  purge-relay    Purges relay log files of the DM-worker according to the
     ↪ specified filename
  query-status   Queries task status
  resume-relay   Resumes DM-worker's relay unit
  resume-task    Resumes a specified paused task or all (sub)tasks bound to a
     ↪ source
```

```
  shard-ddl-lock maintain or show shard-ddl locks information
  start-relay   Starts workers pulling relay log for a source
  start-task    Starts a task as defined in the configuration file
  stop-relay    Stops workers pulling relay log for a source
  stop-task     Stops a specified task or all (sub)tasks bound to a source
  transfer-source Transfers a upstream MySQL/MariaDB source to a free worker

Flags:
  -h, --help          Help for dmctl.
  -s, --source strings MySQL Source ID.

Use "dmctl [command] --help" for more information about a command.
```

Command mode

The command mode differs from the interactive mode in that you need to append the task operation right after the dmctl command. The parameters of the task operation in the command mode are the same as those in the interactive mode.

Note:

- A dmctl command must be followed by only one task operation.
- Starting from v2.0.4, DM supports reading the `-master-addr` parameter from the environment variable `DM_MASTER_ADDR`.

```
./dmctl --master-addr 172.16.30.14:8261 start-task task.yaml
./dmctl --master-addr 172.16.30.14:8261 stop-task task
./dmctl --master-addr 172.16.30.14:8261 query-status

export DM_MASTER_ADDR="172.16.30.14:8261"
./dmctl query-status
```

```
Available Commands:
  binlog        manage or show binlog operations
  binlog-schema manage or show table schema in schema tracker
  check-task    Checks the configuration file of the task
  config        manage config operations
  decrypt       Decrypts cipher text to plain text
  encrypt       Encrypts plain text to cipher text
  help          Gets help about any command
  list-member   Lists member information
  offline-member Offlines member which has been closed
```

```
  operate-leader `evict`/`cancel-evict` the leader
  operate-source `create`/`stop`/`show` upstream MySQL/MariaDB source
  pause-relay   Pauses DM-worker's relay unit
  pause-task    Pauses a specified running task or all (sub)tasks bound to a
     ↪  source
  purge-relay   Purges relay log files of the DM-worker according to the
     ↪ specified filename
  query-status  Queries task status
  resume-relay  Resumes DM-worker's relay unit
  resume-task   Resumes a specified paused task or all (sub)tasks bound to a
     ↪  source
  shard-ddl-lock maintain or show shard-ddl locks information
  start-relay   Starts workers pulling relay log for a source
  start-task    Starts a task as defined in the configuration file
  stop-relay    Stops workers pulling relay log for a source
  stop-task     Stops a specified task or all (sub)tasks bound to a source
  transfer-source Transfers a upstream MySQL/MariaDB source to a free worker

Flags:
      --config string     Path to config file.
  -h, --help              help for dmctl
      --master-addr string Master API server address, this parameter is
         ↪ required when interacting with the dm-master
      --rpc-timeout string RPC timeout, default is 10m. (default "10m")
  -s, --source strings    MySQL Source ID.
      --ssl-ca string     Path of file that contains list of trusted SSL CAs
         ↪  for connection.
      --ssl-cert string   Path of file that contains X509 certificate in PEM
         ↪  format for connection.
      --ssl-key string    Path of file that contains X509 key in PEM format
         ↪ for connection.
  -V, --version           Prints version and exit.

Use "dmctl [command] --help" for more information about a command.
```

13.6.7.3 Performance Tuning

13.6.7.3.1 DM 5.4.0 Benchmark Report

The benchmark data of TiDB Data Migration (DM) v5.4.0 is nearly the same as that of DM v5.3.0. To learn the test purpose, environment, scenarios, and results of DM v5.3.0, refer to DM 5.3.0 Benchmark Report.

13.6.7.3.2 Optimize Configuration of DM

This document introduces how to optimize the configuration of the data migration task to improve the performance of data migration.

Full data export

`mydumpers` is the configuration item related to full data export. This section describes how to configure performance-related options.

`rows`

Setting the `rows` option enables concurrently exporting data from a single table using multi-thread. The value of `rows` is the maximum number of rows contained in each exported chunk. After this option is enabled, DM selects a column as the split benchmark when the data of a MySQL single table is concurrently exported. This column can be one of the following columns: the primary key column, the unique index column, and the normal index column (ordered from highest priority to lowest). Make sure this column is of integer type (for example, `INT`, `MEDIUMINT`, `BIGINT`).

The value of `rows` can be set to 10000. You can change this value according to the total number of rows in the table and the performance of the database. In addition, you need to set `threads` to control the number of concurrent threads. By default, the value of `threads` is 4. You can adjust this value as needed.

`chunk-filesize`

During full backup, DM splits the data of each table into multiple chunks according to the value of the `chunk-filesize` option. Each chunk is saved in a file with a size of about `chunk-filesize`. In this way, data is split into multiple files and you can use the parallel processing of the DM Load unit to improve the import speed. The default value of this option is 64 (in MB). Normally, you do not need to set this option. If you set it, adjust the value of this option according to the size of the full data.

Note:

- You cannot update the value of `mydumpers` after the migration task is created. Be sure about the value of each option before creating the task. If you need to update the value, stop the task using dmctl, update the configuration file, and re-create the task.
- `mydumpers.threads` can be replaced with the `mydumper-thread` configuration item for simplicity.
- If `rows` is set, DM ignores the value of `chunk-filesize`.

Full data import

`loaders` is the configuration item related to full data import. This section describes how to configure performance-related options.

`pool-size`

The `pool-size` option determines the number of threads in the DM Load unit. The default value is 16. Normally, you do not need to set this option. If you set it, adjust the value of this option according to the size of the full data and the performance of the database.

Note:

- You cannot update the value of `loaders` after the migration task is created. Be sure about the value of each option before creating the task. If you need to update the value, stop the task using dmctl, update the configuration file, and re-create the task.
- `loaders.pool-size` can be replaced with the `loader-thread` configuration item for simplicity.

Incremental data replication

`syncers` is the configuration item related to incremental data replication. This section describes how to configure performance-related options.

`worker-count`

`worker-count` determines the number of threads for concurrent replication of DMLs in the DM Sync unit. The default value is 16. To speed up data replication, increase the value of this option appropriately.

`batch`

`batch` determines the number of DMLs included in each transaction when the data is replicated to the downstream database during the DM Sync unit. The default value is 100. Normally, you do not need to change the value of this option.

Note:

- You cannot update the value of `syncers` after the replication task is created. Be sure about the value of each option before creating the task. If you need to update the value, stop the task using dmctl, update the configuration file, and re-create the task.
- `syncers.worker-count` can be replaced with the `syncer-thread` configuration item for simplicity.
- You can change the values of `worker-count` and `batch` according to the actual scenario. For example, if there is a high network delay between DM and the downstream database, you can increase the value of `worker-count` and decrease the value of `batch` appropriately.

13.6.7.3.3 DM Cluster Performance Test

This document describes how to build a test scenario to do a performance test on the DM cluster, including the speed test and latency test regarding data migration.

Migration data flow

You can use a simple migration data flow, that is, MySQL -> DM -> TiDB, to test the data migration performance of the DM cluster.

Deploy test environment

- Deploy the TiDB test cluster using TiUP, with all default configurations.
- Deploy the MySQL service. Enable the `ROW` mode for binlog, and use default configurations for other configuration items.
- Deploy a DM cluster, with a DM-worker and a DM-master.

Performance test

Table schema

Use tables with the following schema for the performance test:

```
CREATE TABLE `sbtest` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) CHARSET utf8mb4 COLLATE utf8mb4_bin NOT NULL DEFAULT '',
  `pad` char(60) CHARSET utf8mb4 COLLATE utf8mb4_bin NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
```

Full import benchmark case

Generate test data

Use `sysbench` to create test tables upstream and generate test data for full import. Execute the following `sysbench` command to generate test data:

```
sysbench --test=oltp_insert --tables=4 --mysql-host=172.16.4.40 --mysql-
    ↪ port=3306 --mysql-user=root --mysql-db=dm_benchmark --db-driver=mysql
    ↪  --table-size=50000000 prepare
```

Create a data migration task

1. Create an upstream MySQL source and set `source-id` to `source-1`. For details, see Load the Data Source Configurations.
2. Create a migration task (in `full` mode). The following is a task configuration template:

“‘yaml

name: test-full
task-mode: full

Configure the migration task using the TiDB information of your actual test environment. target-database: host: “192.168.0.1” port: 4000 user: “root” password: “”

mysql-instances: - source-id: “source-1” block-allow-list: “instance” mydumper-config-name: “global” loader-thread: 16

Configure the name of the database where sysbench generates data. block-allow-list: instance: do-dbs: [“dm_benchmark”]

mydumpers: global: rows: 32000 threads: 32 “‘

For details about how to create a migration task, see Create a Data Migration Task.

Note:

- To enable concurrently exporting data from a single table using multi-thread, you can use the `rows` option in the `mydumpers` configuration item. This speeds up data export.
- To test the performance under different configurations, you can tune `loader-thread` in the `mysql-instances` configuration, as well as `rows` and `threads` in the `mydumpers` configuration item.

Get test results

Observe the DM-worker log. When you see `all data files have been finished`, it means that full data has been imported. In this case, you can see the time consumed to import data. The sample log is as follows:

```
[INFO] [loader.go:604] ["all data files have been finished"] [task=test] [
    ↪ unit=load] ["cost time"=52.439796ms]
```

According to the size of the test data and the time consumed to import data, you can calculate the migration speed of the full data.

Incremental replication benchmark case

Initialize tables

Use `sysbench` to create test tables in the upstream.

Create a data migration task

1. Create the source of the upstream MySQL. Set `source-id` to `source-1` (if the source has been created in the full import benchmark case, you do not need to create it again). For details, see Load the Data Source Configurations.

2. Create a DM migration task (in `all` mode). The following is an example of the task configuration file:

“‘yaml
name: test-all
task-mode: all

Configure the migration task using the TiDB information of your actual test environment. target-database: host: “192.168.0.1” port: 4000 user: “root” password: “”

mysql-instances: - source-id: “source-1” block-allow-list: “instance” syncer-config-name: “global”

Configure the name of the database where sysbench generates data. block-allow-list: instance: do-dbs: [“dm_benchmark”]

syncers: global: worker-count: 16 batch: 100 “‘

For details about how to create a data migration task, see Create a Data Migration Task.

> **Note:**
>
> To test the performance under different configurations, you can tune `worker-count` and `batch` in the `syncers` configuration item.

Generate incremental data

To continuously generate incremental data in the upstream, run the `sysbench` command:

```
sysbench --test=oltp_insert --tables=4 --num-threads=32 --mysql-host
    ↪ =172.17.4.40 --mysql-port=3306 --mysql-user=root --mysql-db=
    ↪ dm_benchmark --db-driver=mysql --report-interval=10 --time=1800 run
```

> **Note:**
>
> You can test the data migration performance under different scenarios by using different `sysbench` statements.

Get test results

To observe the migration status of DM, you can run the `query-status` command. To observe the monitoring metrics of DM, you can use Grafana. Here the monitoring metrics refer to `finished sqls jobs` (the number of jobs finished per unit time), and other related metrics. For more information, see Binlog Migration Monitoring Metrics.

13.6.7.3.4 Handle Performance Issues of TiDB Data Migration

This document introduces common performance issues that might exist in DM and how to deal with them.

Before diagnosing an issue, you can refer to the DM Benchmark Report.

When diagnosing and handling performance issues, make sure that:

- The DM monitoring component is correctly configured and installed.
- You can view monitoring metrics on the Grafana monitoring dashboard.
- The component you diagnose works well; otherwise, possible monitoring metrics exceptions might interfere with the diagnosis of performance issues.

In the case of a large latency in the data migration, to quickly figure out whether the bottleneck is inside the DM component or in the TiDB cluster, you can first check `DML queue remain length` in Write SQL Statements to Downstream.

relay log unit

To diagnose performance issues in the relay log unit, you can check the `binlog file gap between master and relay` monitoring metric. For more information about this metric, refer to monitoring metrics of the relay log. If this metric is greater than 1 for a long time, it usually indicates that there is a performance issue; if this metric is 0, it usually indicates that there is no performance issue.

If the value of `binlog file gap between master and relay` is 0, but you suspect that there is a performance issue, you can check `binlog pos`. If `master` in this metric is much larger than `relay`, a performance issue might exist. In this case, diagnose and handle this issue accordingly.

Read binlog data

`read binlog event duration` refers to the duration that the relay log reads binlog from the upstream database (MySQL/MariaDB). Ideally, this metric is close to the network latency between DM-worker and MySQL/MariaDB instances.

- For data migration in one data center, reading binlog data is not a performance bottleneck. If the value of `read binlog event duration` is too large, check the network connection between DM-worker and MySQL/MariaDB.

- For data migration in the geo-distributed environment, try to deploy DM-worker and MySQL/MariaDB in one data center, while deploying the TiDB cluster in the target data center.

The process of reading binlog data from the upstream database includes the following sub-processes:

- The upstream MySQL/MariaDB reads the binlog data locally and sends it through the network. When no exception occurs in the MySQL/MariaDB load, this sub-process usually does not become a bottleneck.
- The binlog data is transferred from the machine where MySQL/MariaDB is located to the machine where DM-worker is located via the network. Whether this sub-process becomes a bottleneck mainly depends on the network connection between DM-worker and the upstream MySQL/MariaDB.
- DM-worker reads binlog data from the network data stream and constructs it as a binlog event. When no exception occurs in the DM-worker load, this sub-process usually does not become a bottleneck.

> **Note:**
>
> If the value of `read binlog event duration` is large, another possible reason is that the upstream MySQL/MariaDB has a low load. This means that no binlog event needs to be sent to DM for a period of time, and the relay log unit stays in a wait state, thus this value includes additional waiting time.

binlog data decoding and verification

After reading the binlog event into the DM memory, DM's relay processing unit decodes and verifies data. This usually does not lead to performance bottleneck; therefore, there is no related performance metric on the monitoring dashboard by default. If you need to view this metric, you can manually add a monitoring item in Grafana. This monitoring item corresponds to `dm_relay_read_transform_duration`, a metric from Prometheus.

Write relay log files

When writing a binlog event to a relay log file, the relevant performance metric is `write relay log duration`. This value should be microseconds when `binlog event size` is not too large. If `write relay log duration` is too large, check the write performance of the disk. To avoid low write performance, use local SSDs for DM-worker.

Load unit

The main operations of the Load unit are to read the SQL file data from the local and write it to the downstream. The related performance metric is `transaction execution latency`. If this value is too large, check the downstream performance by checking the

monitoring of the downstream database. You can also check whether there is a large network latency between DM and the downstream database.

Binlog replication unit

To diagnose performance issues in the Binlog replication unit, you can check the `binlog file gap between master and syncer` monitoring metric. For more information about this metric, refer to monitoring metrics of the Binlog replication.

- If this metric is greater than 1 for a long time, it usually indicates that there is a performance issue.
- If this metric is 0, it usually indicates that there is no performance issue.

When `binlog file gap between master and syncer` is greater than 1 for a long time, check `binlog file gap between relay and syncer` to figure out which unit the latency mainly exists in. If this value is usually 0, the latency might exist in the relay log unit. Then you can refer to relay log unit to resolve this issue; otherwise, continue checking the Binlog replication unit.

Read binlog data

The Binlog replication unit decides whether to read the binlog event from the upstream MySQL/MariaDB or from the relay log file according to the configuration. The related performance metric is `read binlog event duration`, which generally ranges from a few microseconds to tens of microseconds.

- If DM's Binlog replication processing unit reads the binlog event from upstream MySQL/MariaDB, to locate and resolve the issue, refer to read binlog data in the "relay log unit" section.
- If DM's Binlog replication processing unit reads the binlog event from the relay log file, when `binlog event size` is not too large, the value of `read binlog event duration` ↪ should be microseconds. If `read binlog event duration` is too large, check the read performance of the disk. To avoid low write performance, use local SSDs for DM-worker.

binlog event conversion

The Binlog replication unit constructs DML, parses DDL, and performs table router conversion from binlog event data. The related metric is `transform binlog event duration`.

The duration is mainly affected by the write operations upstream. Take the `INSERT` ↪ `INTO` statement as an example, the time consumed to convert a single `VALUES` greatly differs from that to convert a lot of `VALUES`. The time consumed might range from tens of microseconds to hundreds of microseconds. However, usually this is not a bottleneck of the system.

Write SQL statements to downstream

When the Binlog replication unit writes the converted SQL statements to the downstream, the related performance metrics are `DML queue remain length` and `transaction execution latency`.

After constructing SQL statements from binlog event, DM uses `worker-count` queues to concurrently write these statements to the downstream. However, to avoid too many monitoring entries, DM performs the modulo `8` operation on the IDs of concurrent queues. This means that all concurrent queues correspond to one item from `q_0` to `q_7`.

`DML queue remain length` indicates in the concurrent processing queue, the number of DML statements that have not been consumed and have not started to be written downstream. Ideally, the curves corresponding to each `q_*` are almost the same. If not, it indicates that the concurrent load is extremely unbalanced.

If the load is not balanced, confirm whether tables need to be migrated have primary keys or unique keys. If these keys do not exist, add the primary keys or the unique keys; if these keys do exist while the load is not balanced, upgrade DM to v1.0.5 or later versions.

- When there is no noticeable latency in the entire data migration link, the corresponding curve of `DML queue remain length` is almost always 0, and the maximum does not exceed the value of `batch` in the task configuration file.
- If you find a noticeable latency in the data migration link, and the curve of `DML queue remain length` corresponding to each `q_*` is almost the same and is almost always 0, it means that DM fails to read, convert, or concurrently write the data from the upstream in time (the bottleneck might be in the relay log unit). For troubleshooting, refer to the previous sections of this document.

If the corresponding curve of `DML queue remain length` is not 0 (usually the maximum is not more than 1024), it indicates that there is a bottleneck when writing SQL statements to the downstream. You can use `transaction execution latency` to view the time consumed to execute a single transaction to the downstream.

`transaction execution latency` is usually tens of milliseconds. If this value is too large, check the downstream performance based on the monitoring of the downstream database. You can also check whether there is a large network latency between DM and the downstream database.

To view the time consumed to write a single statement such as `BEGIN`, `INSERT`, `UPDATE`, `DELETE`, or `COMMIT` to the downstream, you can also check `statement execution latency`.

13.6.7.4 Manage Data Sources

13.6.7.4.1 Switch DM-worker Connection between Upstream MySQL Instances

When the upstream MySQL instance that DM-worker connects to needs downtime maintenance or when the instance crashes unexpectedly, you need to switch the DM-worker connection to another MySQL instance within the same migration group.

Note:

- You can switch the DM-worker connection to only an instance within the same primary-secondary migration cluster.
- The MySQL instance to be newly connected to must have the binlog required by DM-worker.
- DM-worker must operate in the GTID sets mode, which means you must specify `enable-gtid: true` in the corresponding source configuration file.
- The connection switch only supports the following two scenarios. Strictly follow the procedures for each scenario. Otherwise, you might have to re-deploy the DM cluster according to the newly connected MySQL instance and perform the data migration task all over again.

For more details on GTID set, refer to MySQL documentation.

Switch DM-worker connection via virtual IP

When DM-worker connects the upstream MySQL instance via a virtual IP (VIP), switching the VIP connection to another MySQL instance means switching the MySQL instance connected to DM-worker, without the upstream connection address changed.

Note:

Make necessary changes to DM in this scenario. Otherwise, when you switch the VIP connection to another MySQL instance, DM might connect to the new and old MySQL instances at the same time in different connections. In this situation, the binlog replicated to DM is not consistent with other upstream status that DM receives, causing unpredictable anomalies and even data damage.

To switch one upstream MySQL instance (when DM-worker connects to it via a VIP) to another, perform the following steps:

1. Use the `query-status` command to get the GTID sets (`syncerBinlogGtid`) corresponding to the binlog that the current processing unit of binlog replication has replicated to the downstream. Mark the sets as `gtid-S`.
2. Use the `SELECT @@GLOBAL.gtid_purged;` command on the new MySQL instance to get the GTID sets corresponding to the purged binlogs. Mark the sets as `gtid-P`.

3. Use the `SELECT @@GLOBAL.gtid_executed;` command on the new MySQL instance to get the GTID sets corresponding to all successfully executed transactions. Mark the sets as `gtid-E`.
4. Make sure that the following conditions are met. Otherwise, you cannot switch the DM-work connection to the new MySQL instance:

    - `gtid-S` contains `gtid-P`. `gtid-P` can be empty.
    - `gtid-E` contains `gtid-S`.

5. Use `pause-task` to pause all running tasks of data migration.
6. Change the VIP for it to direct at the new MySQL instance.
7. Use `resume-task` to resume the previous migration task.

Change the address of the upstream MySQL instance that DM-worker connects to

To make DM-worker connect to a new MySQL instance in the upstream by modifying the DM-worker configuration, perform the following steps:

1. Use the `query-status` command to get the GTID sets (`syncerBinlogGtid`) corresponding to the binlog that the current processing unit of binlog replication has replicated to the downstream. Mark this sets as `gtid-S`.
2. Use the `SELECT @@GLOBAL.gtid_purged;` command on the new MySQL instance to get the GTID sets corresponding to the purged binlogs. Mark this sets as `gtid-P`.
3. Use the `SELECT @@GLOBAL.gtid_executed;` command on the new MySQL instance to get the GTID sets corresponding to all successfully executed transactions. Mark this sets as `gtid-E`.
4. Make sure that the following conditions are met. Otherwise, you cannot switch the DM-work connection to the new MySQL instance:

    - `gtid-S` contains `gtid-P`. `gtid-P` can be empty.
    - `gtid-E` contains `gtid-S`.

5. Use `stop-task` to stop all running tasks of data migration.
6. Use the `operator-source stop` command to remove the source configuration corresponding to the address of the old MySQL instance from the DM cluster.
7. Update the address of the MySQL instance in the source configuration file and use the `operate-source create` command to reload the new source configuration in the DM cluster.
8. Use `start-task` to restart the migration task.

13.6.7.5 Manage Tasks

13.6.7.5.1 Handle Failed DDL Statements in TiDB Data Migration

This document introduces how to handle failed DDL statements when you're using the TiDB Data Migration (DM) tool to migrate data.

Currently, TiDB is not completely compatible with all MySQL syntax (see the DDL statements supported by TiDB). Therefore, when DM is migrating data from MySQL to TiDB and TiDB does not support the corresponding DDL statement, an error might occur and break the migration process. In this case, you can use the `binlog` command of DM to resume the migration.

Restrictions

Do not use this command in the following situations:

- It is unacceptable in the actual production environment that the failed DDL statement is skipped in the downstream TiDB.
- The failed DDL statement cannot be replaced with other DDL statements.
- Other DDL statements must not be injected into the downstream TiDB.

For example, `DROP PRIMARY KEY`. In this scenario, you can only create a new table in the downstream with the new table schema (after executing the DDL statement), and re-import all the data into this new table.

Supported scenarios

During the migration, the DDL statement unsupported by TiDB is executed in the upstream and migrated to the downstream, and as a result, the migration task gets interrupted.

- If it is acceptable that this DDL statement is skipped in the downstream TiDB, then you can use `binlog skip <task-name>` to skip migrating this DDL statement and resume the migration.
- If it is acceptable that this DDL statement is replaced with other DDL statements, then you can use `binlog replace <task-name>` to replace this DDL statement and resume the migration.
- If it is acceptable that other DDL statements are injected to the downstream TiDB, then you can use `binlog inject <task-name>` to inject other DDL statements and resume the migration.

Commands

When you use dmctl to manually handle the failed DDL statements, the commonly used commands include `query-status` and `binlog`.

query-status

The `query-status` command is used to query the current status of items such as the subtask and the relay unit in each MySQL instance. For details, see query status.

binlog

The `binlog` command is used to manage and show binlog operations. This command is only supported in DM v6.0 and later versions. For earlier versions, use the `handle-error` command.

The usage of `binlog` is as follows:

```
binlog -h
```

```
manage or show binlog operations

Usage:
  dmctl binlog [command]

Available Commands:
  inject    inject the current error event or a specific binlog position (
    ↪ binlog-pos) with some ddls
  list      list error handle command at binlog position (binlog-pos) or
    ↪ after binlog position (binlog-pos)
  replace   replace the current error event or a specific binlog position (
    ↪ binlog-pos) with some ddls
  revert    revert the current binlog operation or a specific binlog
    ↪ position (binlog-pos) operation
  skip      skip the current error event or a specific binlog position (
    ↪ binlog-pos) event

Flags:
  -b, --binlog-pos string position used to match binlog event if matched the
    ↪  binlog operation will be applied. The format like "mysql-bin
    ↪ |000001.000003:3270"
  -h, --help            help for binlog

Global Flags:
  -s, --source strings MySQL Source ID.

Use "dmctl binlog [command] --help" for more information about a command.
```

`binlog` supports the following sub-commands:

- `inject`: injects DDL statements into the current error event or a specific binlog position. To specify the binlog position, refer to `-b, --binlog-pos`.
- `list`: lists all valid `inject`, `skip`, and `replace` operations at the current binlog position or after the current binlog position. To specify the binlog position, refer to `-b, --binlog-pos`.
- `replace`: replaces the DDL statement at a specific binlog position with another DDL statement. To specify the binlog position, refer to `-b, --binlog-pos`.
- `revert`: reverts the `inject`, `skip` or `replace` operation at a specified binlog operation, only if the previous operation does not take effect. To specify the binlog position, refer to `-b, --binlog-pos`.
- `skip`: skips the DDL statement at a specific binlog position. To specify the binlog position, refer to `-b, --binlog-pos`.

`binlog` supports the following flags:

- `-b, --binlog-pos`:
  - Type: string.
  - Specifies a binlog position. When the position of the binlog event matches `binlog -pos`, the operation is executed. If it is not specified, DM automatically sets `binlog-pos` to the currently failed DDL statement.
  - Format: `binlog-filename:binlog-pos`, for example, `mysql-bin|000001.000003:3270`.
  - After the migration returns an error, the binlog position can be obtained from `position` in `startLocation` returned by `query-status`. Before the migration returns an error, the binlog position can be obtained by using `SHOW BINLOG EVENTS` in the upstream MySQL instance.
- `-s, --source`:
  - Type: string.
  - Specifies the MySQL instance in which the preset operation is to be executed.

Usage examples

Skip DDL if the migration gets interrupted

If you need to skip the DDL statement when the migration gets interrupted, run the `binlog skip` command:

```
binlog skip -h
```

```
skip the current error event or a specific binlog position (binlog-pos)
   ↪ event

Usage:
  dmctl binlog skip <task-name> [flags]

Flags:
  -h, --help help for skip

Global Flags:
  -b, --binlog-pos string position used to match binlog event if matched the
     ↪  binlog operation will be applied. The format like "mysql-bin
     ↪ |000001.000003:3270"
  -s, --source strings   MySQL Source ID.
```

Non-shard-merge scenario

Assume that you need to migrate the upstream table `db1.tbl1` to the downstream TiDB. The initial table schema is:

```
SHOW CREATE TABLE db1.tbl1;
```

```
+-------+-----------------------------------------------+
| Table | Create Table                                  |
+-------+-----------------------------------------------+
| tbl1 | CREATE TABLE `tbl1` (
  `c1` int(11) NOT NULL,
  `c2` decimal(11,3) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-----------------------------------------------+
```

Now, the following DDL statement is executed in the upstream to alter the table schema (namely, alter DECIMAL(11, 3) of c2 into DECIMAL(10, 3)):

```
ALTER TABLE db1.tbl1 CHANGE c2 c2 DECIMAL (10, 3);
```

Because this DDL statement is not supported by TiDB, the migration task of DM gets interrupted. Execute the `query-status <task-name>` command, and you can see the following error:

```
ERROR 8200 (HY000): Unsupported modify column: can't change decimal column
    ↪ precision
```

Assume that it is acceptable in the actual production environment that this DDL statement is not executed in the downstream TiDB (namely, the original table schema is retained). Then you can use `binlog skip <task-name>` to skip this DDL statement to resume the migration. The procedures are as follows:

1. Execute `binlog skip <task-name>` to skip the currently failed DDL statement:

```
» binlog skip test
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

2. Execute `query-status <task-name>` to view the task status:

```
» query-status test
```

See the execution result.


```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "sourceStatus": {
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
                    "sync": {
                        "totalEvents": "4",
                        "totalTps": "0",
                        "recentTps": "0",
                        "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-10",
                        "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-4",
                        "blockingDDLs": [
                        ],
                        "unresolvedGroups": [
                        ],
                        "synced": true,
                        "binlogType": "remote"
                    }
                }
```

```
            ]
        }
    ]
}
```

You can see that the task runs normally and the wrong DDL is skipped.

Shard merge scenario

Assume that you need to merge and migrate the following four tables in the upstream to one same table `shard_db`.`shard_table` in the downstream. The task mode is "pessimistic".

- MySQL instance 1 contains the shard_db_1 schema, which includes the shard_table_1 ↪ and shard_table_2 tables.
- MySQL instance 2 contains the shard_db_2 schema, which includes the shard_table_1 ↪ and shard_table_2 tables.

The initial table schema is:

```
SHOW CREATE TABLE shard_db.shard_table;
```

```
+--
    ↪ -----+----------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+----------------------------------------------------------------------
    ↪
| tb    | CREATE TABLE `shard_table` (
  `id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+--
    ↪ -----+----------------------------------------------------------------------
    ↪
```

Now, execute the following DDL statement to all upstream sharded tables to alter their character set:

```
ALTER TABLE `shard_db_*`.`shard_table_*` CHARACTER SET LATIN1 COLLATE
    ↪ LATIN1_DANISH_CI;
```

Because this DDL statement is not supported by TiDB, the migration task of DM gets interrupted. Execute the `query-status` command, and you can see the following errors reported by the `shard_db_1.shard_table_1` table in MySQL instance 1 and the `shard_db_2` ↪ `.shard_table_1` table in MySQL instance 2:

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_1`.`shard_table_1`
       ↪ CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to utf8"
}
```

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_2`.`shard_table_1`
       ↪ CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to utf8"
}
```

Assume that it is acceptable in the actual production environment that this DDL statement is not executed in the downstream TiDB (namely, the original table schema is retained). Then you can use `binlog skip <task-name>` to skip this DDL statement to resume the migration. The procedures are as follows:

1. Execute `binlog skip <task-name>` to skip the currently failed DDL statements in MySQL instance 1 and 2:

    ```
    » binlog skip test
    ```

    ```
    {
        "result": true,
        "msg": "",
        "sources": [
            {
                "result": true,
                "msg": "",
                "source": "mysql-replica-01",
                "worker": "worker1"
            },
            {
                "result": true,
                "msg": "",
                "source": "mysql-replica-02",
                "worker": "worker2"
            }
        ]
    }
    ```

2. Execute the `query-status` command, and you can see the errors reported by the `shard_db_1.shard_table_2` table in MySQL instance 1 and the `shard_db_2` ↪ `.shard_table_2` table in MySQL instance 2:

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_1`.`
        ↪ shard_table_2` CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to
        ↪ utf8"
}
```

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_2`.`
        ↪ shard_table_2` CHARACTER SET UTF8 COLLATE UTF8_UNICODE_CI",
    "RawCause": "[ddl:8200]Unsupported modify charset from latin1 to
        ↪ utf8"
}
```

3. Execute `binlog skip <task-name>` again to skip the currently failed DDL statements in MySQL instance 1 and 2:

```
» handle-error test skip
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```

4. Use `query-status <task-name>` to view the task status:

```
» query-status test
```

See the execution result.


```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "sourceStatus": {
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
                    "sync": {
                        "totalEvents": "4",
                        "totalTps": "0",
                        "recentTps": "0",
                        "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-10",
                        "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-4",
                        "blockingDDLs": [
                        ],
                        "unresolvedGroups": [
                        ],
                        "synced": true,
                        "binlogType": "remote"
                    }
                }
            ]
        },
        {
```

```
            "result": true,
            "msg": "",
            "sourceStatus": {
                "source": "mysql-replica-02",
                "worker": "worker2",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
                    "sync": {
                        "totalEvents": "4",
                        "totalTps": "0",
                        "recentTps": "0",
                        "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-10",
                        "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-4",
                        "blockingDDLs": [
                        ],
                        "unresolvedGroups": [
                        ],
                        "synced": true,
                        "binlogType": "remote"
                    }
                }
            ]
        }
    ]
}
```


You can see that the task runs normally with no error and all four wrong DDL statements are skipped.

Replace DDL if the migration gets interrupted

If you need to replace the DDL statement when the migration gets interrupted, run the `binlog replace` command:

```
binlog replace -h
```

```
replace the current error event or a specific binlog position (binlog-pos)
    ↪ with some ddls

Usage:
  dmctl binlog replace <task-name> <replace-sql1> <replace-sql2>... [flags]

Flags:
  -h, --help help for replace

Global Flags:
  -b, --binlog-pos string position used to match binlog event if matched the
     ↪  binlog operation will be applied. The format like "mysql-bin
     ↪ |000001.000003:3270"
  -s, --source strings   MySQL Source ID.
```

Non-shard-merge scenario

Assume that you need to migrate the upstream table `db1.tbl1` to the downstream TiDB. The initial table schema is:

```
SHOW CREATE TABLE db1.tbl1;
```

```
+--
    ↪ -----+----------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+----------------------------------------------------------------------
    ↪
| tb    | CREATE TABLE `tbl1` (
  `id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+--
    ↪ -----+----------------------------------------------------------------------
    ↪
```

Now, perform the following DDL operation in the upstream to add a new column with the UNIQUE constraint:

```
ALTER TABLE `db1`.`tbl1` ADD COLUMN new_col INT UNIQUE;
```

Because this DDL statement is not supported by TiDB, the migration task gets interrupted. Execute the `query-status` command, and you can see the following error:

```
{
    "Message": "cannot track DDL: ALTER TABLE `db1`.`tbl1` ADD COLUMN `
        ↪ new_col` INT UNIQUE KEY",
    "RawCause": "[ddl:8200]unsupported add column 'new_col' constraint
        ↪ UNIQUE KEY when altering 'db1.tbl1'",
}
```

You can replace this DDL statement with two equivalent DDL statements. The steps are as follows:

1. Replace the wrong DDL statement by the following command:

```
» binlog replace test "ALTER TABLE `db1`.`tbl1` ADD COLUMN `new_col`
    ↪ INT;ALTER TABLE `db1`.`tbl1` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

2. Use `query-status <task-name>` to view the task status:

```
» query-status test
```

See the execution result.

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
```

```
            "sourceStatus": {
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
                    "sync": {
                        "totalEvents": "4",
                        "totalTps": "0",
                        "recentTps": "0",
                        "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-10",
                        "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-4",
                        "blockingDDLs": [
                        ],
                        "unresolvedGroups": [
                        ],
                        "synced": true,
                        "binlogType": "remote"
                    }
                }
            ]
        }
    ]
}
```


You can see that the task runs normally and the wrong DDL statement is replaced by new DDL statements that execute successfully.

Shard merge scenario

Assume that you need to merge and migrate the following four tables in the upstream to one same table `` `shard_db`.`shard_table` `` in the downstream. The task mode is "pessimistic".

- In the MySQL instance 1, there is a schema `shard_db_1`, which has two tables `shard_table_1` and `shard_table_2`.
- In the MySQL instance 2, there is a schema `shard_db_2`, which has two tables `shard_table_1` and `shard_table_2`.

The initial table schema is:

```
SHOW CREATE TABLE shard_db.shard_table;
```

```
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| tb    | CREATE TABLE `shard_table` (
  `id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
```

Now, perform the following DDL operation to all upstream sharded tables to add a new column with the UNIQUE constraint:

```
ALTER TABLE `shard_db_*`.`shard_table_*` ADD COLUMN new_col INT UNIQUE;
```

Because this DDL statement is not supported by TiDB, the migration task gets interrupted. Execute the `query-status` command, and you can see the following errors reported by the `shard_db_1.shard_table_1` table in MySQL instance 1 and the `shard_db_2` ↪ `.shard_table_1` table in MySQL instance 2:

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_1`.`shard_table_1`
        ↪ ADD COLUMN `new_col` INT UNIQUE KEY",
    "RawCause": "[ddl:8200]unsupported add column 'new_col' constraint
        ↪ UNIQUE KEY when altering 'shard_db_1.shard_table_1'",
}
```

```
{
    "Message": "cannot track DDL: ALTER TABLE `shard_db_2`.`shard_table_1`
        ↪ ADD COLUMN `new_col` INT UNIQUE KEY",
```

```
    "RawCause": "[ddl:8200]unsupported add column 'new_col' constraint
       ↪ UNIQUE KEY when altering 'shard_db_2.shard_table_1'",
}
```

You can replace this DDL statement with two equivalent DDL statements. The steps are as follows:

1. Replace the wrong DDL statements respectively in MySQL instance 1 and MySQL instance 2 by the following commands:

```
» binlog replace test -s mysql-replica-01 "ALTER TABLE `shard_db_1`.`
   ↪ shard_table_1` ADD COLUMN `new_col` INT;ALTER TABLE `shard_db_1
   ↪ `.`shard_table_1` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

```
» binlog replace test -s mysql-replica-02 "ALTER TABLE `shard_db_2`.`
   ↪ shard_table_1` ADD COLUMN `new_col` INT;ALTER TABLE `shard_db_2
   ↪ `.`shard_table_1` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```

2. Use `query-status <task-name>` to view the task status, and you can see the following errors reported by the `shard_db_1.shard_table_2` table in MySQL instance 1 and the `shard_db_2.shard_table_2` table in MySQL instance 2:

```
{
    "Message": "detect inconsistent DDL sequence from source ... ddls:
       ↪ [ALTER TABLE `shard_db`.`tb` ADD COLUMN `new_col` INT UNIQUE
       ↪ KEY] source: `shard_db_1`.`shard_table_2`], right DDL
       ↪ sequence should be ..."
}
```

```
{
    "Message": "detect inconsistent DDL sequence from source ... ddls:
       ↪ [ALTER TABLE `shard_db`.`tb` ADD COLUMN `new_col` INT UNIQUE
       ↪ KEY] source: `shard_db_2`.`shard_table_2`], right DDL
       ↪ sequence should be ..."
}
```

3. Execute `handle-error <task-name> replace` again to replace the wrong DDL statements in MySQL instance 1 and 2:

```
» binlog replace test -s mysql-replica-01 "ALTER TABLE `shard_db_1`.`
   ↪ shard_table_2` ADD COLUMN `new_col` INT;ALTER TABLE `shard_db_1
   ↪ `.`shard_table_2` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

```
» binlog replace test -s mysql-replica-02 "ALTER TABLE `shard_db_2`.`
   ↪ shard_table_2` ADD COLUMN `new_col` INT;ALTER TABLE `shard_db_2
   ↪ `.`shard_table_2` ADD UNIQUE(`new_col`)";
```

```
{
    "result": true,
    "msg": "",
```

```
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```

4. Use `query-status <task-name>` to view the task status:

```
» query-status test
```

See the execution result.

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "sourceStatus": {
                "source": "mysql-replica-01",
                "worker": "worker1",
                "result": null,
                "relayStatus": null
            },
            "subTaskStatus": [
                {
                    "name": "test",
                    "stage": "Running",
                    "unit": "Sync",
                    "result": null,
                    "unresolvedDDLLockID": "",
                    "sync": {
                        "totalEvents": "4",
                        "totalTps": "0",
                        "recentTps": "0",
                        "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
                        "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                            ↪ de45f57:1-10",
                        "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                            ↪ 2388)",
```

```
                "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                    ↪ de45f57:1-4",
                "blockingDDLs": [
                ],
                "unresolvedGroups": [
                ],
                "unresolvedGroups": [
                ],
                "synced": true,
                "binlogType": "remote"
            }
        }
    ]
},
{
    "result": true,
    "msg": "",
    "sourceStatus": {
        "source": "mysql-replica-02",
        "worker": "worker2",
        "result": null,
        "relayStatus": null
    },
    "subTaskStatus": [
        {
            "name": "test",
            "stage": "Running",
            "unit": "Sync",
            "result": null,
            "unresolvedDDLLockID": "",
            "sync": {
                "totalEvents": "4",
                "totalTps": "0",
                "recentTps": "0",
                "masterBinlog": "(DESKTOP-T561TSO-bin.000001,
                    ↪ 2388)",
                "masterBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                    ↪ de45f57:1-10",
                "syncerBinlog": "(DESKTOP-T561TSO-bin.000001,
                    ↪ 2388)",
                "syncerBinlogGtid": "143bdef3-dd4a-11ea-8b00-00155
                    ↪ de45f57:1-4",
                "blockingDDLs": [
                ],
                "unresolvedGroups": [
```

```
                    ],
                    "unresolvedGroups": [
                    ],
                    "synced": try,
                    "binlogType": "remote"
                }
            }
        ]
    }
]
}
```

You can see that the task runs normally with no error and all four wrong DDL statements are replaced.

Other commands

For the usage of other commands of `binlog`, refer to the `binlog skip` and `binlog replace` examples above.

13.6.7.5.2 Manage Table Schemas of Tables to Be Migrated Using TiDB Data Migration

This document describes how to manage the schema of the table in DM during migration using dmctl.

When DM performs incremental replication, it first reads the upstream binlog, then creates SQL statements and executes them in the downstream. However, the upstream binlog does not contain the complete table schema. To generate the SQL statements, DM maintains internally the schema information of the table to be migrated. This is called the internal table schema.

To deal with some special occasions, or to handle migration interruptions caused by mismatch of the table schemas, DM provides the `binlog-schema` command to obtain, modify, and delete the internal table schema.

Implementation principles

The internal table schema comes from the following sources:

- For full data migration (`task-mode=all`), the migration task goes through three stages: dump/load/sync, which means full export, full import, and incremental replication. In the dump stage, DM exports the table schema information along with the data and automatically creates the corresponding table in the downstream. In the sync stage, this table schema is used as the starting table scheme for incremental replication.
- In the sync stage, when DM handles DDL statements such as `ALTER TABLE`, it updates the internal table schema at the same time.

- If the task is an incremental migration (`task-mode=incremental`), in which the downstream has completed creating the table to be migrated, DM obtains the table schema information from the downstream database. This behavior varies with DM versions.

For incremental replication, schema maintenance is complicated. During the whole data replication, the following four table schemas are involved. These schemas might be the consistent or inconsistent with one another:

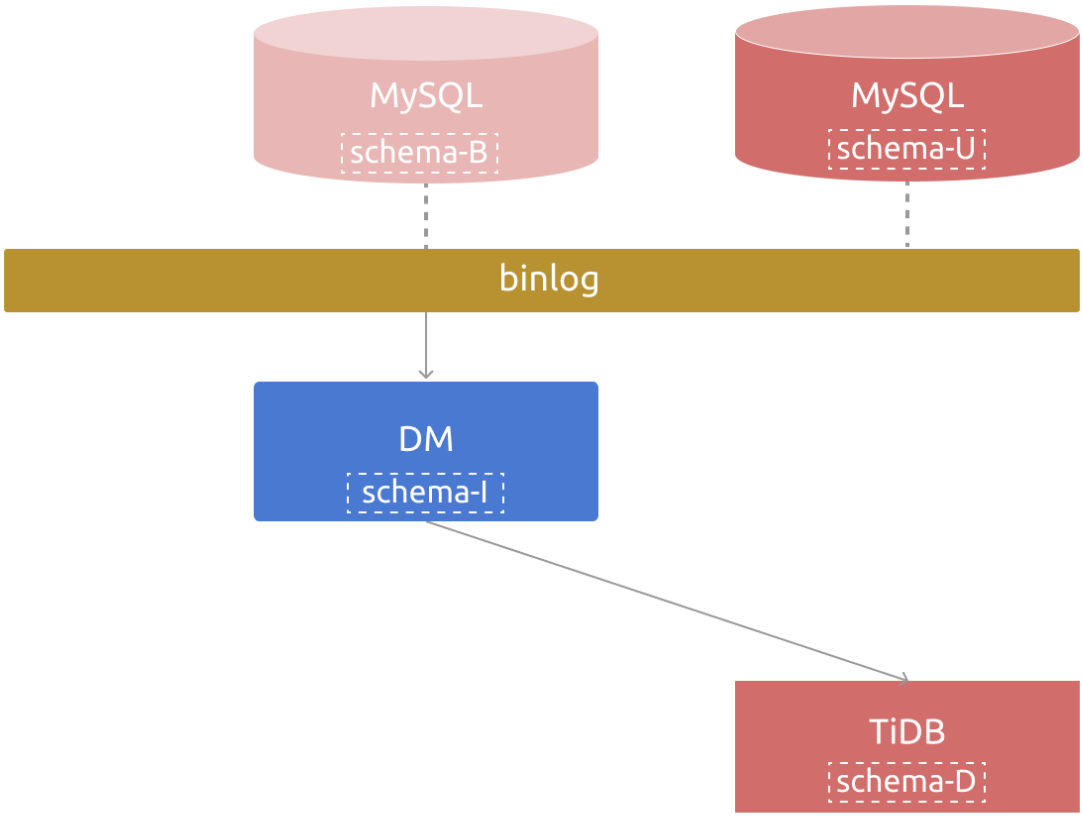


Figure 256: schema

- The upstream table schema at the current time, identified as `schema-U`.
- The table schema of the binlog event currently being consumed by DM, identified as `schema-B`. This schema corresponds to the upstream table schema at a historical time.
- The table schema currently maintained in DM (the schema tracker component), identified as `schema-I`.
- The table schema in the downstream TiDB cluster, identified as `schema-D`.

In most cases, the preceding four table schemas are consistent.

When the upstream database performs a DDL operation to change the table schema, `schema-U` is changed. By applying the DDL operation to the internal schema tracker component and the downstream TiDB cluster, DM updates `schema-I` and `schema-D` in an orderly manner to keep them consistent with `schema-U`. Therefore, DM can then normally consume the binlog event corresponding to the `schema-B` table schema. That is, after the DDL operation is successfully migrated, `schema-U`, `schema-B`, `schema-I`, and `schema-D` are still consistent.

Note the following situations that might cause inconsistency:

- During the migration with optimistic mode sharding DDL support enabled, the `schema-D` of the downstream table might be inconsistent with the `schema-B` and `schema-I` of some upstream sharded tables. In such cases, DM still keeps `schema-I` and `schema-B` consistent to ensure that the binlog event corresponding to DML can be parsed normally.
- When the downstream table has more columns than the upstream table, `schema-D` might be inconsistent with `schema-B` and `schema-I`. In the full data migration (`task-mode=all`), DM automatically handles inconsistency. In the incremental migration (`task-mode=incremental`), because the task is on a first start and there is no internal schema information yet, DM automatically reads the downstream schema (`schema-D`) and updates `schema-I` (this behavior varies with DM versions). After that, if DM uses `schema-I` to parse `schema-B`'s binlog, it will report `Column count doesn't match value count` error. For details, refer to Migrate Data to a Downstream TiDB Table with More Columns.

You can run the `binlog-schema` command to obtain, modify, or delete the `schema-I` table schema maintained in DM.

> **Note:**
>
> The `binlog-schema` command is supported only in DM v6.0 or later versions. For earlier versions, you must use the `operate-schema` command.

Command

```
help binlog-schema
```

```
manage or show table schema in schema tracker

Usage:
  dmctl binlog-schema [command]
```

```
Available Commands:
  delete     delete table schema structure
  list       show table schema structure
  update     update tables schema structure

Flags:
  -h, --help help for binlog-schema

Global Flags:
  -s, --source strings MySQL Source ID.

Use "dmctl binlog-schema [command] --help" for more information about a
    ↪ command.
```

Note:

- Because a table schema might change during data migration, to obtain a predictable table schema, currently the `binlog-schema` command can be used only when the data migration task is in the `Paused` state.
- To avoid data loss due to mishandling, it is **strongly recommended** to get and backup the table schema firstly before you modify the schema.

Parameters

- `delete`: Deletes the table schema.
- `list`: Lists the table schema.
- `update`: Updates the table schema.
- `-s` or `--source`:
  - Required.
  - Specifies the MySQL source that the operation is applied to.

Usage example

Get the table schema

To get the table schema, run the `binlog-schema list` command:

```
help binlog-schema list
```

```
show table schema structure
```

```
Usage:
  dmctl binlog-schema list <task-name> <database> <table> [flags]

Flags:
  -h, --help help for list

Global Flags:
  -s, --source strings MySQL Source ID.
```

If you want to get the table schema of the `db_single`.`t1` table corresponding to the mysql-replica-01 MySQL source in the db_single task, run the following command:

```
binlog-schema list -s mysql-replica-01 task_single db_single t1
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "CREATE TABLE `t1` ( `c1` int(11) NOT NULL, `c2` int(11)
                ↪ DEFAULT NULL, PRIMARY KEY (`c1`)) ENGINE=InnoDB DEFAULT
                ↪ CHARSET=latin1 COLLATE=latin1_bin",
            "source": "mysql-replica-01",
            "worker": "127.0.0.1:8262"
        }
    ]
}
```

Update the table schema

To update the table schema, run the binlog-schema update command:

```
help binlog-schema update
```

```
update tables schema structure

Usage:
  dmctl binlog-schema update <task-name> <database> <table> [schema-file] [
     ↪ flags]

Flags:
      --flush       flush the table info and checkpoint immediately (default
         ↪  true)
      --from-source use the schema from upstream database as the schema of
         ↪ the specified tables
```

```
      --from-target use the schema from downstream database as the schema of
          ↪ the specified tables
  -h, --help        help for update
      --sync        sync the table info to master to resolve shard ddl lock,
          ↪ only for optimistic mode now (default true)

Global Flags:
  -s, --source strings MySQL Source ID.
```

If you want to set the table schema of the `db_single`.`t1` table corresponding to the mysql-replica-01 MySQL source in the db_single task as follows:

```
CREATE TABLE `t1` (
    `c1` int(11) NOT NULL,
    `c2` bigint(11) DEFAULT NULL,
    PRIMARY KEY (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin
```

Save the CREATE TABLE statement above as a file (for example, db_single.t1-schema.
↪ sql), and run the following command:

```
operate-schema set -s mysql-replica-01 task_single -d db_single -t t1
    ↪ db_single.t1-schema.sql
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "127.0.0.1:8262"
        }
    ]
}
```

Delete the table schema

To delete the table schema, run the binlog-schema delete command:

```
help binlog-schema delete
```

```
delete table schema structure

Usage:
  dmctl binlog-schema delete <task-name> <database> <table> [flags]
```

```
Flags:
  -h, --help  help for delete

Global Flags:
  -s, --source strings MySQL Source ID.
```

> **Note:**
>
> After the table schema maintained in DM is deleted, if a DDL/DML statement related to this table needs to be migrated to the downstream, DM will try to get the table schema from the following three sources in an orderly manner:
>
> - The `table_info` field in the checkpoint table
> - The meta information in the optimistic sharding DDL
> - The corresponding table in the downstream TiDB

If you want to delete the table schema of the `` `db_single`.`t1` `` table corresponding to the `mysql-replica-01` MySQL source in the `db_single` task, run the following command:

```
binlog-schema delete -s mysql-replica-01 task_single db_single t1
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "127.0.0.1:8262"
        }
    ]
}
```

13.6.7.6 Export and Import Data Sources and Task Configuration of Clusters

`config` command is used to export and import data sources and task configuration of clusters.

> **Note:**
>
> For clusters earlier than v2.0.5, you can use dmctl v2.0.5 or later to export and import the data source and task configuration files.

```
» help config
Commands to import/export config
Usage:
  dmctl config [command]
Available Commands:
  export     Export the configurations of sources and tasks.
  import     Import the configurations of sources and tasks.
Flags:
  -h, --help help for config
Global Flags:
  -s, --source strings MySQL Source ID.
Use "dmctl config [command] --help" for more information about a command.
```

13.6.7.6.1 Export the data source and task configuration of clusters

You can use `export` command to export the data source and task configuration of clusters to specified files.

```
config export [--dir directory]
```

Parameter explanation

- `dir`:
  - optional
  - specifies the file path for exporting
  - the default value is `./configs`

Returned results

```
config export -d /tmp/configs
```

```
export configs to directory `/tmp/configs` succeed
```

13.6.7.6.2 import the data source and task configuration of clusters

You can use `import` command to import the data source and task configuration of clusters from specified files.

```
config import [--dir directory]
```

> **Note:**
>
> For clusters later than v2.0.2, currently, it is not supported to automatically import the configuration related to relay worker. You can use `start-relay` command to manually start relay log.

Parameter explanation

- `dir`:
  - optional
  - specifies the file path for importing
  - the default value is `./configs`

Returned results

```
config import -d /tmp/configs
```

```
start creating sources
start creating tasks
import configs from directory `/tmp/configs` succeed
```

13.6.7.7 Handle Alerts in TiDB Data Migration

This document introduces how to deal with the alert information in DM.

13.6.7.7.1 Alerts related to high availability

`DM_master_all_down`

- Description:

  If all DM-master nodes are offline, this alert is triggered.

- Solution:

  You can take the following steps to handle the alert:

1. Check the environment of the cluster.
2. Check the logs of all DM-master nodes for troubleshooting.

`DM_worker_offline`

- Description:

  If a DM-worker node is offline for more than one hour, this alert is triggered. In a high-availability architecture, this alert might not directly interrupt the task but increases the risk of interruption.

- Solution:

  You can take the following steps to handle the alert:

  1. View the working status of the corresponding DM-worker node.
  2. Check whether the node is connected.
  3. Troubleshoot errors through logs.

`DM_DDL_error`

- Description:

  This error occurs when DM is processing the sharding DDL operations.

- Solution:

  Refer to Troubleshoot DM.

`DM_pending_DDL`

- Description:

  If a sharding DDL operation is pending for more than one hour, this alert is triggered.

- Solution:

  In some scenarios, the pending sharding DDL operation might be what users expect. Otherwise, refer to Handle Sharding DDL Locks Manually in DM for solution.

13.6.7.7.2 Alert rules related to task status

`DM_task_state`

- Description:

  When a sub-task of DM-worker is in the `Paused` state for over 20 minutes, an alert is triggered.

- Solution:

  Refer to Troubleshoot DM.

13.6.7.7.3 Alert rules related to relay log

`DM_relay_process_exits_with_error`

- Description:

  When the relay log processing unit encounters an error, this unit moves to `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

`DM_remain_storage_of_relay_log`

- Description:

  When the free space of the disk where the relay log is located is less than 10G, an alert is triggered.

- Solutions:

  You can take the following methods to handle the alert:

  - Delete unwanted data manually to increase free disk space.
  - Reconfigure the automatic data purge strategy of the relay log or purge data manually.
  - Execute the command `pause-relay` to pause the relay log pulling process. After there is enough free disk space, resume the process by running the command `resume-relay`. Note that you must not purge upstream binlog files that have not been pulled after the relay log pulling process is paused.

`DM_relay_log_data_corruption`

- Description:

  When the relay log processing unit validates the binlog event read from the upstream and detects abnormal checksum information, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

`DM_fail_to_read_binlog_from_master`

- Description:

  If an error occurs when the relay log processing unit tries to read the binlog event from the upstream, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

`DM_fail_to_write_relay_log`

- Description:

  If an error occurs when the relay log processing unit tries to write the binlog event into the relay log file, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

`DM_binlog_file_gap_between_master_relay`

- Description:

  When the number of the binlog files in the current upstream MySQL/MariaDB exceeds that of the latest binlog files pulled by the relay log processing unit by **more than** 1 for 10 minutes, and an alert is triggered.

- Solution:

  Refer to Troubleshoot DM.

13.6.7.7.4 Alert rules related to Dump/Load

`DM_dump_process_exists_with_error`

- Description:

  When the Dump processing unit encounters an error, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

`DM_load_process_exists_with_error`

- Description:

  When the Load processing unit encounters an error, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

13.6.7.7.5 Alert rules related to binlog replication

`DM_sync_process_exists_with_error`

- Description:

  When the binlog replication processing unit encounters an error, this unit moves to the `Paused` state, and an alert is triggered immediately.

- Solution:

  Refer to Troubleshoot DM.

`DM_binlog_file_gap_between_master_syncer`

- Description:

  When the number of the binlog files in the current upstream MySQL/MariaDB exceeds that of the latest binlog files processed by the relay log processing unit by **more than** 1 for 10 minutes, an alert is triggered.

- Solution:

  Refer to Handle Performance Issues.

`DM_binlog_file_gap_between_relay_syncer`

- Description:

  When the number of the binlog files in the current relay log processing unit exceeds that of the latest binlog files processed by the binlog replication processing unit by **more than** 1 for 10 minutes, an alert is triggered.

- Solution:

  Refer to Handle Performance Issues.

13.6.7.8 Daily Check for TiDB Data Migration

This document summarizes how to perform a daily check on TiDB Data Migration (DM).

- Method 1: Execute the `query-status` command to check the running status of the task and the error output (if any). For details, see Query Status.
- Method 2: If Prometheus and Grafana are correctly deployed when you deploy the DM cluster using TiUP, you can view DM monitoring metrics in Grafana. For example, suppose that the Grafana's address is `172.16.10.71`, go to http://172.16.10.71:3000, enter the Grafana dashboard, and select the DM Dashboard to check monitoring metrics of DM. For more information of these metrics, see DM Monitoring Metrics.

- Method 3: Check the running status of DM and the error (if any) using the log file.

    - DM-master log directory: It is specified by the `--log-file` DM-master process parameter. If DM is deployed using TiUP, the log directory is `{log_dir}` in the DM-master node.
    - DM-worker log directory: It is specified by the `--log-file` DM-worker process parameter. If DM is deployed using TiUP, the log directory is `{log_dir}` in the DM-worker node.

13.6.8 Reference

13.6.8.1 Architecture

13.6.8.1.1 DM-worker Introduction

DM-worker is a tool used to migrate data from MySQL/MariaDB to TiDB.

It has the following features:

- Acts as a secondary database of any MySQL or MariaDB instance
- Reads the binlog events from MySQL/MariaDB and persists them to the local storage
- A single DM-worker supports migrating the data of one MySQL/MariaDB instance to multiple TiDB instances
- Multiple DM-workers support migrating the data of multiple MySQL/MariaDB instances to one TiDB instance

DM-worker processing unit

A DM-worker task contains multiple logic units, including relay log, the dump processing unit, the load processing unit, and binlog replication.

Relay log

The relay log persistently stores the binlog data from the upstream MySQL/MariaDB and provides the feature of accessing binlog events for the binlog replication.

Its rationale and features are similar to the relay log of MySQL. For details, see MySQL Relay Log.

Dump processing unit

The dump processing unit dumps the full data from the upstream MySQL/MariaDB to the local disk.

Load processing unit

The load processing unit reads the dumped files of the dump processing unit and then loads these files to the downstream TiDB.

Binlog replication/sync processing unit

Binlog replication/sync processing unit reads the binlog events of the upstream MySQL/-MariaDB or the binlog events of the relay log, transforms these events to SQL statements, and then applies these statements to the downstream TiDB.

Privileges required by DM-worker

This section describes the upstream and downstream database users' privileges required by DM-worker, and the user privileges required by the respective processing unit.

Upstream database user privileges

The upstream database (MySQL/MariaDB) user must have the following privileges:

Privilege	Scope
SELECT	Tables
RELOAD	Global
REPLICATION SLAVE	Global
REPLICATION CLIENT	Global

If you need to migrate the data from db1 to TiDB, execute the following GRANT statement:

```
GRANT RELOAD,REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'your_user'@'
    ↪ your_wildcard_of_host';
GRANT SELECT ON db1.* TO 'your_user'@'your_wildcard_of_host';
```

If you also need to migrate the data from other databases into TiDB, make sure the same privileges are granted to the user of the respective databases.

Downstream database user privileges

The downstream database (TiDB) user must have the following privileges:

Privilege	Scope
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables
INDEX	Tables

Execute the following GRANT statement for the databases or tables that you need to migrate:

```
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER,INDEX ON db.table TO '
    ↪ your_user'@'your_wildcard_of_host';
```

```
GRANT ALL ON dm_meta.* TO 'your_user'@'your_wildcard_of_host';
```

Minimal privilege required by each processing unit

Processing unit	Minimal upstream (MySQL/MariaDB) privilege	Minimal downstream (TiDB) privilege	Minimal system privilege
Relay log	`REPLICATION SLAVE` (reads the binlog)`REPLICATION CLIENT` (`show master status`, `show slave status`)	NULL	Read/Write local files
Dump	`SELECTRELOAD` (flushes tables with Read lock and unlocks tables)	NULL	Write local files
Load	NULL	`SELECT` (Query the checkpoint history)`CREATE` (creates a database/table)`DELETE` (deletes checkpoint)`INSERT` (Inserts the Dump data)	Read/Write local files

Processing unit	Minimal upstream (MySQL/MariaDB) privilege	Minimal downstream (TiDB) privilege	Minimal system privilege
Binlog replication	`REPLICATION SLAVE` (reads the binlog)`REPLICATION CLIENT` (`show master status`, `show slave status`)	`SELECT` (shows the index and column)`INSERT` (DML)`UPDATE` (DML)`DELETE` (DML)`CREATE` (creates a database/table)`DROP` (drops databases/tables)`ALTER` (alters a table)`INDEX` (creates/drops an index)	Read/Write local files

Note:

These privileges are not immutable and they change as the request changes.

13.6.8.1.2 DM Safe Mode

Safe mode is a special operation mode for DM to perform incremental replication. In safe mode, when the DM incremental replication component replicates binlog events, DM forcibly rewrites all the `INSERT` and `UPDATE` statements before executing them in the downstream.

During safe mode, one binlog event can be replicated repeatedly to the downstream with idempotence guaranteed. Thus, the incremental replication is *safe*.

After resuming a data replication task from a checkpoint, DM might repeatedly replicate some binlog events, which leads to the following issues:

- During incremental replication, the operation of executing DML and the operation of writing checkpoints are not simultaneous. The operation of writing checkpoints and

writing data into the downstream database is not atomic. Therefore, **when DM exits abnormally, checkpoints might only record the restoration point before the exit point**.

- When DM restarts a replication task and resumes incremental replication from a checkpoint, some data between the checkpoint and the exit point might already be processed before the abnormal exit. This causes **some SQL statements to be executed repeatedly**.
- If an `INSERT` statement is executed repeatedly, the primary key or the unique index might encounter a conflict, which leads to a replication failure. If an `UPDATE` statement is executed repeatedly, the filter condition might not be able to locate the previously updated records.

In safe mode, DM can rewrite SQL statements to resolve the preceding issues.

Working principle

In safe mode, DM guarantees the idempotency of binlog events by rewriting SQL statements. Specifically, the following SQL statements are rewritten:

- `INSERT` statements are rewritten to `REPLACE` statements.
- `UPDATE` statements are analyzed to obtain the value of the primary key or the unique index of the row updated. `UPDATE` statements are then rewritten to `DELETE` + `REPLACE` statements in the following two steps: DM deletes the old record using the primary key or unique index, and inserts the new record using the `REPLACE` statement.

`REPLACE` is a MySQL-specific syntax for inserting data. When you insert data using `REPLACE`, and the new data and existing data have a primary key or unique constraint conflict, MySQL deletes all the conflicting records and executes the insert operation, which is equivalent to "force insert". For details, see REPLACE statement in MySQL documentation.

Assume that a `dummydb.dummytbl` table has a primary key `id`. Execute the following SQL statements repeatedly on this table:

```
INSERT INTO dummydb.dummytbl (id, int_value, str_value) VALUES (123, 999, '
    ↪ abc');
UPDATE dummydb.dummytbl SET int_value = 888999 WHERE int_value = 999; --
    ↪ Suppose there is no other record with int_value = 999
UPDATE dummydb.dummytbl SET id = 999 WHERE id = 888; -- Update the primary
    ↪ key
```

With safe mode enabled, when the preceding SQL statements are executed again in the downstream, they are rewritten as follows:

```
REPLACE INTO dummydb.dummytbl (id, int_value, str_value) VALUES (123, 999,
    ↪ 'abc');
DELETE FROM dummydb.dummytbl WHERE id = 123;
```

```
REPLACE INTO dummydb.dummytbl (id, int_value, str_value) VALUES (123,
    ↪ 888999, 'abc');
DELETE FROM dummydb.dummytbl WHERE id = 888;
REPLACE INTO dummydb.dummytbl (id, int_value, str_value) VALUES (999,
    ↪ 888888, 'abc888');
```

In the preceding statements, `UPDATE` is rewritten as `DELETE` + `REPLACE`, rather than `DELETE` + `INSERT`. If `INSERT` is used here, when you insert a duplicate record with `id = 999`, the database reports a primary key conflict. This is why `REPLACE` is used instead. The new record will replace the existing record.

By rewriting SQL statements, DM overwrites the existing row data using the new row data when performing duplicate insert or update operations. This guarantees that insert and update operations are executed repeatedly.

Enable safe mode

You can enable safe mode either automatically or manually. This section describes the detailed steps.

Automatically enable

When DM resumes an incremental replication task from a checkpoint (For example, DM worker restart or network reconnection), DM automatically enables safe mode for a period (60 seconds by default).

Whether to enable safe mode is related to `safemode_exit_point` in the checkpoint. When an incremental replication task is paused abnormally, DM tries to replicate all DML statements in the memory to the downstream and records the latest binlog position among the DML statements as `safemode_exit_point`, which is saved to the last checkpoint.

The detailed logic is as follows:

- If the checkpoint contains `safemode_exit_point`, the incremental replication task is paused abnormally. When DM resumes the task, the binlog position of the checkpoint to be resumed (**begin position**) is earlier than `safemode_exit_point`, which represents the binlog events between the begin position and the `safemode_exit_point` might have been processed in the downstream. So, during the resume process, some binlog events might be executed repeatedly. Therefore, enabling safe mode can make these binlog positions **safe**. After the binlog position exceeds the `safemode_exit_point`, DM automatically disables safe mode unless safe mode is enabled manually.
- If the checkpoint does not contain `safemode_exit_point`, there are two cases:
    1. This is a new task, or this task is paused as expected.
    2. This task is paused abnormally but DM fails to record `safemode_exit_point`, or the DM process exits abnormally.

In the second case, DM does not know which binlog events after the checkpoint are executed in the downstream. To ensure that repeatedly executed binlog events do not cause any problems, DM automatically enables safe mode during the first two checkpoint intervals. The default interval between two checkpoints is 30 seconds, which means when a normal incremental replication task starts, safe mode is enforced for the first 60 seconds (2 * 30 seconds).

Usually, it is not recommended to change the checkpoint interval to adjust the safe mode period at the beginning of the incremental replication task. However, if you do need a change, you can manually enable safe mode (recommended) or change the `checkpoint-flush-interval` item in syncer configuration.

Manually enable

You can set the `safe-mode` item in the syncer configuration to enable safe mode during the entire replication process. `safe-mode` is a bool type parameter and is `false` by default. If it is set to `true`, DM enables safe mode for the whole incremental replication process.

The following is a task configuration example with safe mode enabled:

```
syncers:                           # The running configurations of the sync
   ↪ processing unit.
  global:                          # Configuration name.
    # Other configuration items are not provided in this example.
    safe-mode: true                # Enables safe mode for the whole
       ↪ incremental replication process.
    # Other configuration items are not provided in this example.
##### ----------- Instance configuration -----------
mysql-instances:
  -
    source-id: "mysql-replica-01"
    # Other configuration items are not provided in this example.
    syncer-config-name: "global"       # Name of the syncers configuration.
```

Notes for safe mode

If you want to enable safe mode during the entire replication process for safety reasons, be aware of the following:

- **Incremental replication in safe mode consumes extra overhead.** Frequent `DELETE` + `REPLACE` operations result in frequent changes to primary keys or unique indexes, which creates a greater performance overhead than executing `UPDATE` statements only.
- **Safe mode forces the replacement of records with the same primary key, which might result in data loss in the downstream.** When you merge and migrate shards from the upstream to the downstream, incorrect configuration might lead to a large number of primary key or unique key conflicts. If safe mode is enabled in

this situation, the downstream might lose lots of data without showing any exception, resulting in severe data inconsistency.

- **Safe mode relies on the primary key or unique index to detect conflicts.** If the downstream table has no primary key or unique index, DM cannot use `REPLACE` ↪ to replace and insert records. In this case, even if safe mode is enabled and DM rewrites `INSERT` to `REPLACE` statements, duplicate records are still inserted into the downstream.

In summary, if the upstream database has data with duplicate primary keys, and your application tolerates loss of duplicate records and performance overhead, you can enable safe mode to ignore data duplication.

13.6.8.1.3 Data Migration Relay Log

The Data Migration (DM) relay log consists of several sets of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

After relay log is enabled, DM-worker automatically migrates the upstream binlog to the local configuration directory (the default migration directory is `<deploy_dir>/<relay_log>` if DM is deployed using TiUP). The default value of `<relay_log>` is `relay-dir` and can be modified in Upstream Database Configuration File. Since v5.4.0, you can configure the local configuration directory through `relay-dir` in the DM-worker configuration file, which takes precedence over the configuration file of the upstream database.

User scenarios

In MySQL, storage space is limited, so the binlog is automatically purged when the maximum retention time is reached. After the upstream database purges the binlog, DM fails to pull the purged binlog and the migration task fails. For each migration task, DM creates a connection in the upstream to pull binlog. Too many connections might cause a heavy workload on the upstream database.

When the relay log is enabled, multiple migration task with the same upstream database can reuse the relay log that has been pulled to the local disk. This **relieves the pressure on the upstream database**.

For full and incremental data migration tasks (`task-mode=all`), DM needs to first migrate full data and then perform incremental migration based on binlog. If the full migration phase takes long, the upstream binlog might be purged, which results in incremental migration failure. To avoid this situation, you can enable the relay log feature so that DM automatically retains enough log in the local disk and **ensures the incremental migration task can be performed normally**.

It is generally recommended to enable relay log, but be aware of the following potential issue:

Because relay log must be written to the disk, it consumes external IO and CPU resources. This prolongs the whole data replication process and increases the data replication latency. For **latency-sensitive** scenarios, it is not recommended to enable relay log.

Note:

In DM v2.0.7 and later versions, relay log writes are optimized. The latency and CPU resource consumption is relatively low.

Use relay log

This section describes how to enable and disable relay log, query relay log status, and purge relay log.

Enable and disable relay log

In v5.4.0 and later versions, you can enable relay log by setting `enable-relay` to `true`. Since v5.4.0, when binding the upstream data source, DM-worker checks the `enable-relay` item in the configuration of the data source. If `enable-relay` is `true`, the relay log feature is enabled for this data source.

For the detailed configuration method, see Upstream Database Configuration File.

In addition, you can also dynamically adjust the `enable-relay` configuration of the data source using the `start-relay` or `stop-relay` command to enable or disable relay log in time.

```
start-relay -s mysql-replica-01
```

```
{
    "result": true,
    "msg": ""
}
```

Note:

In DM v2.0.x later than DM v2.0.2 and in v5.3.0, the configuration item `enable-relay` in the source configuration file is no longer valid, and you can only use `start-relay` and `stop-relay` to enable and disable relay log. If DM finds that `enable-relay` is set to `true` when loading the data source configuration, it outputs the following message:

```
Please use `start-relay` to specify which workers should pull
    ↪ relay log of relay-enabled sources.
```

Warning:

This startup method is marked as deprecated in v6.1 and might be removed in a future release. You can see the following prompt in the output of the relevant command: `start-relay/stop-relay with worker name will be deprecated soon. You can try stopping relay first and use start-relay without worker name instead.`

In the command `start-relay`, you can configure one or more DM-workers to migrate relay logs for the specified data source, but the DM-workers specified in the parameter must be free or have been bound to the upstream data source. Examples are as follows:

```
start-relay -s mysql-replica-01 worker1 worker2
```

```
{
    "result": true,
    "msg": ""
}
```

```
stop-relay -s mysql-replica-01 worker1 worker2
```

```
{
    "result": true,
    "msg": ""
}
```

In DM versions earlier than v2.0.2 (not including v2.0.2), DM checks the configuration item `enable-relay` in the source configuration file when binding a DM-worker to an upstream data source. If `enable-relay` is set to `true`, DM enables the relay log feature for the data source.

See Upstream Database Configuration File for how to set the configuration item `enable-relay`.

Query relay log status

You can use the command `query-status -s` to query the status of the relay log:

```
query-status -s mysql-replica-01
```

Expected output

```
{
    "result": true,
    "msg": "",
```

```
"sources": [
    {
        "result": true,
        "msg": "no sub task started",
        "sourceStatus": {
            "source": "mysql-replica-01",
            "worker": "worker2",
            "result": null,
            "relayStatus": {
                "masterBinlog": "(mysql-bin.000005, 916)",
                "masterBinlogGtid": "09bec856-ba95-11ea-850a-58f2b4af5188
                    ↪ :1-28",
                "relaySubDir": "09bec856-ba95-11ea-850a-58f2b4af5188
                    ↪ .000001",
                "relayBinlog": "(mysql-bin.000005, 4)",
                "relayBinlogGtid": "09bec856-ba95-11ea-850a-58f2b4af5188
                    ↪ :1-28",
                "relayCatchUpMaster": false,
                "stage": "Running",
                "result": null
            }
        },
        "subTaskStatus": [
        ]
    },
    {
        "result": true,
        "msg": "no sub task started",
        "sourceStatus": {
            "source": "mysql-replica-01",
            "worker": "worker1",
            "result": null,
            "relayStatus": {
                "masterBinlog": "(mysql-bin.000005, 916)",
                "masterBinlogGtid": "09bec856-ba95-11ea-850a-58f2b4af5188
                    ↪ :1-28",
                "relaySubDir": "09bec856-ba95-11ea-850a-58f2b4af5188
                    ↪ .000001",
                "relayBinlog": "(mysql-bin.000005, 916)",
                "relayBinlogGtid": "",
                "relayCatchUpMaster": true,
                "stage": "Running",
                "result": null
            }
        },
```

```
            "subTaskStatus": [
            ]
        }
    ]
}
```

Pause and resume relay log

You can use the command `pause-relay` to pause the pulling process of relay logs and use the command `resume-relay` to resume the process. You need to specify the `source-id` of the upstream data source when executing these two commands. See the following examples:

```
pause-relay -s mysql-replica-01 -s mysql-replica-02
```

Expected output

```
{
    "op": "PauseRelay",
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```

```
resume-relay -s mysql-replica-01
```

Expected output

```
{
    "op": "ResumeRelay",
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
```

```
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        }
    ]
}
```

Purge relay logs

DM provides two ways to purge relay logs: manual purge and automatic purge. Neither of these two methods purges active relay logs.

Note:

- Active relay log: The relay log is being used by a data migration task. An active relay log is currently only updated and written in the Syncer Unit. If a data migration task in All mode spends more time on full export/import than the expiration time configured in the purge of the data source, the relay log is still purged.
- Expired relay log: The difference between the last modification time of the relay log file and the current time is greater than the value of the `expires` field in the configuration file.

Automatic purge

You can enable automatic purge and configure its strategy in the source configuration file. See the following example:

```
##### relay log purge strategy
purge:
    interval: 3600
    expires: 24
    remain-space: 15
```

- `purge.interval`
    - The interval of automatic purge in the background, in seconds.
    - "3600" by default, indicating a background purge task is performed every 3600 seconds.
- `purge.expires`

    - The number of hours for which the relay log (that has been previously written to the relay processing unit, and that is not being used or will not be read later by the currently running data migration task) can be retained before being purged in the automatic background purge.
    - "0" by default, indicating data purge is not performed according to the update time of the relay log.

- `purge.remain-space`

    - The amount of remaining disk space in GB less than which the specified DM-worker machine tries to purge the relay log that can be purged securely in the automatic background purge. If it is set to `0`, data purge is not performed according to the remaining disk space.
    - "15" by default, indicating when the available disk space is less than 15 GB, DM-master tries to purge the relay log securely.

Manual purge

Manual purge means using the `purge-relay` command provided by dmctl to specify `subdir` and the binlog name thus to purge all the relay logs **before** the specified binlog. If the `-subdir` option in the command is not specified, all relay logs **before** the current relay log sub-directory are purged.

Assuming that the directory structure of the current relay log is as follows:

```
$ tree .
.
|-- deb76a2b-09cc-11e9-9129-5242cf3bb246.000001
|   |-- mysql-bin.000001
|   |-- mysql-bin.000002
|   |-- mysql-bin.000003
|   `-- relay.meta
|-- deb76a2b-09cc-11e9-9129-5242cf3bb246.000003
|   |-- mysql-bin.000001
|   `-- relay.meta
|-- e4e0e8ab-09cc-11e9-9220-82cc35207219.000002
|   |-- mysql-bin.000001
|   `-- relay.meta
`-- server-uuid.index

$ cat server-uuid.index
deb76a2b-09cc-11e9-9129-5242cf3bb246.000001
e4e0e8ab-09cc-11e9-9220-82cc35207219.000002
deb76a2b-09cc-11e9-9129-5242cf3bb246.000003
```

- Executing the following `purge-relay` command in dmctl purges all relay log files **before** `e4e0e8ab-09cc-11e9-9220-82cc35207219.000002/mysql-bin.000001`, which

are all relay log files in `deb76a2b-09cc-11e9-9129-5242cf3bb246.000001`. Files in `e4e0e8ab-09cc-11e9-9220-82cc35207219.000002` and `deb76a2b-09cc-11e9-9129-5242cf3bb246.000003` are retained.

```
purge-relay -s mysql-replica-01 --filename mysql-bin.000001 --sub-dir
    ↪ e4e0e8ab-09cc-11e9-9220-82cc35207219.000002
```

- Executing the following `purge-relay` command in dmctl purges all relay log files **before the current** (`deb76a2b-09cc-11e9-9129-5242cf3bb246.000003`) directory's `mysql-bin.000001`, which are all relay log files in `deb76a2b-09cc-11e9-9129-5242cf3bb246.000001` and `e4e0e8ab-09cc-11e9-9220-82cc35207219.000002`. Files in `deb76a2b-09cc-11e9-9129-5242cf3bb246.000003` are retained.

```
purge-relay -s mysql-replica-01 --filename mysql-bin.000001
```

Internal mechanism of relay log

This section introduces the internal mechanism of relay log.

Directory structure

An example of the directory structure of the local storage for a relay log:

```
<deploy_dir>/<relay_log>/
|-- 7e427cc0-091c-11e9-9e45-72b7c59d52d7.000001
|   |-- mysql-bin.000001
|   |-- mysql-bin.000002
|   |-- mysql-bin.000003
|   |-- mysql-bin.000004
|   `-- relay.meta
|-- 842965eb-091c-11e9-9e45-9a3bff03fa39.000002
|   |-- mysql-bin.000001
|   `-- relay.meta
`-- server-uuid.index
```

- `subdir`:
    - DM-worker stores the binlog migrated from the upstream database in the same directory. Each directory is a `subdir`.
    - `subdir` is named in the format of `<Upstream database UUID>.<Local subdir serial number>`.
    - After a switch between primary and secondary instances in the upstream, DM-worker generates a new `subdir` directory with an incremental serial number.
    - In the above example, for the `7e427cc0-091c-11e9-9e45-72b7c59d52d7.000001` directory, `7e427cc0-091c-11e9-9e45-72b7c59d52d7` is the upstream database UUID and `000001` is the local `subdir` serial number.

- `server-uuid.index`: records a list of the currently available `subdir` directories.
- `relay.meta`: stores the information of the migrated binlog in each `subdir`. For example,

```
cat c0149e17-dff1-11e8-b6a8-0242ac110004.000001/relay.meta
```

```
binlog-name = "mysql-bin.000010"                          # The name of the
    ↪ currently migrated binlog.
binlog-pos = 63083620                                     # The position of
    ↪ the currently migrated binlog.
binlog-gtid = "c0149e17-dff1-11e8-b6a8-0242ac110004:1-3328" # GTID of
    ↪ the currently migrated binlog.
```

There might also be multiple GTIDs:

```
cat 92acbd8a-c844-11e7-94a1-1866daf8accc.000001/relay.meta
```

```
binlog-name = "mysql-bin.018393"
binlog-pos = 277987307
binlog-gtid = "3ccc475b-2343-11e7-be21-6c0b84d59f30:1-14,406a3f61-690d
    ↪ -11e7-87c5-6c92bf46f384:1-94321383,53bfca22-690d-11e7-8a62-18
    ↪ ded7a37b78:1-495,686e1ab6-c47e-11e7-a42c-6c92bf46f384
    ↪ :1-34981190,03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474
    ↪ d3c-28c7-11e7-8352-203db246dd3d:1-170,10b039fc-c843-11e7-8f6a
    ↪ -1866daf8d810:1-308290454"
```

The position where DM receives the binlog

- DM obtains the earliest position that each migration task needs from the saved checkpoint (in the downstream `dm_meta` schema by default). If this position is later than any of the following positions, DM starts to migrate from this position.
- If the local relay log is valid, which means that the relay log contains valid `server-uuid.index`, `subdir`, and `relay.meta` files, DM-worker recovers the migration from the position recorded in `relay.meta`.
- If there is no valid local relay log, but the upstream data source configuration file specifies `relay-binlog-name` or `relay-binlog-gtid`:
  - In non-GTID mode, if `relay-binlog-name` is specified, DM-worker starts to migrate from the specified binlog file.
  - In GTID mode, if `relay-binlog-gtid` is specified, DM-worker starts to migrate from the specified GTID.
- If there is no valid local relay log and the `relay-binlog-name` or `relay-binlog-gtid` is not specified in the DM configuration file:

- In non-GTID mode, DM-worker starts to migrate from the earliest binlog that each subtask is migrating, until the latest binlog is migrated.
- In GTID mode, DM-worker starts to migrate from the earliest GTID that each subtask is migrating, until the latest GTID is migrated.

> **Note:**
>
> If the upstream relay log is purged, an error occurs. In this case, you need to configure `relay-binlog-gtid` to specify the start position of the migration.

13.6.8.1.4 Special Handling of DM DDLs

When TiDB Data Migration (DM) migrates data, it parses the DDL statements and handles them according to the statement type and the current migration stage.

Skip DDL statements

The following statements are not supported by DM, so DM skips them directly after parsing.

```
<tr>
    <th>Description</th>
    <th>SQL</th>
</tr>
<tr>
    <td>transaction</td>
    <td><code>^SAVEPOINT</code></td>
</tr>
<tr>
    <td>skip all flush sqls</td>
    <td><code>^FLUSH</code></td>
</tr>
<tr>
    <td rowspan="3">table maintenance</td>
    <td><code>^OPTIMIZE\\s+TABLE</code></td>
</tr>
<tr>
    <td><code>^ANALYZE\\s+TABLE</code></td>
</tr>
<tr>
    <td><code>^REPAIR\\s+TABLE</code></td>
</tr>
<tr>
    <td>temporary table</td>
```

```
    <td><code>^DROP\\s+(\\/\\*\\!40005\\s+)?TEMPORARY\\s+(\\*\\/\\s+)?TABLE
       ↪ </code></td>
</tr>
<tr>
    <td rowspan="2">trigger</td>
    <td><code>^CREATE\\s+(DEFINER\\s?=.+?)?TRIGGER</code></td>
</tr>
<tr>
    <td><code>^DROP\\s+TRIGGER</code></td>
</tr>
<tr>
    <td rowspan="3">procedure</td>
    <td><code>^DROP\\s+PROCEDURE</code></td>
</tr>
<tr>
    <td><code>^CREATE\\s+(DEFINER\\s?=.+?)?PROCEDURE</code></td>
</tr>
<tr>
    <td><code>^ALTER\\s+PROCEDURE</code></td>
</tr>
<tr>
    <td rowspan="3">view</td>
    <td><code>^CREATE\\s*(OR REPLACE)?\\s+(ALGORITHM\\s?=.+?)?(DEFINER\\s
       ↪ ?=.+?)?\\s+(SQL SECURITY DEFINER)?VIEW</code></td>
</tr>
<tr>
    <td><code>^DROP\\s+VIEW</code></td>
</tr>
<tr>
    <td><code>^ALTER\\s+(ALGORITHM\\s?=.+?)?(DEFINER\\s?=.+?)?(SQL SECURITY
       ↪ DEFINER)?VIEW</code></td>
</tr>
<tr>
    <td rowspan="4">function</td>
    <td><code>^CREATE\\s+(AGGREGATE)?\\s*?FUNCTION</code></td>
</tr>
<tr>
    <td><code>^CREATE\\s+(DEFINER\\s?=.+?)?FUNCTION</code></td>
</tr>
<tr>
    <td><code>^ALTER\\s+FUNCTION</code></td>
</tr>
<tr>
    <td><code>^DROP\\s+FUNCTION</code></td>
</tr>
```

```
<tr>
    <td rowspan="3">tableSpace</td>
    <td><code>^CREATE\\s+TABLESPACE</code></td>
</tr>
<tr>
    <td><code>^ALTER\\s+TABLESPACE</code></td>
</tr>
<tr>
    <td><code>^DROP\\s+TABLESPACE</code></td>
</tr>
<tr>
    <td rowspan="3">event</td>
    <td><code>^CREATE\\s+(DEFINER\\s?=.+?)?EVENT</code></td>
</tr>
<tr>
    <td><code>^ALTER\\s+(DEFINER\\s?=.+?)?EVENT</code></td>
</tr>
<tr>
    <td><code>^DROP\\s+EVENT</code></td>
</tr>
<tr>
    <td rowspan="7">account management</td>
    <td><code>^GRANT</code></td>
</tr>
<tr>
    <td><code>^REVOKE</code></td>
</tr>
<tr>
    <td><code>^CREATE\\s+USER</code></td>
</tr>
<tr>
    <td><code>^ALTER\\s+USER</code></td>
</tr>
<tr>
    <td><code>^RENAME\\s+USER</code></td>
</tr>
<tr>
    <td><code>^DROP\\s+USER</code></td>
</tr>
<tr>
    <td><code>^DROP\\s+USER</code></td>
</tr>
```

Rewrite DDL statements

The following statements are rewritten before being replicated to the downstream.

Original statement	Rewritten statement
`^CREATE DATABASE...`	`^CREATE DATABASE...IF NOT EXISTS`
`^CREATE TABLE...`	`^CREATE TABLE..IF NOT EXISTS`
`^DROP DATABASE...`	`^DROP DATABASE...IF EXISTS`
`^DROP TABLE...`	`^DROP TABLE...IF EXISTS`
`^DROP INDEX...`	`^DROP INDEX...IF EXISTS`

Shard merge migration tasks

When DM merges and migrates tables in pessimistic or optimistic mode, the behavior of DDL replication is different from that in other scenarios. For details, refer to Pessimistic Mode and Optimistic Mode.

Online DDL

The Online DDL feature also handles DDL events in a special way. For details, refer to Migrate from Databases that Use GH-ost/PT-osc.

13.6.8.2 Mechanism

13.6.8.2.1 DML Replication Mechanism in Data Migration

This document introduces how the core processing unit Sync in DM processes DML statements read from the data source or relay log. This document introduces the complete processing flow of DML events in DM, including the logic of binlog reading, filtering, routing, transformation, optimization, and execution. This document also explains the DML optimization logic and DML execution logic in detail.

DML processing flow

The Sync unit processes DML statements as follows:

1. Read the binlog event from the MySQL, MariaDB, or relay log.
2. Transform the binlog event read from the data source:
    1. Binlog filter: filter binlog events according to binlog expressions, configured by `filters`.
    2. Table routing: transform the "database/table" name according to the "database/table" routing rule, configured by `routes`.
    3. Expression filter: filter binlog events according to SQL expressions, configured by `expression-filter`.
3. Optimize the DML execution plan:

1. Compactor: merge multiple operations on the same record (with the same primary key) into one operation. This feature is enabled by `syncer.compact`.
2. Causality: perform conflict detection on different records (with different primary keys) to improve the concurrency of replication.
3. Merger: merge multiple binlog events into one DML statement, enabled by `syncer.multiple-rows`.

4. Execute the DML to the downstream.

5. Periodically save the binlog position or GTID to the checkpoint.

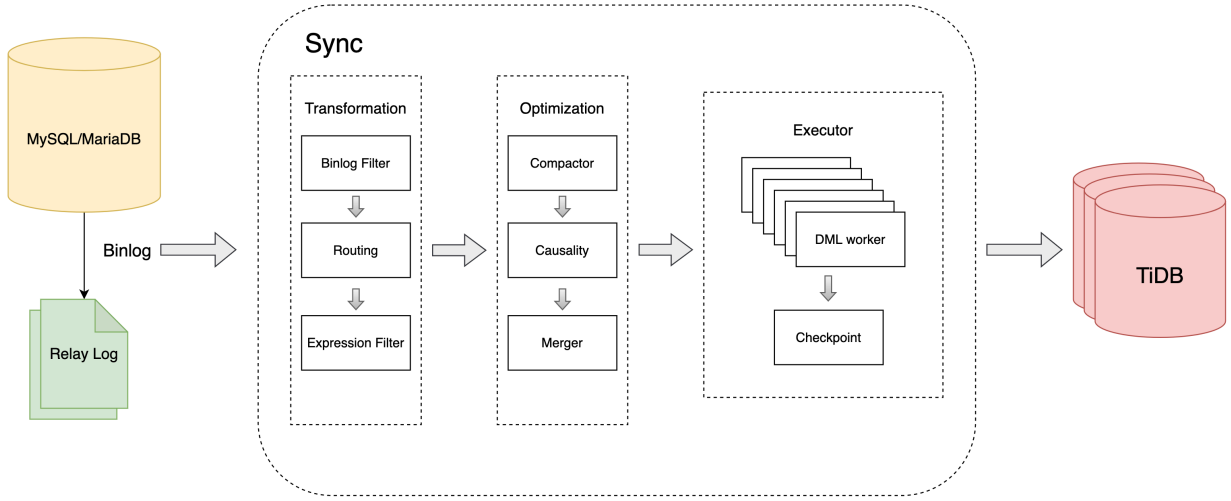


Figure 257: DML processing logic

DML optimization logic

The Sync unit implements the DML optimization logic through three steps: Compactor, Causality, and Merger.

Compactor

According to the binlog records of the upstream, DM captures the changes of the records and replicates them to the downstream. When the upstream makes multiple changes to the same record (`INSERT`/`UPDATE`/`DELETE`) in a short period of time, DM can compress multiple changes into one change through Compactor to reduce the pressure on the downstream and improve the throughput. For example:

```
INSERT + UPDATE => INSERT
INSERT + DELETE => DELETE
UPDATE + UPDATE => UPDATE
UPDATE + DELETE => DELETE
DELETE + INSERT => UPDATE
```

The Compactor feature is disabled by default. To enable it, you can set `syncer.compact` to `true` in the `sync` configuration module of the replication task, as shown below:

```
syncers:                          # The configuration parameters of the sync
    ↪ processing unit
  global:                         # Configuration name
    ...                            # Other configurations are omitted
    compact: true
```

Causality

The sequential replication model of MySQL binlog requires that binlog events be replicated in the order of binlog. This replication model cannot meet the requirements of high QPS and low replication latency. In addition, because not all operations involved in binlog have conflicts, sequential replication is not necessary in those cases.

DM recognizes the binlog that needs to be executed sequentially through conflict detection, and ensures that these binlog are executed sequentially while maximizing the concurrency of other binlog. This helps improve the performance of binlog replication.

Causality adopts an algorithm similar to the union-find algorithm to classify each DML and group DMLs that are related to each other.

Merger

According to the MySQL binlog protocol, each binlog corresponds to a change operation of one row of data. Through Merger, DM can merge multiple binlogs into one DML and execute it to the downstream, reducing the network interaction. For example:

```
  INSERT tb(a,b) VALUES(1,1);
+ INSERT tb(a,b) VALUES(2,2);
= INSERT tb(a,b) VALUES(1,1),(2,2);
  UPDATE tb SET a=1, b=1 WHERE a=1;
+ UPDATE tb SET a=2, b=2 WHERE a=2;
= INSERT tb(a,b) VALUES(1,1),(2,2) ON DUPLICATE UPDATE a=VALUES(a), b=VALUES
    ↪ (b)
  DELETE tb WHERE a=1
+ DELETE tb WHERE a=2
= DELETE tb WHERE (a) IN (1),(2);
```

The Merger feature is disabled by default. To enable it, you can set `syncer.multiple-rows` to `true` in the `sync` configuration module of the replication task, as shown below:

```
syncers:                          # The configuration parameters of the sync
    ↪ processing unit
  global:                         # Configuration name
    ...                            # Other configurations are omitted
    multiple-rows: true
```

DML execution logic

After the Sync unit optimizes the DML, it performs the execution logic.

DML generation

DM has an embedded schema tracker that records the schema information of the upstream and downstream:

- When DM receives a DDL statement, DM updates the table schema of the internal schema tracker.
- When DM receives a DML statement, DM generates the corresponding DML according to the table schema of the schema tracker.

The logic of generating DML is as follows:

1. The Sync unit records the initial table structure of the upstream:
    - When starting a full and incremental task, Sync uses the **table structure exported during the upstream full data migration** as the initial table structure of the upstream.
    - When starting an incremental task, because MySQL binlog does not record the table structure information, Sync uses the **table structure of the corresponding table in the downstream** as the initial table structure of the upstream.
2. The user's upstream and downstream table structures might be inconsistent, for example, the downstream might have additional columns than the upstream, or the upstream and downstream primary keys are inconsistent. Therefore, to ensure the correctness of data replication, DM records the **primary key and unique key information of the corresponding table in the downstream**.
3. DM generates DML:
    - Use the **the upstream table structure recorded in the schema tracker** to generate the column names of the DML statement.
    - Use the **column values recorded in the binlog** to generate the column values of the DML statement.
    - Use the **downstream primary key or unique key recorded in the schema tracker** to generate the `WHERE` condition of the DML statement. When the table structure has no unique key, DM uses all the column values recorded in the binlog as the `WHERE` condition.

Worker count

Causality can divide binlog into multiple groups through conflict detection and execute them concurrently to the downstream. DM controls the concurrency by setting `worker-count`. When the CPU usage of the downstream TiDB is not high, increasing the concurrency can effectively improve the throughput of data replication.

You can modify the number of threads that concurrently migrate DML by modifying the `syncer.worker-count` configuration item.

Batch

DM batches multiple DMLs into a single transaction and executes it to the downstream. When a DML worker receives a DML, it adds the DML to the cache. When the number of DMLs in the cache reaches the preset threshold, or the DML worker does not receive DML for a long time, the DML worker executes the DMLs in the cache to the downstream.

You can modify the number of DMLs contained in a transaction by modifying the `syncer.batch` configuration item.

checkpoint

The operation of executing DML and updating checkpoint is not atomic.

In DM, checkpoint is updated every 30 seconds by default. Because there are multiple DML worker processes, the checkpoint process calculates the binlog position of the earliest replication progress of all DML workers, and uses this position as the current replication checkpoint. All binlogs earlier than this position are guaranteed to be successfully executed to the downstream.

Notes

Transaction consistency

DM replicates data at the row level and does not guarantee transaction consistency. In DM, an upstream transaction is split into multiple rows and distributed to different DML workers for concurrent execution. Therefore, when the DM replication task reports an error and pauses, or when the user manually pauses the task, the downstream might be in an intermediate state. That is, the DML statements in an upstream transaction might be partially replicated to the downstream, which might cause the downstream to be in an inconsistent state.

To ensure that the downstream is in a consistent state when the task is paused as much as possible, starting from DM v5.3.0, DM waits for 10 seconds before pausing the task to ensure that all transactions from the upstream are replicated to the downstream. However, if a transaction is not replicated to the downstream within 10 seconds, the downstream might still be in an inconsistent state.

Safe mode

The operation of DML execution and checkpoint update is not atomic, and the operation of checkpoint update and writing data to the downstream is also not atomic. When DM exits abnormally, the checkpoint might only record a recovery point before the exit time. Therefore, when the task is restarted, DM might write the same data multiple times, which means that DM actually provides the “at least once processing” logic, and the same data might be processed more than once.

To make sure the data is reentrant, DM enters the safe mode when it restarts from an abnormal exit.

When the safe mode is enabled, to make sure that data can be processed multiple times, DM performs the following conversions:

- Rewrite the `INSERT` statement of the upstream to the `REPLACE` statement.
- Rewrite the `UPDATE` statement of the upstream to the `DELETE` + `REPLACE` statement.

Exactly-once processing

Currently, DM only guarantees eventual consistency and does not support "exactly-once processing" and "keeping the original order of transactions".

13.6.8.3 Command Line

13.6.8.3.1 TiDB Data Migration Command-line Flags

This document introduces DM's command-line flags.

DM-master

`--advertise-addr`

- The external address of DM-master used to receive client requests
- The default value is `"{master-addr}"`
- Optional flag. It can be in the form of `"domain-name:port"`

`--advertise-peer-urls`

- The external address for communication between DM-master nodes
- The default value is `"{peer-urls}"`
- Optional flag. It can be in the form of `"http(s)://domain-name:port"`

`--config`

- The configuration file path of DM-master
- The default value is `""`
- Optional flag

`--data-dir`

- The directory used to store data of DM-master
- The default value is `"default.{name}"`
- Optional flag

`--initial-cluster`

- The "{node name}={external address}" list used to bootstrap DM-master cluster
- The default value is "{name}={advertise-peer-urls}"
- This flag needs to be specified if the join flag is not specified. A configuration example of a 3-node cluster is "dm-master-1=http://172.16.15.11:8291,dm-master-2=http ↪ ://172.16.15.12:8291,dm-master-3=http://172.16.15.13:8291"

--join

- The existing cluster's advertise-addr list when a DM-master node joins this cluster
- The default value is ""
- This flag needs to be specified if the initial-cluster flag is not specified. Suppose a new node joins a cluster that has 2 nodes, a configuration example is "172.16.15.11:8261,172.16.15.12:8261"

--log-file

- The output file name of the log
- The default value is ""
- Optional flag

-L

- The log level
- The default value is "info"
- Optional flag

--master-addr

- The address on which DM-master listens to the client's requests
- The default value is ""
- Required flag

--name

- The name of a DM-master node
- The default value is "dm-master-{hostname}"
- Required flag

--peer-urls

- The listening address for communications between DM-master nodes
- The default value is "http://127.0.0.1:8291"

- Required flag

DM-worker

`--advertise-addr`

- The external address of DM-worker used to receive client requests
- The default value is `"{worker-addr}"`
- Optional flag. It can be in the form of `"domain-name:port"`

`--config`

- The configuration file path of DM-worker
- The default value is `""`
- Optional flag

`--join`

- The `{advertise-addr}` list of DM-master nodes in a cluster when a DM-worker registers to this cluster
- The default value is `""`
- Required flag. A configuration example of 3-node (DM-master node) cluster is `"172.16.15.11:8261,172.16.15.12:8261,172.16.15.13:8261"`

`--log-file`

- The output file name of the log
- The default value is `""`
- Optional flag

`-L`

- The log level
- The default value is `"info"`
- Optional flag

`--name`

- The name of a DM-worker node
- The default value is `"{advertise-addr}"`
- Required flag

`--worker-addr`

- The address on which DM-worker listens to the client's requests
- The default value is `""`
- Required flag

dmctl

`--config`

- The configuration file path of dmctl
- The default value is `""`
- Optional flag

`--master-addr`

- The `{advertise-addr}` of any DM-master node in the cluster to be connected by dmctl
- The default value is `""`
- It is a required flag when dmctl interacts with DM-master

`--encrypt`

- Encrypts the plaintext database password into ciphertext
- The default value is `""`
- When this flag is specified, it is only used to encrypt the plaintext without interacting with the DM-master

`--decrypt`

- Decrypts ciphertext encrypted with dmctl into plaintext
- The default value is `""`
- When this flag is specified, it is only used to decrypt the ciphertext without interacting with the DM-master

13.6.8.4 Configuration Files

13.6.8.4.1 Data Migration Configuration File Overview

This document gives an overview of configuration files of DM (Data Migration).

DM process configuration files

- `dm-master.toml`: The configuration file of running the DM-master process, including the topology information and the logs of the DM-master. For more details, refer to DM-master Configuration File.
- `dm-worker.toml`: The configuration file of running the DM-worker process, including the topology information and the logs of the DM-worker. For more details, refer to DM-worker Configuration File.
- `source.yaml`: The configuration of the upstream database such as MySQL and MariaDB. For more details, refer to Upstream Database Configuration File.

DM migration task configuration

Data migration task creation

You can take the following steps to create a data migration task:

1. Load the data source configuration into the DM cluster using dmctl.
2. Refer to the description in the Task Configuration Guide and create the configuration file `your_task.yaml`.
3. Create the data migration task using dmctl.

Important concepts

This section shows description of some important concepts.

Concept	Description	Configuration File
`source-id`	Uniquely represents a MySQL or MariaDB instance, or a migration group with the primary-secondary structure. The maximum length of `source-id` is 32.	`source_id` of `source.yaml`; `source-id` of `task.yaml`
DM-master ID	Uniquely represents a DM-master (by the `master-addr` parameter of `dm-master.toml`)	`master-addr` of `dm-master.toml`
DM-worker ID	Uniquely represents a DM-worker (by the `worker-addr` parameter of `dm-worker.toml`)	`worker-addr` of `dm-worker.toml`

13.6.8.4.2 Upstream Database Configuration File of TiDB Data Migration

This document introduces the configuration file of the upstream database, including a configuration file template and the description of each configuration parameter in this file.

Configuration file template

The following is a configuration file template of the upstream database:

```
source-id: "mysql-replica-01"

##### Whether to enable GTID.
enable-gtid: false

##### Whether to enable relay log.
enable-relay: false
relay-binlog-name: ""  # The file name from which DM-worker starts to pull
    ↪ the binlog.
relay-binlog-gtid: ""  # The GTID from which DM-worker starts to pull the
    ↪ binlog.
##### relay-dir: "relay-dir" # The directory used to store relay log. The
    ↪ default value is "relay-dir". This configuration item is marked as
    ↪ deprecated since v6.1 and replaced by a parameter of the same name in
    ↪  the dm-worker configuration.


from:
  host: "127.0.0.1"
  port: 3306
  user: "root"
  password: "ZqMLjZ2j5khNelDEfDoUhkD5aV5fIJOe0fiog9w=" # The user password
      ↪ of the upstream database. It is recommended to use the password
      ↪ encrypted with dmctl.
  security:                   # The TLS configuration of the upstream
      ↪ database
    ssl-ca: "/path/to/ca.pem"
    ssl-cert: "/path/to/cert.pem"
    ssl-key: "/path/to/key.pem"

##### purge:
#####  interval: 3600
#####  expires: 0
#####  remain-space: 15

##### checker:
#####  check-enable: true
#####  backoff-rollback: 5m0s
#####  backoff-max: 5m0s    # The maximum value of backoff, should be larger
    ↪  than 1s

##### Configure binlog event filters. New in DM v2.0.2
```

```
##### case-sensitive: false
##### filters:
##### - schema-pattern: dmctl
#####  table-pattern: t_1
#####  events: []
#####  sql-pattern:
#####  - alter table .* add column `aaa` int
#####  action: Ignore
```

Note:

In DM v2.0.1, DO NOT set `enable-gtid` and `enable-relay` to `true` at the same time. Otherwise, it may cause loss of incremental data.

Configuration parameters

This section describes each configuration parameter in the configuration file.

Global configuration

Parameter	Description
`source-id`	Represents a MySQL instance ID.
`enable-gtid`	Determines whether to pull binlog from the upstream using GTID. The default value is `false`. In general, you do not need to configure `enable-gtid` manually. However, if GTID is enabled in the upstream database, and the primary/secondary switch is required, you need to set `enable-gtid` to `true`.
`enable-relay`	Determines whether to enable the relay log feature. The default value is `false`. This parameter takes effect from v5.4. Additionally, you can enable relay log dynamically using the `start-relay` command.

Parameter	Description
`relay-binlog-name`	Specifies the file name from which DM-worker starts to pull the binlog. For example, `"mysql-bin.000002"`. It only works when `enable_gtid` is `false`. If this parameter is not specified, DM-worker will start pulling from the earliest binlog file being replicated. Manual configuration is generally not required.
`relay-binlog-gtid`	Specifies the GTID from which DM-worker starts to pull the binlog. For example, `"e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849"`. It only works when `enable_gtid` is `true`. If this parameter is not specified, DM-worker will start pulling from the latest GTID being replicated. Manual configuration is generally not required.
`relay-dir`	Specifies the relay log directory.
`host`	Specifies the host of the upstream database.
`port`	Specifies the port of the upstream database.
`user`	Specifies the username of the upstream database.
`password`	Specifies the user password of the upstream database. It is recommended to use the password encrypted with dmctl.
`security`	Specifies the TLS config of the upstream database. The configured file paths of the certificates must be accessible to all nodes. If the configured file paths are local paths, then all the nodes in the cluster need to store a copy of the certificates in the same path of each host.

Relay log cleanup strategy configuration (`purge`)

Generally, there is no need to manually configure these parameters unless there is a large amount of relay logs and disk capacity is insufficient.

Parameter	Description	Default value
`interval`	Sets the time interval at which relay logs are regularly checked for expiration, in seconds.	`3600`

Parameter	Description	Default value
`expires`	Sets the expiration time for relay logs, in hours. The relay log that is not written by the relay processing unit, or does not need to be read by the existing data migration task will be deleted by DM if it exceeds the expiration time. If this parameter is not specified, the automatic purge is not performed.	`0`
`remain-space`	Sets the minimum amount of free disk space, in gigabytes. When the available disk space is smaller than this value, DM-worker tries to delete relay logs.	`15`

Note:

The automatic data purge strategy only takes effect when `interval` is not 0 and at least one of the two configuration items `expires` and `remain-space` is not 0.

Task status checker configuration (`checker`)

DM periodically checks the current task status and error message to determine if resuming the task will eliminate the error. If needed, DM automatically retries to resume the task. DM adjusts the checking interval using the exponential backoff strategy. Its behaviors can be adjusted by the following configuration.

Parameter	Description
`check-enable`	Whether to enable this feature.
`backoff-` ↪ `rollback`	If the current checking interval of backoff strategy is larger than this value and the task status is normal, DM will try to decrease the interval.
`backoff-max`	The maximum value of checking interval of backoff strategy, must be larger than 1 second.

Binlog event filter

Starting from DM v2.0.2, you can configure binlog event filters in the source configuration file.

Parameter	Description
`case-sensitive`	Determines whether the filtering rules are case-sensitive. The default value is `false`.
`filters`	Sets binlog event filtering rules. For details, see Binlog event filter parameter explanation.

13.6.8.4.3 DM Advanced Task Configuration File

This document introduces the advanced task configuration file of Data Migration (DM), including global configuration and instance configuration.

Important concepts

For description of important concepts including `source-id` and the DM-worker ID, see Important concepts.

Task configuration file template (advanced)

The following is the task configuration file template which allows you to perform **advanced** data migration tasks.

```
---

##### ----------- Global setting -----------
###### ********* Basic configuration *********
name: test                  # The name of the task. Should be globally
   ↪ unique.
task-mode: all              # The task mode. Can be set to `full`(only
   ↪ migrates full data)/`incremental`(replicates binlogs synchronously)/`
   ↪ all` (replicates both full data and incremental binlogs).
shard-mode: "pessimistic"   # The shard merge mode. Optional modes are ""/"
   ↪ pessimistic"/"optimistic". The "" mode is used by default which means
   ↪  sharding DDL merge is disabled. If the task is a shard merge task,
   ↪ set it to the "pessimistic" mode.
                            # After understanding the principles and
                               ↪ restrictions of the "optimistic" mode,
                               ↪ you can set it to the "optimistic" mode.
meta-schema: "dm_meta"      # The downstream database that stores the `meta`
   ↪  information.
timezone: "Asia/Shanghai"   # The timezone used in SQL Session. By default,
   ↪ DM uses the global timezone setting in the target cluster, which
   ↪ ensures the correctness automatically. A customized timezone does not
   ↪  affect data migration but is unnecessary.
case-sensitive: false       # Determines whether the schema/table is case-
   ↪ sensitive.
```

```
online-ddl: true            # Supports automatic processing of upstream "gh-
    ↪ ost" and "pt".
online-ddl-scheme: "gh-ost" # `online-ddl-scheme` is deprecated, so it is
    ↪ recommended to use `online-ddl`.
clean-dump-file: true       # Whether to clean up the files generated during
    ↪  data dump. Note that these include `metadata` files.
collation_compatible: "loose" # The mode to sync the default collation in `
    ↪ CREATE` SQL statements. The supported values are "loose" (by default)
    ↪  or "strict". When the value is "strict", DM explicitly appends the
    ↪ corresponding collation of the upstream to the SQL statements; when
    ↪ the value is "loose", DM does not modify the SQL statements. In "
    ↪ strict" mode, if the downstream does not support the default
    ↪ collation in the upstream, the downstream might report an error.
ignore-checking-items: []   # Ignorable checking items. For the complete
    ↪ list of ignorable checking items, see DM precheck: https://docs.
    ↪ pingcap.com/tidb/stable/dm-precheck#ignorable-checking-items.

target-database:            # Configuration of the downstream database
    ↪ instance.
  host: "192.168.0.1"
  port: 4000
  user: "root"
  password: "/Q7B9DizNLLTTfiZHv9WoEAKamfpIUs=" # It is recommended to use a
     ↪ password encrypted with `dmctl encrypt`.
  max-allowed-packet: 67108864            # Sets the "max_allowed_packet"
     ↪ limit of the TiDB client (that is, the limit of the maximum accepted
     ↪  packet) when DM internally connects to the TiDB server. The unit is
     ↪  bytes. (67108864 by default)
                                          # Since DM v2.0.0, this
                                             ↪ configuration item is
                                             ↪ deprecated, and DM
                                             ↪ automatically obtains the "
                                             ↪ max_allowed_packet" value
                                             ↪ from TiDB.
  session:                                 # The session variables of TiDB,
     ↪ supported since v1.0.6. For details, go to `https://pingcap.com/docs
     ↪ /stable/system-variables`.
    sql_mode: "ANSI_QUOTES,NO_ZERO_IN_DATE,NO_ZERO_DATE" # Since DM v2.0.0,
       ↪ if this item does not appear in the configuration file, DM
       ↪ automatically fetches a proper value for "sql_mode" from the
       ↪ downstream TiDB. Manual configuration of this item has a higher
       ↪ priority.
    tidb_skip_utf8_check: 1               # Since DM v2.0.0, if this item
       ↪ does not appear in the configuration file, DM automatically
       ↪ fetches a proper value for "tidb_skip_utf8_check" from the
```

```
      ↪ downstream TiDB. Manual configuration of this item has a higher
      ↪ priority.
    tidb_constraint_check_in_place: 0
  security:                    # The TLS configuration of the downstream TiDB
    ssl-ca: "/path/to/ca.pem"
    ssl-cert: "/path/to/cert.pem"
    ssl-key: "/path/to/key.pem"


###### ******** Feature configuration set **********
##### The routing mapping rule set between the upstream and downstream
    ↪ tables.
routes:
  route-rule-1:              # The name of the routing mapping rule.
    schema-pattern: "test_*" # The pattern of the upstream schema name,
      ↪ wildcard characters (*?) are supported.
    table-pattern: "t_*"     # The pattern of the upstream table name,
      ↪ wildcard characters (*?) are supported.
    target-schema: "test"    # The name of the downstream schema.
    target-table: "t"        # The name of the downstream table.
  route-rule-2:
    schema-pattern: "test_*"
    target-schema: "test"

##### The binlog event filter rule set of the matched table of the upstream
    ↪ database instance.
filters:
  filter-rule-1:                          # The name of the filtering rule.
    schema-pattern: "test_*"              # The pattern of the upstream
      ↪ schema name, wildcard characters (*?) are supported.
    table-pattern: "t_*"                  # The pattern of the upstream
      ↪ schema name, wildcard characters (*?) are supported.
    events: ["truncate table", "drop table"] # What event types to match.
    action: Ignore                        # Whether to migrate (Do) or
      ↪ ignore (Ignore) the binlog that matches the filtering rule.
  filter-rule-2:
    schema-pattern: "test_*"
    events: ["all dml"]
    action: Do

expression-filter:               # Defines the filter rules for row changes
    ↪ when migrating data. Supports defining multiple rules.
  # Filter the value of inserted `c` in `expr_filter`.`tbl` when it is even.
  even_c:                        # The name of the filter rule.
    schema: "expr_filter"        # The name of upstream database to be
```

```
      ↪ matched. Wildcard match or regular match is not supported.
    table: "tbl"                # The name of upstream table to be matched.
      ↪  Wildcard match or regular match is not supported.
    insert-value-expr: "c % 2 = 0"

##### The filter rule set of tables to be migrated from the upstream
   ↪ database instance. You can set multiple rules at the same time.
block-allow-list:               # Use black-white-list if the DM version is
   ↪  earlier than or equal to v2.0.0-beta.2.
  bw-rule-1:                    # The name of the block allow list rule.
    do-dbs: ["~^test.*", "user"] # The allow list of upstream schemas needs
      ↪ to be migrated.
    ignore-dbs: ["mysql", "account"] # The block list of upstream schemas
      ↪ needs to be migrated.
    do-tables:                  # The allow list of upstream tables needs
      ↪ to be migrated.
    - db-name: "~^test.*"
      tbl-name: "~^t.*"
    - db-name: "user"
      tbl-name: "information"
  bw-rule-2:                    # The name of the block allow list rule.
    ignore-tables:              # The block list of upstream tables needs
      ↪ to be migrated.
    - db-name: "user"
      tbl-name: "log"

##### Configuration arguments of the dump processing unit.
mydumpers:
  global:                       # The configuration name of the processing
     ↪ unit.
    threads: 4                  # The number of threads that access the
      ↪ upstream when the dump processing unit performs the precheck and
      ↪ exports data from the upstream database (4 by default)
    chunk-filesize: 64          # The size of the file generated by the
      ↪ dump processing unit (64 MB by default).
    extra-args: "--consistency none" # Other arguments of the dump
      ↪ processing unit. You do not need to manually configure table-list
      ↪ in `extra-args`, because it is automatically generated by DM.

##### Configuration arguments of the load processing unit.
loaders:
  global:                       # The configuration name of the processing
     ↪ unit.
    pool-size: 16               # The number of threads that concurrently
      ↪ execute dumped SQL files in the load processing unit (16 by
```

```
      ↪ default). When multiple instances are migrating data to TiDB at
      ↪ the same time, slightly reduce the value according to the load.
   # The directory that stores full data exported from the upstream ("./
      ↪ dumped_data" by default).
   # Supoprts a local filesystem path or an Amazon S3 path. For example, "
      ↪ s3://dm_bucket/dumped_data?region=us-west-2&endpoint=s3-website.us
      ↪ -east-2.amazonaws.com&access_key=s3accesskey&secret_access_key=
      ↪ s3secretkey&force_path_style=true"
   dir: "./dumped_data"
   # The import mode during the full import phase. In most cases you don't
      ↪ need to care about this configuration.
   # - "sql" (default). Use [TiDB Lightning](#tidb-lightning-overview) TiDB
      ↪ -backend mode to import data.
   # - "loader". Use Loader mode to import data. This mode is only for
      ↪ compatibility with features that TiDB Lightning does not support
      ↪ yet. It will be deprecated in the future.
   import-mode: "sql"
   #  Methods to resolve conflicts during the full import phase. You can
      ↪ set it to the following:
   # - "replace" (default). Only supports the import mode "sql". In this
      ↪ method, it uses the new data to replace the existing data.
   # - "ignore". Only supports the import mode "sql". It keeps the existing
      ↪  data, and ignores the new data.
   # - "error". Only supports the import mode "loader". It reports errors
      ↪ when inserting duplicated data, and then stops the replication
      ↪ task.
   on-duplicate: "replace"

##### Configuration arguments of the sync processing unit.
syncers:
  global:                       # The configuration name of the processing
     ↪ unit.
   worker-count: 16             # The number of concurrent threads that
      ↪ apply binlogs which have been transferred to the local (16 by
      ↪ default). Adjusting this parameter has no effect on the
      ↪ concurrency of upstream pull logs, but has a significant effect on
      ↪  the downstream database.
   batch: 100                   # The number of SQL statements in a
      ↪ transaction batch that the sync processing unit replicates to the
      ↪ downstream database (100 by default). Generally, it is recommended
      ↪  to set the value less than 500.
   enable-ansi-quotes: true     # Enable this argument if `sql-mode: "
      ↪ ANSI_QUOTES"` is set in the `session`

   # If set to true, `INSERT` statements from upstream are rewritten to `
```

```
      ↪ REPLACE` statements, and `UPDATE` statements are rewritten to `
      ↪ DELETE` and `REPLACE` statements. This ensures that DML statements
      ↪  can be imported repeatedly during data migration when there is
      ↪ any primary key or unique index in the table schema.
    safe-mode: false
    # If set to true, DM compacts as many upstream statements on the same
      ↪ rows as possible into a single statements without increasing
      ↪ latency.
    # For example, `INSERT INTO tb(a,b) VALUES(1,1); UPDATE tb SET b=11
      ↪ WHERE a=1`;` will be compacted to `INSERT INTO tb(a,b) VALUES
      ↪ (1,11);`, where "a" is the primary key
    # `UPDATE tb SET b=1 WHERE a=1; UPDATE tb(a,b) SET b=2 WHERE a=1;` will
      ↪ be compacted to `UPDATE tb(a,b) SET b=2 WHERE a=1;`, where "a" is
      ↪ the primary key
    # `DELETE FROM tb WHERE a=1; INSERT INTO tb(a,b) VALUES(1,1);` will be
      ↪ compacted to `REPLACE INTO tb(a,b) VALUES(1,1);`, where "a" is the
      ↪  primary key
    compact: false
    # If set to true, DM combines as many statements of the same type as
      ↪ possible into a single statement and generates a single SQL
      ↪ statement with multiple rows of data.
    # For example, `INSERT INTO tb(a,b) VALUES(1,1); INSERT INTO tb(a,b)
      ↪ VALUES(2,2);` will become `INSERT INTO tb(a,b) VALUES(1,1),(2,2);`
    # `UPDATE tb SET b=11 WHERE a=1; UPDATE tb(a,b) set b=22 WHERE a=2;`
      ↪ will become `INSERT INTO tb(a,b) VALUES(1,11),(2,22) ON DUPLICATE
      ↪ KEY UPDATE a=VALUES(a), b= VALUES(b);`, where "a" is the primary
      ↪ key
    # `DELETE FROM tb WHERE a=1; DELETE FROM tb WHERE a=2` will become `
      ↪ DELETE FROM tb WHERE (a) IN (1),(2)`, where "a" is the primary key
    multiple-rows: true

##### ----------- Instance configuration -----------
mysql-instances:
  -
    source-id: "mysql-replica-01"             # The `source-id` in source.
      ↪ toml.
    meta:                                     # The position where the
      ↪ binlog replication starts when `task-mode` is `incremental` and
      ↪ the downstream database checkpoint does not exist. If the
      ↪ checkpoint exists, the checkpoint is used.

      binlog-name: binlog.000001
      binlog-pos: 4
      binlog-gtid: "03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474d3c
        ↪ -28c7-11e7-8352-203db246dd3d:1-170" # You need to set this
```

```
      ↪ argument if you specify `enable-gtid: true` for the source of
      ↪ the incremental task.

  route-rules: ["route-rule-1", "route-rule-2"] # The name of the mapping
     ↪ rule between the table matching the upstream database instance and
     ↪  the downstream database.
  filter-rules: ["filter-rule-1", "filter-rule-2"]         # The name of
     ↪ the binlog event filtering rule of the table matching the upstream
     ↪  database instance.
  block-allow-list: "bw-rule-1"             # The name of the block and
     ↪ allow lists filtering rule of the table matching the upstream
     ↪ database instance. Use black-white-list if the DM version is
     ↪ earlier than or equal to v2.0.0-beta.2.
  expression-filters: ["even_c"]            # Use expression filter rule
     ↪ named even_c.
  mydumper-config-name: "global"            # The name of the mydumpers
     ↪ configuration.
  loader-config-name: "global"              # The name of the loaders
     ↪ configuration.
  syncer-config-name: "global"              # The name of the syncers
     ↪ configuration.

-
  source-id: "mysql-replica-02"             # The `source-id` in source.
     ↪ toml.
  mydumper-thread: 4                        # The number of threads that
     ↪ the dump processing unit uses for dumping data. `mydumper-thread`
     ↪ corresponds to the `threads` configuration item of the mydumpers
     ↪ configuration. `mydumper-thread` has overriding priority when the
     ↪ two items are both configured.
  loader-thread: 16                         # The number of threads that
     ↪ the load processing unit uses for loading data. `loader-thread`
     ↪ corresponds to the `pool-size` configuration item of the loaders
     ↪ configuration. `loader-thread` has overriding priority when the
     ↪ two items are both configured. When multiple instances are
     ↪ migrating data to TiDB at the same time, reduce the value
     ↪ according to the load.
  syncer-thread: 16                         # The number of threads that
     ↪ the sync processing unit uses for replicating incremental data. `
     ↪ syncer-thread` corresponds to the `worker-count` configuration
     ↪ item of the syncers configuration. `syncer-thread` has overriding
     ↪ priority when the two items are both configured. When multiple
     ↪ instances are migrating data to TiDB at the same time, reduce the
     ↪ value according to the load.
```

Configuration order

1. Edit the global configuration.
2. Edit the instance configuration based on the global configuration.

Global configuration

Basic configuration

Refer to the comments in the template to see more details. Detailed explanations about `task-mode` are as follows:

- Description: the task mode that can be used to specify the data migration task to be executed.
- Value: string (`full`, `incremental`, or `all`).
    - `full` only makes a full backup of the upstream database and then imports the full data to the downstream database.
    - `incremental`: Only replicates the incremental data of the upstream database to the downstream database using the binlog. You can set the `meta` configuration item of the instance configuration to specify the starting position of incremental replication.
    - `all`: `full` + `incremental`. Makes a full backup of the upstream database, imports the full data to the downstream database, and then uses the binlog to make an incremental replication to the downstream database starting from the exported position during the full backup process (binlog position).

Feature configuration set

Arguments in each feature configuration set are explained in the comments in the template.

Parameter	Description
`routes`	The routing mapping rule set between the upstream and downstream tables. If the names of the upstream and downstream schemas and tables are the same, this item does not need to be configured. See Table Routing for usage scenarios and sample configurations.
`filters`	The binlog event filter rule set of the matched table of the upstream database instance. If binlog filtering is not required, this item does not need to be configured. See Binlog Event Filter for usage scenarios and sample configurations.

Parameter	Description
`block-allow-list`	The filter rule set of the block allow list of the matched table of the upstream database instance. It is recommended to specify the schemas and tables that need to be migrated through this item, otherwise all schemas and tables are migrated. See Binlog Event Filter and Block & Allow Lists for usage scenarios and sample configurations.
`mydumpers`	Configuration arguments of dump processing unit. If the default configuration is sufficient for your needs, this item does not need to be configured. Or you can configure `thread` only using `mydumper-thread`.
`loaders`	Configuration arguments of load processing unit. If the default configuration is sufficient for your needs, this item does not need to be configured. Or you can configure `pool-size` only using `loader-thread`.
`syncers`	Configuration arguments of sync processing unit. If the default configuration is sufficient for your needs, this item does not need to be configured. Or you can configure `worker-count` only using `syncer-thread`.

Instance configuration

This part defines the subtask of data migration. DM supports migrating data from one or multiple MySQL instances in the upstream to the same instance in the downstream.

For the configuration details of the above options, see the corresponding part in Feature configuration set, as shown in the following table.

Option	Corresponding part
`route-rules`	`routes`
`filter-rules`	`filters`
`block-allow-list`	`block-allow-list`
`mydumper-config-name`	`mydumpers`
`loader-config-name`	`loaders`
`syncer-config-name`	`syncers`

13.6.8.4.4 DM-master Configuration File

This document introduces the configuration of DM-master, including a configuration file template and a description of each configuration parameter in this file.

Configuration file template

The following is a configuration file template of DM-master.

```
name = "dm-master"

##### log configuration
log-level = "info"
log-file = "dm-master.log"

##### DM-master listening address
master-addr = ":8261"
advertise-addr = "127.0.0.1:8261"

##### URLs for peer traffic
peer-urls = "http://127.0.0.1:8291"
advertise-peer-urls = "http://127.0.0.1:8291"

##### cluster configuration
initial-cluster = "master1=http://127.0.0.1:8291,master2=http
    ↪ ://127.0.0.1:8292,master3=http://127.0.0.1:8293"
join = ""

ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/cert.pem"
ssl-key = "/path/to/key.pem"
cert-allowed-cn = ["dm"]
```

Configuration parameters

This section introduces the configuration parameters of DM-master.

Global configuration

Parameter	Description
`name`	The name of the DM-master.
`log-level`	Specifies a log level from `debug`, `info`, `warn`, `error`, and `fatal`. The default log level is `info`.
`log-file`	Specifies the log file directory. If the parameter is not specified, the logs are printed onto the standard output.

Parameter	Description
`master-addr`	Specifies the address of DM-master which provides services. You can omit the IP address and specify the port number only, such as ":8261".
`advertise-addr`	Specifies the address that DM-master advertises to the outside world.
`peer-urls`	Specifies the peer URL of the DM-master node.
`advertise-peer-urls`	Specifies the peer URL that DM-master advertises to the outside world. The value of `advertise-peer-urls` is by default the same as that of `peer-urls`.
`initial-cluster`	The value of `initial-cluster` is the combination of the `advertise-peer-urls` value of all DM-master nodes in the initial cluster.
`join`	The value of `join` is the combination of the `advertise-peer-urls` value of the existed DM-master nodes in the cluster. If the DM-master node is newly added, replace `initial-cluster` with `join`.
`ssl-ca`	The path of the file that contains list of trusted SSL CAs for DM-master to connect with other components.
`ssl-cert`	The path of the file that contains X509 certificate in PEM format for DM-master to connect with other components.
`ssl-key`	The path of the file that contains X509 key in PEM format for DM-master to connect with other components.
`cert-allowed-cn`	Common Name list.

13.6.8.4.5 DM-worker Configuration File

This document introduces the configuration of DM worker, including a configuration file template and a description of each configuration parameter in this file.

Configuration file template

The following is a configuration file template of the DM-worker:

```
##### Worker Configuration.
name = "worker1"
```

```
##### Log configuration.
log-level = "info"
log-file = "dm-worker.log"

##### DM-worker listen address.
worker-addr = ":8262"
advertise-addr = "127.0.0.1:8262"
join = "http://127.0.0.1:8261,http://127.0.0.1:8361,http://127.0.0.1:8461"

keepalive-ttl = 60
relay-keepalive-ttl = 1800 # New in DM v2.0.2.
##### relay-dir = "relay_log" # New in 5.4.0. When you use a relative path,
    ↪ check the deployment and start method of DM-worker to determine the
    ↪ full path.

ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/cert.pem"
ssl-key = "/path/to/key.pem"
cert-allowed-cn = ["dm"]
```

Configuration parameters

Global

Parameter	Description
`name`	The name of the DM-worker.
`log-level`	Specifies a log level from `debug`, `info`, `warn`, `error`, and `fatal`. The default log level is `info`.
`log-file`	Specifies the log file directory. If this parameter is not specified, the logs are printed onto the standard output.
`worker-addr`	Specifies the address of DM-worker which provides services. You can omit the IP address and specify the port number only, such as ":8262".
`advertise-addr`	Specifies the address that DM-worker advertises to the outside world.
`join`	Corresponds to one or more `master-addrs` in the DM-master configuration file.
`keepalive-ttl`	The keepalive time (in seconds) of a DM-worker node to the DM-master node if the upstream data source of the DM-worker node does not enable the relay log. The default value is 60s.

Parameter	Description
`relay-keepalive-ttl`	The keepalive time (in seconds) of a DM-worker node to the DM-master node if the upstream data source of the DM-worker node enables the relay log. The default value is 1800s. This parameter is added since DM v2.0.2.
`relay-dir`	When relay log is enabled in the bound upstream data source, DM-worker stores the relay log in this directory. This parameter is new in v5.4.0 and takes precedence over the configuration of the upstream data source.
`ssl-ca`	The path of the file that contains list of trusted SSL CAs for DM-worker to connect with other components.
`ssl-cert`	The path of the file that contains X509 certificate in PEM format for DM-worker to connect with other components.
`ssl-key`	The path of the file that contains X509 key in PEM format for DM-worker to connect with other components.
`cert-allowed-cn`	Common Name list.

13.6.8.4.6 Table Selector of TiDB Data Migration

Table selector provides a match rule based on wildcard characters for schema/table. To match a specified table, configure `schema-pattern/table-pattern`.

Wildcard character

Table selector uses the following two wildcard characters in `schema-pattern/table-pattern`:

- The asterisk character (`*`, also called "star")
  - `*` matches zero or more characters. For example, `doc*` matches `doc` and `document` but not `dodo`.
  - `*` can only be placed at the end of the word. For example, `doc*` is supported, while `do*c` is not supported.
- The question mark (`?`)

  `?` matches exactly one character except the empty character.

Match rules

- `schema-pattern` cannot be empty.
- `table-pattern` can be empty. When you configure it as empty, only `schema` is matched according to `schema-pattern`.
- When `table-pattern` is not empty, the `schema` is matched according to `schema-pattern` and `table` is matched according to `table-pattern`. Only when both `schema` and `table` are successfully matched, you can get the match result.

Usage examples

- Matching all schemas and tables that have a `schema_` prefix in the schema name:

```
schema-pattern: "schema_*"
table-pattern: ""
```

- Matching all tables that have a `schema_` prefix in the schema name and a `table_` prefix in the table name:

```
schema-pattern = "schema_*"
table-pattern = "table_*"
```

13.6.8.5 Maintain DM Clusters Using OpenAPI

DM provides the OpenAPI feature for easily querying and operating the DM cluster, which is similar to the feature of dmctl tools.

To enable OpenAPI, perform one of the following operations:

- If your DM cluster has been deployed directly using binary, add the following configuration to the DM-master configuration file.

```
openapi = true
```

- If your DM cluster has been deployed using TiUP, add the following configuration to the topology file:

```
server_configs:
  master:
    openapi: true
```

Note:

- DM provides the specification document that meets the OpenAPI 3.0.0 standard. This document contains all the request parameters and returned values. You can copy the document yaml and preview it in Swagger Editor.
- After you deploy the DM-master nodes, you can access `http://{master-addr}/api/v1/docs` to preview the documentation online.

You can use the APIs to perform the following maintenance operations on the DM cluster:

13.6.8.5.1 APIs for managing clusters

- Get the information of a DM-master node
- Stop a DM-master node
- Get the information of a DM-worker node
- Stop a DM-worker node

13.6.8.5.2 APIs for managing data sources

- Create a data source
- Get a data source
- Delete the data source
- Update a data source
- Enable a data source
- Disable a data source
- Get the information of a data source
- Get the data source list
- Start the relay-log feature for data sources
- Stop the relay-log feature for data sources
- Purge relay-log files that are no longer required
- Change the bindings between the data source and DM-workers
- Get the list of schema names of a data source
- Get the list of table names of a specified schema in a data source

13.6.8.5.3 APIs for managing replication tasks

- Create a replication task
- Get a replication task
- Delete a replication task
- Update a replication task

- Start a replication task
- Stop a replication task
- Get the information of a replication task
- Get the replication task list
- Get the migration rules of a replication task
- Get the list of schema names of the data source that is associated with a replication task
- Get the list of table names of a specified schema in the data source that is associated with a replication task
- Get the CREATE statement for schemas of the data source that is associated with a replication task
- Update the CREATE statement for schemas of the data source that is associated with a replication task
- Delete a schema of the data source that is associated with a replication task

The following sections describe the specific usage of the APIs.

13.6.8.5.4 API error message template

After sending an API request, if an error occurs, the returned error message is in the following format:

```
{
    "error_msg": "",
    "error_code": ""
}
```

From the above JSON output, `error_msg` describes the error message and `error_code` is the corresponding error code.

13.6.8.5.5 Get the information of a DM-master node

This API is a synchronous interface. If the request is successful, the information of the corresponding node is returned.

Request URI

`GET /api/v1/cluster/masters`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/cluster/masters' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
```

```
    {
      "name": "master1",
      "alive": true,
      "leader": true,
      "addr": "127.0.0.1:8261"
    }
  ]
}
```

13.6.8.5.6 Stop a DM-master node

This API is a synchronous interface. If the request is successful, the status code of the returned body is 204.

Request URI

`DELETE /api/v1/cluster/masters/{master-name}`

Example

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/cluster/masters/master1' \
  -H 'accept: */*'
```

13.6.8.5.7 Get the information of a DM-worker node

This API is a synchronous interface. If the request is successful, the information of the corresponding node is returned.

Request URI

`GET /api/v1/cluster/workers`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/cluster/workers' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
    {
      "name": "worker1",
      "addr": "127.0.0.1:8261",
      "bound_stage": "bound",
      "bound_source_name": "mysql-01"
    }
```

```
  ]
}
```

13.6.8.5.8 Stop a DM-worker node

This API is a synchronous interface. If the request is successful, the status code of the returned body is 204.

Request URI

`DELETE /api/v1/cluster/workers/{worker-name}`

Example

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/cluster/workers/worker1' \
  -H 'accept: */*'
```

13.6.8.5.9 Create a data source

This API is a synchronous interface. If the request is successful, the information of the corresponding data source is returned.

Request URI

`POST /api/v1/sources`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "source_name": "mysql-01",
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "enable": true,
  "enable_gtid": false,
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  },
```

```
  "purge": {
    "interval": 3600,
    "expires": 0,
    "remain_space": 15
  }
}'
```

```
{
  "source_name": "mysql-01",
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "enable": true,
  "enable_gtid": false,
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  },
  "purge": {
    "interval": 3600,
    "expires": 0,
    "remain_space": 15
  },
  "status_list": [
    {
      "source_name": "mysql-replica-01",
      "worker_name": "worker-1",
      "relay_status": {
        "master_binlog": "(mysql-bin.000001, 1979)",
        "master_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_dir": "./sub_dir",
        "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_catch_up_master": true,
        "stage": "Running"
      },
      "error_msg": "string"
    }
  ]
}
```

13.6.8.5.10 Get a data source

This API is a synchronous interface. If the request is successful, the information of the corresponding data source is returned.

Request URI

`GET /api/v1/sources/{source-name}`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources/source-1?with_status=true' \
  -H 'accept: application/json'
```

```
{
  "source_name": "mysql-01",
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "enable_gtid": false,
  "enable": false,
  "flavor": "mysql",
  "task_name_list": [
    "task1"
  ],
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  },
  "purge": {
    "interval": 3600,
    "expires": 0,
    "remain_space": 15
  },
  "status_list": [
    {
      "source_name": "mysql-replica-01",
      "worker_name": "worker-1",
      "relay_status": {
        "master_binlog": "(mysql-bin.000001, 1979)",
        "master_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_dir": "./sub_dir",
```

```
        "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_catch_up_master": true,
        "stage": "Running"
      },
      "error_msg": "string"
    }
  ],
  "relay_config": {
    "enable_relay": true,
    "relay_binlog_name": "mysql-bin.000002",
    "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
    "relay_dir": "./relay_log"
  }
}
```

13.6.8.5.11 Delete the data source

This API is a synchronous interface. If the request is successful, the status code of the returned body is 204.

Request URI

`DELETE /api/v1/sources/{source-name}`

Example

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01?force=true' \
  -H 'accept: application/json'
```

13.6.8.5.12 Update a data source

This API is a synchronous interface. If the request is successful, the information of the corresponding data source is returned.

> **Note:**
>
> When you use this API to update the data source configuration, make sure that there are no running tasks under the current data source.

Request URI

`PUT /api/v1/sources/{source-name}`

Example

```
curl -X 'PUT' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "source": {
    "source_name": "mysql-01",
    "host": "127.0.0.1",
    "port": 3306,
    "user": "root",
    "password": "123456",
    "enable_gtid": false,
    "enable": false,
    "flavor": "mysql",
    "task_name_list": [
      "task1"
    ],
    "security": {
      "ssl_ca_content": "",
      "ssl_cert_content": "",
      "ssl_key_content": "",
      "cert_allowed_cn": [
        "string"
      ]
    },
    "purge": {
      "interval": 3600,
      "expires": 0,
      "remain_space": 15
    },
    "relay_config": {
      "enable_relay": true,
      "relay_binlog_name": "mysql-bin.000002",
      "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
      "relay_dir": "./relay_log"
    }
  }
}'
```

```
{
  "source_name": "mysql-01",
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
```

```
  "password": "123456",
  "enable": true,
  "enable_gtid": false,
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  },
  "purge": {
    "interval": 3600,
    "expires": 0,
    "remain_space": 15
  }
}
```

13.6.8.5.13 Enable a data source

This is a synchronous interface that enables a data source on a successful request and starts all subtasks of the task that rely on this data source in batch.

Request URI

`POST /api/v1/sources/{source-name}/enable`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/enable' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json'
```

13.6.8.5.14 Disable a data source

This is a synchronous interface that deactivates this data source on a successful request and stops all subtasks of the task that rely on it in batch.

Request URI

`POST /api/v1/sources/{source-name}/disable`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/disable' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json'
```

13.6.8.5.15 Get the data source list

This API is a synchronous interface. If the request is successful, the data source list is returned.

Request URI

`GET /api/v1/sources`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources?with_status=true' \
  -H 'accept: application/json'
```


```
{
  "data": [
    {
      "enable_gtid": false,
      "host": "127.0.0.1",
      "password": "******",
      "port": 3306,
      "purge": {
        "expires": 0,
        "interval": 3600,
        "remain_space": 15
      },
      "security": null,
      "source_name": "mysql-01",
      "user": "root"
    },
    {
      "enable_gtid": false,
      "host": "127.0.0.1",
      "password": "******",
      "port": 3307,
      "purge": {
        "expires": 0,
        "interval": 3600,
        "remain_space": 15
      },
      "security": null,
      "source_name": "mysql-02",
      "user": "root"
    }
  ],
  "total": 2
}
```

13.6.8.5.16 Get the information of a data source

This API is a synchronous interface. If the request is successful, the information of the corresponding node is returned.

Request URI

`GET /api/v1/sources/{source-name}/status`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-replica-01/status' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
    {
      "source_name": "mysql-replica-01",
      "worker_name": "worker-1",
      "relay_status": {
        "master_binlog": "(mysql-bin.000001, 1979)",
        "master_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_dir": "./sub_dir",
        "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
        "relay_catch_up_master": true,
        "stage": "Running"
      },
      "error_msg": "string"
    }
  ]
}
```

13.6.8.5.17 Start the relay-log feature for data sources

This API is an asynchronous interface. If the request is successful, the status code of the returned body is 200. To learn about its latest status, You can get the information of a data source.

Request URI

`POST /api/v1/sources/{source-name}/relay/enable`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/relay/enable' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
```

```
  -d '{
  "worker_name_list": [
    "worker-1"
  ],
  "relay_binlog_name": "mysql-bin.000002",
  "relay_binlog_gtid": "e9a1fc22-ec08-11e9-b2ac-0242ac110003:1-7849",
  "relay_dir": "./relay_log"
}'
```

13.6.8.5.18 Stop the relay-log feature for data sources

This API is an asynchronous interface. If the request is successful, the status code of the returned body is 200. To learn about its latest status, You can get the information of a data source.

Request URI

`POST /api/v1/sources/{source-name}/relay/disable`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/relay/disable' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "worker_name_list": [
    "worker-1"
  ]
}'
```

13.6.8.5.19 Purge relay log files that are no longer required

This API is an asynchronous interface. If the request is successful, the status code of the returned body is 200. To learn about its latest status, You can get the information of a data source.

Request URI

`POST /api/v1/sources/{source-name}/relay/purge`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/relay/purge' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "relay_binlog_name": "mysql-bin.000002",
```

```
  "relay_dir": "string"
}'
```

13.6.8.5.20 Change the bindings between the data source and DM-workers

This API is an asynchronous interface. If the request is successful, the status code of the returned body is 200. To learn about its latest status, You can get the information of a DM-worker node.

Request URI

`POST /api/v1/sources/{source-name}/transfer`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/sources/mysql-01/transfer' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "worker_name": "worker-1"
}'
```

13.6.8.5.21 Get the list of schema names of a data source

This API is a synchronous interface. If the request is successful, the corresponding list is returned.

Request URI

`GET /api/v1/sources/{source-name}/schemas`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources/source-1/schemas' \
  -H 'accept: application/json'
```

```
[
  "db1"
]
```

13.6.8.5.22 Get the list of table names of a specified schema in a data source

This API is a synchronous interface. If the request is successful, the corresponding list is returned.

Request URI

`GET /api/v1/sources/{source-name}/schemas/{schema-name}`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/sources/source-1/schemas/db1' \
  -H 'accept: application/json'
```

```
[
  "table1"
]
```

13.6.8.5.23 Create a replication task

This API is a synchronous interface. If the request is successful, the status code of the returned body is 200. A successful request will return the information of the corresponding replication task.

Request URI

`POST /api/v1/tasks`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/tasks' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task": {
    "name": "task-1",
    "task_mode": "all",
    "shard_mode": "pessimistic",
    "meta_schema": "dm-meta",
    "enhance_online_schema_change": true,
    "on_duplicate": "overwrite",
    "target_config": {
      "host": "127.0.0.1",
      "port": 3306,
      "user": "root",
      "password": "123456",
      "security": {
        "ssl_ca_content": "",
        "ssl_cert_content": "",
        "ssl_key_content": "",
        "cert_allowed_cn": [
```

```
      "string"
    ]
  }
},
"binlog_filter_rule": {
  "rule-1": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-2": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-3": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  }
},
"table_migrate_rule": [
  {
    "source": {
      "source_name": "source-name",
      "schema": "db-*",
      "table": "tb-*"
    },
    "target": {
      "schema": "db1",
      "table": "tb1"
    },
    "binlog_filter_rule": [
      "rule-1",
      "rule-2",
      "rule-3",
```

```
      ]
    }
  ],
  "source_config": {
    "full_migrate_conf": {
      "export_threads": 4,
      "import_threads": 16,
      "data_dir": "./exported_data",
      "consistency": "auto"
    },
    "incr_migrate_conf": {
      "repl_threads": 16,
      "repl_batch": 100
    },
    "source_conf": [
      {
        "source_name": "mysql-replica-01",
        "binlog_name": "binlog.000001",
        "binlog_pos": 4,
        "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30
            ↪ :1-7041423,05474d3c-28c7-11e7-8352-203db246dd3d:1-170"
      }
    ]
  }
 }
}'
```

```
{
  "name": "task-1",
  "task_mode": "all",
  "shard_mode": "pessimistic",
  "meta_schema": "dm-meta",
  "enhance_online_schema_change": true,
  "on_duplicate": "overwrite",
  "target_config": {
    "host": "127.0.0.1",
    "port": 3306,
    "user": "root",
    "password": "123456",
    "security": {
      "ssl_ca_content": "",
      "ssl_cert_content": "",
      "ssl_key_content": "",
      "cert_allowed_cn": [
        "string"
```

```
    ]
  }
},
"binlog_filter_rule": {
  "rule-1": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-2": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-3": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  }
},
"table_migrate_rule": [
  {
    "source": {
      "source_name": "source-name",
      "schema": "db-*",
      "table": "tb-*"
    },
    "target": {
      "schema": "db1",
      "table": "tb1"
    },
    "binlog_filter_rule": [
      "rule-1",
      "rule-2",
      "rule-3",
    ]
```

```
    }
  ],
  "source_config": {
    "full_migrate_conf": {
      "export_threads": 4,
      "import_threads": 16,
      "data_dir": "./exported_data",
      "consistency": "auto"
    },
    "incr_migrate_conf": {
      "repl_threads": 16,
      "repl_batch": 100
    },
    "source_conf": [
      {
        "source_name": "mysql-replica-01",
        "binlog_name": "binlog.000001",
        "binlog_pos": 4,
        "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474
            ↪ d3c-28c7-11e7-8352-203db246dd3d:1-170"
      }
    ]
  }
}
```

13.6.8.5.24 Get a replication task

This API is a synchronous interface. If the request is successful, the information of the corresponding replication task is returned.

Request URI

`GET /api/v1/tasks/{task-name}?with_status=true`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1?with_status=true' \
  -H 'accept: application/json'
```

```
{
  "name": "task-1",
  "task_mode": "all",
  "shard_mode": "pessimistic",
  "meta_schema": "dm-meta",
  "enhance_online_schema_change": true,
  "on_duplicate": "overwrite",
```

```
"target_config": {
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  }
},
"binlog_filter_rule": {
  "rule-1": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-2": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-3": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  }
},
"table_migrate_rule": [
  {
    "source": {
      "source_name": "source-name",
      "schema": "db-*",
```

```
      "table": "tb-*"
    },
    "target": {
      "schema": "db1",
      "table": "tb1"
    },
    "binlog_filter_rule": [
      "rule-1",
      "rule-2",
      "rule-3",
    ]
  }
 ],
 "source_config": {
  "full_migrate_conf": {
    "export_threads": 4,
    "import_threads": 16,
    "data_dir": "./exported_data",
    "consistency": "auto"
  },
  "incr_migrate_conf": {
    "repl_threads": 16,
    "repl_batch": 100
  },
  "source_conf": [
    {
      "source_name": "mysql-replica-01",
      "binlog_name": "binlog.000001",
      "binlog_pos": 4,
      "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474
          ↪ d3c-28c7-11e7-8352-203db246dd3d:1-170"
    }
  ]
 }
}
```

13.6.8.5.25 Delete a replication task

This interface is a synchronous interface and the Status Code of the returned body is 204 upon successful request.

Request URI

`DELETE /api/v1/tasks/{task-name}`

Example

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1' \
  -H 'accept: application/json'
```

13.6.8.5.26 Update a replication task

This interface is a synchronous interface and a successful request returns the information of the task.

> **Note:**
>
> When you use this API to update the task configuration, make sure that the task is stopped and has run into incremental sync and that only some of the fields can be updated.

Request URI

`PUT /api/v1/tasks/{task-name}`

Example

```
curl -X 'PUT' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "task": {
    "name": "task-1",
    "task_mode": "all",
    "shard_mode": "pessimistic",
    "meta_schema": "dm-meta",
    "enhance_online_schema_change": true,
    "on_duplicate": "overwrite",
    "target_config": {
      "host": "127.0.0.1",
      "port": 3306,
      "user": "root",
      "password": "123456",
      "security": {
        "ssl_ca_content": "",
        "ssl_cert_content": "",
        "ssl_key_content": "",
        "cert_allowed_cn": [
```

```
        "string"
      ]
    }
  },
  "binlog_filter_rule": {
    "rule-1": {
      "ignore_event": [
        "all dml"
      ],
      "ignore_sql": [
        "^Drop"
      ]
    },
    "rule-2": {
      "ignore_event": [
        "all dml"
      ],
      "ignore_sql": [
        "^Drop"
      ]
    },
    "rule-3": {
      "ignore_event": [
        "all dml"
      ],
      "ignore_sql": [
        "^Drop"
      ]
    }
  },
  "table_migrate_rule": [
    {
      "source": {
        "source_name": "source-name",
        "schema": "db-*",
        "table": "tb-*"
      },
      "target": {
        "schema": "db1",
        "table": "tb1"
      },
      "binlog_filter_rule": [
        "rule-1",
        "rule-2",
        "rule-3",
```

```
        ]
      }
    ],
    "source_config": {
      "full_migrate_conf": {
        "export_threads": 4,
        "import_threads": 16,
        "data_dir": "./exported_data",
        "consistency": "auto"
      },
      "incr_migrate_conf": {
        "repl_threads": 16,
        "repl_batch": 100
      },
      "source_conf": [
        {
          "source_name": "mysql-replica-01",
          "binlog_name": "binlog.000001",
          "binlog_pos": 4,
          "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30
             ↪ :1-7041423,05474d3c-28c7-11e7-8352-203db246dd3d:1-170"
        }
      ]
    }
  }
}'
```

```
{
  "name": "task-1",
  "task_mode": "all",
  "shard_mode": "pessimistic",
  "meta_schema": "dm-meta",
  "enhance_online_schema_change": true,
  "on_duplicate": "overwrite",
  "target_config": {
    "host": "127.0.0.1",
    "port": 3306,
    "user": "root",
    "password": "123456",
    "security": {
      "ssl_ca_content": "",
      "ssl_cert_content": "",
      "ssl_key_content": "",
      "cert_allowed_cn": [
        "string"
```

```
    ]
  }
},
"binlog_filter_rule": {
  "rule-1": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-2": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-3": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  }
},
"table_migrate_rule": [
  {
    "source": {
      "source_name": "source-name",
      "schema": "db-*",
      "table": "tb-*"
    },
    "target": {
      "schema": "db1",
      "table": "tb1"
    },
    "binlog_filter_rule": [
      "rule-1",
      "rule-2",
      "rule-3",
    ]
```

```
    }
  ],
  "source_config": {
    "full_migrate_conf": {
      "export_threads": 4,
      "import_threads": 16,
      "data_dir": "./exported_data",
      "consistency": "auto"
    },
    "incr_migrate_conf": {
      "repl_threads": 16,
      "repl_batch": 100
    },
    "source_conf": [
      {
        "source_name": "mysql-replica-01",
        "binlog_name": "binlog.000001",
        "binlog_pos": 4,
        "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30:1-7041423,05474
            ↪ d3c-28c7-11e7-8352-203db246dd3d:1-170"
      }
    ]
  }
}
```

13.6.8.5.27 Start a replication task

This API is an asynchronous interface. If the request is successful, the status code of the returned body is 204. To learn the latest status of a task, You can get the information of a replication task.

Request URI

`POST /api/v1/tasks/{task-name}/start`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/start' \
  -H 'accept: */*'
```

13.6.8.5.28 Stop a replication task

This API is an asynchronous interface. If the request is successful, the status code of the returned body is 200. To learn the latest status of a task, You can get the information of a replication task.

Request URI

`POST /api/v1/tasks/{task-name}/stop`

Example

```
curl -X 'POST' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/stop' \
  -H 'accept: */*'
```

13.6.8.5.29 Get the information of a replication task

This API is a synchronous interface. If the request is successful, the information of the corresponding node is returned.

Request URI

`GET /api/v1/tasks/task-1/status`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/status?stage=running' \
  -H 'accept: application/json'
```

```
{
  "total": 1,
  "data": [
    {
      "name": "string",
      "source_name": "string",
      "worker_name": "string",
      "stage": "runing",
      "unit": "sync",
      "unresolved_ddl_lock_id": "string",
      "load_status": {
        "finished_bytes": 0,
        "total_bytes": 0,
        "progress": "string",
        "meta_binlog": "string",
        "meta_binlog_gtid": "string"
      },
      "sync_status": {
        "total_events": 0,
        "total_tps": 0,
        "recent_tps": 0,
        "master_binlog": "string",
        "master_binlog_gtid": "string",
```

```
      "syncer_binlog": "string",
      "syncer_binlog_gtid": "string",
      "blocking_ddls": [
        "string"
      ],
      "unresolved_groups": [
        {
          "target": "string",
          "ddl_list": [
            "string"
          ],
          "first_location": "string",
          "synced": [
            "string"
          ],
          "unsynced": [
            "string"
          ]
        }
      ],
      "synced": true,
      "binlog_type": "string",
      "seconds_behind_master": 0
    }
  }
]
}
```

13.6.8.5.30 Get the replication task list

This API is a synchronous interface and a successful request returns a list of the corresponding tasks.

Request URI

`GET /api/v1/tasks`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks' \
  -H 'accept: application/json'
```

```
{
  "total": 2,
  "data": [
    {
```

```
"name": "task-1",
"task_mode": "all",
"shard_mode": "pessimistic",
"meta_schema": "dm-meta",
"enhance_online_schema_change": true,
"on_duplicate": "overwrite",
"target_config": {
  "host": "127.0.0.1",
  "port": 3306,
  "user": "root",
  "password": "123456",
  "security": {
    "ssl_ca_content": "",
    "ssl_cert_content": "",
    "ssl_key_content": "",
    "cert_allowed_cn": [
      "string"
    ]
  }
},
"binlog_filter_rule": {
  "rule-1": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-2": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  },
  "rule-3": {
    "ignore_event": [
      "all dml"
    ],
    "ignore_sql": [
      "^Drop"
    ]
  }
```

```
    },
    "table_migrate_rule": [
      {
        "source": {
          "source_name": "source-name",
          "schema": "db-*",
          "table": "tb-*"
        },
        "target": {
          "schema": "db1",
          "table": "tb1"
        },
        "binlog_filter_rule": [
          "rule-1",
          "rule-2",
          "rule-3",
        ]
      }
    ],
    "source_config": {
      "full_migrate_conf": {
        "export_threads": 4,
        "import_threads": 16,
        "data_dir": "./exported_data",
        "consistency": "auto"
      },
      "incr_migrate_conf": {
        "repl_threads": 16,
        "repl_batch": 100
      },
      "source_conf": [
        {
          "source_name": "mysql-replica-01",
          "binlog_name": "binlog.000001",
          "binlog_pos": 4,
          "binlog_gtid": "03fc0263-28c7-11e7-a653-6c0b84d59f30
              ↪ :1-7041423,05474d3c-28c7-11e7-8352-203db246dd3d:1-170"
        }
      ]
    }
  }
 ]
}
```

13.6.8.5.31 Get the migration rules of a replication task

This API is a synchronous interface and a successful request returns a list of the migration rules of this task.

Request URI

`GET /api/v1/tasks/{task-name}/sources/{source-name}/migrate_targets`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/
     ↪ migrate_targets' \
  -H 'accept: application/json'
```

```
{
  "total": 0,
  "data": [
    {
      "source_schema": "db1",
      "source_table": "tb1",
      "target_schema": "db1",
      "target_table": "tb1"
    }
  ]
}
```

13.6.8.5.32 Get the list of schema names of the data source that is associated with a replication task

This API is a synchronous interface. If the request is successful, the corresponding list is returned.

Request URI

`GET /api/v1/tasks/{task-name}/sources/{source-name}/schemas`

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas' \
  -H 'accept: application/json'
```

```
[
  "db1"
]
```

13.6.8.5.33 Get the list of table names of a specified schema in the data source that is associated with a replication task

This API is a synchronous interface. If the request is successful, the corresponding list is returned.

Request URI

```
GET /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-name
↪ }
```

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas/db1' \
  -H 'accept: application/json'
```

```
[
  "table1"
]
```

13.6.8.5.34 Get the CREATE statement for schemas of the data source that is associated with a replication task

This API is a synchronous interface. If the request is successful, the corresponding CREATE statement is returned.

Request URI

```
GET /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-name
↪ }/{table-name}
```

Example

```
curl -X 'GET' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas/db1/
     ↪ table1' \
  -H 'accept: application/json'
```

```
{
  "schema_name": "db1",
  "table_name": "table1",
  "schema_create_sql": "CREATE TABLE `t1` (`id` int(11) NOT NULL
     ↪ AUTO_INCREMENT,PRIMARY KEY (`id`) /*T![clustered_index] CLUSTERED
     ↪ */) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin"
}
```

13.6.8.5.35 Update the CREATE statement for schemas of the data source that is associated with a replication task

This API is a synchronous interface. If the request is successful, the status code of the returned body is 200.

Request URI

```
POST /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-
↪ name}/{table-name}
```

Example

```
curl -X 'PUT' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/task-1/schemas/db1/
     ↪ table1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "sql_content": "CREATE TABLE `t1` ( `c1` int(11) DEFAULT NULL, `c2` int
     ↪ (11) DEFAULT NULL, `c3` int(11) DEFAULT NULL) ENGINE=InnoDB DEFAULT
     ↪ CHARSET=utf8mb4 COLLATE=utf8mb4_bin;",
  "flush": true,
  "sync": true
}'
```

13.6.8.5.36 Delete a schema of the data source that is associated with a replication task

This API is a synchronous interface. If the request is successful, the status code of the returned body is 200.

Request URI

```
DELETE /api/v1/tasks/{task-name}/sources/{source-name}/schemas/{schema-
↪ name}/{table-name}
```

Example

```
curl -X 'DELETE' \
  'http://127.0.0.1:8261/api/v1/tasks/task-1/sources/source-1/schemas/db1/
     ↪ table1' \
  -H 'accept: */*'
```

13.6.8.6 Compatibility Catalog of TiDB Data Migration

DM supports migrating data from different sources to TiDB clusters. Based on the data source type, DM has four compatibility levels:

- **Generally available (GA)**: The application scenario has been verified and passed the GA test.
- **Experimental**: Although the application scenario has been verified, the test does not cover all scenarios or involves only a limited number of users. The application scenario might encounter problems occasionally.
- **Not tested**: DM is expected to be always compatible with MySQL during iteration. However, due to resource constraints, not all MySQL forks are tested with DM. Therefore, the *not tested* source or target is technically compatible with DM, but is not fully tested, which means you need to verify its compatibility before you use.
- **Incompatible**: DM is proved to be incompatible with the data source and the application is not recommended for use in production environments.

13.6.8.6.1 Data sources

Data source	Compatibility level	Remarks
MySQL 5.5	Not tested	
MySQL 5.6	GA	
MySQL 5.7	GA	
MySQL 8.0	Experimental	
MariaDB < 10.1.2	Incompatible	Incompatible with binlog of the time type
MariaDB 10.1.2 ~ 10.5.10	Experimental	
MariaDB > 10.5.10	Incompatible	Permission errors reported in the check procedure

13.6.8.6.2 Target databases

Warning:

DM v5.3.0 is not recommended. If you have enabled GTID replication but do not enable relay log in DM v5.3.0, data replication fails with low probability.

Target database	Compatibility level	DM version
TiDB 8.x	GA	5.3.1
TiDB 7.x	GA	5.3.1
TiDB 6.x	GA	5.3.1
TiDB 5.4	GA	5.3.1
TiDB 5.3	GA	5.3.1
TiDB 5.2	GA	2.0.7, recommended: 5.4
TiDB 5.1	GA	2.0.4, recommended: 5.4
TiDB 5.0	GA	2.0.4, recommended: 5.4
TiDB 4.x	GA	2.0.1, recommended: 2.0.7

Target database	Compatibility level	DM version
TiDB 3.x	GA	2.0.1, recommended: 2.0.7
MySQL	Experimental	
MariaDB	Experimental	

13.6.8.7 Secure

13.6.8.7.1 Enable TLS for DM Connections

This document describes how to enable encrypted data transmission for DM connections, including connections between the DM-master, DM-worker, and dmctl components, and connections between DM and the upstream or downstream database.

Enable encrypted data transmission between DM-master, DM-worker, and dmctl

This section introduces how to enable encrypted data transmission between DM-master, DM-worker, and dmctl.

Configure and enable encrypted data transmission

1. Prepare certificates.

   It is recommended to prepare a server certificate for DM-master and DM-worker separately. Make sure that the two components can authenticate each other. You can choose to share one client certificate for dmctl.

   To generate self-signed certificates, you can use `openssl`, `cfssl` and other tools based on `openssl`, such as `easy-rsa`.

   If you choose `openssl`, you can refer to generating self-signed certificates.

2. Configure certificates.

> **Note:**
> You can configure DM-master, DM-worker, and dmctl to use the same set of certificates.

- DM-master

  Configure in the configuration file or command-line arguments:

```
ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/master-cert.pem"
ssl-key = "/path/to/master-key.pem"
```

- DM-worker

  Configure in the configuration file or command-line arguments:

```
ssl-ca = "/path/to/ca.pem"
ssl-cert = "/path/to/worker-cert.pem"
ssl-key = "/path/to/worker-key.pem"
```

- dmctl

  After enabling encrypted transmission in a DM cluster, if you need to connect to the cluster using dmctl, specify the client certificate. For example:

```
./dmctl --master-addr=127.0.0.1:8261 --ssl-ca /path/to/ca.pem --ssl
    ↪ -cert /path/to/client-cert.pem --ssl-key /path/to/client-key
    ↪ .pem
```

Verify component caller's identity

The Common Name is used for caller verification. In general, the callee needs to verify the caller's identity, in addition to verifying the key, the certificates, and the CA provided by the caller. For example, DM-worker can only be accessed by DM-master, and other visitors are blocked even though they have legitimate certificates.

To verify component caller's identity, you need to mark the certificate user identity using `Common Name` (CN) when generating the certificate, and to check the caller's identity by configuring the `Common Name` list for the callee.

- DM-master

  Configure in the configuration file or command-line arguments:

```
cert-allowed-cn = ["dm"]
```

- DM-worker

  Configure in the configuration file or command-line arguments:

```
cert-allowed-cn = ["dm"]
```

Reload certificates

To reload the certificates and the keys, DM-master, DM-worker, and dmctl reread the current certificates and the key files each time a new connection is created.

When the files specified by `ssl-ca`, `ssl-cert` or `ssl-key` are updated, restart DM components to reload the certificates and the key files and reconnect with each other.

Enable encrypted data transmission between DM components and the upstream or downstream database

This section introduces how to enable encrypted data transmission between DM components and the upstream or downstream database.

Enable encrypted data transmission for upstream database

1. Configure the upstream database, enable the encryption support, and set the server certificate. For detailed operations, see Using encrypted connections.

2. Set the MySQL client certificate in the source configuration file:

> **Note:**
> Make sure that all DM-master and DM-worker components can read the certificates and the key files via specified paths.

```
from:
    security:
        ssl-ca: "/path/to/mysql-ca.pem"
        ssl-cert: "/path/to/mysql-cert.pem"
        ssl-key: "/path/to/mysql-key.pem"
```

Enable encrypted data transmission for downstream TiDB

1. Configure the downstream TiDB to use encrypted connections. For detailed operatons, refer to Configure TiDB server to use secure connections.

2. Set the TiDB client certificate in the task configuration file:

> **Note:**
> Make sure that all DM-master and DM-worker components can read the certificates and the key files via specified paths.

```
target-database:
    security:
        ssl-ca: "/path/to/tidb-ca.pem"
        ssl-cert: "/path/to/tidb-client-cert.pem"
        ssl-key: "/path/to/tidb-client-key.pem"
```

13.6.8.7.2 Generate Self-signed Certificates for TiDB Data Migration

This document provides an example of using `openssl` to generate a self-signed certificate for TiDB Data Migration (DM). You can also generate certificates and keys that meet requirements according to your demands.

Assume that the topology of the instance cluster is as follows:

Name	Host IP	Services
node1	172.16.10.11	DM-master1
node2	172.16.10.12	DM-master2
node3	172.16.10.13	DM-master3
node4	172.16.10.14	DM-worker1
node5	172.16.10.15	DM-worker2
node6	172.16.10.16	DM-worker3

Install OpenSSL

- For Debian or Ubuntu OS:

```
apt install openssl
```

- For RedHat or CentOS OS:

```
yum install openssl
```

You can also refer to OpenSSL's official download document for installation.

Generate the CA certificate

A certificate authority (CA) is a trusted entity that issues digital certificates. In practice, contact your administrator to issue the certificate or use a trusted CA. CA manages multiple certificate pairs. Here you only need to generate an original pair of certificates as follows.

1. Generate the CA key:

```
openssl genrsa -out ca-key.pem 4096
```

2. Generate the CA certificates:

```
openssl req -new -x509 -days 1000 -key ca-key.pem -out ca.pem
```

3. Validate the CA certificates:

```
openssl x509 -text -in ca.pem -noout
```

Issue certificates for individual components

Certificates that might be used in the cluster

- The `master` certificate used by DM-master to authenticate DM-master for other components.

- The `worker` certificate used by DM-worker to authenticate DM-worker for other components.
- The `client` certificate used by dmctl to authenticate clients for DM-master and DM-worker.

Issue certificates for DM-master

To issue a certificate to a DM-master instance, perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out master-key.pem 2048
```

2. Make a copy of the OpenSSL configuration template file (Refer to the actual location of your template file because it might have more than one location):

```
cp /usr/lib/ssl/openssl.cnf .
```

If you do not know the actual location, look for it in the root directory:

```
find / -name openssl.cnf
```

3. Edit `openssl.cnf`, add `req_extensions = v3_req` under the `[ req ]` field, and add `subjectAltName = @alt_names` under the `[ v3_req ]` field. Finally, create a new field and edit the information of `Subject Alternative Name` (SAN) according to the cluster topology description above.

```
[ alt_names ]
IP.1 = 127.0.0.1
IP.2 = 172.16.10.11
IP.3 = 172.16.10.12
IP.4 = 172.16.10.13
```

The following checking items of SAN are currently supported:

- `IP`
- `DNS`
- `URI`

> **Note:**
> If a special IP such as `0.0.0.0` is to be used for connection or communication, you must also add it to `alt_names`.

4. Save the `openssl.cnf` file, and generate the certificate request file: (When giving input to `Common Name (e.g. server FQDN or YOUR name)[]:`, you assign a Common Name (CN) to the certificate, such as `dm`. It is used by the server to validate the identity of the client. Each component does not enable the validation by default. You can enable it in the configuration file.)

```
openssl req -new -key master-key.pem -out master-cert.pem -config
    ↪ openssl.cnf
```

5. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA ca.pem -CAkey ca-key.pem -
    ↪ CAcreateserial -in master-cert.pem -out master-cert.pem -
    ↪ extensions v3_req -extfile openssl.cnf
```

6. Verify that the certificate includes the SAN field (optional):

```
openssl x509 -text -in master-cert.pem -noout
```

7. Confirm that the following files exist in your current directory:

```
ca.pem
master-cert.pem
master-key.pem
```

> **Note:**
>
> The process of issuing certificates for the DM-worker instance is similar and will not be repeated in this document.

Issue certificates for the client (dmctl)

To issue a certificate to the client (dmctl), perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out client-key.pem 2048
```

2. Generate the certificate request file (in this step, you can also assign a Common Name to the certificate, which is used to allow the server to validate the identity of the client. Each component does not enable the validation by default, and you can enable it in the configuration file):

```
openssl req -new -key client-key.pem -out client-cert.pem
```

3. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA ca.pem -CAkey ca-key.pem -
    ↪ CAcreateserial -in client-cert.pem -out client-cert.pem
```

13.6.8.8 Monitoring and Alerts

13.6.8.8.1 Data Migration Monitoring Metrics

If your DM cluster is deployed using TiUP, the monitoring system is also deployed at the same time. This document describes the monitoring metrics provided by DM-worker.

Task

In the Grafana dashboard, the default name of DM is `DM-task`.

`overview`

`Overview` contains some monitoring metrics of all the DM-worker and DM-master instances or sources in the currently selected task. The current default alert rule is only for a single DM-worker/DM-master instance/source.

Metric name	Description	Alert	Severity level
task state	The state of subtasks for migration	N/A	N/A
storage capacity	The total storage capacity of the disk occupied by relay logs	N/A	N/A
storage remain	The remaining storage capacity of the disk occupied by relay logs	N/A	N/A

Metric name	Description	Alert	Severity level
binlog file gap between master and relay	The number of binlog files by which the `relay` processing unit is behind the upstream master	N/A	N/A
load progress	The percentage of the completed loading process of the load unit. The value is between 0%~100%	N/A	N/A
binlog file gap between master and syncer	The number of binlog files by which the binlog replication unit is behind the upstream master	N/A	N/A

Metric name	Description	Alert	Severity level
shard lock resolving	Whether the current subtask is waiting for sharding DDL migration. A value greater than 0 means that the current subtask is waiting for sharding DDL migration	N/A	N/A

Operation errors

Metric name	Description	Alert	Severity level
before any operate error	The number of errors before any operation	N/A	N/A
source bound error	The number of errors of data source binding operations	N/A	N/A
start error	The number of errors during the start of a subtask	N/A	N/A
pause error	The number of errors during the pause of a subtask	N/A	N/A

Metric name	Description	Alert	Severity level
resume error	The number of errors during the resuming of a subtask	N/A	N/A
auto-resume error	The number of errors during the auto-resuming of a subtask	N/A	N/A
update error	The number of errors during the update of a subtask	N/A	N/A
stop error	The number of errors during the stop of a subtask	N/A	N/A

High availability

Metric name	Description	Alert	Severity level
number of dm-masters start leader components per minute	The number of DM-master attempts to enable leader related components per minute	N/A	N/A

Metric name	Description	Alert	Severity level
number of workers in different state	The number of DM-workers in different states	Some DM-worker(s) has (have) been offline for more than one hour	critical
workers' state	The state of the DM-worker	N/A	N/A
number of worker event error	The number of different types of DM-worker errors	N/A	N/A
shard ddl error per minute	The number of different types of sharding DDL errors per minute	Any sharding DDL error occurs	critical
number of pending sharding ddl	The number of pending sharding DDL operations	Any pending sharding DDL operation has existed for more than one hour	critical

Task state

Metric name	Description	Alert	Severity level
task state	The state of subtasks	An alert occurs when the sub-task has been in the `Paused` ↪ state for more than 20 min-utes	critical

Dump/Load unit

The following metrics show only when `task-mode` is in the `full` or `all` mode.

Metric name	Description	Alert	Severity level
load progress	The percentage of the completed loading process of the load unit. The value range is 0%~100%	N/A	N/A

Metric name	Description	Alert	Severity level
data file size	The total size of the data files (includes the `INSERT INTO` statement) in the full data imported by the load unit	N/A	N/A
dump process exits with error	The dump unit encounters an error within the DM-worker and exits	Immediate alerts	critical
load process exits with error	The load unit encounters an error within the DM-worker and exits	Immediate alerts	critical
table count	The total number of tables in the full data imported by the load unit	N/A	N/A
data file count	The total number of data files (includes the `INSERT INTO` statement) in the full data imported by the load unit	N/A	N/A

Metric name	Description	Alert	Severity level
transaction execution latency	The latency of executing a transaction by the load unit (in seconds)	N/A	N/A
statement execution latency	The duration of executing a statement by the load unit (in seconds)	N/A	N/A
remaining time	The remaining time of replicating data by the load unit (in seconds)	N/A	N/A

Binlog replication

The following metrics show only when `task-mode` is in the `incremental` or `all` mode.

Metric name	Description	Alert	Severity level
remaining time to sync	The predicted remaining time it takes for `syncer` to be completely migrated with the upstream master (in minutes)	N/A	N/A

Metric name	Description	Alert	Severity level
replicate lag gauge	The latency time it takes to replicate the binlog from upstream to downstream (in seconds)	N/A	N/A
replicate lag histogram	The histogram of replicating the binlog from upstream to downstream (in seconds). Note that due to different statistical mechanisms, the data might be inaccurate	N/A	N/A
process exist with error	The binlog replication unit encounters an error within the DM-worker and exits	Immediate alerts	critical

Metric name	Description	Alert	Severity level
binlog file gap between master and syncer	The number of binlog files by which the `syncer` processing unit is behind the upstream master	An alert occurs when the number of binlog files by which the `syncer` ↪ processing unit is behind the upstream master exceeds one (>1) and the condition lasts over 10 minutes	critical

Metric name	Description	Alert	Severity level
binlog file gap between relay and syncer	The number of binlog files by which `syncer` is behind `relay`	An alert occurs when the number of binlog files by which the `syncer` ↪ processing unit is behind the `relay` processing unit exceeds one (>1) and the condition lasts over 10 minutes	critical

Metric name	Description	Alert	Severity level
binlog event QPS	The number of binlog events received per unit of time (this number does not include the events that need to be skipped)	N/A	N/A
skipped binlog event QPS	The number of binlog events received per unit of time that need to be skipped	N/A	N/A
read binlog event duration	The duration that the binlog replication unit reads the binlog from the relay log or the upstream MySQL (in seconds)	N/A	N/A
transform binlog event duration	The duration that the binlog replication unit parses and transforms the binlog into SQL statements (in seconds)	N/A	N/A

Metric name	Description	Alert	Severity level
dispatch binlog event duration	The duration that the binlog replication unit dispatches a binlog event (in seconds)	N/A	N/A
transaction execution latency	The duration that the binlog replication unit executes the transaction to the downstream (in seconds)	N/A	N/A
binlog event size	The size of a binlog event that the binlog replication unit reads from the relay log or the upstream MySQL	N/A	N/A
DML queue remain length	The length of the remaining DML job queue	N/A	N/A
total sqls jobs	The number of newly added jobs per unit of time	N/A	N/A
finished sqls jobs	The number of finished jobs per unit of time	N/A	N/A

Metric name	Description	Alert	Severity level
statement execution latency	The duration that the binlog replication unit executes the statement to the downstream (in seconds)	N/A	N/A
add job duration	The duration tht the binlog replication unit adds a job to the queue (in seconds)	N/A	N/A
DML conflict detect duration	The duration that the binlog replication unit detects the conflict in DML (in seconds)	N/A	N/A
skipped event duration	The duration that the binlog replication unit skips a binlog event (in seconds)	N/A	N/A
unsynced tables	The number of tables that have not received the shard DDL statement in the current subtask	N/A	N/A

Metric name	Description	Alert	Severity level
shard lock resolving	Whether the current subtask is waiting for the shard DDL lock to be resolved. A value greater than 0 indicates that it is waiting for the shard DDL lock to be resolved	N/A	N/A
ideal QPS	The highest QPS that can be achieved when the running time of DM is 0	N/A	N/A
binlog event row	The number of rows in a binlog event	N/A	N/A
finished transaction total	The number of finished transactions in total	N/A	N/A
replication transaction batch	The number of sql rows in the transaction executed to the downstream	N/A	N/A
flush checkpoints time interval	The time interval for flushing the checkpoints (in seconds)	N/A	N/A

Relay log

> **Note:**
>
> Currently, DM v2.0 does not support enabling the relay log feature.

Metric name	Description	Alert	Severity level
storage capacity	The storage capacity of the disk occupied by the relay log	N/A	N/A
storage remain	The remaining storage capacity of the disk occupied by the relay log	An alert is needed once the value is smaller than 10G	critical
process exits with error	The relay log encounters an error within the DM-worker and exits	Immediate alerts	critical
relay log data corruption	The number of corrupted relay log files	Immediate alerts	emergency

Metric name	Description	Alert	Severity level
fail to read binlog from master	The number of errors encountered when the relay log reads the binlog from the upstream MySQL	Immediate alerts	critical
fail to write relay log	The number of errors encountered when the relay log writes the binlog to disks	Immediate alerts	critical
binlog file index	The largest index number of relay log files. For example, “value = 1” indicates “relay-log.000001”	N/A	N/A

Metric name	Description	Alert	Severity level
binlog file gap between master and relay	The number of binlog files in the relay log that are behind the upstream master	An alert occurs when the number of binlog files by which the `relay` processing unit is behind the upstream master exceeds one (>1) and the condition lasts over 10 minutes	critical
binlog pos	The write offset of the latest relay log file	N/A	N/A

Metric name	Description	Alert	Severity level
read binlog event duration	The duration that the relay log reads binlog from the upstream MySQL (in seconds)	N/A	N/A
write relay log duration	The duration that the relay log writes binlog into the disks each time (in seconds)	N/A	N/A
binlog event size	The size of a single binlog event that the relay log writes into the disks	N/A	N/A

Instance

In the Grafana dashboard, the default name of an instance is `DM-instance`.

Relay log

Metric name	Description	Alert	Severity level
storage capacity	The total storage capacity of the disk occupied by the relay log	N/A	N/A

Metric name	Description	Alert	Severity level
storage remain	The remaining storage capacity within the disk occupied by the relay log	An alert occurs once the value is smaller than 10G	critical
process exits with error	The relay log encounters an error in DM-worker and exits	Immediate alerts	critical
relay log data corruption	The number of corrupted relay logs	Immediate alerts	emergency
fail to read binlog from master	The number of errors encountered when relay log reads the binlog from the upstream MySQL	Immediate alerts	critical
fail to write relay log	The number of errors encountered when the relay log writes the binlog to disks	Immediate alerts	critical

Metric name	Description	Alert	Severity level
binlog file index	The largest index number of relay log files. For example, “value = 1” indicates “relay-log.000001”	N/A	N/A

Metric name	Description	Alert	Severity level
binlog file gap between master and relay	The number of binlog files by which the `relay` processing unit is behind the upstream master	An alert occurs when the number of binlog files by which the `relay` processing unit is behind the upstream master exceeds one (>1) and the condition lasts over 10 minutes	critical
binlog pos	The write offset of the latest relay log file	N/A	N/A

Metric name	Description	Alert	Severity level
read binlog duration	The duration that the relay log reads the binlog from the upstream MySQL (in seconds)	N/A	N/A
write relay log duration	The duration that the relay log writes the binlog into the disk each time (in seconds)	N/A	N/A
binlog size	The size of a single binlog event that the relay log writes into the disks	N/A	N/A

Task

Metric name	Description	Alert	Severity level
task state	The state of subtasks for migration	An alert occurs when the sub-task has been paused for more than 10 min-utes	critical

Metric name	Description	Alert	Severity level
load progress	The percentage of the completed loading process of the load unit. The value range is 0%~100%	N/A	N/A
binlog file gap between master and syncer	The number of binlog files by which the binlog replication unit is behind the upstream master	N/A	N/A
shard lock resolving	Whether the current subtask is waiting for sharding DDL migration. A value greater than 0 means that the current subtask is waiting for sharding DDL migration	N/A	N/A

13.6.8.8.2 DM Alert Information

The alert system is deployed by default when you deploy a DM cluster using TiUP. For more information about DM alert rules and the solutions, refer to handle alerts.

Both DM alert information and monitoring metrics are based on Prometheus. For more

information about their relationship, refer to DM monitoring metrics.

13.6.8.9 Handle Errors in TiDB Data Migration

This document introduces the error system and how to handle common errors when you use DM.

13.6.8.9.1 Error system

In the error system, usually, the information of a specific error is as follows:

- `code`: error code.

  DM uses the same error code for the same error type. An error code does not change as the DM version changes.

  Some errors might be removed during the DM iteration, while the error codes are not. DM uses a new error code instead of an existing one for a new error.

- `class`: error type.

  It is used to mark the component where an error occurs (error source).

  The following table displays all error types, error sources, and error samples.

  |

  Error Type

Error Source	Error Sample
database	Database operations	[code=10003:class=database:scope=downstream:level=medium] database driver: invalid connection
functional	Underlying functions of DM	[code=11005:class=functional:scope=internal:level=high] not allowed operation: alter multiple tables in one statement
config	Incorrect configuration	[code=20005:class=config:scope=internal:level=medium] empty source-id not valid
binlog-op	Binlog operations	[code=22001:class=binlog-op:scope=internal:level=high] empty UUIDs not valid
checkpoint	checkpoint operations	[code=24002:class=checkpoint:scope=internal:level=high] save point bin.1234 is older than current pos bin.1371
task-check	Performing task check	[code=26003:class=task-check:scope=internal:level=medium] new table router error

```
relay-event-lib| Executing the basic functions of the relay module | [code=28001:
↪ class=relay-event-lib:scope=internal:level=high] parse server-uuid.
↪ index |
relay-unit | relay processing unit | [code=30015:class=relay-unit:scope=
↪ upstream:level=high] TCPReader get event: ERROR 1236 (HY000): Could
↪ not open log file |
dump-unit | dump processing unit | [code=32001:class=dump-unit:scope=internal
↪ :level=high] mydumper runs with error: CRITICAL **: 15:12:17.559:
↪ Error connecting to database: Access denied for user 'root'@'172.17.0.1'
↪  (using password: NO) |
load-unit | load processing unit | [code=34002:class=load-unit:scope=internal
↪ :level=high] corresponding ending of sql: ')' not found |
sync-unit | sync processing unit | [code=36027:class=sync-unit:scope=internal
↪ :level=high] Column count doesn't match value count: 9 (columns)vs
↪ 10 (values) |
dm-master | DM-master service | [code=38008:class=dm-master:scope=internal
↪ :level=high] grpc request error: rpc error: code = Unavailable desc
↪  = all SubConns are in TransientFailure, latest connection error:
↪ connection error: desc = "transport: Error while dialing dial tcp
↪ 172.17.0.2:8262: connect: connection refused" |
dm-worker | DM-worker service | [code=40066:class=dm-worker:scope=internal
↪ :level=high] ExecuteDDL timeout, try use query-status to query
↪ whether the DDL is still blocking |
dm-tracer | DM-tracer service | [code=42004:class=dm-tracer:scope=internal:
↪ level=medium] trace event test.1 not found |
schema-tracker | schema-tracker (during incremental data replication) | [code
↪ =44006:class=schema-tracker:scope=internal:level=high],"cannot track
↪  DDL: ALTER TABLE test DROP COLUMN col1" |
scheduler | Scheduling operations (of data migration tasks) | [code=46001:class=
↪ scheduler:scope=internal:level=high],"the scheduler has not started"
|
dmctl | An error occurs within dmctl or when it interacts with other components |
[code=48001:class=dmctl:scope=internal:level=high],"can not create grpc
↪  connection" |
```

- `scope`: Error scope.

  It is used to mark the scope and source of DM objects when an error occurs. `scope` includes four types: `not-set`, `upstream`, `downstream`, and `internal`.

  If the logic of the error directly involves requests between upstream and downstream databases, the scope is set to `upstream` or `downstream`; otherwise, it is currently set to `internal`.

- `level`: Error level.

  The severity level of the error, including `low`, `medium`, and `high`.

  - The `low` level error usually relates to user operations and incorrect inputs. It does not affect migration tasks.
  - The `medium` level error usually relates to user configurations. It affects some newly started services; however, it does not affect the existing DM migration status.
  - The `high` level error usually needs your attention, since you need to resolve it to avoid the possible interruption of a migration task.

- `message`: Error descriptions.

  Detailed descriptions of the error. To wrap and store every additional layer of error message on the error call chain, the errors.Wrap mode is adopted. The message description wrapped at the outermost layer indicates the error in DM and the message description wrapped at the innermost layer indicates the error source.

- `workaround`: Error handling methods (optional)

  The handling methods for this error. For some confirmed errors (such as configuration errors), DM gives the corresponding manual handling methods in `workaround`.

- Error stack information (optional)

  Whether DM outputs the error stack information depends on the error severity and the necessity. The error stack records the complete stack call information when the error occurs. If you cannot figure out the error cause based on the basic information and the error message, you can trace the execution path of the code when the error occurs using the error stack.

For the complete list of error codes, refer to the error code lists.

13.6.8.9.2 Troubleshooting

If you encounter an error while running DM, take the following steps to troubleshoot this error:

1. Execute the `query-status` command to check the task running status and the error output.
2. Check the log files related to the error. The log files are on the DM-master and DM-worker nodes. To get key information about the error, refer to the error system. Then check the Handle Common Errors section to find the solution.
3. If the error is not covered in this document, and you cannot solve the problem by checking the log or monitoring metrics, get support from PingCAP or the community.
4. After the error is resolved, restart the task using dmctl.

```
resume-task ${task name}
```

However, you need to reset the data migration task in some cases. For details, refer to Reset the Data Migration Task.

13.6.8.9.3 Handle common errors

|

Error Code

Error Description	How to Handle
code=10001	Abnormal database operation.	Further analyze the error message and error stack.
code=10002	The `bad connection` error from the underlying database. It usually indicates that the connection between DM and the downstream TiDB instance is abnormal (possibly caused by network failure or TiDB restart) and the currently requested data is not sent to TiDB.	DM provides automatic recovery for such error. If the recovery is not successful for a long time, check the network or TiDB status.
code=10003	The `invalid connection` error from the underlying database. It usually indicates that the connection between DM and the downstream TiDB instance is abnormal (possibly caused by network failure or TiDB restart) and the currently requested data is partly sent to TiDB.	DM provides automatic recovery for such error. If the recovery is not successful for a long time, further check the error message and analyze the information based on the actual situation.
code=10005	Occurs when performing the `QUERY` type SQL statements.	
code=10006	Occurs when performing the `EXECUTE` type SQL statements, including DDL statements and DML statements of the `INSERT`, `UPDATE`or `DELETE` type. For more detailed error information, check the error message which usually includes the error code and error information returned for database operations.	
code=11006	Occurs when the built-in parser of DM parses the incompatible DDL statements.	Refer to Data Migration - incompatible DDL statements for solution.
code=20010	Occurs when decrypting the database password that is provided in task configuration.	Check whether the downstream database password provided in the configuration task is correctly encrypted using dmctl.
code=26002	The task check fails to establish database connection. For more detailed error information, check the error message which usually includes the error code and error information returned for database operations.	Check whether the machine where DM-master is located has permission to access the upstream.
code=32001	Abnormal dump processing unit	If the error message contains `mydumper: argument list too long.`, configure the table to be exported by manually adding the `--regex` regular expression in the Mydumper argument `extra-args` in the `task.yaml` file according to the block-allow list. For example, to export all tables named `hello`, add `--regex '.*\\.hello$'`; to export all tables, add `--regex '.*'`.
code=38008	An error occurs in the gRPC communication among DM components.	Check `class`. Find out the error occurs in the interaction of which components. Determine the type of communication error. If the error occurs when establishing gRPC connection, check

whether the communication server is working normally. |

What can I do when a migration task is interrupted with the `invalid connection` error returned?

Reason

The `invalid connection` error indicates that anomalies have occurred in the connection between DM and the downstream TiDB database (such as network failure, TiDB restart, and TiKV busy) and that a part of the data for the current request has been sent to TiDB.

Solutions

Because DM has the feature of concurrently migrating data to the downstream in migration tasks, several errors might occur when a task is interrupted. You can check these errors by using `query-status`.

- If only the `invalid connection` error occurs during the incremental replication process, DM retries the task automatically.
- If DM does not or fails to retry automatically because of version problems, use `stop-task` to stop the task and then use `start-task` to restart the task.

A migration task is interrupted with the `driver: bad connection` error returned

Reason

The `driver: bad connection` error indicates that anomalies have occurred in the connection between DM and the upstream TiDB database (such as network failure and TiDB restart) and that the data of the current request has not yet been sent to TiDB at that moment.

Solution

The current version of DM automatically retries on error. If you use the previous version which does not support automatically retry, you can execute the `stop-task` command to stop the task. Then execute `start-task` to restart the task.

The relay unit throws error `event from * in * diff from passed-in event *` or a migration task is interrupted with failing to get or parse binlog errors like `get binlog error ERROR 1236 (HY000)` and `binlog checksum mismatch, data may be corrupted` returned

Reason

During the DM process of relay log pulling or incremental replication, this two errors might occur if the size of the upstream binlog file exceeds **4 GB**.

Cause: When writing relay logs, DM needs to perform event verification based on binlog positions and the size of the binlog file, and store the replicated binlog positions as checkpoints. However, the official MySQL uses `uint32` to store binlog positions. This means the binlog position for a binlog file over 4 GB overflows, and then the errors above occur.

Solutions

For relay units, manually recover migration using the following solution:

1. Identify in the upstream that the size of the corresponding binlog file has exceeded 4GB when the error occurs.
2. Stop the DM-worker.
3. Copy the corresponding binlog file in the upstream to the relay log directory as the relay log file.
4. In the relay log directory, update the corresponding `relay.meta` file to pull from the next binlog file. If you have specified `enable_gtid` to `true` for the DM-worker, you need to modify the GTID corresponding to the next binlog file when updating the `relay.meta` file. Otherwise, you don't need to modify the GTID.

   Example: when the error occurs, `binlog-name = "mysql-bin.004451"` and `binlog-pos = 2453`. Update them respectively to `binlog-name = "mysql-bin.004452"` and `binlog-pos = 4`, and update `binlog-gtid` to `f0e914ef-54cf-11e7-813d-6c92bf2fa791:1-138218058`.
5. Restart the DM-worker.

For binlog replication processing units, manually recover migration using the following solution:

1. Identify in the upstream that the size of the corresponding binlog file has exceeded 4GB when the error occurs.
2. Stop the migration task using `stop-task`.
3. Update the `binlog_name` in the global checkpoints and in each table checkpoint of the downstream `dm_meta` database to the name of the binlog file in error; update `binlog_pos` to a valid position value for which migration has completed, for example, 4.

   Example: the name of the task in error is `dm_test`, the corresponding `source-id` is `replica-1`, and the corresponding binlog file is `mysql-bin|000001.004451`. Execute the following command:

```
UPDATE dm_test_syncer_checkpoint SET binlog_name='mysql-bin
    ↪ |000001.004451', binlog_pos = 4 WHERE id='replica-1';
```

4. Specify `safe-mode: true` in the `syncers` section of the migration task configuration to ensure re-entrant.
5. Start the migration task using `start-task`.
6. View the status of the migration task using `query-status`. You can restore `safe-mode` to the original value and restart the migration task when migration is done for the original error-triggering relay log files.

`Access denied for user 'root'@'172.31.43.27' (using password: YES)` shows when you query the task or check the log

For database related passwords in all the DM configuration files, it is recommended to use the passwords encrypted by `dmctl`. If a database password is empty, it is unnecessary to encrypt it. For how to encrypt the plaintext password, see Encrypt the database password using dmctl.

In addition, the user of the upstream and downstream databases must have the corresponding read and write privileges. Data Migration also prechecks the corresponding privileges automatically while starting the data migration task.

The `load` processing unit reports the error `packet for query is too large. Try adjusting the 'max_allowed_packet' variable`

Reasons

- Both MySQL client and MySQL/TiDB server have the quota limits for `max_allowed_packet` ↪ . If any `max_allowed_packet` exceeds a limit, the client receives the error message. Currently, for the latest version of DM and TiDB server, the default value of `max_allowed_packet` is `64M`.
- The full data import processing unit in DM does not support splitting the SQL file exported by the Dump processing unit in DM.

Solutions

- It is recommended to set the `statement-size` option of `extra-args` for the Dump processing unit:

  According to the default `--statement-size` setting, the default size of `Insert` ↪ `Statement` generated by the Dump processing unit is about `1M`. With this default setting, the load processing unit does not report the error `packet for query is` ↪ `too large. Try adjusting the 'max_allowed_packet' variable` in most cases.

  Sometimes you might receive the following `WARN` log during the data dump. This `WARN` log does not affect the dump process. This only means that wide tables are dumped.

  ```
  Row bigger than statement_size for xxx
  ```

- If the single row of the wide table exceeds `64M`, you need to modify the following configurations and make sure the configurations take effect.
  - Execute `set @@global.max_allowed_packet=134217728` (`134217728` = 128 MB) in the TiDB server.
  - First add the `max-allowed-packet: 134217728` (128 MB) to the `target-` ↪ `database` section in the DM task configuration file. Then, execute the `stop-task` command and execute the `start-task` command.

13.6.8.10 TiDB Data Migration Glossary

This document lists the terms used in the logs, monitoring, configurations, and documentation of TiDB Data Migration (DM).

13.6.8.10.1 B

Binlog

In TiDB DM, binlogs refer to the binary log files generated in the TiDB database. It has the same indications as that in MySQL or MariaDB. Refer to MySQL Binary Log and MariaDB Binary Log for details.

Binlog event

Binlog events are information about data modification made to a MySQL or MariaDB server instance. These binlog events are stored in the binlog files. Refer to MySQL Binlog Event and MariaDB Binlog Event for details.

Binlog event filter

Binlog event filter is a more fine-grained filtering feature than the block and allow lists filtering rule. Refer to binlog event filter for details.

Binlog position

The binlog position is the offset information of a binlog event in a binlog file. Refer to MySQL `SHOW BINLOG EVENTS` and MariaDB `SHOW BINLOG EVENTS` for details.

Binlog replication processing unit/sync unit

Binlog replication processing unit is the processing unit used in DM-worker to read upstream binlogs or local relay logs, and to migrate these logs to the downstream. Each subtask corresponds to a binlog replication processing unit. In the current documentation, the binlog replication processing unit is also referred to as the sync processing unit.

Block & allow table list

Block & allow table list is the feature that filters or only migrates all operations of some databases or some tables. Refer to block & allow table lists for details. This feature is similar to MySQL Replication Filtering and MariaDB Replication Filters.

13.6.8.10.2 C

Checkpoint

A checkpoint indicates the position from which a full data import or an incremental replication task is paused and resumed, or is stopped and restarted.

- In a full import task, a checkpoint corresponds to the offset and other information of the successfully imported data in a file that is being imported. A checkpoint is updated synchronously with the data import task.

- In an incremental replication, a checkpoint corresponds to the binlog position and other information of a binlog event that is successfully parsed and migrated to the downstream. A checkpoint is updated after the DDL operation is successfully migrated or 30 seconds after the last update.

In addition, the `relay.meta` information corresponding to a relay processing unit works similarly to a checkpoint. A relay processing unit pulls the binlog event from the upstream and writes this event to the relay log, and writes the binlog position or the GTID information corresponding to this event to `relay.meta`.

13.6.8.10.3 D

Dump processing unit/dump unit

The dump processing unit is the processing unit used in DM-worker to export all data from the upstream. Each subtask corresponds to a dump processing unit.

13.6.8.10.4 G

GTID

The GTID is the global transaction ID of MySQL or MariaDB. With this feature enabled, the GTID information is recorded in the binlog files. Multiple GTIDs form a GTID set. Refer to MySQL GTID Format and Storage and MariaDB Global Transaction ID for details.

13.6.8.10.5 L

Load processing unit/load unit

The load processing unit is the processing unit used in DM-worker to import the fully exported data to the downstream. Each subtask corresponds to a load processing unit. In the current documentation, the load processing unit is also referred to as the import processing unit.

13.6.8.10.6 M

Migrate/migration

The process of using the TiDB Data Migration tool to copy the **full data** of the upstream database to the downstream database.

In the case of clearly mentioning “full”, not explicitly mentioning “full or incremental”, and clearly mentioning “full + incremental”, use migrate/migration instead of replicate/replication.

13.6.8.10.7 R

Relay log

The relay log refers to the binlog files that DM-worker pulls from the upstream MySQL or MariaDB, and stores in the local disk. The format of the relay log is the standard binlog file, which can be parsed by tools such as mysqlbinlog of a compatible version. Its role is similar to MySQL Relay Log and MariaDB Relay Log.

For more details such as the relay log's directory structure, initial migration rules, and data purge in TiDB DM, see TiDB DM relay log.

Relay processing unit

The relay processing unit is the processing unit used in DM-worker to pull binlog files from the upstream and write data into relay logs. Each DM-worker instance has only one relay processing unit.

Replicate/replication

The process of using the TiDB Data Migration tool to copy the **incremental data** of the upstream database to the downstream database.

In the case of clearly mentioning "incremental", use replicate/replication instead of migrate/migration.

13.6.8.10.8 S

Safe mode

Safe mode is the mode in which DML statements can be imported more than once when the primary key or unique index exists in the table schema. In this mode, some statements from the upstream are migrated to the downstream only after they are re-written. The `INSERT` statement is re-written as `REPLACE`; the `UPDATE` statement is re-written as `DELETE` and `REPLACE`.

This mode is enabled in any of the following situations:

- The safe mode remains enabled when the `safe-mode` parameter in the task configuration file is set to `true`.
- In shard merge scenarios, the safe mode remains enabled before DDL statements are replicated in all sharded tables.
- If the argument `--consistency none` is configured for the dump processing unit of a full migration task, it cannot be determined whether the binlog changes at the beginning of the export affect the exported data or not. Therefore, the safe mode remains enabled for the incremental replication of these binlog changes.
- If the task is paused by error and then resumed, the operations on some data might be executed twice.

Shard DDL

The shard DDL is the DDL statement that is executed on the upstream sharded tables. It needs to be coordinated and migrated by TiDB DM in the process of merging the sharded tables. In the current documentation, the shard DDL is also referred to as the sharding DDL.

Shard DDL lock

The shard DDL lock is the lock mechanism that coordinates the migration of shard DDL. Refer to the implementation principles of merging and migrating data from sharded tables in the pessimistic mode for details. In the current documentation, the shard DDL lock is also referred to as the sharding DDL lock.

Shard group

A shard group is all the upstream sharded tables to be merged and migrated to the same table in the downstream. Two-level shard groups are used for implementation of TiDB DM. Refer to the implementation principles of merging and migrating data from sharded tables in the pessimistic mode for details. In the current documentation, the shard group is also referred to as the sharding group.

Subtask

The subtask is a part of a data migration task that is running on each DM-worker instance. In different task configurations, a single data migration task might have one subtask or multiple subtasks.

Subtask status

The subtask status is the status of a data migration subtask. The current status options include `New`, `Running`, `Paused`, `Stopped`, and `Finished`. Refer to subtask status for more details about the status of a data migration task or subtask.

13.6.8.10.9 T

Table routing

The table routing feature enables DM to migrate a certain table of the upstream MySQL or MariaDB instance to the specified table in the downstream, which can be used to merge and migrate sharded tables. Refer to table routing for details.

Task

The data migration task, which is started after you successfully execute a `start-task` ↪ command. In different task configurations, a single migration task can run on a single DM-worker instance or on multiple DM-worker instances at the same time.

Task status

The task status refers to the status of a data migration task. The task status depends on the statuses of all its subtasks. Refer to subtask status for details.

13.6.8.11 Example

13.6.8.11.1 Migrate Data Using Data Migration

This guide shows how to migrate data using the Data Migration (DM) tool.

Step 1: Deploy the DM cluster

It is recommended to deploy the DM cluster using TiUP. You can also deploy the DM cluster using binary for trial or test.

Note:

- For database passwords in all the DM configuration files, it is recommended to use the passwords encrypted by `dmctl`. If a database password is empty, it is unnecessary to encrypt it. See Encrypt the database password using dmctl.
- The user of the upstream and downstream databases must have the corresponding read and write privileges.

Step 2: Check the cluster information

After the DM cluster is deployed using TiUP, the configuration information is like what is listed below.

- The configuration information of related components in the DM cluster:

Component	Host	Port
dm_worker1	172.16.10.72	8262
dm_worker2	172.16.10.73	8262
dm_master	172.16.10.71	8261

- The information of upstream and downstream database instances:

Database instance	Host	Port	Username	Encrypted password
Upstream MySQL-1	172.16.10.81	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
Upstream MySQL-2	172.16.10.82	3306	root	VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU=
Downstream TiDB	172.16.10.83	4000	root	

The list of privileges needed on the MySQL host can be found in the precheck documentation.

Step 3: Create data source

1. Write MySQL-1 related information to `conf/source1.yaml`:

```
# MySQL1 Configuration.

source-id: "mysql-replica-01"
# This indicates that whether DM-worker uses Global Transaction
    ↪ Identifier (GTID) to pull binlog. Before you use this
    ↪ configuration item, make sure that the GTID mode is enabled in
    ↪ the upstream MySQL.
enable-gtid: false

from:
  host: "172.16.10.81"
  user: "root"
  password: "VjX8cEeTX+qcvZ3bPaO4h0C80pe/1aU="
  port: 3306
```

2. Execute the following command in the terminal, and use `tiup dmctl` to load the MySQL-1 data source configuration to the DM cluster:

```
tiup dmctl --master-addr 172.16.10.71:8261 operate-source create conf/
    ↪ source1.yaml
```

3. For MySQL-2, modify the relevant information in the configuration file and execute the same `dmctl` command.

Step 4: Configure the data migration task

The following example assumes that you need to migrate all the `test_table` table data in the `test_db` database of both the upstream MySQL-1 and MySQL-2 instances, to the downstream `test_table` table in the `test_db` database of TiDB, in the full data plus incremental data mode.

Edit the `task.yaml` task configuration file as below:

```
##### The task name. You need to use a different name for each of the
    ↪ multiple tasks that
##### run simultaneously.
name: "test"
##### The full data plus incremental data (all) migration mode.
task-mode: "all"
##### The downstream TiDB configuration information.
target-database:
  host: "172.16.10.83"
  port: 4000
  user: "root"
  password: ""
```

```
##### Configuration of all the upstream MySQL instances required by the
    ↪ current data migration task.
mysql-instances:
-
  # The ID of upstream instances or the migration group. You can refer to
     ↪ the configuration of `source_id` in the "inventory.ini" file or in
     ↪ the "dm-master.toml" file.
  source-id: "mysql-replica-01"
  # The configuration item name of the block and allow lists of the name of
     ↪ the
  # database/table to be migrated, used to quote the global block and allow
  # lists configuration that is set in the global block-allow-list below.
  block-allow-list: "global" # Use black-white-list if the DM version is
     ↪ earlier than or equal to v2.0.0-beta.2.
  # The configuration item name of the dump processing unit, used to quote
     ↪ the global configuration of the dump unit.
  mydumper-config-name: "global"

-
  source-id: "mysql-replica-02"
  block-allow-list: "global" # Use black-white-list if the DM version is
     ↪ earlier than or equal to v2.0.0-beta.2.
  mydumper-config-name: "global"

##### The global configuration of block and allow lists. Each instance can
    ↪ quote it by the
##### configuration item name.
block-allow-list:                  # Use black-white-list if the DM version
    ↪ is earlier than or equal to v2.0.0-beta.2.
  global:
    do-tables:                     # The allow list of upstream tables to be
       ↪ migrated.
    - db-name: "test_db"           # The database name of the table to be
       ↪ migrated.
      tbl-name: "test_table"       # The name of the table to be migrated.

##### The global configuration of the dump unit. Each instance can quote it
    ↪ by the configuration item name.
mydumpers:
  global:
    extra-args: ""
```

Step 5: Start the data migration task

To detect possible errors of data migration configuration in advance, DM provides the

precheck feature:

- DM automatically checks the corresponding privileges and configuration while starting the data migration task.
- You can also use the `check-task` command to manually precheck whether the upstream MySQL instance configuration satisfies the DM requirements.

For details about the precheck feature, see Precheck the upstream MySQL instance configuration.

> **Note:**
>
> Before starting the data migration task for the first time, you should have got the upstream configured. Otherwise, an error is reported while you start the task.

Run the `tiup dmctl` command to start the data migration tasks. `task.yaml` is the configuration file that is edited above.

```
tiup dmctl --master-addr 172.16.10.71:8261 start-task ./task.yaml
```

- If the above command returns the following result, it indicates the task is successfully started.

```
{
    "result": true,
    "msg": "",
    "workers": [
        {
            "result": true,
            "worker": "172.16.10.72:8262",
            "msg": ""
        },
        {
            "result": true,
            "worker": "172.16.10.73:8262",
            "msg": ""
        }
    ]
}
```

- If you fail to start the data migration task, modify the configuration according to the returned prompt and then run the `start-task task.yaml` command to restart the task.

Step 6: Check the data migration task

If you need to check the task state or whether a certain data migration task is running in the DM cluster, run the following command in `tiup dmctl`:

```
tiup dmctl --master-addr 172.16.10.71:8261 query-status
```

Step 7: Stop the data migration task

If you do not need to migrate data any more, run the following command in `tiup dmctl` to stop the task:

```
tiup dmctl --master-addr 172.16.10.71:8261 stop-task test
```

`test` is the task name that you set in the `name` configuration item of the `task.yaml` configuration file.

Step 8: Monitor the task and check logs

Assuming that Prometheus, Alertmanager, and Grafana are successfully deployed along with the DM cluster deployment using TiUP, and the Grafana address is `172.16.10.71`. To view the alert information related to DM, you can open http://172.16.10.71:9093 in a browser and enter into Alertmanager; to check monitoring metrics, go to http://172.16.10.71:3000, and choose the DM dashboard.

While the DM cluster is running, DM-master, DM-worker, and dmctl output the monitoring metrics information through logs. The log directory of each component is as follows:

- DM-master log directory: It is specified by the `--log-file` DM-master process parameter. If DM is deployed using TiUP, the log directory is `{log_dir}` in the DM-master node.
- DM-worker log directory: It is specified by the `--log-file` DM-worker process parameter. If DM is deployed using TiUP, the log directory is `{log_dir}` in the DM-worker node.

13.6.8.11.2 Create a Data Migration Task

This document describes how to create a simple data migration task after the DM cluster is successfully deployed.

Sample scenario

Suppose that you create a data migration task based on this sample scenario:

- Deploy two MySQL instances with binlog enabled and one TiDB instance locally
- Use a DM-master of the DM cluster to manage the cluster and data migration tasks.

The information of each node is as follows.

Instance	Server Address	Port
MySQL1	127.0.0.1	3306
MySQL2	127.0.0.1	3307
TiDB	127.0.0.1	4000
DM-master	127.0.0.1	8261

Based on this scenario, the following sections describe how to create a data migration task.

Start upstream MySQL

Prepare 2 runnable MySQL instances. You can also use Docker to quickly start MySQL. The commands are as follows:

```
docker run --rm --name mysql-3306 -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD
    ↪ =true mysql:5.7.22 --log-bin=mysql-bin --port=3306 --bind-address
    ↪ =0.0.0.0 --binlog-format=ROW --server-id=1 --gtid_mode=ON --enforce-
    ↪ gtid-consistency=true > mysql.3306.log 2>&1 &
docker run --rm --name mysql-3307 -p 3307:3307 -e MYSQL_ALLOW_EMPTY_PASSWORD
    ↪ =true mysql:5.7.22 --log-bin=mysql-bin --port=3307 --bind-address
    ↪ =0.0.0.0 --binlog-format=ROW --server-id=1 --gtid_mode=ON --enforce-
    ↪ gtid-consistency=true > mysql.3307.log 2>&1 &
```

Prepare data

- Write example data into mysql-3306:

```
drop database if exists `sharding1`;
create database `sharding1`;
use `sharding1`;
create table t1 (id bigint, uid int, name varchar(80), info varchar
    ↪ (100), primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=
    ↪ utf8mb4;
create table t2 (id bigint, uid int, name varchar(80), info varchar
    ↪ (100), primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=
    ↪ utf8mb4;
insert into t1 (id, uid, name) values (1, 10001, 'Gabriel García
    ↪ Márquez'), (2 ,10002, 'Cien años de soledad');
insert into t2 (id, uid, name) values (3,20001, 'José Arcadio Buendía'
    ↪ ), (4,20002, 'Úrsula Iguarán'), (5,20003, 'José Arcadio');
```

- Write example data into mysql-3307:

```
drop database if exists `sharding2`;
create database `sharding2`;
use `sharding2`;
```

```
create table t2 (id bigint, uid int, name varchar(80), info varchar
    ↪ (100), primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=
    ↪ utf8mb4;
create table t3 (id bigint, uid int, name varchar(80), info varchar
    ↪ (100), primary key (`id`), unique key(`uid`)) DEFAULT CHARSET=
    ↪ utf8mb4;
insert into t2 (id, uid, name, info) values (6, 40000, 'Remedios
    ↪ Moscote', '{}');
insert into t3 (id, uid, name, info) values (7, 30001, 'Aureliano José
    ↪ ', '{}'), (8, 30002, 'Santa Sofía de la Piedad', '{}'), (9,
    ↪ 30003, '17 Aurelianos', NULL);
```

Start downstream TiDB

To run a TiDB server, use the following command:

```
wget https://download.pingcap.org/tidb-community-server-v6.1.7-linux-amd64.
    ↪ tar.gz
tar -xzvf tidb-latest-linux-amd64.tar.gz
mv tidb-latest-linux-amd64/bin/tidb-server ./
./tidb-server
```

> **Warning:**
>
> The deployment method of TiDB in this document **do not apply** to production or development environments.

Configure the MySQL data source

Before starting a data migration task, you need to configure the MySQL data source.

Encrypt the password

> **Note:**
>
> - You can skip this step if the database does not have a password.
> - You can use the plaintext password to configure the source information in DM v1.0.6 and later versions.

For safety reasons, it is recommended to configure and use encrypted passwords. You can use dmctl to encrypt the MySQL/TiDB password. Suppose the password is "123456":

```
./dmctl encrypt "123456"
```

```
fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg=
```

Save this encrypted value, and use it for creating a MySQL data source in the following steps.

Edit the source configuration file

Write the following configurations to `conf/source1.yaml`.

```
##### MySQL1 Configuration.

source-id: "mysql-replica-01"

##### Indicates whether GTID is enabled
enable-gtid: true

from:
  host: "127.0.0.1"
  user: "root"
  password: "fCxfQ9XKCezSzuCD0Wf5dUD+LsKegSg="
  port: 3306
```

In MySQL2 data source, copy the above configurations to `conf/source2.yaml`. You need to change `name` to `mysql-replica-02` and change `password` and `port` to appropriate values.

Create a source

To load the data source configurations of MySQL1 into the DM cluster using dmctl, run the following command in the terminal:

```
./dmctl --master-addr=127.0.0.1:8261 operate-source create conf/source1.
    ↪ yaml
```

For MySQL2, replace the configuration file in the above command with that of MySQL2.

Create a data migration task

After importing prepared data, there are several sharded tables on both MySQL1 and MySQL2 instances. These tables have identical structure and the same prefix "t" in the table names; the databases where these tables are located are all prefixed with "sharding"; and there is no conflict between the primary keys or the unique keys (in each sharded table, the primary keys or the unique keys are different from those of other tables).

Now, suppose that you need to migrate these sharded tables to the `db_target.t_target` table in TiDB. The steps are as follows.

1. Create the configuration file of the task:

```
---
name: test
task-mode: all
shard-mode: "pessimistic"
target-database:
  host: "127.0.0.1"
  port: 4000
  user: "root"
  password: "" # It is recommended to use password encrypted with dmctl
     ↪  if the password is not empty.

mysql-instances:
  - source-id: "mysql-replica-01"
    block-allow-list: "instance" # This configuration applies to DM
       ↪ versions higher than v2.0.0-beta.2. Use black-white-list
       ↪ otherwise.
    route-rules: ["sharding-route-rules-table", "sharding-route-rules-
       ↪ schema"]
    mydumper-thread: 4
    loader-thread: 16
    syncer-thread: 16
  - source-id: "mysql-replica-02"
    block-allow-list: "instance" # This configuration applies to DM
       ↪ versions higher than v2.0.0-beta.2. Use black-white-list
       ↪ otherwise.
    route-rules: ["sharding-route-rules-table", "sharding-route-rules-
       ↪ schema"]
    mydumper-thread: 4
    loader-thread: 16
    syncer-thread: 16
block-allow-list: # This configuration applies to DM versions higher
    ↪ than v2.0.0-beta.2. Use black-white-list otherwise.
  instance:
    do-dbs: ["~^sharding[\\d]+"]
    do-tables:
    - db-name: "~^sharding[\\d]+"
      tbl-name: "~^t[\\d]+"
routes:
  sharding-route-rules-table:
    schema-pattern: sharding*
    table-pattern: t*
    target-schema: db_target
    target-table: t_target
```

```
  sharding-route-rules-schema:
    schema-pattern: sharding*
    target-schema: db_target
```

2. To create a task using dmctl, write the above configurations to the `conf/task.yaml` file:

```
./dmctl --master-addr 127.0.0.1:8261 start-task conf/task.yaml
```

```
{
    "result": true,
    "msg": "",
    "sources": [
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-01",
            "worker": "worker1"
        },
        {
            "result": true,
            "msg": "",
            "source": "mysql-replica-02",
            "worker": "worker2"
        }
    ]
}
```

Now, you have successfully created a task to migrate the sharded tables from the MySQL1 and MySQL2 instances to TiDB.

Verify data

You can modify data in the upstream MySQL sharded tables. Then use sync-diff-inspector to check whether the upstream and downstream data are consistent. Consistent data means that the migration task works well, which also indicates that the cluster works well.

13.6.8.11.3 Best Practices of Data Migration in the Shard Merge Scenario

This document describes the features and limitations of TiDB Data Migration (DM) in the shard merge scenario and provides a data migration best practice guide for your application (the default "pessimistic" mode is used).

Use a separate data migration task

In the Merge and Migrate Data from Sharded Tables document, the definition of "sharding group" is given: A sharding group consists of all upstream tables that need to be merged and migrated into the same downstream table.

The current sharding DDL mechanism has some usage restrictions to coordinate the schema changes brought by DDL operations in different sharded tables. If these restrictions are violated due to unexpected reasons, you need to handle sharding DDL locks manually in DM, or even redo the entire data migration task.

To mitigate the impact on data migration when an exception occurs, it is recommended to merge and migrate each sharding group as a separate data migration task. **This might enable that only a small number of data migration tasks need to be handled manually while others remain unaffected.**

Handle sharding DDL locks manually

You can easily conclude from Merge and Migrate Data from Sharded Tables that DM's sharding DDL lock is a mechanism for coordinating the execution of DDL operations to the downstream from multiple upstream sharded tables.

Therefore, when you find any sharding DDL lock on `DM-master` through `shard-ddl-lock` ↪ command, or any `unresolvedGroups` or `blockingDDLs` on some DM-workers through `query-status` command, do not rush to manually release the sharding DDL lock through `shard-ddl-lock unlock` commands.

Instead, you can:

- Follow the corresponding manual solution to handle the scenario if the failure of automatically releasing the sharding DDL lock is one of the listed abnormal scenarios.
- Redo the entire data migration task if it is an unsupported scenario: First, empty the data in the downstream database and the `dm_meta` information associated with the migration task; then, re-execute the full and incremental data replication.

Handle conflicts between primary keys or unique indexes across multiple sharded tables

Data from multiple sharded tables might cause conflicts between the primary keys or unique indexes. You need to check each primary key or unique index based on the sharding logic of these sharded tables. The following are three cases related to primary keys or unique indexes:

- Shard key: Usually, the same shard key only exists in one sharded table, which means no data conflict is caused on shard key.
- Auto-increment primary key: The auto-increment primary key of each sharded tables counts separately, so their range might overlap. In this case, you need to refer to the next section Handle conflicts of auto-increment primary key to solve it.
- Other primary keys or unique indexes: you need to analyze them based on the business logic. If data conflict, you can also refer to the next section Handle conflicts of auto-increment primary key to solve it.

Handle conflicts of auto-increment primary key

This section introduces two recommended solutions to handle conflicts of auto-increment primary key.

Remove the `PRIMARY KEY` attribute from the column

Assume that the upstream schemas are as follows:

```
CREATE TABLE `tbl_no_pk` (
  `auto_pk_c1` bigint(20) NOT NULL,
  `uk_c2` bigint(20) NOT NULL,
  `content_c3` text,
  PRIMARY KEY (`auto_pk_c1`),
  UNIQUE KEY `uk_c2` (`uk_c2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

If the following requirements are satisfied:

- The `auto_pk_c1` column has no impact on the application and does not depend on the column's `PRIMARY KEY` attribute.
- The `uk_c2` column has the `UNIQUE KEY` attribute, and it is globally unique in all upstream sharded tables.

Then you can perform the following steps to fix the `ERROR 1062 (23000): Duplicate entry '***' for key 'PRIMARY'` error that is possibly caused by the `auto_pk_c1` column when you merge sharded tables.

1. Before the full data migration, create a table in the downstream database for merging and migrating data, and modify the `PRIMARY KEY` attribute of the `auto_pk_c1` column to normal index.

   ```
   CREATE TABLE `tbl_no_pk_2` (
     `auto_pk_c1` bigint(20) NOT NULL,
     `uk_c2` bigint(20) NOT NULL,
     `content_c3` text,
     INDEX (`auto_pk_c1`),
     UNIQUE KEY `uk_c2` (`uk_c2`)
   ) ENGINE=InnoDB DEFAULT CHARSET=latin1
   ```

2. Add the following configuration in `task.yaml` to skip the check of auto-increment primary key conflict:

   ```
   ignore-checking-items: ["auto_increment_ID"]
   ```

3. Start the full and incremental data replication task.

4. Run `query-status` to verify whether the data migration task is successfully processed and whether the data from the upstream has already been merged and migrated to the downstream database.

Use a composite primary key

Assume that the upstream schemas are as follows:

```
CREATE TABLE `tbl_multi_pk` (
  `auto_pk_c1` bigint(20) NOT NULL,
  `uuid_c2` bigint(20) NOT NULL,
  `content_c3` text,
  PRIMARY KEY (`auto_pk_c1`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

If the following requirements are satisfied:

- The application does not depend on the `PRIMARY KEY` attribute of the `auto_pk_c1` column.
- The composite primary key that consists of the `auto_pk_c1` and `uuid_c2` columns is globally unique.
- It is acceptable to use a composite primary key in the application.

Then you can perform the following steps to fix the `ERROR 1062 (23000): Duplicate entry '***' for key 'PRIMARY'` error that is possibly caused by the `auto_pk_c1` column when you merge sharded tables.

1. Before the full data migration, create a table in the downstream database for merging and migrating data. Do not specify the `PRIMARY KEY` attribute for the `auto_pk_c1` column, but use the `auto_pk_c1` and `uuid_c2` columns to make up a composite primary key.

   ```
   CREATE TABLE `tbl_multi_pk_c2` (
     `auto_pk_c1` bigint(20) NOT NULL,
     `uuid_c2` bigint(20) NOT NULL,
     `content_c3` text,
     PRIMARY KEY (`auto_pk_c1`,`uuid_c2`)
   ) ENGINE=InnoDB DEFAULT CHARSET=latin1
   ```

2. Start the full and incremental data migration task.

3. Run `query-status` to verify whether the data migration task is successfully processed and whether the data from upstream has already been merged and migrated to the downstream database.

Special processing when the upstream RDS contains sharded tables

If the upstream data source is an RDS and it contains sharded tables, the table names in MySQL binlog might be invisible when connecting to a SQL client. For example, if the upstream is a UCloud distributed database, the table name in the binlog might have an extra prefix `_0001`. Therefore, you need to configure table routing based on the table names in binlog, instead of those in the SQL client.

Create/drop tables in the upstream

In Merge and Migrate Data from Sharded Tables, it is clear that the coordination of sharding DDL lock depends on whether the downstream database receives the DDL statements of all upstream sharded tables. In addition, DM currently **does not support** dynamically creating or dropping sharded tables in the upstream. Therefore, to create or drop sharded tables in the upstream, it is recommended to perform the following steps.

Create sharded tables in the upstream

If you need to create a new sharded table in the upstream, perform the following steps:

1. Wait for the coordination of all executed sharding DDL in the upstream sharded tables to finish.
2. Run `stop-task` to stop the data migration task.
3. Create a new sharded table in the upstream.
4. Make sure that the configuration in the `task.yaml` file allows the newly added sharded table to be merged in one downstream table with other existing sharded tables.
5. Run `start-task` to start the task.
6. Run `query-status` to verify whether the data migration task is successfully processed and whether the data from upstream has already been merged and migrated to the downstream database.

Drop sharded tables in the upstream

If you need to drop a sharded table in the upstream, perform the following steps:

1. Drop the sharded table, run `SHOW BINLOG EVENTS` to fetch the `End_log_pos` corresponding to the `DROP TABLE` statement in the binlog events, and mark it as *Pos-M*.
2. Run `query-status` to fetch the position (`syncerBinlog`) corresponding to the binlog event that has been processed by DM, and mark it as *Pos-S*.
3. When *Pos-S* is greater than *Pos-M*, it means that DM has processed all of the `DROP TABLE` statements, and the data of the table before dropping has been migrated to the downstream, so the subsequent operation can be performed. Otherwise, wait for DM to finish migrating the data.

4. Run `stop-task` to stop the task.
5. Make sure that the configuration in the `task.yaml` file ignores the dropped sharded table in the upstream.
6. Run `start-task` to start the task.
7. Run `query-status` to verify whether the data migration task is successfully processed.

Speed limits and traffic flow control

When data from multiple upstream MySQL or MariaDB instances is merged and migrated to the same TiDB cluster in the downstream, every DM-worker corresponding to each upstream instance executes full and incremental data replication concurrently. This means that the default degree of concurrency (`pool-size` in full data migration and `worker-count` in incremental data replication) accumulates as the number of DM-workers increases, which might overload the downstream database. In this case, you need to conduct a preliminary performance analysis based on TiDB and DM monitoring metrics and adjust the value of each concurrency parameter. In the future, DM is expected to support partially automated traffic flow control.

13.6.8.12 Troubleshoot

13.6.8.12.1 TiDB Data Migration FAQs

This document collects the frequently asked questions (FAQs) about TiDB Data Migration (DM).

Does DM support migrating data from Alibaba RDS or other cloud databases?

Currently, DM only supports decoding the standard version of MySQL or MariaDB binlog. It has not been tested for Alibaba Cloud RDS or other cloud databases. If you are confirmed that its binlog is in standard format, then it is supported.

It is a known issue that for an upstream table with no primary key in Alibaba Cloud RDS, its binlog still contains a hidden primary key column, which is inconsistent with the original table structure.

Here are some known incompatible issues:

- In **Alibaba Cloud RDS**, for an upstream table with no primary key, its binlog still contains a hidden primary key column, which is inconsistent with the original table structure.
- In **HUAWEI Cloud RDS**, directly reading binlog files is not supported. For more details, see Can HUAWEI Cloud RDS Directly Read Binlog Backup Files?

Does the regular expression of the block and allow list in the task configuration support `non-capturing (?!)`?

Currently, DM does not support it and only supports the regular expressions of the Golang standard library. See regular expressions supported by Golang via re2-syntax.

If a statement executed upstream contains multiple DDL operations, does DM support such migration?

DM will attempt to split a single statement containing multiple DDL change operations into multiple statements containing only one DDL operation, but might not cover all cases. It is recommended to include only one DDL operation in a statement executed upstream, or verify it in the test environment. If it is not supported, you can file an issue to the DM repository.

How to handle incompatible DDL statements?

When you encounter a DDL statement unsupported by TiDB, you need to manually handle it using dmctl (skipping the DDL statement or replacing the DDL statement with a specified DDL statement). For details, see Handle failed DDL statements.

> **Note:**
>
> Currently, TiDB is not compatible with all the DDL statements that MySQL supports. See MySQL Compatibility.

Does DM replicate view-related DDL statements and DML statements to TiDB?

Currently, DM does not replicate view-related DDL statements to the downstream TiDB cluster, nor does it replicate view-related DML statements to the downstream TiDB cluster.

How to reset the data migration task?

When an exception occurs during data migration and the data migration task cannot be resumed, you need to reset the task and re-migrate the data:

1. Execute the `stop-task` command to stop the abnormal data migration task.
2. Purge the data migrated to the downstream.
3. Use one of the following ways to restart the data migration task.

- Specify a new task name in the task configuration file. Then execute `start-task {task-config-file}`.
- Execute `start-task --remove-meta {task-config-file}`.

How to handle the error returned by the DDL operation related to the gh-ost table, after `online-ddl-scheme: "gh-ost"` is set?

```
[unit=Sync] ["error information"="{\"msg\":\"[code=36046:class=sync-unit:
    ↪ scope=internal:level=high] online ddls on ghost table `xxx`.`
    ↪ _xxxx_gho`\\ngithub.com/pingcap/dm/pkg/terror.(*Error).Generate
    ↪ ......
```

The above error can be caused by the following reason:

In the last `rename ghost_table to origin table` step, DM reads the DDL information in memory, and restores it to the DDL of the origin table.

However, the DDL information in memory is obtained in either of the two ways:

- DM processes the gh-ost table during the `alter ghost_table` operation and records the DDL information of `ghost_table`;
- When DM-worker is restarted to start the task, DM reads the DDL from `dm_meta.{task_name}_onlineddl`.

Therefore, in the process of incremental replication, if the specified Pos has skipped the `alter ghost_table` DDL but the Pos is still in the online-ddl process of gh-ost, the ghost_table is not written into memory or `dm_meta.{task_name}_onlineddl` correctly. In such cases, the above error is returned.

You can avoid this error by the following steps:

1. Remove the `online-ddl-scheme` configuration of the task.
2. Configure `_{table_name}_gho`, `_{table_name}_ghc`, and `_{table_name}_del` in `block-allow-list.ignore-tables`.
3. Execute the upstream DDL in the downstream TiDB manually.
4. After the Pos is replicated to the position after the gh-ost process, re-enable the `online-ddl-scheme` and comment out `block-allow-list.ignore-tables`.

How to add tables to the existing data migration tasks?

If you need to add tables to a data migration task that is running, you can address it in the following ways according to the stage of the task.

> **Note:**
>
> Because adding tables to an existing data migration task is complex, it is recommended that you perform this operation only when necessary.

In the `Dump` stage

Since MySQL cannot specify a snapshot for export, it does not support updating data migration tasks during the export and then restarting to resume the export through the checkpoint. Therefore, you cannot dynamically add tables that need to be migrated at the `Dump` stage.

If you really need to add tables for migration, it is recommended to restart the task directly using the new configuration file.

In the `Load` stage

During the export, multiple data migration tasks usually have different binlog positions. If you merge the tasks in the `Load` stage, they might not be able to reach consensus on binlog positions. Therefore, it is not recommended to add tables to a data migration task in the `Load` stage.

In the `Sync` stage

When the data migration task is in the `Sync` stage, if you add additional tables to the configuration file and restart the task, DM does not re-execute full export and import for the newly added tables. Instead, DM continues incremental replication from the previous checkpoint.

Therefore, if the full data of the newly added table has not been imported to the downstream, you need to use a separate data migration task to export and import the full data to the downstream.

Record the position information in the global checkpoint (`is_global=1`) corresponding to the existing migration task as `checkpoint-T`, such as `(mysql-bin.000100, 1234)`. Record the position information of the full export `metedata` (or the checkpoint of another data migration task in the `Sync` stage) of the table to be added to the migration task as `checkpoint-S`, such as `(mysql-bin.000099, 5678)`. You can add the table to the migration task by the following steps:

1. Use `stop-task` to stop an existing migration task. If the table to be added belongs to another running migration task, stop that task as well.
2. Use a MySQL client to connect the downstream TiDB database and manually update the information in the checkpoint table corresponding to the existing migration task to the smaller value between `checkpoint-T` and `checkpoint-S`. In this example, it is `(mysql- bin.000099, 5678)`.
    - The checkpoint table to be updated is `{task-name}_syncer_checkpoint` in the `{dm_meta}` schema.
    - The checkpoint rows to be updated match `id=(source-id)` and `is_global=1`.
    - The checkpoint columns to be updated are `binlog_name` and `binlog_pos`.
3. Set `safe-mode: true` for the `syncers` in the task to ensure reentrant execution.

4. Start the task using `start-task`.

5. Observe the task status through `query-status`. When `syncerBinlog` exceeds the larger value of `checkpoint-T` and `checkpoint-S`, restore `safe-mode` to the original value and restart the task. In this example, it is `(mysql-bin.000100, 1234)`.

How to handle the error `packet for query is too large. Try adjusting the 'max_allowed_packet' variable` that occurs during the full import?

Set the parameters below to a value larger than the default 67108864 (64M).

- The global variable of the TiDB server: `max_allowed_packet`.
- The configuration item in the task configuration file: `target-database.max-allowed-packet`. For details, refer to DM Advanced Task Configuration File.

How to handle the error `Error 1054: Unknown column 'binlog_gtid' in 'field list'` that occurs when existing DM migration tasks of an DM 1.0 cluster are running on a DM 2.0 or newer cluster?

Since DM v2.0, if you directly run the `start-task` command with the task configuration file of the DM 1.0 cluster to continue the incremental data replication, the error `Error 1054: Unknown column 'binlog_gtid' in 'field list'` occurs.

This error can be handled by manually importing DM migration tasks of a DM 1.0 cluster to a DM 2.0 cluster.

Why does TiUP fail to deploy some versions of DM (for example, v2.0.0-hotfix)?

You can use the `tiup list dm-master` command to view the DM versions that TiUP supports to deploy. TiUP does not manage DM versions which are not shown by this command.

How to handle the error `parse mydumper metadata error: EOF` that occurs when DM is replicating data?

You need to check the error message and log files to further analyze this error. The cause might be that the dump unit does not produce the correct metadata file due to a lack of permissions.

Why does DM report no fatal error when replicating sharded schemas and tables, but downstream data is lost?

Check the configuration items `block-allow-list` and `table-route`:

- You need to configure the names of upstream databases and tables under `block-allow-list`. You can add "~" before `do-tables` to use regular expressions to match names.
- `table-route` uses wildcard characters instead of regular expressions to match table names. For example, `table_parttern_[0-63]` only matches 7 tables, from `table_parttern_0` to `table_pattern_6`.

Why does the `replicate lag` monitor metric show no data when DM is not replicating from upstream?

In DM 1.0, you need to enable `enable-heartbeat` to generate the monitor data. In DM 2.0 and later versions, it is expected to have no data in the monitor metric `replicate lag` because this feature is not supported.

How to handle the error `fail to initial unit Sync of subtask` when DM is starting a task, with the `RawCause` in the error message showing `context deadline exceeded`?

This is a known issue in DM 2.0.0 version and will be fixed in DM 2.0.1 version. It is likely to be triggered when a replication task has a lot of tables to process. If you use TiUP to deploy DM, you can upgrade DM to the nightly version to fix this issue. Or you can download the 2.0.0-hotfix version from the release page of DM on GitHub and manually replace the executable files.

How to handle the error `duplicate entry` when DM is replicating data?

You need to first check and confirm the following things:

- `disable-detect` is not configured in the replication task (in v2.0.7 and earlier versions).
- The data is not inserted manually or by other replication programs.
- No DML filter associated with this table is configured.

To facilitate troubleshooting, you can first collect general log files of the downstream TiDB instance and then ask for technical support at TiDB Community slack channel. The following example shows how to collect general log files:

```
##### Enable general log collection
curl -X POST -d "tidb_general_log=1" http://{TiDBIP}:10080/settings
##### Disable general log collection
curl -X POST -d "tidb_general_log=0" http://{TiDBIP}:10080/settings
```

When the `duplicate entry` error occurs, you need to check the log files for the records that contain conflict data.

Why do some monitoring panels show `No data point`?

It is normal for some panels to have no data. For example, when there is no error reported, no DDL lock, or the relay log feature is not enabled, the corresponding panels show `No data point`. For detailed description of each panel, see DM Monitoring Metrics.

In DM v1.0, why does the command `sql-skip` fail to skip some statements when the task is in error?

You need to first check whether the binlog position is still advancing after you execute `sql-skip`. If so, it means that `sql-skip` has taken effect. The reason why this error keeps occurring is that the upstream sends multiple unsupported DDL statements. You can use `sql-skip -s <sql-pattern>` to set a pattern to match these statements.

Sometimes, the error message contains the `parse statement` information, for example:

```
if the DDL is not needed, you can use a filter rule with \"*\" schema-
    ↪ pattern to ignore it.\n\t : parse statement: line 1 column 11 near \"
    ↪ EVENT `event_del_big_table` \r\nDISABLE\" %!!(MISSING)(EXTRA string=
    ↪ ALTER EVENT `event_del_big_table` \r\nDISABLE
```

The reason for this type of error is that the TiDB parser cannot parse DDL statements sent by the upstream, such as `ALTER EVENT`, so `sql-skip` does not take effect as expected. You can add binlog event filters in the configuration file to filter those statements and set `schema-pattern: "*"`. Starting from DM v2.0.1, DM pre-filters statements related to `EVENT`.

Since DM v6.0, `binlog` replaces `sql-skip` and `handle-error`. You can use the `binlog` command instead to avoid this issue.

Why do `REPLACE` statements keep appearing in the downstream when DM is replicating?

You need to check whether the safe mode is automatically enabled for the task. If the task is automatically resumed after an error, or if there is high availability scheduling, then the safe mode is enabled because it is within 1 minutes after the task is started or resumed.

You can check the DM-worker log file and search for a line containing `change count`. If the `new count` in the line is not zero, the safe mode is enabled. To find out why it is enabled, check when it happens and if any errors are reported before.

In DM v2.0, why does the full import task fail if DM restarts during the task?

In DM v2.0.1 and earlier versions, if DM restarts before the full import completes, the bindings between upstream data sources and DM-worker nodes might change. For example, it is possible that the intermediate data of the dump unit is on DM-worker node A but the load unit is run by DM-worker node B, thus causing the operation to fail.

The following are two solutions to this issue:

- If the data volume is small (less than 1 TB) or the task merges sharded tables, take these steps:

    1. Clean up the imported data in the downstream database.
    2. Remove all files in the directory of exported data.
    3. Delete the task using dmctl and run the command `start-task --remove-meta` to create a new task.

    After the new task starts, it is recommended to ensure that there is no redundant DM worker node and avoid restarting or upgrading the DM cluster during the full import.

- If the data volume is large (more than 1 TB), take these steps:

    1. Clean up the imported data in the downstream database.
    2. Deploy TiDB-Lightning to the DM worker nodes that process the data.

3. Use the Local-backend mode of TiDB-Lightning to import data that DM dump units export.
4. After the full import completes, edit the task configuration file in the following ways and restart the task:
    - Change `task-mode` to `incremental`.
    - Set the value of `mysql-instance.meta.pos` to the position recorded in the metadata file that the dump unit outputs.

Why does DM report the error `ERROR 1236 (HY000): The slave is connecting using CHANGE MASTER TO MASTER_AUTO_POSITION = 1, but the master has purged binary logs containing GTIDs that the slave requires.` if it restarts during an incremental task?

This error indicates that the upstream binlog position recorded in the metadata file output by the dump unit has been purged during the full migration.

If this issue occurs, you need to pause the task, delete all migrated data in the downstream database, and start a new task with the `--remove-meta` option.

You can avoid this issue in advance by configuring in the following ways:

1. Increase the value of `expire_logs_days` in the upstream MySQL database to avoid wrongly purging needed binlog files before the full migration task completes. If the data volume is large, it is recommended to use dumpling and TiDB-Lightning at the same time to speed up the task.
2. Enable the relay log feature for this task so that DM can read data from relay logs even though the binlog position is purged.

Why does the Grafana dashboard of a DM cluster display `failed to fetch dashboard` if the cluster is deployed using TiUP v1.3.0 or v1.3.1?

This is a known bug of TiUP, which is fixed in TiUP v1.3.2. The following are two solutions to this issue:

- Solution one:
    1. Upgrade TiUP to a later version using the command `tiup update --self && tiup update dm`.
    2. Scale in and then scale out Grafana nodes in the cluster to restart the Grafana service.
- Solution two:
    1. Back up the `deploy/grafana-$port/bin/public` folder.
    2. Download the TiUP DM offline package and unpack it.
    3. Unpack the `grafana-v4.0.3-**.tar.gz` in the offline package.
    4. Replace the folder `deploy/grafana-$port/bin/public` with the `public` folder in `grafana-v4.0.3-**.tar.gz`.

5. Execute `tiup dm restart $cluster_name -R grafana` to restart the Grafana service.

In DM v2.0, why does the query result of the command `query-status` show that the Syncer checkpoint GTIDs are inconsecutive if the task has `enable-relay` and `enable-gtid` enabled at the same time?

This is a known bug in DM, which is fixed in DM v2.0.2. The bug is triggered when the following two conditions are fully met at the same time:

1. Parameters `enable-relay` and `enable-gtid` are set to `true` in the source configuration file.
2. The upstream database is a **MySQL secondary database**. If you execute the command `show binlog events in '<newest-binlog>' limit 2` to query the `previous_gtids` of the database, the result is inconsecutive, such as the following example:

```
mysql> show binlog events in 'mysql-bin.000005' limit 2;
+------------------+------+---------------+-----------+-------------+------------------
   ↪
| Log_name       | Pos | Event_type   | Server_id | End_log_pos | Info
   ↪                                                  |
+------------------+------+---------------+-----------+-------------+------------------
   ↪
| mysql-bin.000005 | 4 | Format_desc  |    123452 |       123 | Server ver:
   ↪ 5.7.32-35-log, Binlog ver: 4                    |
| mysql-bin.000005 | 123 | Previous_gtids | 123452 |     194 | d3618e68
   ↪ -6052-11eb-a68b-0242ac110002:6-7                |
+------------------+------+---------------+-----------+-------------+------------------
   ↪
```

The bug occurs if you run `query-status <task>` in dmctl to query task information and find that `subTaskStatus.sync.syncerBinlogGtid` is inconsecutive but `subTaskStatus` ↪ `.sync.masterBinlogGtid` is consecutive. See the following example:

```
query-status test
{
    ...
    "sources": [
        {
            ...
            "sourceStatus": {
                "source": "mysql1",
                ...
                "relayStatus": {
```

```
            "masterBinlog": "(mysql-bin.000006, 744)",
            "masterBinlogGtid": "f8004e25-6067-11eb-9fa3-0242ac110003
                ↪ :1-50",
            ...
        }
    },
    "subTaskStatus": [
        {
            ...
            "sync": {
                ...
                "masterBinlog": "(mysql-bin.000006, 744)",
                "masterBinlogGtid": "f8004e25-6067-11eb-9fa3-0242
                    ↪ ac110003:1-50",
                "syncerBinlog": "(mysql-bin|000001.000006, 738)",
                "syncerBinlogGtid": "f8004e25-6067-11eb-9fa3-0242
                    ↪ ac110003:1-20:40-49",
                ...
                "synced": false,
                "binlogType": "local"
            }
        }
    ]
},
{
    ...
    "sourceStatus": {
        "source": "mysql2",
        ...
        "relayStatus": {
            "masterBinlog": "(mysql-bin.000007, 1979)",
            "masterBinlogGtid": "ddb8974e-6064-11eb-8357-0242ac110002
                ↪ :1-25",
            ...
        }
    },
    "subTaskStatus": [
        {
            ...
            "sync": {
                "masterBinlog": "(mysql-bin.000007, 1979)",
                "masterBinlogGtid": "ddb8974e-6064-11eb-8357-0242
                    ↪ ac110002:1-25",
                "syncerBinlog": "(mysql-bin|000001.000008, 1979)",
                "syncerBinlogGtid": "ddb8974e-6064-11eb-8357-0242
```

```
                    ↪ ac110002:1-25",
                ...
                "synced": true,
                "binlogType": "local"
            }
        }
    ]
  }
]
}
```

In the example, the `syncerBinlogGtid` of the data source `mysql1` is inconsecutive. In this case, you can do one of the following to handle the data loss:

- If upstream binlogs from the current time to the position recorded in the metadata of the full export task have not been purged, you can take these steps:
    1. Stop the current task and delete all data sources with inconsecutive GTIDs.
    2. Set `enable-relay` to `false` in all source configuration files.
    3. For data sources with inconsecutive GTIDs (such as `mysql1` in the above example), change the task to an incremental task and configure related `mysql-instances.meta` with metadata information of each full export task, including the `binlog-name`, `binlog-pos`, and `binlog-gtid` information.
    4. Set `syncers.safe-mode` to `true` in `task.yaml` of the incremental task and restart the task.
    5. After the incremental task replicates all missing data to the downstream, stop the task and change `safe-mode` to `false` in the `task.yaml`.
    6. Restart the task again.
- If upstream binlogs have been purged but local relay logs remain, you can take these steps:
    1. Stop the current task.
    2. For data sources with inconsecutive GTIDs (such as `mysql1` in the above example), change the task to an incremental task and configure related `mysql-instances.meta` with metadata information of each full export task, including the `binlog-name`, `binlog-pos`, and `binlog-gtid` information.
    3. In the `task.yaml` of the incremental task, change the previous value of `binlog-gtid` to the previous value of `previous_gtids`. For the above example, change `1-y` to `6-y`.
    4. Set `syncers.safe-mode` to `true` in the `task.yaml` and restart the task.
    5. After the incremental task replicates all missing data to the downstream, stop the task and change `safe-mode` to `false` in the `task.yaml`.
    6. Restart the task again.
    7. Restart the data source and set either `enable-relay` or `enable-gtid` to `false` in the source configuration file.

- If none of the above conditions is met or if the data volume of the task is small, you can take these steps:

    1. Clean up imported data in the downstream database.
    2. Restart the data source and set either `enable-relay` or `enable-gtid` to `false` in the source configuration file.
    3. Create a new task and run the command `start-task task.yaml --remove-meta` to migrate data from the beginning again.

For data sources that can be replicated normally (such as `mysql2` in the above example) in the first and second solutions above, configure related `mysql-instances.meta` with `syncerBinlog` and `syncerBinlogGtid` information from `subTaskStatus.sync` when setting the incremental task.

In DM v2.0, how do I handle the error "heartbeat config is different from previous used: serverID not equal" when switching the connection between DM-workers and MySQL instances in a virtual IP environment with the `heartbeat` feature enabled?

The `heartbeat` feature is disabled by default in DM v2.0 and later versions. If you enable the feature in the task configuration file, it interferes with the high availability feature. To solve this issue, you can disable the `heartbeat` feature by setting `enable-heartbeat` to `false` in the task configuration file, and then reload the task configuration file. DM will forcibly disable the `heartbeat` feature in subsequent releases.

Why does a DM-master fail to join the cluster after it restarts and DM reports the error "fail to start embed etcd, RawCause: member xxx has already been bootstrapped"?

When a DM-master starts, DM records the etcd information in the current directory. If the directory changes after the DM-master restarts, DM cannot get access to the etcd information, and thus the restart fails.

To solve this issue, you are recommended to maintain DM clusters using TiUP. In the case that you need to deploy using binary files, you need to configure `data-dir` with absolute paths in the configuration file of the DM-master, or pay attention to the current directory where you run the command.

Why DM-master cannot be connected when I use dmctl to execute commands?

When using dmctl execute commands, you might find the connection to DM master fails (even if you have specified the parameter value of `--master-addr` in the command), and the error message is like `RawCause: context deadline exceeded, Workaround: please check your network connection.`. But afer checking the network connection using commands like `telnet <master-addr>`, no exception is found.

In this case, you can check the environment variable `https_proxy` (note that it is **https**). If this variable is configured, dmctl automatically connects the host and port specified by `https_proxy`. If the host does not have a corresponding `proxy` forwarding service, the connection fails.

To solve this issue, check whether `https_proxy` is mandatory. If not, cancel the setting. Otherwise, add the environment variable setting `https_proxy="" ./dmctl --master-addr "x.x.x.x:8261"` before the oringial dmctl commands.

> **Note:**
>
> The environment variables related to `proxy` include `http_proxy`, `https_proxy`, and `no_proxy`. If the connection error persists after you perform the above steps, check whether the configuration parameters of `http_proxy` and `no_proxy` are correct.

How to handle the returned error when executing start-relay command for DM versions from 2.0.2 to 2.0.6?

```
flush local meta, Rawcause: open relay-dir/xxx.000001/relay.metayyyy: no such file or directory
```

The above error might be made in the following cases:

- DM has been upgraded from v2.0.1 and earlier to v2.0.2 - v2.0.6, and relay log is started before the upgrade and restarted after the upgrade.
- Execute the stop-relay command to pause the relay log and then restart it.

You can avoid this error by the following options:

- Restart relay log:

```
» stop-relay -s sourceID workerName
» start-relay -s sourceID workerName
```

- Upgrade DM to v2.0.7 or later versions.

13.6.8.12.2 Handle Errors in TiDB Data Migration

This document introduces the error system and how to handle common errors when you use DM.

Error system

In the error system, usually, the information of a specific error is as follows:

- `code`: error code.

  DM uses the same error code for the same error type. An error code does not change as the DM version changes.

  Some errors might be removed during the DM iteration, while the error codes are not. DM uses a new error code instead of an existing one for a new error.

- `class`: error type.

  It is used to mark the component where an error occurs (error source).

  The following table displays all error types, error sources, and error samples.

  |

  Error Type

Error Source	Error Sample
database	Database operations	[code=10003:class=database:scope=downstream:level=medium] database driver: invalid connection
functional	Underlying functions of DM	[code=11005:class=functional:scope=internal:level=high] not allowed operation: alter multiple tables in one statement
config	Incorrect configuration	[code=20005:class=config:scope=internal:level=medium] empty source-id not valid
binlog-op	Binlog operations	[code=22001:class=binlog-op:scope=internal:level=high] empty UUIDs not valid
checkpoint	checkpoint operations	[code=24002:class=checkpoint:scope=internal:level=high] save point bin.1234 is older than current pos bin.1371
task-check	Performing task check	[code=26003:class=task-check:scope=internal:level=medium] new table router error
relay-event-lib	Executing the basic functions of the relay module	[code=28001:class=relay-event-lib:scope=internal:level=high] parse server-uuid.index
relay-unit	relay processing unit	[code=30015:class=relay-unit:scope=upstream:level=high] TCPReader get event: ERROR 1236 (HY000): Could not open log file
dump-unit	dump processing unit	[code=32001:class=dump-unit:scope=internal:level=high] mydumper runs with error: CRITICAL **: 15:12:17.559: Error connecting to database: Access denied for user 'root'@'172.17.0.1' (using password: NO)
load-unit	load processing unit	[code=34002:class=load-unit:scope=internal:level=high] corresponding ending of sql: ')' not found

```
sync-unit | sync processing unit | [code=36027:class=sync-unit:scope=internal
↪ :level=high] Column count doesn't match value count: 9 (columns)vs
↪ 10 (values) |
dm-master | DM-master service | [code=38008:class=dm-master:scope=internal
↪ :level=high] grpc request error: rpc error: code = Unavailable desc
↪  = all SubConns are in TransientFailure, latest connection error:
↪ connection error: desc = "transport: Error while dialing dial tcp
↪ 172.17.0.2:8262: connect: connection refused" |
dm-worker | DM-worker service | [code=40066:class=dm-worker:scope=internal
↪ :level=high] ExecuteDDL timeout, try use query-status to query
↪ whether the DDL is still blocking |
dm-tracer | DM-tracer service | [code=42004:class=dm-tracer:scope=internal:
↪ level=medium] trace event test.1 not found |
schema-tracker | schema-tracker (during incremental data replication) | [code
↪ =44006:class=schema-tracker:scope=internal:level=high],"cannot track
↪  DDL: ALTER TABLE test DROP COLUMN col1" |
scheduler | Scheduling operations (of data migration tasks) | [code=46001:class=
↪ scheduler:scope=internal:level=high],"the scheduler has not started"
|
dmctl | An error occurs within dmctl or when it interacts with other components |
[code=48001:class=dmctl:scope=internal:level=high],"can not create grpc
↪  connection" |
```

- `scope`: Error scope.

  It is used to mark the scope and source of DM objects when an error occurs. `scope` includes four types: `not-set`, `upstream`, `downstream`, and `internal`.

  If the logic of the error directly involves requests between upstream and downstream databases, the scope is set to `upstream` or `downstream`; otherwise, it is currently set to `internal`.

- `level`: Error level.

  The severity level of the error, including `low`, `medium`, and `high`.

  - The `low` level error usually relates to user operations and incorrect inputs. It does not affect migration tasks.
  - The `medium` level error usually relates to user configurations. It affects some newly started services; however, it does not affect the existing DM migration status.
  - The `high` level error usually needs your attention, since you need to resolve it to avoid the possible interruption of a migration task.

- `message`: Error descriptions.

  Detailed descriptions of the error. To wrap and store every additional layer of error message on the error call chain, the errors.Wrap mode is adopted. The message description wrapped at the outermost layer indicates the error in DM and the message description wrapped at the innermost layer indicates the error source.

- `workaround`: Error handling methods (optional)

  The handling methods for this error. For some confirmed errors (such as configuration errors), DM gives the corresponding manual handling methods in `workaround`.

- Error stack information (optional)

  Whether DM outputs the error stack information depends on the error severity and the necessity. The error stack records the complete stack call information when the error occurs. If you cannot figure out the error cause based on the basic information and the error message, you can trace the execution path of the code when the error occurs using the error stack.

For the complete list of error codes, refer to the error code lists.

Troubleshooting

If you encounter an error while running DM, take the following steps to troubleshoot this error:

1. Execute the `query-status` command to check the task running status and the error output.
2. Check the log files related to the error. The log files are on the DM-master and DM-worker nodes. To get key information about the error, refer to the error system. Then check the Handle Common Errors section to find the solution.
3. If the error is not covered in this document, and you cannot solve the problem by checking the log or monitoring metrics, get support from PingCAP or the community.
4. After the error is resolved, restart the task using dmctl.

```
resume-task ${task name}
```

However, you need to reset the data migration task in some cases. For details, refer to Reset the Data Migration Task.

Handle common errors

|

Error Code

```
| Error Description                                                    |  How to
    ↪ Handle                                                  |
```

:———— | :———————————————————————— | :————————————————————————————————— |

`code=10001` | Abnormal database operation. | Further analyze the error message and error stack. |

`code=10002` | The `bad connection` error from the underlying database. It usually indicates that the connection between DM and the downstream TiDB instance is abnormal (possibly caused by network failure or TiDB restart) and the currently requested data is not sent to TiDB. | DM provides automatic recovery for such error. If the recovery is not successful for a long time, check the network or TiDB status. |
`code=10003` | The `invalid connection` error from the underlying database. It usually indicates that the connection between DM and the downstream TiDB instance is abnormal (possibly caused by network failure or TiDB restart) and the currently requested data is partly sent to TiDB. | DM provides automatic recovery for such error. If the recovery is not successful for a long time, further check the error message and analyze the information based on the actual situation. |
`code=10005` | Occurs when performing the `QUERY` type SQL statements. | |
`code=10006` | Occurs when performing the `EXECUTE` type SQL statements, including DDL statements and DML statements of the `INSERT`, `UPDATE`or `DELETE` type. For more detailed error information, check the error message which usually includes the error code and error information returned for database operations.
|
`code=11006` | Occurs when the built-in parser of DM parses the incompatible DDL statements. | Refer to Data Migration - incompatible DDL statements for solution. |
`code=20010` | Occurs when decrypting the database password that is provided in task configuration. | Check whether the downstream database password provided in the configuration task is correctly encrypted using dmctl. |
`code=26002` | The task check fails to establish database connection. For more detailed error information, check the error message which usually includes the error code and error information returned for database operations. | Check whether the machine where DM-master is located has permission to access the upstream. |
`code=32001` | Abnormal dump processing unit | If the error message contains `mydumper: argument list too long.`, configure the table to be exported by manually adding the `--regex` regular expression in the Mydumper argument `extra-args` in the `task.yaml` file according to the block-allow list. For example, to export all tables named `hello`, add `--regex '.*\\.hello$'`; to export all tables, add `--regex '.*'`. |
`code=38008` | An error occurs in the gRPC communication among DM components. | Check `class`. Find out the error occurs in the interaction of which components. Determine the type of communication error. If the error occurs when establishing gRPC connection, check whether the communication server is working normally. |

What can I do when a migration task is interrupted with the `invalid connection` error returned?

Reason

The `invalid connection` error indicates that anomalies have occurred in the connection between DM and the downstream TiDB database (such as network failure, TiDB restart, and TiKV busy) and that a part of the data for the current request has been sent to TiDB.

Solutions

Because DM has the feature of concurrently migrating data to the downstream in migra-

tion tasks, several errors might occur when a task is interrupted. You can check these errors by using `query-status`.

- If only the `invalid connection` error occurs during the incremental replication process, DM retries the task automatically.
- If DM does not or fails to retry automatically because of version problems, use `stop-task` to stop the task and then use `start-task` to restart the task.

A migration task is interrupted with the `driver: bad connection` error returned

Reason

The `driver: bad connection` error indicates that anomalies have occurred in the connection between DM and the upstream TiDB database (such as network failure and TiDB restart) and that the data of the current request has not yet been sent to TiDB at that moment.

Solution

The current version of DM automatically retries on error. If you use the previous version which does not support automatically retry, you can execute the `stop-task` command to stop the task. Then execute `start-task` to restart the task.

The relay unit throws error `event from * in * diff from passed-in event *` or a migration task is interrupted with failing to get or parse binlog errors like `get binlog error ERROR 1236 (HY000)` and `binlog checksum mismatch, data may be corrupted` returned

Reason

During the DM process of relay log pulling or incremental replication, this two errors might occur if the size of the upstream binlog file exceeds **4 GB**.

Cause: When writing relay logs, DM needs to perform event verification based on binlog positions and the size of the binlog file, and store the replicated binlog positions as checkpoints. However, the official MySQL uses `uint32` to store binlog positions. This means the binlog position for a binlog file over 4 GB overflows, and then the errors above occur.

Solutions

For relay units, manually recover migration using the following solution:

1. Identify in the upstream that the size of the corresponding binlog file has exceeded 4GB when the error occurs.
2. Stop the DM-worker.
3. Copy the corresponding binlog file in the upstream to the relay log directory as the relay log file.

4. In the relay log directory, update the corresponding `relay.meta` file to pull from the next binlog file. If you have specified `enable_gtid` to `true` for the DM-worker, you need to modify the GTID corresponding to the next binlog file when updating the `relay.meta` file. Otherwise, you don't need to modify the GTID.

   Example: when the error occurs, `binlog-name = "mysql-bin.004451"` and `binlog-pos = 2453`. Update them respectively to `binlog-name = "mysql-bin.004452"` and `binlog-pos = 4`, and update `binlog-gtid` to `f0e914ef-54cf-11e7-813d-6c92bf2fa791:1-138218058`.

5. Restart the DM-worker.

For binlog replication processing units, manually recover migration using the following solution:

1. Identify in the upstream that the size of the corresponding binlog file has exceeded 4GB when the error occurs.
2. Stop the migration task using `stop-task`.
3. Update the `binlog_name` in the global checkpoints and in each table checkpoint of the downstream `dm_meta` database to the name of the binlog file in error; update `binlog_pos` to a valid position value for which migration has completed, for example, 4.

   Example: the name of the task in error is `dm_test`, the corresponding `source-id` is `replica-1`, and the corresponding binlog file is `mysql-bin|000001.004451`. Execute the following command:

   ```
   UPDATE dm_test_syncer_checkpoint SET binlog_name='mysql-bin|000001.004451', binlog_pos = 4 WHERE id='replica-1';
   ```

4. Specify `safe-mode: true` in the `syncers` section of the migration task configuration to ensure re-entrant.
5. Start the migration task using `start-task`.
6. View the status of the migration task using `query-status`. You can restore `safe-mode` to the original value and restart the migration task when migration is done for the original error-triggering relay log files.

`Access denied for user 'root'@'172.31.43.27' (using password: YES)` shows when you query the task or check the log

For database related passwords in all the DM configuration files, it is recommended to use the passwords encrypted by `dmctl`. If a database password is empty, it is unnecessary to encrypt it. For how to encrypt the plaintext password, see Encrypt the database password using dmctl.

In addition, the user of the upstream and downstream databases must have the corresponding read and write privileges. Data Migration also prechecks the corresponding privileges automatically while starting the data migration task.

The `load` processing unit reports the error `packet for query is too large. Try adjusting the 'max_allowed_packet' variable`

Reasons

- Both MySQL client and MySQL/TiDB server have the quota limits for `max_allowed_packet` ↪ . If any `max_allowed_packet` exceeds a limit, the client receives the error message. Currently, for the latest version of DM and TiDB server, the default value of `max_allowed_packet` is `64M`.
- The full data import processing unit in DM does not support splitting the SQL file exported by the Dump processing unit in DM.

Solutions

- It is recommended to set the `statement-size` option of `extra-args` for the Dump processing unit:

  According to the default `--statement-size` setting, the default size of `Insert` ↪ `Statement` generated by the Dump processing unit is about `1M`. With this default setting, the load processing unit does not report the error `packet for query is` ↪ `too large. Try adjusting the 'max_allowed_packet' variable` in most cases.

  Sometimes you might receive the following `WARN` log during the data dump. This `WARN` log does not affect the dump process. This only means that wide tables are dumped.

  ```
  Row bigger than statement_size for xxx
  ```

- If the single row of the wide table exceeds `64M`, you need to modify the following configurations and make sure the configurations take effect.

  - Execute `set @@global.max_allowed_packet=134217728` (`134217728` = 128 MB) in the TiDB server.
  - First add the `max-allowed-packet: 134217728` (128 MB) to the `target-` ↪ `database` section in the DM task configuration file. Then, execute the `stop-task` command and execute the `start-task` command.

13.6.8.13 TiDB Data Migration Release Notes

Since DM v5.4, the Release Notes of TiDB Data Migration have been merged into TiDB Release Notes of the same version number.

- To read the DM Release Notes of v5.4 or a later version, see the DM related content in the corresponding TiDB Release Notes.
- To read the DM Release Notes of v5.3.0 or an earlier version, refer to the following links.

13.6.8.13.1 5.3

- 5.3.0

13.6.8.13.2 2.0

- 2.0.7
- 2.0.6
- 2.0.5
- 2.0.4
- 2.0.3
- 2.0.2
- 2.0.1
- 2.0 GA
- 2.0.0-rc.2
- 2.0.0-rc

13.6.8.13.3 1.0

- 1.0.7
- 1.0.6
- 1.0.5
- 1.0.4
- 1.0.3
- 1.0.2

13.7 TiDB Lightning

13.7.1 TiDB Lightning Overview

TiDB Lightning is a tool used for importing data at TB scale to TiDB clusters. It is often used for initial data import to TiDB clusters.

TiDB Lightning supports the following file formats:

- Files exported by Dumpling
- CSV files
- Apache Parquet files generated by Amazon Aurora

TiDB Lightning can read data from the following sources:

- Local
- Amazon S3
- Google Cloud Storage

13.7.1.1 TiDB Lightning architecture

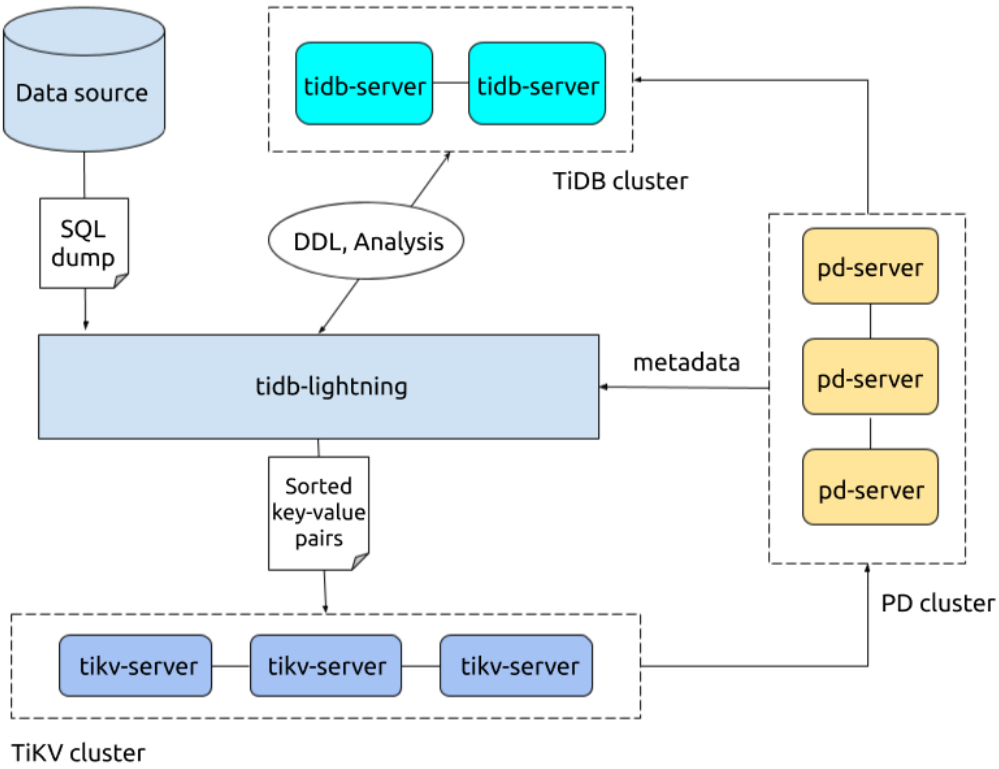


Figure 258: Architecture of TiDB Lightning tool set

TiDB Lightning supports two import modes, configured by `backend`. The import mode determines the way data is imported into TiDB.

- Physical Import Mode: TiDB Lightning first encodes data into key-value pairs and stores them in a local temporary directory, then uploads these key-value pairs to each TiKV node, and finally calls the TiKV Ingest interface to insert data into TiKV's RocksDB. If you need to perform initial import, consider physical import mode, which has higher import speed. The backend for the physical import mode is `local`.

- Logical Import Mode: TiDB Lightning first encodes the data into SQL statements and then runs these SQL statements directly for data import. If the cluster to be imported is in production, or if the target table to be imported already contains data, use logical import mode. The backend for the logical import mode is `tidb`.

Import mode	Physical Import Mode	Logical Import
Backend	`local`	`tidb`
Speed	Fast (100~500 GiB/hour)	Low (10~50 G
Resource consumption	High	Low
Network bandwidth consumption	High	Low
ACID compliance during import	No	Yes
Target tables	Must be empty	Can contain d
TiDB cluster version	>= 4.0.0	All
Whether the TiDB cluster can provide service during import	No	Yes

The preceding performance data is used to compare the import performance difference between the two modes. The actual import speed is affected by various factors such as hardware configuration, table schema, and the number of indexes.

13.7.2 Quick Start for TiDB Lightning

This document provides a quick guide on getting started with TiDB Lightning by importing MySQL data into a TiDB cluster.

> **Warning:**
>
> The deployment method in this tutorial is only recommended for test and trial. **Do not apply it in the production or development environment.**

13.7.2.1 Step 1: Prepare full backup data

First, you can use dumpling to export data from MySQL.

1. Run `tiup --version` to check if TiUP is already installed. If TiUP is installed, skip this step. If TiUP is not installed, run the following command:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

2. Using TiUP to install Dumpling:

```
tiup install dumpling
```

3. To export data from MySQL, you can refer to the detailed steps provided in Use Dumpling to Export Data:

```
tiup dumpling -h 127.0.0.1 -P 3306 -u root -t 16 -F 256MB -B test -f '
    ↪ test.t[12]' -o /data/my_database/
```

In the above command:

- `-t 16`: Export data using 16 threads.
- `-F 256MB`: Split each table into multiple files, with each file approximately 256 MB in size.
- `-B test`: Export from the `test` database.
- `-f 'test.t[12]'`: Export only the two tables `test.t1` and `test.t2`.

The full backup data exported will be saved in the `/data/my_database` directory.

13.7.2.2 Step 2: Deploy the TiDB cluster

Before starting the data import, you need to deploy a TiDB cluster for the import. If you already have a TiDB cluster, you can skip this step.

For the steps on deploying a TiDB cluster, refer to the Quick Start Guide for the TiDB Database Platform.

13.7.2.3 Step 3: Install TiDB Lightning

Run the following command to install the latest version of TiDB Lightning:

```
tiup install tidb-lightning
```

13.7.2.4 Step 4: Start TiDB Lightning

> **Note:**
>
> The import method in this section is only suitable for testing and functional experience. For production environments, refer to Migrate Large Datasets from MySQL to TiDB

1. Create the configuration file `tidb-lightning.toml` and fill in the following settings based on your cluster information:

```
[lightning]
# Logging
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
# Configure the import mode
backend = "local"
# Sets the directory for temporarily storing the sorted key-value pairs
    ↪ . The target directory must be empty.
sorted-kv-dir = "/mnt/ssd/sorted-kv-dir"

[mydumper]
# Local source data directory
data-source-dir = "/data/my_datasource/"

# Configures the wildcard rule. By default, all tables in the mysql,
    ↪ sys, INFORMATION_SCHEMA, PERFORMANCE_SCHEMA, METRICS_SCHEMA, and
    ↪ INSPECTION_SCHEMA system databases are filtered.
# If this item is not configured, the "cannot find schema" error occurs
    ↪  when system tables are imported.
filter = ['*.*', '!mysql.*', '!sys.*', '!INFORMATION_SCHEMA.*', '!
    ↪ PERFORMANCE_SCHEMA.*', '!METRICS_SCHEMA.*', '!INSPECTION_SCHEMA
    ↪ .*']
[tidb]
# Information of the target cluster
host = "172.16.31.2"
port = 4000
user = "root"
password = "rootroot"
# Table schema information is fetched from TiDB via this status-port.
status-port = 10080
# The PD address of the cluster
pd-addr = "172.16.31.3:2379"
```

2. Run `tidb-lightning`. To avoid the program exiting due to the `SIGHUP` signal when starting the program directly in the command line using `nohup`, it is recommended to put the `nohup` command in a script. For example:

```
#!/bin/bash
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out &
```

13.7.2.5 Step 5: Check data integrity

After the import is completed, TiDB Lightning exits automatically. If the import is successful, you can find `tidb lightning exit` in the last line of the log file.

If any error occurs, refer to TiDB Lightning FAQs.

13.7.2.6 Summary

This tutorial briefly introduces what TiDB Lightning is and how to quickly deploy a TiDB Lightning cluster to import full backup data to the TiDB cluster.

For detailed features and usage about TiDB Lightning, refer to TiDB Lightning Overview.

13.7.3 Deploy TiDB Lightning

This document describes the hardware requirements of using TiDB Lightning to import data, and how to deploy it manually. Requirements on hardware resources vary with the import modes. For details, refer to the following docs:

- Physical Import Mode Requirements and Limitations
- Logical Import Mode Requirements and Limitations

13.7.3.1 Online deployment using TiUP (recommended)

1. Install TiUP using the following command:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
    ↪ install.sh | sh
```

   This command automatically adds TiUP to the `PATH` environment variable. You need to start a new terminal session or run `source ~/.bashrc` before you can use TiUP. (According to your environment, you may need to run `source ~/.profile`. For the specific command, check the output of TiUP.)

2. Install TiDB Lightning using TiUP:

```
tiup install tidb-lightning
```

13.7.3.2 Manual deployment

13.7.3.2.1 Download TiDB Lightning binaries

Refer to Download TiDB Tools and download TiDB Lightning binaries. TiDB Lightning is completely compatible with early versions of TiDB. It is recommended to use the latest version of TiDB Lightning.

Unzip the TiDB Lightning binary package to obtain the `tidb-lightning` executable file:

```
tar -zxvf tidb-lightning-${version}-linux-amd64.tar.gz
chmod +x tidb-lightning
```

13.7.3.2.2 Upgrade TiDB Lightning

You can upgrade TiDB Lightning by replacing the binaries alone without further configurations. After the upgrade, you need to restart TiDB Lightning. For details, see How to properly restart TiDB Lightning.

If an import task is running, we recommend you to wait until it finishes before upgrading TiDB Lightning. Otherwise, there might be chances that you need to reimport from scratch, because there is no guarantee that checkpoints work across versions.

13.7.4 TiDB Lightning Requirements for the Target Database

Before using TiDB Lightning, you need to check whether the environment meets the requirements. This helps reduce errors during import and ensures import success.

13.7.4.1 Privileges of the target database

Based on the import mode and features enabled, the target database users should be granted with different privileges. The following table provides a reference.

```
<td></td>
<td>Feature</td>
<td>Scope</td>
<td>Required privilege</td>
<td>Remarks</td>
```

```
<td rowspan="2">Mandatory</td>
<td rowspan="2">Basic functions</td>
<td>Target table</td>
<td>CREATE, SELECT, INSERT, UPDATE, DELETE, DROP, ALTER, REFERENCES</td>
<td>DROP is required only when tidb-lightning-ctl runs the checkpoint-
   ↪ destroy-all command</td>
```

```
<td>Target database</td>
<td>CREATE</td>
<td></td>
```

```
<td rowspan="4">Mandatory</td>
<td>Logical Import Mode</td>
<td>information_schema.columns</td>
<td>SELECT</td>
<td></td>
```

```
<td rowspan="3">Physical Import Mode</td>
<td>mysql.tidb</td>
<td>SELECT</td>
<td></td>
```

```
<td>-</td>
<td>SUPER</td>
<td></td>
```

```
<td>-</td>
<td>RESTRICTED_VARIABLES_ADMIN,RESTRICTED_TABLES_ADMIN</td>
<td>Required when the target TiDB enables SEM</td>
```

```
<td>Recommended</td>
<td>Conflict detection, max-error</td>
<td>Schema configured for lightning.task-info-schema-name</td>
<td>SELECT, INSERT, UPDATE, DELETE, CREATE, DROP</td>
<td>If not required, the value must be set to ""</td>
```

```
<td>Optional</td>
<td>Parallel import</td>
<td>Schema configured for lightning.meta-schema-name</td>
<td>SELECT, INSERT, UPDATE, DELETE, CREATE, DROP</td>
<td>If not required, the value must be set to ""</td>
```

```
<td>Optional</td>
<td>checkpoint.driver = "mysql"</td>
<td>checkpoint.schema setting</td>
<td>SELECT,INSERT,UPDATE,DELETE,CREATE,DROP</td>
<td>Required when checkpoint information is stored in databases, instead
   ↪ of files</td>
```

Note:

- If the target table does not use the foreign key, the `REFERENCES` privilege is not required.
- If the target table uses the foreign key, you need to grant the `REFERENCES` ↪ privilege for the downstream database user, or manually create the target table in the downstream database in advance to ensure smooth data import.

13.7.4.2 Storage space of the target database

The target TiKV cluster must have enough disk space to store the imported data. In addition to the standard hardware requirements, the storage space of the target TiKV cluster must be larger than **the size of the data source x the number of replicas x 2**. For example, if the cluster uses 3 replicas by default, the target TiKV cluster must have a storage space larger than 6 times the size of the data source. The formula has x 2 because:

- Indexes might take extra space.
- RocksDB has a space amplification effect.

It is difficult to calculate the exact data volume exported by Dumpling from MySQL. However, you can estimate the data volume by using the following SQL statement to summarize the data-length field in the information_schema.tables table:

Calculate the size of all schemas, in MiB. Replace ${schema_name} with your schema name.

```
SELECT table_schema, SUM(data_length)/1024/1024 AS data_length, SUM(
    ↪ index_length)/1024/1024 AS index_length, SUM(data_length+index_length
    ↪ )/1024/1024 AS sum FROM information_schema.tables WHERE table_schema
    ↪ = "${schema_name}" GROUP BY table_schema;
```

Calculate the size of the largest table, in MiB. Replace ${schema_name} with your schema name.

```
SELECT table_name, table_schema, SUM(data_length)/1024/1024 AS data_length,
    ↪  SUM(index_length)/1024/1024 AS index_length,sum(data_length+
    ↪ index_length)/1024/1024 AS sum FROM information_schema.tables WHERE
    ↪ table_schema = "${schema_name}" GROUP BY table_name,table_schema
    ↪ ORDER BY sum DESC LIMIT 5;
```

13.7.5 Data Sources

13.7.5.1 TiDB Lightning Data Sources

TiDB Lightning supports importing data from multiple data sources to TiDB clusters, including CSV, SQL, and Parquet files.

To specify the data source for TiDB Lightning, use the following configuration:

```
[mydumper]
#### Local source data directory or the URL of the external storage such as
    ↪ S3.
data-source-dir = "/data/my_database"
```

When TiDB Lightning is running, it looks for all files that match the pattern of `data-source-dir`.

File	Type	Pattern
Schema file	Contains the CREATE TABLE DDL statement	${ db_name }.${ table_name }-schema.sql
Schema file	Contains the CREATE DATABASE DDL statement	${ db_name }-schema.create.sql
Data file	If the data file contains data for a whole table, the file is imported into a table named ${ db_name }.${ table_name }	${db_name}.${table_name}.${csv|sql|parquet}

File	Type	Pattern
Data file	If the data for a table is split into multiple data files, each data file must be suffixed with a number in its filename	${db_name}.${table_name}.001.${csv|sql|parquet}

TiDB Lightning processes data in parallel as much as possible. Because files must be read in sequence, the data processing concurrency is at the file level (controlled by `region-concurrency`). Therefore, when the imported file is large, the import performance is poor. It is recommended to limit the size of the imported file to no greater than 256 MiB to achieve the best performance.

13.7.5.1.1 CSV

Schema

CSV files are schema-less. To import CSV files into TiDB, you must provide a table schema. You can provide schema by either of the following methods:

- Create files named `${db_name}.${table_name}-schema.sql` and `${db_name}-schema-create.sql` that contain DDL statements.
- Manually create the table schema in TiDB.

Configuration

You can configure the CSV format in the `[mydumper.csv]` section in the `tidb-lightning.toml` file. Most settings have a corresponding option in the LOAD DATA statement of MySQL.

```
[mydumper.csv]
#### The field separator. Can be one or multiple characters. The default is
    ↪ ','.
#### If the data might contain commas, it is recommended to use '|+|' or
    ↪ other uncommon
```

```
#### character combinations as a separator.
separator = ','
#### Quoting delimiter. Empty value means no quoting.
delimiter = '"'
#### Line terminator. Can be one or multiple characters. Empty value (
   ↪ default) means
#### both "\n" (LF) and "\r\n" (CRLF) are line terminators.
terminator = ''
#### Whether the CSV file contains a header.
#### If `header` is true, the first line is skipped and mapped
#### to the table columns.
header = true
#### Whether the CSV file contains any NULL value.
#### If `not-null` is true, all columns from CSV cannot be parsed as NULL.
not-null = false
#### When `not-null` is false (that is, CSV can contain NULL),
#### fields equal to this value will be treated as NULL.
null = '\N'
#### Whether to parse backslash as escape character.
backslash-escape = true
#### Whether to treat `separator` as the line terminator and trim all
   ↪ trailing separators.
trim-last-separator = false
```

If the input of a string field such as `separator`, `delimiter`, or `terminator` involves special characters, you can use a backslash to escape the special characters. The escape sequence must be a *double-quoted* string (`"⋯"`). For example, `separator = "\u001f"` means using the ASCII character `0X1F` as the separator.

You can use *single-quoted* strings (`'⋯'`) to suppress backslash escaping. For example, `terminator = '\n'` means using the two-character string, a backslash (\) followed by the letter `n`, as the terminator, rather than the LF `\n`.

For more details, see the TOML v1.0.0 specification.

`separator`

- Defines the field separator.
- Can be one or multiple characters, but must not be empty.
- Common values:
  - `','` for CSV (comma-separated values).
  - `"\t"` for TSV (tab-separated values).
  - `"\u0001"` to use the ASCII character `0x01`.
- Corresponds to the `FIELDS TERMINATED BY` option in the LOAD DATA statement.

`delimiter`

- Defines the delimiter used for quoting.
- If `delimiter` is empty, all fields are unquoted.
- Common values:
  - `'"'` quotes fields with double-quote. The same as RFC 4180.
  - `''` disables quoting.
- Corresponds to the `FIELDS ENCLOSED BY` option in the `LOAD DATA` statement.

`terminator`

- Defines the line terminator.
- If `terminator` is empty, both `"\n"` (Line Feed) and `"\r\n"` (Carriage Return + Line Feed) are used as the line terminator.
- Corresponds to the `LINES TERMINATED BY` option in the `LOAD DATA` statement.

`header`

- Whether *all* CSV files contain a header row.
- If `header` is `true`, the first row is used as the *column names*. If `header` is `false`, the first row is treated as an ordinary data row.

`not-null` and `null`

- The `not-null` setting controls whether all fields are non-nullable.
- If `not-null` is `false`, the string specified by `null` is transformed to the SQL NULL instead of a specific value.
- Quoting does not affect whether a field is null.

  For example, in the following CSV file:

```
A,B,C
\N,"\N",
```

  In the default settings (`not-null = false; null = '\N'`), the columns `A` and `B` are both converted to NULL after being imported to TiDB. The column `C` is an empty string `''` but not NULL.

`backslash-escape`

- Whether to parse backslash inside fields as escape characters.
- If `backslash-escape` is true, the following sequences are recognized and converted:

Sequence	Converted to
\0	Null character (U+0000)
\b	Backspace (U+0008)
\n	Line feed (U+000A)
\r	Carriage return (U+000D)
\t	Tab (U+0009)
\Z	Windows EOF (U+001A)

In all other cases (for example, \"), the backslash is stripped, leaving the next character (") in the field. The character left has no special roles (for example, delimiters) and is just an ordinary character.

- Quoting does not affect whether backslash is parsed as an escape character.
- Corresponds to the FIELDS ESCAPED BY '\' option in the LOAD DATA statement.

trim-last-separator

- Whether to treat separator as the line terminator and trim all trailing separators.

  For example, in the following CSV file:

```
A,,B,,
```

  - When trim-last-separator = false, this is interpreted as a row of 5 fields ('A', '', 'B', '', '').
  - When trim-last-separator = true, this is interpreted as a row of 3 fields ('A ↪ ', '', 'B').

- This option is deprecated. Use the terminator option instead.

  If your existing configuration is:

```
separator = ','
trim-last-separator = true
```

  It is recommended to change the configuration to:

```
separator = ','
terminator = ",\n" # Use ",\n" or ",'\r\n" according to your actual
   ↪ file.
```

Non-configurable options

TiDB Lightning does not support every option supported by the LOAD DATA statement. For example:

- There cannot be line prefixes (`LINES STARTING BY`).
- The header cannot be skipped (`IGNORE n LINES`) and must be valid column names.

Strict format

TiDB Lightning works best when the input files have a uniform size of around 256 MiB. When the input is a single huge CSV file, TiDB Lightning can only process the file in one thread, which slows down the import speed.

This can be fixed by splitting the CSV into multiple files first. For the generic CSV format, there is no way to quickly identify where a row starts or ends without reading the whole file. Therefore, TiDB Lightning by default does *not* automatically split a CSV file. However, if you are certain that the CSV input adheres to certain restrictions, you can enable the `strict-format` setting to allow TiDB Lightning to split the file into multiple 256 MiB-sized chunks for parallel processing.

```
[mydumper]
strict-format = true
```

In a strict CSV file, every field occupies only a single line. In other words, one of the following must be true:

- Delimiter is empty.
- Every field does not contain the terminator itself. In the default configuration, this means every field does not contain CR (`\r`) or LF (`\n`).

If a CSV file is not strict, but `strict-format` is wrongly set to `true`, a field spanning multiple lines may be cut in half into two chunks, causing parse failure, or even quietly importing corrupted data.

Common configuration examples

CSV

The default setting is already tuned for CSV following RFC 4180.

```
[mydumper.csv]
separator = ',' # If the data might contain a comma (','), it is recommended
   ↪  to use '|+|' or other uncommon character combinations as the
   ↪ separator.
delimiter = '"'
header = true
not-null = false
null = '\N'
backslash-escape = true
```

Example content:

```
ID,Region,Count
1,"East",32
2,"South",\N
3,"West",10
4,"North",39
```

TSV

```
[mydumper.csv]
separator = "\t"
delimiter = ''
header = true
not-null = false
null = 'NULL'
backslash-escape = false
```

Example content:

```
ID    Region    Count
1     East      32
2     South     NULL
3     West      10
4     North     39
```

TPC-H DBGEN

```
[mydumper.csv]
separator = '|'
delimiter = ''
terminator = "|\n"
header = false
not-null = true
backslash-escape = false
```

Example content:

```
1|East|32|
2|South|0|
3|West|10|
4|North|39|
```

13.7.5.1.2 SQL

When TiDB Lightning processes a SQL file, because TiDB Lightning cannot quickly split a single SQL file, it cannot improve the import speed of a single file by increasing concurrency. Therefore, when you import data from SQL files, avoid a single huge SQL file. TiDB Lightning works best when the input files have a uniform size of around 256 MiB.

13.7.5.1.3 Parquet

TiDB Lightning currently only supports Parquet files generated by Amazon Aurora. To identify the file structure in S3, use the following configuration to match all data files:

```
[[mydumper.files]]
#### The expression needed for parsing Amazon Aurora parquet files
pattern = '(?i)^(?:[^/]*/)*([a-z0-9_]+)\.([a-z0-9_]+)/(?:[^/]*/)*(?:[a-z0
    ↪ -9\-_.]+\.(parquet))$'
schema = '$1'
table = '$2'
type = '$3'
```

Note that this configuration only shows how to match the parquet files exported by Aurora snapshot. You need to export and process the schema file separately.

For more information on `mydumper.files`, refer to Match customized file.

13.7.5.1.4 Match customized files

TiDB Lightning only recognizes data files that follow the naming pattern. In some cases, your data file might not follow the naming pattern, and thus data import is completed in a short time without importing any file.

To resolve this issue, you can use `[[mydumper.files]]` to match data files in your customized expression.

Take the Aurora snapshot exported to S3 as an example. The complete path of the Parquet file is `S3://some-bucket/some-subdir/some-database/some-database.some-table
↪ /part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet`.

Usually, `data-source-dir` is set to `S3://some-bucket/some-subdir/some-database/` to import the `some-database` database.

Based on the preceding Parquet file path, you can write a regular expression like `(?i)^(?:[^/]*/)*([a-z0-9_]+)\.([a-z0-9_]+)/(?:[^/]*/)*(?:[a-z0-9\-_.]+\.(
↪ parquet))$` to match the files. In the match group, `index=1` is `some-database`, `index=2` is `some-table`, and `index=3` is `parquet`.

You can write the configuration file according to the regular expression and the corresponding index so that TiDB Lightning can recognize the data files that do not follow the default naming convention. For example:

```
[[mydumper.files]]
#### The expression needed for parsing the Amazon Aurora parquet file
pattern = '(?i)^(?:[^/]*/)*([a-z0-9_]+)\.([a-z0-9_]+)/(?:[^/]*/)*(?:[a-z0
    ↪ -9\-_.]+\.(parquet))$'
schema = '$1'
table = '$2'
type = '$3'
```

- **schema**: The name of the target database. The value can be:
  - The group index obtained by using a regular expression, such as `$1`.
  - The name of the database that you want to import, such as `db1`. All matched files are imported into `db1`.
- **table**: The name of the target table. The value can be:
  - The group index obtained by using a regular expression, such as `$2`.
  - The name of the table that you want to import, such as `table1`. All matched files are imported into `table1`.
- **type**: The file type. Supports `sql`, `parquet`, and `csv`. The value can be:
  - The group index obtained by using a regular expression, such as `$3`.
- **key**: The file number, such as `001` in `${db_name}.${table_name}.001.csv`.
  - The group index obtained by using a regular expression, such as `$4`.

13.7.5.1.5 More resources

- Export to CSV files Using Dumpling
- `LOAD DATA`

13.7.6 Physical Import Mode

13.7.6.1 Physical Import Mode

Physical import mode is an efficient and fast import mode that inserts data directly into TiKV nodes as key-value pairs without going through the SQL interface. When using the physical import mode, a single instance of Lightning can import up to 10 TiB of data. The supported amount of imported data theoretically increases as the number of Lightning instances increases. It is verified by users that parallel importing based on Lightning can effectively handle up to 20 TiB of data.

Before you use the physical import mode, make sure to read Requirements and restrictions.

The backend for the physical import mode is `local`.

13.7.6.1.1 Implementation

1. Before importing data, TiDB Lightning automatically switches the TiKV nodes to "import mode", which improves write performance and stops PD scheduling and auto-compaction.
2. `tidb-lightning` creates table schemas in the target database and fetches the meta-data.

3. Each table is divided into multiple contiguous **blocks**, so that Lightning can import data data from large tables (200 GB+) in parallel.

4. `tidb-lightning` prepares an "engine file" for each block to handle key-value pairs. `tidb-lightning` reads the SQL dump in parallel, converts the data source to key-value pairs in the same encoding as TiDB, sorts the key-value pairs and writes them to a local temporary storage file.

5. When an engine file is written, `tidb-lightning` starts to split and schedule data on the target TiKV cluster, and then imports data to TiKV cluster.

   The engine file contains two types of engines: **data engine** and **index engine**. Each engine corresponds to a type of key-value pairs: row data and secondary index. Normally, row data is completely ordered in the data source, and the secondary index is unordered. Therefore, the data engine files are imported immediately after the corresponding block is written, and all index engine files are imported only after the entire table is encoded.

6. After all engine files are imported, `tidb-lightning` compares the checksum between the local data source and the downstream cluster, and ensures that the imported data is not corrupted. Then `tidb-lightning` analyzes the new data (`ANALYZE`) to optimize the future operations. Meanwhile, `tidb-lightning` adjusts the `AUTO_INCREMENT` value to prevent conflicts in the future.

   The auto-increment ID is estimated by the **upper bound** of the number of rows, and is proportional to the total size of the table data file. Therefore, the auto-increment ID is usually larger than the actual number of rows. This is normal because the auto-increment ID is not necessarily contiguous.

7. After all steps are completed, `tidb-lightning` automatically switches the TiKV nodes to "normal mode", and the TiDB cluster can provide services normally.

13.7.6.1.2 Requirements and restrictions

Environment requirements

Operating system:

It is recommended to use fresh CentOS 7 instances. You can deploy a virtual machine either on your local host or in the cloud. Because TiDB Lightning consumes as much CPU resources as needed by default, it is recommended that you deploy it on a dedicated server. If this is not possible, you can deploy it on a single server together with other TiDB components (for example, tikv-server) and then configure `region-concurrency` to limit the CPU usage from TiDB Lightning. Usually, you can configure the size to 75% of the logical CPU.

Memory and CPU:

It is recommended that you allocate CPU more than 32 cores and memory greater than 64 GiB to get better performance.

Note:

When you import a large amount of data, one concurrent import may consume about 2 GiB memory. The total memory usage can be `region-concurrency * 2 GiB`. `region-concurrency` is the same as the number of logical CPUs by default. If the memory size (GiB) is less than twice of the CPU or OOM occurs during the import, you can decrease `region-concurrency` to avoid OOM.

Storage: The `sorted-kv-dir` configuration item specifies the temporary storage directory for the sorted key-value files. The directory must be empty, and the storage space must be greater than the size of the dataset to be imported. For better import performance, it is recommended to use a directory different from `data-source-dir` and use flash storage and exclusive I/O for the directory.

Network: A 10Gbps Ethernet card is recommended.

Version requirements

- TiDB Lightning >= v4.0.3.
- TiDB >= v4.0.0.
- If the target TiDB cluster is v3.x or earlier, you need to use Importer-backend to complete the data import. In this mode, `tidb-lightning` needs to send the parsed key-value pairs to `tikv-importer` via gRPC, and `tikv-importer` will complete the data import.

Limitations

- Do not use physical import mode to import data to TiDB clusters in production. It has severe performance implications.
- Do not use multiple TiDB Lightning instances to import data to the same TiDB cluster by default. Use Parallel Import instead.
- When you use multiple TiDB Lightning to import data to the same target cluster, do not mix the import modes. That is, do not use the physical import mode and the logical import mode at the same time.
- A single Lightning process can import a single table of 10 TB at most. Parallel import can use 10 Lightning instances at most.

Tips for using with other components

- When you use TiDB Lightning with TiFlash, note the following:

    - Whether you have created a TiFlash replica for a table, you can use TiDB Lightning to import data to the table. However, the import might take longer than the normal import. The import time is influenced by the network bandwidth of the server TiDB Lightning is deployed on, the CPU and disk load on the TiFlash node, and the number of TiFlash replicas.

- TiDB Lightning character sets:

    - TiDB Lightning earlier than v5.4.0 cannot import tables of `charset=GBK`.

- When you use TiDB Lightning with TiCDC, note the following:

    - TiCDC cannot capture the data inserted in the physical import mode.

13.7.6.2 Use Physical Import Mode

This document introduces how to use the physical import mode in TiDB Lightning, including writing the configuration file and tuning performance.

There are limitations on the physical import mode. Before using the physical import mode, make sure to read Limitations.

13.7.6.2.1 Configure and use the physical import mode

You can use the following configuration file to execute data import using the physical import mode:

```
[lightning]
#### log
level = "info"
file = "tidb-lightning.log"
max-size = 128 # MB
max-days = 28
max-backups = 14

#### Checks the cluster minimum requirements before start.
check-requirements = true

[mydumper]
#### The local data source directory or the external storage URL.
data-source-dir = "/data/my_database"

[tikv-importer]
#### Import mode. "local" means using the physical import mode.
backend = "local"

#### The method to resolve the conflicting data.
```

```
duplicate-resolution = 'remove'

#### The directory of local KV sorting.
sorted-kv-dir = "./some-dir"

[tidb]
#### The information of the target cluster. The address of any tidb-server
   ↪ from the cluster.
host = "172.16.31.1"
port = 4000
user = "root"
#### Configure the password to connect to TiDB. Either plaintext or Base64
   ↪ encoded.
password = ""
#### Required. Table schema information is fetched from TiDB via this status
   ↪ -port.
status-port = 10080
#### Required. The address of any pd-server from the cluster.
pd-addr = "172.16.31.4:2379"
#### tidb-lightning imports the TiDB library, and generates some logs.
#### Set the log level of the TiDB library.
log-level = "error"

[post-restore]
#### Specifies whether to perform `ADMIN CHECKSUM TABLE <table>` for each
   ↪ table to verify data integrity after importing.
#### The following options are available:
#### - "required" (default): Perform admin checksum after importing. If
   ↪ checksum fails, TiDB Lightning will exit with failure.
#### - "optional": Perform admin checksum. If checksum fails, TiDB Lightning
   ↪  will report a WARN log but ignore any error.
#### - "off": Do not perform checksum after importing.
#### Note that since v4.0.8, the default value has changed from "true" to "
   ↪ required".
#### # Note:
#### 1. Checksum failure usually means import exception (data loss or data
   ↪ inconsistency), so it is recommended to always enable Checksum.
#### 2. For backward compatibility, bool values "true" and "false" are also
   ↪ allowed for this field.
#### "true" is equivalent to "required" and "false" is equivalent to "off".
checksum = "required"

#### Specifies whether to perform `ANALYZE TABLE <table>` for each table
   ↪ after checksum is done.
#### Options available for this field are the same as `checksum`. However,
```

```
    ↪ the default value for this field is "optional".
analyze = "optional"
```

For the complete configuration file, refer to the configuration file and command line parameters.

13.7.6.2.2 Conflict detection

Conflicting data refers to two or more records with the same PK/UK column data. When the data source contains conflicting data, the actual number of rows in the table is different from the total number of rows returned by the query using unique index.

TiDB Lightning offers the following strategies for detecting conflicting data:

- `remove` (recommended): records and removes all conflicting records from the target table to ensure a consistent state in the target TiDB.
- `none`: does not detect duplicate records. `none` has the best performance in the two strategies, but might lead to inconsistent data in the target TiDB.

Before v5.3, TiDB Lightning does not support conflict detection. If there is conflicting data, the import process fails at the checksum step. When conflict detection is enabled, if there is conflicting data, TiDB Lightning skips the checksum step (because it always fails).

Suppose an `order_line` table has the following schema:

```
CREATE TABLE IF NOT EXISTS `order_line` (
  `ol_o_id` int(11) NOT NULL,
  `ol_d_id` int(11) NOT NULL,
  `ol_w_id` int(11) NOT NULL,
  `ol_number` int(11) NOT NULL,
  `ol_i_id` int(11) NOT NULL,
  `ol_supply_w_id` int(11) DEFAULT NULL,
  `ol_delivery_d` datetime DEFAULT NULL,
  `ol_quantity` int(11) DEFAULT NULL,
  `ol_amount` decimal(6,2) DEFAULT NULL,
  `ol_dist_info` char(24) DEFAULT NULL,
  PRIMARY KEY (`ol_w_id`,`ol_d_id`,`ol_o_id`,`ol_number`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

If Lightning detects conflicting data during the import, you can query the `lightning_task_info ↪ .conflict_error_v1` table as follows:

```
mysql> select table_name,index_name,key_data,row_data from
    ↪ conflict_error_v1 limit 10;
+--
    ↪ ------------------+------------+----------+-----------------------------------
    ↪
```

```
| table_name       | index_name | key_data | row_data
   ↪                                                        |
+--
   ↪ -------------------+------------+----------+-----------------------------------
   ↪
| `tpcc`.`order_line` | PRIMARY | 21829216 | (2677, 10, 10, 11, 75656, 10,
   ↪ NULL, 5, 5831.97, "HT5DN3EVb6kWTd4L37bsbogj") |
| `tpcc`.`order_line` | PRIMARY | 49931672 | (2677, 10, 10, 11, 75656, 10,
   ↪ NULL, 5, 5831.97, "HT5DN3EVb6kWTd4L37bsbogj") |
| `tpcc`.`order_line` | PRIMARY | 21829217 | (2677, 10, 10, 12, 76007, 10,
   ↪ NULL, 5, 9644.36, "bHuVoRfidQ0q2rJ6ZC9Hd12E") |
| `tpcc`.`order_line` | PRIMARY | 49931673 | (2677, 10, 10, 12, 76007, 10,
   ↪ NULL, 5, 9644.36, "bHuVoRfidQ0q2rJ6ZC9Hd12E") |
| `tpcc`.`order_line` | PRIMARY | 21829218 | (2677, 10, 10, 13, 85618, 10,
   ↪ NULL, 5, 7427.98, "t3rsesgi9rVAKi9tf6an5Rpv") |
| `tpcc`.`order_line` | PRIMARY | 49931674 | (2677, 10, 10, 13, 85618, 10,
   ↪ NULL, 5, 7427.98, "t3rsesgi9rVAKi9tf6an5Rpv") |
| `tpcc`.`order_line` | PRIMARY | 21829219 | (2677, 10, 10, 14, 15873, 10,
   ↪ NULL, 5, 133.21, "z1vH0e31tQydJGhfNYNa4ScD") |
| `tpcc`.`order_line` | PRIMARY | 49931675 | (2677, 10, 10, 14, 15873, 10,
   ↪ NULL, 5, 133.21, "z1vH0e31tQydJGhfNYNa4ScD") |
| `tpcc`.`order_line` | PRIMARY | 21829220 | (2678, 10, 10, 1, 44644, 10,
   ↪ NULL, 5, 8463.76, "TWKJBt5iJA4eF7FIVxnugNmz") |
| `tpcc`.`order_line` | PRIMARY | 49931676 | (2678, 10, 10, 1, 44644, 10,
   ↪ NULL, 5, 8463.76, "TWKJBt5iJA4eF7FIVxnugNmz") |
+--
   ↪ -------------------+------------+-----------------------------------------------
   ↪
10 rows in set (0.14 sec)
```

You can manually identify the records that need to be retained and insert these records into the table.

13.7.6.2.3 Performance tuning

The most direct and effective ways to improve import performance of the physical import mode are as follows:

- **Upgrade the hardware of the node where Lightning is deployed, especially the CPU and the storage device of `sorted-key-dir`.**
- **Use the parallel import feature to achieve horizontal scaling.**

TiDB Lightning provides some concurrency-related configurations to affect import performance in the physical import mode. However, from long-term experience, it is recommended to keep the following four configuration items in the default value. Adjusting the four configuration items does not bring significant performance boost.

```
[lightning]
#### The maximum concurrency of engine files.
#### Each table is split into one "index engine" to store indices, and
   ↪ multiple
#### "data engines" to store row data. These settings control the maximum
#### concurrent number for each type of engines.
#### The two settings controls the maximum concurrency of the two engine
   ↪ files.
index-concurrency = 2
table-concurrency = 6

#### The concurrency of data. The default value is the number of logical
   ↪ CPUs.
region-concurrency =

#### The maximum concurrency of I/O. When the concurrency is too high, the
   ↪ disk
#### cache may be frequently refreshed, causing the cache miss and read
   ↪ speed
#### to slow down. For different storage mediums, this parameter may need to
   ↪  be
#### adjusted to achieve the best performance.
io-concurrency = 5
```

During the import, each table is split into one "index engine" to store indices, and multiple "data engines" to store row data.

`index-concurrency` controls the maximum concurrency of the index engine. When you adjust `index-concurrency`, make sure that `index-concurrency * the number of source files of each table > region-concurrency` to ensure that the CPU is fully utilized. The ratio is usually between 1.5 ~ 2. Do not set `index-concurrency` too high and not lower than 2 (default). Too high `index-concurrency` causes too many pipelines to be built, which causes the index-engine import stage to pile up.

The same goes for `table-concurrency`. Make sure that `table-concurrency * the number of source files of each table > region-concurrency` to ensure that the CPU is fully utilized. A recommended value is around `region-concurrency * 4 / the number of source files of each table` and not lower than 4.

If the table is large, Lightning will split the table into multiple batches of 100 GiB. The concurrency is controlled by `table-concurrency`.

`index-concurrency` and `table-concurrency` has little effect on the import speed. You can leave them in the default value.

`io-concurrency` controls the concurrency of file read. The default value is 5. At any given time, only 5 handles are performing read operations. Because the file read speed is

usually not a bottleneck, you can leave this configuration in the default value.

After the file data is read, Lightning needs to do some post-processing, such as encoding and sorting the data locally. The concurrency of these operations is controlled by `region-concurrency`. The default value is the number of CPU cores. You can leave this configuration in the default value. It is recommended to deploy Lightning on a separate server from other components. If you must deploy Lightning together with other components, you need to lower the value of `region-concurrency` according to the load.

The `num-threads` configuration of TiKV can also affect the performance. For new clusters, it is recommended to set `num-threads` to the number of CPU cores.

13.7.7 Logical Import Mode

13.7.7.1 Logical Import Mode Introduction

The logical import mode is one of the two import modes supported by TiDB Lightning. In the logical import mode, TiDB Lightning first encodes data into SQL statements and then runs the SQL statements to import data.

If your TiDB cluster already contains data and provides service for external applications, it is recommended to import data in the logical import mode. The behavior of the logical import mode is the same as executing normal SQL statements, and thus it guarantees ACID compliance.

The backend for the logical import mode is `tidb`.

13.7.7.1.1 Environment requirements

Operating system:

It is recommended to use fresh CentOS 7 instances. You can deploy a virtual machine either on your local host or in the cloud. Because TiDB Lightning consumes as much CPU resources as needed by default, it is recommended that you deploy it on a dedicated server. If this is not possible, you can deploy it on a single server together with other TiDB components (for example, tikv-server) and then configure `region-concurrency` to limit the CPU usage from TiDB Lightning. Usually, you can configure the size to 75% of the logical CPU.

Memory and CPU:

It is recommended that you allocate CPU more than 4 cores and memory greater than 8 GiB to get better performance. It is verified that TiDB Lightning does not have significant memory usage (no more than 5 GiB) in the logical import mode. However, if you increase the value of `region-concurrency`, TiDB Lightning might consume more memory.

Network: A 1 Gbps or 10 Gbps Ethernet card is recommended.

13.7.7.1.2 Limitations

When you use multiple TiDB Lightning to import data to the same target, do not mix the backends. That is, do not use the physical import mode and the logical import mode to import data to a single TiDB cluster at the same time.

13.7.7.2 Use Logical Import Mode

This document introduces how to use the logical import mode in TiDB Lightning, including writing the configuration file and tuning performance.

13.7.7.2.1 Configure and use the logical import mode

You can use the logical import mode via the following configuration file to import data:

```
[lightning]
#### log
level = "info"
file = "tidb-lightning.log"
max-size = 128 # MB
max-days = 28
max-backups = 14

#### Checks the cluster minimum requirements before start.
check-requirements = true

[mydumper]
#### The local data source directory or the external storage URL.
data-source-dir = "/data/my_database"

[tikv-importer]
#### Import mode. "tidb" means using the logical import mode.
backend = "tidb"

#### The operation of inserting duplicate data in the logical import mode.
#### - replace: replace existing data with new data
#### - ignore: keep existing data and ignore new data
#### - error: pause the import and report an error
on-duplicate = "replace"

[tidb]
#### The information of the target cluster. The address of any tidb-server
   ↪ from the cluster.
host = "172.16.31.1"
port = 4000
user = "root"
#### Configure the password to connect to TiDB. Either plaintext or Base64
   ↪ encoded.
```

```
password = ""
#### tidb-lightning imports the TiDB library, and generates some logs.
#### Set the log level of the TiDB library.
log-level = "error"
```

For the complete configuration file, refer to TiDB Lightning Configuration.

13.7.7.2.2 Conflict detection

Conflicting data refers to two or more records with the same data in the PK or UK column. When the data source contains conflicting data, the actual number of rows in the table is different from the total number of rows returned by the query using the unique index.

In the logical import mode, you can configure the strategy for resolving conflicting data by setting the `on-duplicate` configuration item. Based on the strategy, TiDB Lightning imports data with different SQL statements.

Strategy	Default behavior of conflicting data	The corresponding SQL statement
`replace`	Replacing existing data with new data.	`REPLACE INTO ...`
`ignore`	Keeping existing data and ignoring new data.	`INSERT IGNORE INTO ...`
`error`	Pausing the import and reporting an error.	`INSERT INTO ...`

13.7.7.2.3 Performance tuning

- In the logical import mode, the performance of TiDB Lightning largely depends on the write performance of the target TiDB cluster. If the cluster hits a performance bottleneck, refer to Highly Concurrent Write Best Practices.
- If the target TiDB cluster does not hit a write bottleneck, consider increasing the value of `region-concurrency` in TiDB Lightning configuration. The default value of `region-concurrency` is the number of CPU cores. The meaning of `region-concurrency` is different between the physical import mode and the logical import mode. In the logical import mode, `region-concurrency` is the write concurrency.

  Example configuration:

  ```
  [lightning]
  region-concurrency = 32
  ```

- Adjusting the `raftstore.apply-pool-size` and `raftstore.store-pool-size` configuration items in the target TiDB cluster might improve the import speed.

13.7.8 TiDB Lightning Prechecks

Starting from TiDB 5.3.0, TiDB Lightning provides the ability to check the configuration before running a migration task. It is enabled by default. This feature automatically

performs some routine checks for disk space and execution configuration. The main purpose is to ensure that the whole subsequent import process goes smoothly.

The following table describes each check item and detailed explanation.

Check Items	Supported Version	Description
Cluster version and status	>= 5.3.0	Check whether the cluster can be connected in the configuration, and whether the TiKV/PD/TiFlash version supports the physical import mode.

Check Items	Supported Version	Description
Permissions	5.3.0	When the data source is cloud storage (Amazon S3), check whether TiDB Lightning has the necessary permissions and make sure that the import will not fail due to lack of permissions.

Check Items	Supported Version	Description
Disk space	>= 5.3.0	Check whether there is enough space on the local disk and on the TiKV cluster for importing data. TiDB Lightning samples the data sources and estimates the percentage of the index size from the sample result.

Check Items	Supported Version	Description
Region distribution status	>= 5.3.0	Check whether the Regions in the TiKV cluster are distributed evenly and whether there are too many empty Regions. If the number of empty Regions exceeds max(1000, number of tables * 3), i.e. greater than the bigger one of “1000”
	2092	

Check Items	Supported Version	Description
Exceedingly Large CSV files in the data file	5.3.0	When there are CSV files larger than 10 GiB in the backup file and auto-slicing is not enabled (StrictFormat=false), it will impact the import performance. The purpose of this check is to remind you to ensure the
	2093	

Check Items	Supported Version	Description
Recovery from breakpoints	>= 5.3.0	This check ensures that no changes are made to the source file or schema in the database during the breakpoint recovery process that would result in importing the wrong data.

Check Items	Supported Version	Description
Import into an existing table	>= 5.3.0	When importing into an already created table, it checks, as much as possible, whether the source file matches the existing table. Check if the number of columns matches. If the source file has column names, check if the column

Check Items	Supported Version	Description
Whether the target table is empty	>= 5.3.1	TiDB Lightning automatically exits with an error if the target table is not empty. If parallel import mode is enabled (`incremental - import = true`), this check item will be skipped.

13.7.9 Table Filter

The TiDB migration tools operate on all the databases by default, but oftentimes only a subset is needed. For example, you only want to work with the schemas in the form of `foo*` and `bar*` and nothing else.

Since TiDB 4.0, all TiDB migration tools share a common filter syntax to define subsets. This document describes how to use the table filter feature.

13.7.9.1 Usage

13.7.9.1.1 CLI

Table filters can be applied to the tools using multiple `-f` or `--filter` command line parameters. Each filter is in the form of `db.table`, where each part can be a wildcard (further explained in the next section). The following lists the example usage.

- BR:

```
./br backup full -f 'foo*.*' -f 'bar*.*' -s 'local:///tmp/backup'
```

```
./br restore full -f 'foo*.*' -f 'bar*.*' -s 'local:///tmp/backup'
```

- Dumpling:

```
./dumpling -f 'foo*.*' -f 'bar*.*' -P 3306 -o /tmp/data/
```

- TiDB Lightning:

```
./tidb-lightning -f 'foo*.*' -f 'bar*.*' -d /tmp/data/ --backend tidb
```

13.7.9.1.2 TOML configuration files

Table filters in TOML files are specified as array of strings. The following lists the example usage.

- TiDB Lightning:

```
[mydumper]
filter = ['foo*.*', 'bar*.*']
```

- TiCDC:

```
[filter]
rules = ['foo*.*', 'bar*.*']

[[sink.dispatchers]]
matcher = ['db1.*', 'db2.*', 'db3.*']
dispatcher = 'ts'
```

13.7.9.2 Syntax

13.7.9.2.1 Plain table names

Each table filter rule consists of a "schema pattern" and a "table pattern", separated by a dot (`.`). Tables whose fully-qualified name matches the rules are accepted.

```
db1.tbl1
db2.tbl2
db3.tbl3
```

A plain name must only consist of valid identifier characters, such as:

- digits (`0` to `9`)
- letters (`a` to `z`, `A` to `Z`)
- `$`
- `_`
- non ASCII characters (U+0080 to U+10FFFF)

All other ASCII characters are reserved. Some punctuations have special meanings, as described in the next section.

13.7.9.2.2 Wildcards

Each part of the name can be a wildcard symbol described in fnmatch(3):

- `*` — matches zero or more characters
- `?` — matches one character
- `[a-z]` — matches one character between "a" and "z" inclusively
- `[!a-z]` — matches one character except "a" to "z".

```
db[0-9].tbl[0-9a-f][0-9a-f]
data.*
*.backup_*
```

"Character" here means a Unicode code point, such as:

- U+00E9 (é) is 1 character.
- U+0065 U+0301 (é) are 2 characters.
- U+1F926 U+1F3FF U+200D U+2640 U+FE0F (🤦🏿‍♀️) are 5 characters.

13.7.9.2.3 File import

To import a file as the filter rule, include an `@` at the beginning of the rule to specify the file name. The table filter parser treats each line of the imported file as additional filter rules.

For example, if a file `config/filter.txt` has the following content:

```
employees.*
*.WorkOrder
```

the following two invocations are equivalent:

```
./dumpling -f '@config/filter.txt'
./dumpling -f 'employees.*' -f '*.WorkOrder'
```

A filter file cannot further import another file.

13.7.9.2.4 Comments and blank lines

Inside a filter file, leading and trailing white-spaces of every line are trimmed. Furthermore, blank lines (empty strings) are ignored.

A leading `#` marks a comment and is ignored. `#` not at start of line is considered syntax error.

```
### this line is a comment
db.table  # but this part is not comment and may cause error
```

13.7.9.2.5 Exclusion

An `!` at the beginning of the rule means the pattern after it is used to exclude tables from being processed. This effectively turns the filter into a block list.

```
*.*
#^ note: must add the *.* to include all tables first
!*.Password
!employees.salaries
```

13.7.9.2.6 Escape character

To turn a special character into an identifier character, precede it with a backslash `\`.

```
db\.with\.dots.*
```

For simplicity and future compatibility, the following sequences are prohibited:

- `\` at the end of the line after trimming whitespaces (use `[ ]` to match a literal whitespace at the end).
- `\` followed by any ASCII alphanumeric character (`[0-9a-zA-Z]`). In particular, C-like escape sequences like `\0`, `\r`, `\n` and `\t` currently are meaningless.

13.7.9.2.7 Quoted identifier

Besides `\`, special characters can also be suppressed by quoting using `"` or `` ` ``.

```
"db.with.dots"."tbl\1"
`db.with.dots`.`tbl\2`
```

The quotation mark can be included within an identifier by doubling itself.

```
"foo""bar".`foo``bar`
### equivalent to:
foo\"bar.foo\`bar
```

Quoted identifiers cannot span multiple lines.

It is invalid to partially quote an identifier:

```
"this is "invalid*.*
```

13.7.9.2.8 Regular expression

In case very complex rules are needed, each pattern can be written as a regular expression delimited with `/`:

```
/^db\d{2,}$/./^tbl\d{2,}$/
```

These regular expressions use the Go dialect. The pattern is matched if the identifier contains a substring matching the regular expression. For instance, `/b/` matches `db01`.

> **Note:**
>
> Every `/` in the regular expression must be escaped as `\/`, including inside `[⋯]`. You cannot place an unescaped `/` between `\Q⋯\E`.

13.7.9.3 Multiple rules

When a table name matches none of the rules in the filter list, the default behavior is to ignore such unmatched tables.

To build a block list, an explicit `*.*` must be used as the first rule, otherwise all tables will be excluded.

```
### every table will be filtered out
./dumpling -f '!*.Password'

### only the "Password" table is filtered out, the rest are included.
./dumpling -f '*.*' -f '!*.Password'
```

In a filter list, if a table name matches multiple patterns, the last match decides the outcome. For instance:

```
### rule 1
employees.*
### rule 2
!*.dep*
### rule 3
*.departments
```

The filtered outcome is as follows:

Table name	Rule 1	Rule 2	Rule 3	Outcome
irrelevant.table				Default (reject)
employees.employees				Rule 1 (accept)
employees.dept_emp				Rule 2 (reject)
employees.departments				Rule 3 (accept)
else.departments				Rule 3 (accept)

Note:

In TiDB tools, the system schemas are always excluded in the default configuration. The system schemas are:

- `INFORMATION_SCHEMA`
- `PERFORMANCE_SCHEMA`
- `METRICS_SCHEMA`
- `INSPECTION_SCHEMA`
- `mysql`
- `sys`

13.7.10 TiDB Lightning Checkpoints

Importing a large database usually takes hours or days, and if such long running processes spuriously crashes, it can be very time-wasting to redo the previously completed tasks. To solve this, TiDB Lightning uses *checkpoints* to store the import progress, so that `tidb-lightning` continues importing from where it lefts off after restarting.

This document describes how to enable, configure, store, and control *checkpoints*.

13.7.10.1 Enable and configure checkpoints

```
[checkpoint]
### Whether to enable checkpoints.
### While importing data, TiDB Lightning records which tables have been imported, so
### even if TiDB Lightning or some other component crashes, you can start from a known
### good state instead of restarting from scratch.
enable = true

### Where to store the checkpoints.
###  - file: store as a local file (requires v2.1.1 or later)
###  - mysql: store into a remote MySQL-compatible database
driver = "file"

### The schema name (database name) to store the checkpoints
### Enabled only when `driver = "mysql"`.
### schema = "tidb_lightning_checkpoint"

### The data source name (DSN) indicating the location of the checkpoint storage.
### # For the "file" driver, the DSN is a path. If the path is not specified, Lightning would
### default to "/tmp/CHECKPOINT_SCHEMA.pb".
### # For the "mysql" driver, the DSN is a URL in the form of "USER:PASS@tcp(HOST:PORT)/".
### If the URL is not specified, the TiDB server from the [tidb] section is used to
### store the checkpoints. You should specify a different MySQL-compatible
### database server to reduce the load of the target TiDB cluster.
#dsn = "/tmp/tidb_lightning_checkpoint.pb"

### Whether to keep the checkpoints after all data are imported. If false, the
### checkpoints are deleted. Keeping the checkpoints can aid debugging but
### might leak metadata about the data source.
```

```
### keep-after-success = false
```

13.7.10.2 Checkpoints storage

TiDB Lightning supports two kinds of checkpoint storage: a local file or a remote MySQL-compatible database.

- With `driver = "file"`, checkpoints are stored in a local file at the path given by the `dsn` setting. Checkpoints are updated rapidly, so we highly recommend placing the checkpoint file on a drive with very high write endurance, such as a RAM disk.
- With `driver = "mysql"`, checkpoints can be saved in any databases compatible with MySQL 5.7 or later, including MariaDB and TiDB. By default, the checkpoints are saved in the target database.

While using the target database as the checkpoints storage, Lightning is importing large amounts of data at the same time. This puts extra stress on the target database and sometimes leads to communication timeout. Therefore, **it is strongly recommended to install a temporary MySQL server to store these checkpoints**. This server can be installed on the same host as `tidb-lightning` and can be uninstalled after the importer progress is completed.

13.7.10.3 Checkpoints control

If `tidb-lightning` exits abnormally due to unrecoverable errors (for example, data corruption), it refuses to reuse the checkpoints until the errors are resolved. This is to prevent worsening the situation. The checkpoint errors can be resolved using the `tidb-lightning-ctl` program.

13.7.10.3.1 `--checkpoint-error-destroy`

```
tidb-lightning-ctl --checkpoint-error-destroy='`schema`.`table`'
```

This option allows you to restart importing the table from scratch. The schema and table names must be quoted with backquotes and are case-sensitive.

- If importing the table `` `schema`.`table` `` failed previously, this option executes the following operations:
    1. DROPs the table `` `schema`.`table` `` from the target database, which means removing all imported data.
    2. Resets the checkpoints record of this table to be "not yet started".
- If there is no errors involving the table `` `schema`.`table` ``, this operation does nothing.

It is the same as applying the above on every table. This is the most convenient, safe and conservative solution to fix the checkpoint error problem:

```
tidb-lightning-ctl --checkpoint-error-destroy=all
```

13.7.10.3.2 `--checkpoint-error-ignore`

```
tidb-lightning-ctl --checkpoint-error-ignore='`schema`.`table`'
tidb-lightning-ctl --checkpoint-error-ignore=all
```

If importing the table `` `schema`.`table` `` failed previously, this clears the error status as if nothing ever happened. The `all` variant applies this operation to all tables.

> **Note:**
>
> Use this option only when you are sure that the error can indeed be ignored. If not, some imported data can be lost. The only safety net is the final "checksum" check, and thus you need to keep the "checksum" option always enabled when using `--checkpoint-error-ignore`.

13.7.10.3.3 `--checkpoint-remove`

```
tidb-lightning-ctl --checkpoint-remove='`schema`.`table`'
tidb-lightning-ctl --checkpoint-remove=all
```

This option simply removes all checkpoint information about one table or all tables, regardless of their status.

13.7.10.3.4 `--checkpoint-dump`

```
tidb-lightning-ctl --checkpoint-dump=output/directory
```

This option dumps the content of the checkpoint into the given directory, which is mainly used for debugging by the technical staff. This option is only enabled when `driver = "mysql ↪ "`.

13.7.11 Use TiDB Lightning to Import Data in Parallel

Since v5.3.0, the physical import mode of TiDB Lightning supports the parallel import of a single table or multiple tables. By simultaneously running multiple TiDB Lightning instances, you can import data in parallel from different single tables or multiple tables. In

this way, TiDB Lightning provides the ability to scale horizontally, which greatly reduces the time required to import large amount of data.

In technical implementation, TiDB Lightning records the meta data of each instance and the data of each imported table in the target TiDB, and coordinates the Row ID allocation range of different instances, the record of global Checksum, and the configuration changes and recovery of TiKV and PD.

You can use TiDB Lightning to import data in parallel in the following scenarios:

- Import sharded schemas and sharded tables. In this scenario, multiple tables from multiple upstream database instances are imported into the downstream TiDB database by different TiDB Lightning instances in parallel.
- Import single tables in parallel. In this scenario, single tables stored in a certain directory or cloud storage (such as Amazon S3) are imported into the downstream TiDB cluster by different TiDB Lightning instances in parallel. This is a new feature introduced in TiDB 5.3.0.

Note:

- Parallel import only supports initialized empty tables in TiDB and does not support migrating data to tables with data written by existing services. Otherwise, data inconsistencies may occur.
- Parallel import is usually used in the physical import mode. You need to configure `incremental-import = true`.
- Apply only one backend at a time when using multiple TiDB Lightning instances to import data to the same target. For example, you cannot import data to the same TiDB cluster in both the physical and logical import modes at the same time.

13.7.11.1 Considerations

To use parallel import, you need to configure `incremental-import = true`. When TiDB Lightning is started, it registers meta data in the downstream TiDB cluster and automatically detects whether there are other instances migrating data to the target cluster at the same time. If there is, it automatically enters the parallel import mode.

But when migrating data in parallel, you need to take the following into consideration:

- Handle conflicts between primary keys or unique indexes across multiple sharded tables
- Optimize import performance

13.7.11.1.1 Handle conflicts between primary keys or unique indexes

When using the physical import mode to import data in parallel, ensure that there are no primary key or unique index conflicts between data sources, and between the tables in the target TiDB cluster, and there are no data writes in the target table during import. Otherwise, TiDB Lightning will fail to guarantee the correctness of the imported data, and the target table will contain inconsistent indexes after the import is completed.

13.7.11.1.2 Optimize import performance

Because TiDB Lightning needs to upload the generated Key-Value data to the TiKV node where each copy of the corresponding Region is located, the import speed is limited by the size of the target cluster. It is recommended to ensure that the number of TiKV instances in the target TiDB cluster and the number of TiDB Lightning instances are greater than n:1 (n is the number of copies of the Region). At the same time, the following requirements should be met to achieve optimal import performance:

- Deploy each TiDB Lightning instance on a dedicated machine. Since one TiDB Lightning instance consumes all CPU resources by default, deploying multiple instances on a single machine cannot improve performance.
- The total size of source files for each TiDB Lightning instance performing parallel import should be smaller than 5 TiB
- The total number of TiDB Lightning instances should be smaller than 10.

When using TiDB Lightning to import shared databases and tables in parallel, choose an appropriate number of TiDB Lightning instances according to the amount of data.

- If the MySQL data volume is less than 2 TiB, you can use one TiDB Lightning instance for parallel import.
- If the MySQL data volume exceeds 2 TiB and the total number of MySQL instances is smaller than 10, it is recommended that you use one TiDB Lightning instance for each MySQL instance, and the number of parallel TiDB Lightning instances should not exceed 10.
- If the MySQL data volume exceeds 2 TiB and the total number of MySQL instances exceeds 10, it is recommended that you allocate 5 to 10 TiDB Lightning instances for importing the data exported by these MySQL instances.

Next, this document uses two examples to detail the operation steps of parallel import in different scenarios:

- Example 1: Use Dumpling + TiDB Lightning to import sharded databases and tables into TiDB in parallel
- Example 2: Import single tables in parallel

13.7.11.1.3 Restrictions

TiDB Lightning exclusively uses some resources when it is running. If you need to deploy multiple TiDB Lightning instances on a single machine (which is not recommended for production environments), or on a disk shared by multiple machines, be aware of the following usage restrictions.

- Set `tikv-importer.sorted-kv-dir` to a unique path for each TiDB Lightning instance. Multiple instances sharing the same path can lead to unintended behavior and may result in import failures or data errors.
- Store each TiDB Lightning checkpoint separately. For more information about checkpoint configurations, see TiDB Lightning Checkpoints.
    - If you set checkpoint.driver = "file" (default), make sure that the path to the checkpoint is unique for each instance.
    - If you set checkpoint.driver = "mysql", you need to set a unique schema for each instance.
- The log file for each TiDB Lightning should be set to a unique path. Sharing the same log file will impact log querying and troubleshooting.
- If you use the Web Interface or Debug API, you need to set `lightning.status-addr` to a unique address for each instance; otherwise, the TiDB Lightning process fails to start due to port conflict.

13.7.11.2 Example 1: Use Dumpling + TiDB Lightning to Import Sharded Databases and Tables into TiDB in Parallel

In this example, assume that the upstream is a MySQL cluster with 10 sharded tables, with a total size of 10 TiB. You can Use 5 TiDB Lightning instances to perform parallel import, and each instance imports 2 TiB. It is estimated that the total import time (excluding the time required for Dumpling export) can be reduced from about 40 hours to about 10 hours.

Assume that the upstream library is named `my_db`, and the name of each sharded table is `my_table_01` ~ `my_table_10`. You want to merge and import them into the downstream `my_db.my_table` table. The specific steps are described in the following sections.

13.7.11.2.1 Step 1: Use Dumpling to export data

Export two sharded tables on the 5 nodes where TiDB Lightning is deployed:

- If the two sharded tables are in the same MySQL instance, you can use the `--filter` ↪ parameter of Dumpling to directly export them. When using TiDB Lightning to import, you can specify `data-source-dir` as the directory where Dumpling exports data to;

- If the data of the two sharded tables are distributed on different MySQL nodes, you need to use Dumpling to separately export them. The exported data needs to be placed in the same parent directory but in different sub-directories. When using TiDB Lightning to perform parallel import, you need to specify `data-source-dir` as the parent directory.

For more information on how to use Dumpling to export data, see Dumpling.

13.7.11.2.2 Step 2: Configure TiDB Lightning data sources

Create a configuration file `tidb-lightning.toml`, and then add the following content:

```
[lightning]
status-addr = ":8289"

[mydumper]
### Specify the path for Dumpling to export data. If Dumpling performs
    ↪ several times and the data belongs to different directories, you can
    ↪ place all the exported data in the same parent directory and specify
    ↪ this parent directory here.
data-source-dir = "/path/to/source-dir"

[tikv-importer]
### Whether to allow importing data into tables that already have data. The
    ↪ default value is `false`.
### When using parallel import, because multiple TiDB Lightning instances
    ↪ import a table at the same time, this configuration item must be set
    ↪ to `true`.
incremental-import = true
### "local": The default mode. It applies to large dataset import, for
    ↪ example, greater than 1 TiB. However, during the import, downstream
    ↪ TiDB is not available to provide services.
### "tidb": You can use this mode for small dataset import, for example,
    ↪ smaller than 1 TiB. During the import, downstream TiDB is available
    ↪ to provide services.
backend = "local"

### Specify the path for local sorting data.
sorted-kv-dir = "/path/to/sorted-dir"
```

If the data source is stored in external storage such as Amazon S3 or GCS, see External Storages.

13.7.11.2.3 Step 3: Start TiDB Lightning to import data

During parallel import, the server configuration requirements for each TiDB Lightning node are the same as the non-parallel import mode. Each TiDB Lightning node needs to consume the same resources. It is recommended to deploy them on different servers. For detailed deployment steps, see Deploy TiDB Lightning.

Start TiDB Lightning on each server in turn. If you use `nohup` to directly start it from the command line, it might exit due to the SIGHUP signal. So it is recommended to put `nohup` in the script, for example:

```
### !/bin/bash
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out &
```

During parallel import, TiDB Lightning automatically performs the following checks after starting the task.

- Check whether there is enough space on the local disk (controlled by the `sort-kv-dir` configuration) and on the TiKV cluster for importing data. To learn the required disk space, see Downstream storage space requirements and Resource requirements. TiDB Lightning samples the data sources and estimates the percentage of the index size from the sample result. Because indexes are included in the estimation, there might be cases where the size of the source data is less than the available space on the local disk, but still the check fails.
- Check whether the regions in the TiKV cluster are distributed evenly and whether there are too many empty regions. If the number of empty regions exceeds max(1000, number of tables * 3), i.e. greater than the bigger one of "1000" or "3 times the number of tables", then the import cannot be executed.
- Check whether the data is imported in order from the data sources. The size of `mydumper.batch-size` is automatically adjusted based on the result of the check. Therefore, the `mydumper.batch-size` configuration is no longer available.

You can also turn off the check and perform a forced import with the `lightning.check-requirements` configuration. For more detailed checks, see TiDB Lightning prechecks

13.7.11.2.4 Step 4: Check the import progress

After starting the import, you can check the progress in either of the following ways:

- Check the progress through the `grep` log keyword `progress`. It is updated every 5 minutes by default.
- Check the progress through the monitoring console. For details, see TiDB Lightning Monitoring.

Wait for all TiDB Lightning instances to finish, then the entire import is completed.

13.7.11.3 Example 2: Import single tables in parallel

TiDB Lightning also supports parallel import of single tables. For example, import multiple single tables stored in Amazon S3 by different TiDB Lightning instances into the downstream TiDB cluster in parallel. This method can speed up the overall import speed. For more information on external storages, see External Storages).

> **Note:**
>
> In the local environment, you can use the `--filesize` or `--where` parameter of Dumpling to divide the data of a single table into different parts and export it to the local disks of multiple servers in advance. This way, you can still perform parallel import. The configuration is the same as Example 1.

Assuming that the source files are stored in Amazon S3, the table files are `my_db.my_table.00001.sql` ~ `my_db.my_table.10000.sql`, a total of 10,000 SQL files. If you want to use 2 TiDB Lightning instances to speed up the import, you need to add the following settings in the configuration file:

```
[[mydumper.files]]
### the db schema file
pattern = '(?i)^(?:[^/]*/)*my_db-schema-create\.sql'
schema = "my_db"
type = "schema-schema"

[[mydumper.files]]
### the table schema file
pattern = '(?i)^(?:[^/]*/)*my_db\.my_table-schema\.sql'
schema = "my_db"
table = "my_table"
type = "table-schema"

[[mydumper.files]]
### Only import 00001~05000 and ignore other files
pattern = '(?i)^(?:[^/]*/)*my_db\.my_table\.(0[0-4][0-9][0-9][0-9]|05000)\.
    ↪ sql'
schema = "my_db"
table = "my_table"
type = "sql"

[tikv-importer]
### Whether to allow importing data into tables that already have data. The
    ↪ default value is `false`.
```

```
### When using parallel import, because multiple TiDB Lightning instances
    ↪ import a table at the same time, this configuration item must be set
    ↪ to `true`.
incremental-import = true
```

You can modify the configuration of the other instance to only import the `05001 ~ 10000` data files.

For other steps, see the relevant steps in Example 1.

13.7.11.4 Handle errors

13.7.11.4.1 Some TiDB Lightning nodes exit abnormally

If one or more TiDB Lightning nodes exit abnormally during a parallel import, identify the cause based on the logged error, and handle the error according to the error type:

- If the error shows normal exit (for example, exit in response to a kill command) or termination by the operating system due to OOM, adjust the configuration and then restart the TiDB Lightning nodes.
- If the error has no impact on data accuracy, for example, network timeout, perform the following steps:
    1. Run the `checkpoint-error-ignore` command with the setting `--checkpoint-error-ignore=all` on all failed nodes to clean errors in the checkpoint source data.
    2. Restart these nodes to continue importing data from checkpoints.
- If you see errors in the log that result in data inaccuracies, such as a checksum mismatch indicating invalid data in the source file, you can perform the following steps to resolve this issue:
    1. Run the `checkpoint-error-destroy` command on all Lightning nodes, including successful nodes. This command removes the imported data from failed tables and resets the checkpoint status of these tables to "not yet started".
    2. Reconfigure and import the data of failed tables by using the `filter` parameter on all TiDB Lightning nodes, including normally exiting nodes.

       When you reconfigure the Lightning parallel import task, do not include the `checkpoint-error-destroy` command in the startup script of each Lightning node. Otherwise, this command deletes shared metadata used by multiple parallel import tasks, which might cause issues during data import. For example, if a second Lightning import task is started, it will delete the metadata written by the first task, leading to abnormal data import.

13.7.11.4.2 During an import, an error "Target table is calculating checksum. Please wait until the checksum is finished and try again" is reported

Some parallel imports involve a large number of tables or tables with a small volume of data. In this case, it is possible that before one or more tasks start processing a table, other tasks of this table have finished and data checksum is in progress. At this time, an error `Target table is calculating checksum. Please wait until the checksum is finished and try again` is reported. In this case, you can wait for the completion of checksum and then restart the failed tasks. The error disappears and data accuracy is not affected.

13.7.12 TiDB Lightning Error Resolution

Starting from v5.4.0, you can configure TiDB Lightning to skip errors like invalid type conversion and unique key conflicts, and to continue the data processing as if those wrong row data does not exist. A report will be generated for you to read and manually fix errors afterward. This is ideal for importing from a slightly dirty data source, where locating the errors manually is difficult and restarting TiDB Lightning on every encounter is costly.

This document introduces how to use the type error feature (`lightning.max-error`) and the duplicate resolution feature (`tikv-importer.duplicate-resolution`). It also introduces the database where these errors are stored (`lightning.task-info-schema-name`). At the end of this document, an example is provided.

13.7.12.1 Type error

You can use the `lightning.max-error` configuration to increase the tolerance of errors related to data types. If this configuration is set to *N*, TiDB Lightning allows and skips up to *N* errors from the data source before it exists. The default value `0` means that no error is allowed.

These errors are recorded in a database. After the import is completed, you can view the errors in the database and process them manually. For more information, see Error Report.

```
[lightning]
max-error = 0
```

The above configuration covers the following errors:

- Invalid values (example: set `'Text'` to an INT column).
- Numeric overflow (example: set `500` to a TINYINT column)
- String overflow (example: set `'Very Long Text'` to a VARCHAR(5) column).
- Zero date-time (namely `'0000-00-00'` and `'2021-12-00'`).
- Set NULL to a NOT NULL column.
- Failed to evaluate a generated column expression.
- Column count mismatch. The number of values in the row does not match the number of columns of the table.

- Unique/Primary key conflict in TiDB-backend, when `on-duplicate = "error"`.
- Any other SQL errors.

The following errors are always fatal, and cannot be skipped by changing `max-error`:

- Syntax error (such as unclosed quotation marks) in the original CSV, SQL or Parquet file.
- I/O, network or system permission errors.

Unique/Primary key conflict in the physical import mode is handled separately and explained in the next section.

13.7.12.2 Error report

If TiDB Lightning encounters errors during the import, it outputs a statistics summary about these errors in both your terminal and the log file when it exits.

- The error report in the terminal is similar to the following table:

#	ERROR TYPE	ERROR COUNT	ERROR DATA TABLE
1	Data Type	1000	`lightning_task_info.type_error_v1`

- The error report in the TiDB Lightning log file is as follows:

```
[2022/03/13 05:33:57.736 +08:00] [WARN] [errormanager.go:459] ["Detect
    ↪ 1000 data type errors in total, please refer to table `
    ↪ lightning_task_info`.`type_error_v1` for more details"]
```

All errors are written to tables in the `lightning_task_info` database in the downstream TiDB cluster. After the import is completed, if the error data is collected, you can view the errors in the database and process them manually.

You can change the database name by configuring `lightning.task-info-schema-name`.

```
[lightning]
task-info-schema-name = 'lightning_task_info'
```

TiDB Lightning creates 3 tables in this database:

```
CREATE TABLE syntax_error_v1 (
    task_id    bigint NOT NULL,
    create_time datetime(6) NOT NULL DEFAULT now(6),
    table_name varchar(261) NOT NULL,
    path       varchar(2048) NOT NULL,
```

```
    offset     bigint NOT NULL,
    error      text NOT NULL,
    context    text
);

CREATE TABLE type_error_v1 (
    task_id    bigint NOT NULL,
    create_time datetime(6) NOT NULL DEFAULT now(6),
    table_name varchar(261) NOT NULL,
    path       varchar(2048) NOT NULL,
    offset     bigint NOT NULL,
    error      text NOT NULL,
    row_data   text NOT NULL
);

CREATE TABLE conflict_error_v1 (
    task_id    bigint NOT NULL,
    create_time datetime(6) NOT NULL DEFAULT now(6),
    table_name varchar(261) NOT NULL,
    index_name varchar(128) NOT NULL,
    key_data   text NOT NULL,
    row_data   text NOT NULL,
    raw_key    mediumblob NOT NULL,
    raw_value  mediumblob NOT NULL,
    raw_handle mediumblob NOT NULL,
    raw_row    mediumblob NOT NULL,
    KEY (task_id, table_name)
);
```

type_error_v1 records all type errors managed by the `max-error` configuration. There is one row per error.

conflict_error_v1 records all unique/primary key conflict in the Local-backend. There are 2 rows per pair of conflicts.

Column Syntax Type Conflict Description	
task_id	The TiDB Lightning task ID that generates this error
create_time	The time at which the error is recorded
table_name	The name of the table that contains the error, in the form of '`db`.`tbl`'
path	The path of the file that contains the error
offset	The byte position in the file where the error is found
error	The error message
context	The text that surrounds the error

Column Syntax	Type	Conflict	Description
index_name			The name of the unique key in conflict. It is `'PRIMARY'` for primary key conflicts.
key_data			The formatted key handle of the row that causes the error. The content is for human reference only, and not intended to be machine-readable.
row_data			The formatted row data that causes the error. The content is for human reference only, and not intended to be machine-readable
raw_key			The key of the conflicted KV pair
raw_value			The value of the conflicted KV pair
raw_handle			The row handle of the conflicted row
raw_row			The encoded value of the conflicted row

Note:

The error report records the file offset, not line/column number which is inefficient to obtain. You can quickly jump near a byte position (using 183 as example) using the following commands:

- shell, printing the first several lines.

```
head -c 183 file.csv | tail
```

- shell, printing the next several lines:

```
tail -c +183 file.csv | head
```

- vim — `:goto 183` or `183go`

13.7.12.3 Example

In this example, a data source is prepared with some known errors.

1. Prepare the database and table schema.

```
mkdir example && cd example

echo 'CREATE SCHEMA example;' > example-schema-create.sql
echo 'CREATE TABLE t(a TINYINT PRIMARY KEY, b VARCHAR(12) NOT NULL
    ↪ UNIQUE);' > example.t-schema.sql
```

2. Prepare the data.

```
cat <<EOF > example.t.1.sql

    INSERT INTO t (a, b) VALUES
    (0, NULL),           -- column is NOT NULL
    (1, 'one'),
    (2, 'two'),
    (40, 'forty'),       -- conflicts with the other 40 below
    (54, 'fifty-four'),  -- conflicts with the other 'fifty-four' below
    (77, 'seventy-seven'), -- the string is longer than 12 characters
    (600, 'six hundred'), -- the number overflows TINYINT
    (40, 'forty'),      -- conflicts with the other 40 above
    (42, 'fifty-four');  -- conflicts with the other 'fifty-four' above

EOF
```

3. Configure TiDB Lightning to enable strict SQL mode, use the Local-backend to import data, delete duplicates, and skip up to 10 errors.

```
cat <<EOF > config.toml

    [lightning]
    max-error = 10

    [tikv-importer]
    backend = 'local'
    sorted-kv-dir = '/tmp/lightning-tmp/'
    duplicate-resolution = 'remove'

    [mydumper]
    data-source-dir = '.'
    [tidb]
    host = '127.0.0.1'
    port = 4000
    user = 'root'
    password = ''
    sql-mode = 'STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE'

EOF
```

4. Run TiDB Lightning. This command will exit successfully because all errors are skipped.

```
tiup tidb-lightning -c config.toml
```

5. Verify that the imported table only contains the two normal rows:

```
$ mysql -u root -h 127.0.0.1 -P 4000 -e 'select * from example.t'
+---+-----+
| a | b   |
+---+-----+
| 1 | one |
| 2 | two |
+---+-----+
```

6. Check whether the `type_error_v1` table has caught the three rows involving type conversion:

```
$ mysql -u root -h 127.0.0.1 -P 4000 -e 'select * from
   ↪ lightning_task_info.type_error_v1;' -E

*************************** 1. row ***************************
    task_id: 1635888701843303564
create_time: 2021-11-02 21:31:42.620090
 table_name: `example`.`t`
       path: example.t.1.sql
     offset: 46
      error: failed to cast value as varchar(12) CHARACTER SET utf8mb4
         ↪ COLLATE utf8mb4_bin for column `b` (#2): [table:1048]Column
         ↪  'b' cannot be null
   row_data: (0,NULL)

*************************** 2. row ***************************
    task_id: 1635888701843303564
create_time: 2021-11-02 21:31:42.627496
 table_name: `example`.`t`
       path: example.t.1.sql
     offset: 183
      error: failed to cast value as varchar(12) CHARACTER SET utf8mb4
         ↪ COLLATE utf8mb4_bin for column `b` (#2): [types:1406]Data
         ↪ Too Long, field len 12, data len 13
   row_data: (77,'seventy-seven')

*************************** 3. row ***************************
    task_id: 1635888701843303564
create_time: 2021-11-02 21:31:42.629929
 table_name: `example`.`t`
       path: example.t.1.sql
     offset: 253
      error: failed to cast value as tinyint(4) for column `a` (#1): [
         ↪ types:1690]constant 600 overflows tinyint
```

```
   row_data: (600,'six hundred')
```

7. Check whether the `conflict_error_v1` table has caught the four rows that have unique/primary key conflicts:

```
$ mysql -u root -h 127.0.0.1 -P 4000 -e 'select * from
    ↪ lightning_task_info.conflict_error_v1;' --binary-as-hex -E

*************************** 1. row ***************************
    task_id: 1635888701843303564
create_time: 2021-11-02 21:31:42.669601
 table_name: `example`.`t`
 index_name: PRIMARY
   key_data: 40
   row_data: (40, "forty")
    raw_key: 0x7480000000000000C15F728000000000000028
  raw_value: 0x800001000000020500666F727479
 raw_handle: 0x7480000000000000C15F728000000000000028
    raw_row: 0x800001000000020500666F727479

*************************** 2. row ***************************
    task_id: 1635888701843303564
create_time: 2021-11-02 21:31:42.674798
 table_name: `example`.`t`
 index_name: PRIMARY
   key_data: 40
   row_data: (40, "forty")
    raw_key: 0x7480000000000000C15F728000000000000028
  raw_value: 0x800001000000020600666F75727479
 raw_handle: 0x7480000000000000C15F728000000000000028
    raw_row: 0x800001000000020600666F75727479

*************************** 3. row ***************************
    task_id: 1635888701843303564
create_time: 2021-11-02 21:31:42.680332
 table_name: `example`.`t`
 index_name: b
   key_data: 54
   row_data: (54, "fifty-four")
    raw_key: 0
        ↪ x7480000000000000C15F698000000000000001016669667479 2D666FFF7572000000000
        ↪
  raw_value: 0x0000000000000036
 raw_handle: 0x7480000000000000C15F728000000000000036
    raw_row: 0x800001000000020A0066696674792D666F7572
```

```
*************************** 4. row ***************************
   task_id: 1635888701843303564
create_time: 2021-11-02 21:31:42.681073
 table_name: `example`.`t`
 index_name: b
  key_data: 42
  row_data: (42, "fifty-four")
   raw_key: 0
       ↪ x74800000000000000C15F698000000000000000101666966747 92D666FFF7572000000000000
       ↪
 raw_value: 0x000000000000002A
raw_handle: 0x74800000000000000C15F72800000000000002A
   raw_row: 0x800001000000020A0066696674792D666F7572
```

13.7.13 Troubleshoot TiDB Lightning

This document summarizes the common problems you might encounter when you use TiDB Lightning and their solutions.

13.7.13.1 Import speed is too slow

Normally it takes 2 minutes per thread for TiDB Lightning to import a 256 MB data file. If the speed is much slower than this, there is an error. You can check the time taken for each data file from the log mentioning `restore chunk ⋯ takes`. This can also be observed from metrics on Grafana.

There are several reasons why TiDB Lightning becomes slow:

Cause 1: `region-concurrency` is set too high, which causes thread contention and reduces performance.

1. The setting can be found from the start of the log by searching `region-concurrency`.
2. If TiDB Lightning shares the same machine with other services (for example, TiKV Importer), `region-concurrency` must be **manually** set to 75% of the total number of CPU cores.
3. If there is a quota on CPU (for example, limited by Kubernetes settings), TiDB Lightning may not be able to read this out. In this case, `region-concurrency` must also be **manually** reduced.

Cause 2: The table schema is too complex.

Every additional index introduces a new KV pair for each row. If there are N indices, the actual size to be imported would be approximately (N+1) times the size of the Dumpling output. If the indices are negligible, you may first remove them from the schema, and add them back using `CREATE INDEX` after the import is complete.

Cause 3: Each file is too large.

TiDB Lightning works the best when the data source is broken down into multiple files of size around 256 MB so that the data can be processed in parallel. If each file is too large, TiDB Lightning might not respond.

If the data source is CSV, and all CSV files have no fields containing newline control characters (U+000A and U+000D), you can turn on "strict format" to let TiDB Lightning automatically split the large files.

```
[mydumper]
strict-format = true
```

Cause 4: TiDB Lightning is too old.

Try the latest version. Maybe there is new speed improvement.

13.7.13.2 The `tidb-lightning` process suddenly quits while running in background

It is potentially caused by starting `tidb-lightning` incorrectly, which causes the system to send a SIGHUP signal to stop the `tidb-lightning` process. In this situation, `tidb-lightning.log` usually outputs the following log:

```
[2018/08/10 07:29:08.310 +08:00] [INFO] [main.go:41] ["got signal to exit"]
    ↪ [signal=hangup]
```

It is not recommended to directly use `nohup` in the command line to start `tidb-lightning`. You can start `tidb-lightning` by executing a script.

In addition, if the last log of TiDB Lightning shows that the error is "Context canceled", you need to search for the first "ERROR" level log. This "ERROR" level log is usually followed by "got signal to exit", which indicates that TiDB Lightning received an interrupt signal and then exited.

13.7.13.3 The TiDB cluster uses lots of CPU resources and runs very slowly after using TiDB Lightning

If `tidb-lightning` abnormally exited, the cluster might be stuck in the "import mode", which is not suitable for production. The current mode can be retrieved using the following command:

```
tidb-lightning-ctl --config tidb-lightning.toml --fetch-mode
```

You can force the cluster back to "normal mode" using the following command:

```
tidb-lightning-ctl --config tidb-lightning.toml --fetch-mode
```

13.7.13.4 TiDB Lightning reports an error

13.7.13.4.1 `could not find first pair, this shouldn't happen`

This error occurs possibly because the number of files opened by TiDB Lightning exceeds the system limit when TiDB Lightning reads the sorted local files. In the Linux system, you can use the `ulimit -n` command to confirm whether the value of this system limit is too small. It is recommended that you adjust this value to `1000000` (`ulimit -n 1000000`) during the import.

13.7.13.4.2 `checksum failed: checksum mismatched remote vs local`

Cause: The checksum of a table in the local data source and the remote imported database differ. This error has several deeper reasons. You can further locate the reason by checking the log that contains `checksum mismatched`.

The lines that contain `checksum mismatched` provide the information `total_kvs: x vs y`, where `x` indicates the number of key-value pairs (KV pairs) calculated by the target cluster after the import is completed, and `y` indicates the number of key-value pairs generated by the local data source.

- If `x` is greater, it means that there are more KV pairs in the target cluster.
    - It is possible that this table is not empty before the import and therefore affects the data checksum. It is also possible that TiDB Lightning has previously failed and shut down, but did not restart correctly.
- If `y` is greater, it means that there are more KV pairs in the local data source.
    - If the checksum of the target database is all 0, it means that no import has occurred. It is possible that the cluster is too busy to receive any data.
    - It is possible that the exported data contains duplicate data, such as the UNIQUE and PRIMARY KEYs with duplicate values, or that the downstream table structure is case-insensitive while the data is case-sensitive.
- Other possible reasons
    - If the data source is machine-generated and not backed up by Dumpling, make sure the data conforms to the table limits. For example, the AUTO_INCREMENT column needs to be positive and not 0.

Solutions:

1. Delete the corrupted data using `tidb-lightning-ctl`, check the table structure and the data, and restart TiDB Lightning to import the affected tables again.

```
tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-error
    ↪ -destroy=all
```

2. Consider using an external database to store the checkpoints (change `[checkpoint] dsn`) to reduce the target database's load.

3. If TiDB Lightning was improperly restarted, see also the "How to properly restart TiDB Lightning" section in the FAQ.

13.7.13.4.3 `Checkpoint for … has invalid status:` (error code)

Cause: Checkpoint is enabled, and TiDB Lightning or TiKV Importer has previously abnormally exited. To prevent accidental data corruption, TiDB Lightning will not start until the error is addressed.

The error code is an integer smaller than 25, with possible values of 0, 3, 6, 9, 12, 14, 15, 17, 18, 20, and 21. The integer indicates the step where the unexpected exit occurs in the import process. The larger the integer is, the later step the exit occurs at.

Solutions:

If the error was caused by invalid data source, delete the imported data using `tidb-lightning-ctl` and start Lightning again.

```
tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-error-destroy=all
```

See the Checkpoints control section for other options.

13.7.13.4.4 `ResourceTemporarilyUnavailable("Too many open engines …: …")`

Cause: The number of concurrent engine files exceeds the limit specified by `tikv-importer`. This could be caused by misconfiguration. Additionally, if `tidb-lightning` exited abnormally, an engine file might be left at a dangling open state, which could cause this error as well.

Solutions:

1. Increase the value of `max-open-engines` setting in `tikv-importer.toml`. This value is typically dictated by the available memory. This could be calculated by using:

   Max Memory Usage ≈ `max-open-engines` × `write-buffer-size` × `max-write-buffer-number`

2. Decrease the value of `table-concurrency` + `index-concurrency` so it is less than `max-open-engines`.

3. Restart `tikv-importer` to forcefully remove all engine files (default to `./data.import/`). This also removes all partially imported tables, which requires TiDB Lightning to clear the outdated checkpoints.

   ```
   tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-error-destroy=all
   ```

13.7.13.4.5 `cannot guess encoding for input file, please convert to UTF-8 manually`

Cause: TiDB Lightning only recognizes the UTF-8 and GB-18030 encodings for the table schemas. This error is emitted if the file isn't in any of these encodings. It is also possible that the file has mixed encoding, such as containing a string in UTF-8 and another string in GB-18030, due to historical `ALTER TABLE` executions.

Solutions:

1. Fix the schema so that the file is entirely in either UTF-8 or GB-18030.
2. Manually `CREATE` the affected tables in the target database.
3. Set `[mydumper] character-set = "binary"` to skip the check. Note that this might introduce mojibake into the target database.

13.7.13.4.6 `[sql2kv] sql encode error = [types:1292]invalid time format: '{1970 1 1 …}'`

Cause: A table contains a column with the `timestamp` type, but the time value itself does not exist. This is either because of DST changes or the time value has exceeded the supported range (Jan 1, 1970 to Jan 19, 2038).

Solutions:

1. Ensure TiDB Lightning and the source database are using the same time zone.

   When executing TiDB Lightning directly, the time zone can be forced using the `$TZ` environment variable.

   ```
   # Manual deployment, and force Asia/Shanghai.
   TZ='Asia/Shanghai' bin/tidb-lightning -config tidb-lightning.toml
   ```

2. When exporting data using Mydumper, make sure to include the `--skip-tz-utc` flag.
3. Ensure the entire cluster is using the same and latest version of `tzdata` (version 2018i or above).

   On CentOS, run `yum info tzdata` to check the installed version and whether there is an update. Run `yum upgrade tzdata` to upgrade the package.

13.7.13.4.7 `[Error 8025: entry too large, the max entry size is 6291456]`

Cause: A single row of key-value pairs generated by TiDB Lightning exceeds the limit set by TiDB.

Solution:

Currently, the limitation of TiDB cannot be bypassed. You can only ignore this table to ensure the successful import of other tables.

13.7.13.4.8 Encounter `rpc error: code = Unimplemented ...` when TiDB Lightning switches the mode

Cause: Some node(s) in the cluster does not support `switch-mode`. For example, if the TiFlash version is earlier than `v4.0.0-rc.2`, `switch-mode` is not supported.

Solutions:

- If there are TiFlash nodes in the cluster, you can update the cluster to `v4.0.0-rc.2` or higher versions.
- Temporarily disable TiFlash if you do not want to upgrade the cluster.

13.7.13.4.9 `tidb lightning encountered error: TiDB version too old, expected '>=4.0.0', found '3.0.18'`

TiDB Lightning Local-backend only supports importing data to TiDB clusters of v4.0.0 and later versions. If you try to use Local-backend to import data to a v2.x or v3.x cluster, the above error is reported. At this time, you can modify the configuration to use Importer-backend or TiDB-backend for data import.

Some `nightly` versions might be similar to v4.0.0-beta.2. These `nightly` versions of TiDB Lightning actually support Local-backend. If you encounter this error when using a `nightly` version, you can skip the version check by setting the configuration `check-requirements = false`. Before setting this parameter, make sure that the configuration of TiDB Lightning supports the corresponding version; otherwise, the import might fail.

13.7.13.4.10 `restore table test.district failed: unknown columns in header [...]`

This error occurs usually because the CSV data file does not contain a header (the first row is not column names but data). Therefore, you need to add the following configuration to the TiDB Lightning configuration file:

```
[mydumper.csv]
header = false
```

13.7.13.5 TiDB Lightning Configuration

This document provides samples for global configuration and task configuration, and describes the usage of command-line parameters.

13.7.13.5.1 Configuration files

TiDB Lightning has two configuration classes: "global" and "task", and they have compatible structures. Their distinction arises only when the server mode is enabled. When server mode is disabled (the default), TiDB Lightning will only execute one task, and the same configuration file is used for both global and task configurations.

TiDB Lightning (Global)

```
###### tidb-lightning global configuration

[lightning]
#### The HTTP port for displaying the web interface, pulling Prometheus
   ↪ metrics, exposing debug data, and submitting import tasks (in server
   ↪ mode). Setting it to 0 disables the port.
status-addr = ':8289'

#### Server mode. Defaults to false, which means an import task starts
   ↪ immediately after you execute the command.
#### If this value is set to true, after you execute the command, TiDB
   ↪ Lightning waits until you submit an import task in the web interface.
#### See the "TiDB Lightning Web Interface" section for details.
server-mode = false

#### Logging
level = "info"
file = "tidb-lightning.log"
max-size = 128 # MB
max-days = 28
max-backups = 14
```

TiDB Lightning (Task)

```
###### tidb-lightning task configuration

[lightning]
#### Checks whether the cluster satisfies the minimum requirement before
   ↪ starting the task, and check whether TiKV has more than 10% free
   ↪ space left during running time.
#check-requirements = true

#### The maximum number of engines to be opened concurrently.
#### Each table is split into one "index engine" to store indices, and
   ↪ multiple
#### "data engines" to store row data. These settings control the maximum
#### concurrent number for each type of engines.
#### These values affect the memory and disk usage of tikv-importer.
#### The sum of these two values must not exceed the max-open-engines
   ↪ setting
#### for tikv-importer.
index-concurrency = 2
table-concurrency = 6

#### The concurrency number of data. It is set to the number of logical CPU
```

```
#### cores by default. When deploying together with other components, you
    ↪ can
#### set it to 75% of the size of logical CPU cores to limit the CPU usage.
#region-concurrency =

#### The maximum I/O concurrency. Excessive I/O concurrency causes an
    ↪ increase in
#### I/O latency because the disk's internal buffer is frequently refreshed,
#### which causes the cache miss and slows down the read speed. Depending on
    ↪  the storage
#### medium, this value might need to be adjusted for optimal performance.
io-concurrency = 5

#### The maximum number of non-fatal errors to tolerate before stopping TiDB
    ↪  Lightning.
#### Non-fatal errors are localized to a few rows, and ignoring those rows
    ↪ allows the import process to continue.
#### Setting this to N means that TiDB Lightning will stop as soon as
    ↪ possible when the (N+1)-th error is encountered.
#### The skipped rows will be inserted into tables inside the "task info"
    ↪ schema on the target TiDB, which can be configured below.
max-error = 0
#### task-info-schema-name is the name of the schema or database that stores
    ↪  TiDB Lightning execution results.
#### To disable error recording, set this to an empty string.
#### task-info-schema-name = 'lightning_task_info'

#### In parallel import mode, the schema name that stores the meta
    ↪ information for each TiDB Lightning instance in the target cluster.
    ↪ By default, the value is "lightning_metadata".
#### Configure this parameter only if parallel import is enabled.
#### **Note:**
#### - The value set for this parameter must be the same for each TiDB
    ↪ Lightning instance that participates in the same parallel import;
    ↪ otherwise, the correctness of the imported data cannot be ensured.
#### - If parallel import mode is enabled, make sure that the user used for
    ↪ import (for the tidb.user configuration) has permissions to create
    ↪ and access the databases corresponding to this configuration.
#### - TiDB Lightning removes this schema after the import is completed. So
    ↪ do not use any existing schema name to configure this parameter.
meta-schema-name = "lightning_metadata"

[security]
#### Specifies certificates and keys for TLS connections within the cluster.
#### Public certificate of the CA. Leave empty to disable TLS.
```

```
#### ca-path = "/path/to/ca.pem"
#### Public certificate of this service.
#### cert-path = "/path/to/lightning.pem"
#### Private key of this service.
#### key-path = "/path/to/lightning.key"

[checkpoint]
#### Whether to enable checkpoints.
#### While importing data, TiDB Lightning records which tables have been
   ↪ imported, so
#### even if TiDB Lightning or another component crashes, you can start from
   ↪  a known
#### good state instead of restarting from scratch.
enable = true
#### The schema name (database name) to store the checkpoints.
schema = "tidb_lightning_checkpoint"
#### Where to store the checkpoints.
#### - file: store as a local file.
#### - mysql: store into a remote MySQL-compatible database
driver = "file"
#### The data source name (DSN) indicating the location of the checkpoint
   ↪ storage.
#### For the "file" driver, the DSN is a path. If the path is not specified,
   ↪  TiDB Lightning would
#### default to "/tmp/CHECKPOINT_SCHEMA.pb".
#### For the "mysql" driver, the DSN is a URL in the form of "USER:PASS@tcp(
   ↪ HOST:PORT)/".
#### If the URL is not specified, the TiDB server from the [tidb] section is
   ↪  used to
#### store the checkpoints. You should specify a different MySQL-compatible
#### database server to reduce the load of the target TiDB cluster.
#### dsn = "/tmp/tidb_lightning_checkpoint.pb"
#### Whether to keep the checkpoints after all data are imported. If false,
   ↪ the
#### checkpoints will be deleted. Keeping the checkpoints can aid debugging
   ↪ but
#### will leak metadata about the data source.
#### keep-after-success = false

[tikv-importer]
#### "local": Physical import mode, used by default. It applies to large
   ↪ dataset import, for example, greater than 1 TiB. However, during the
   ↪ import, downstream TiDB is not available to provide services.
#### "tidb": Logical import mode. You can use this mode for small dataset
   ↪ import, for example, smaller than 1 TiB. During the import,
```

```
   ↪ downstream TiDB is available to provide services.
#### backend = "local"
#### Whether to enable multiple TiDB Lightning instances (in physical import
   ↪  mode) to import data to one or more target tables in parallel. The
   ↪ default value is `false`.
#### When you use parallel import mode, you must set the parameter to `true
   ↪ `, but the premise is that no data exists in the target table, that
   ↪ is, all data can only be imported by TiDB Lightning. Note that this
   ↪ parameter **is not for incremental data import** and is only used in
   ↪ scenarios where the target table is empty.
#### incremental-import = false

#### The listening address of tikv-importer when backend is "importer".
   ↪ Change it to the actual address.
addr = "172.16.31.10:8287"
#### Action to do when trying to insert a conflicting record in the logical
   ↪ import mode. For more information on the conflict detection, see the
   ↪ document: https://docs.pingcap.com/tidb/dev/tidb-lightning-logical-
   ↪ import-mode-usage#conflict-detection
####  - replace: use new entry to replace the existing entry
####  - ignore: keep the existing entry, and ignore the new entry
####  - error: report error and quit the program
#### on-duplicate = "replace"

#### Whether to detect and resolve duplicate records (unique key conflict)
   ↪ in the physical import mode.
#### The following resolution algorithms are supported:
####  - record: After the data is written to the target table, add the
   ↪ duplicate records from the target table to the `lightning_task_info.
   ↪ conflict_error_v1` table in the target TiDB. Note that the required
   ↪ version of the target TiKV is no earlier than v5.2.0; otherwise it
   ↪ falls back to 'none'.
####  - none: does not detect duplicate records, which has the best
   ↪ performance of the three algorithms. But if there are duplicate
   ↪ records in the data source, it might lead to inconsistent data in the
   ↪  target TiDB.
####  - remove: records all duplicate records in the target table to the
   ↪ lightning_task_info database, like the 'record' algorithm. But it
   ↪ removes all duplicate records from the target table to ensure a
   ↪ consistent state in the target TiDB.
#### duplicate-resolution = 'none'
#### The number of KV pairs sent in one request in the physical import mode.
#### send-kv-pairs = 32768
#### The directory of local KV sorting in the physical import mode. If the
   ↪ disk
```

```
#### performance is low (such as in HDD), it is recommended to set the
    ↪ directory
#### on a different disk from `data-source-dir` to improve import speed.
#### sorted-kv-dir = ""
#### The concurrency that TiKV writes KV data in the physical import mode.
#### When the network transmission speed between TiDB Lightning and TiKV
#### exceeds 10 Gigabit, you can increase this value accordingly.
#### range-concurrency = 16

[mydumper]
#### Block size for file reading. Keep it longer than the longest string of
    ↪ the data source.
read-block-size = "64KiB" # default value

#### The engine file needs to be imported sequentially. Due to parallel
    ↪ processing,
#### multiple data engines will be imported at nearly the same time, and
    ↪ this
#### creates a queue and wastes resources. Therefore, TiDB Lightning
    ↪ slightly
#### increases the size of the first few batches to properly distribute
#### resources. The scale up factor is controlled by this parameter, which
#### expresses the ratio of duration between the "import" and "write" steps
#### with full concurrency. This can be calculated by using the ratio
#### (import duration/write duration) of a single table of size around 1 GiB
    ↪ .
#### The exact timing can be found in the log. If "import" is faster, the
    ↪ batch
#### size variance is smaller, and a ratio of zero means a uniform batch
    ↪ size.
#### This value should be in the range (0 <= batch-import-ratio < 1).
batch-import-ratio = 0.75

#### Local source data directory or the URL of the external storage.
data-source-dir = "/data/my_database"

#### The character set of the schema files, containing CREATE TABLE
    ↪ statements;
#### only supports one of:
#### - utf8mb4: the schema files must be encoded as UTF-8; otherwise, an
    ↪ error is reported.
#### - gb18030: the schema files must be encoded as GB-18030; otherwise,
####           an error is reported
#### - auto:   (default) automatically detects whether the schema is UTF-8
    ↪ or
```

```
####           GB-18030. An error is reported if the encoding is neither.
#### - binary: do not try to decode the schema files
character-set = "auto"

#### Specifies the character set of the source data file. Lightning converts
   ↪  the source file from the specified character set to UTF-8 encoding
   ↪ when importing.
#### Currently, this configuration only specifies the character set of the
   ↪ CSV files with the following options supported:
#### - utf8mb4: Indicates that the source data file uses UTF-8 encoding.
#### - GB18030: Indicates that the source data file uses the GB-18030
   ↪ encoding.
#### - GBK: The source data file uses GBK encoding (GBK encoding is an
   ↪ extension of the GB-2312 character set, also known as Code Page 936).
#### - binary: Indicates that Lightning does not convert the encoding (by
   ↪ default).
#### If left blank, the default value "binary" is used, that is to say,
   ↪ Lightning does not convert the encoding.
#### Note that Lightning does not predict about the character set of the
   ↪ source data file and only converts the source file and import the
   ↪ data based on this configuration.
#### If the value of this configuration is not the same as the actual
   ↪ encoding of the source data file, a failed import, data loss or data
   ↪ disorder might appear.
data-character-set = "binary"
#### Specifies the replacement character in case of incompatible characters
   ↪ during the character set conversion of the source data file.
#### This configuration must not be duplicated with field separators, quote
   ↪ definers, and line breaks.
#### The default value is "\uFFFD", which is the "error" Rune or Unicode
   ↪ replacement character in UTF-8 encoding.
#### Changing the default value might result in potential degradation of
   ↪ parsing performance for the source data file.
data-invalid-char-replace = "\uFFFD"

#### the input data in a "strict" format speeds up processing.
#### "strict-format = true" requires that:
#### in CSV, every value cannot contain literal new lines (U+000A and U+000D
   ↪ , or \r and \n) even
#### when quoted, which means new lines are strictly used to separate rows.
#### "Strict" format allows TiDB Lightning to quickly locate split positions
   ↪  of a large file for parallel processing.
#### However, if the input data is not "strict", it may split a valid data
   ↪ in half and
#### corrupt the result.
```

```
#### The default value is false for safety instead of speed.
strict-format = false

#### If strict-format is true, TiDB Lightning splits large CSV files into
   ↪ multiple chunks to process in
#### parallel. max-region-size is the maximum size of each chunk after
   ↪ splitting.
#### max-region-size = "256MiB" # default value

#### Only import tables if these wildcard rules are matched. See the
   ↪ corresponding section for details.
filter = ['*.*', '!mysql.*', '!sys.*', '!INFORMATION_SCHEMA.*', '!
   ↪ PERFORMANCE_SCHEMA.*', '!METRICS_SCHEMA.*', '!INSPECTION_SCHEMA.*']

#### Configures how CSV files are parsed.
[mydumper.csv]
#### Separator between fields. Must not be empty.
separator = ','
#### Quoting delimiter. Empty value means no quoting.
delimiter = '"'
#### Line terminator. Empty value means both "\n" (LF) and "\r\n" (CRLF) are
   ↪  line terminators.
terminator = ''
#### Whether the CSV files contain a header.
#### If `header` is true, the first line will be skipped.
header = true
#### Whether the CSV contains any NULL value.
#### If `not-null` is true, all columns from CSV cannot be NULL.
not-null = false
#### When `not-null` is false (that is, CSV can contain NULL),
#### fields equal to this value will be treated as NULL.
null = '\N'
#### Whether to interpret backslash escapes inside fields.
backslash-escape = true
#### If a line ends with a separator, remove it.
trim-last-separator = false

#### [[mydumper.files]]
#### Expression used for parsing AWS Aurora parquet files
#### pattern = '(?i)^(?:[^/]*/)*([a-z0-9_]+)\.([a-z0-9_]+)/(?:[^/]*/)*(?:[a-
   ↪ z0-9\-_.]+\.(parquet))$'
#### schema = '$1'
#### table = '$2'
#### type = '$3'
```

```
[tidb]
#### Configuration of any TiDB server from the cluster.
host = "172.16.31.1"
port = 4000
user = "root"
#### Configure the password to connect to TiDB. The password can either be
   ↪ plaintext or Base64 encoded.
password = ""
#### Table schema information is fetched from TiDB via this status-port.
status-port = 10080
#### Address of any PD server from the cluster.
pd-addr = "172.16.31.4:2379"
#### tidb-lightning imports TiDB as a library and generates some logs itself
   ↪ .
#### This setting controls the log level of the TiDB library.
log-level = "error"

#### Sets the TiDB session variable to speed up the Checksum and Analyze
   ↪ operations. Note that if checksum-via-sql is set to "true", TiDB
   ↪ Lightning will execute the ADMIN CHECKSUM TABLE <table> SQL statement
   ↪  to perform the Checksum operation on TiDB. In this case, the
   ↪ following parameters `distsql-scan-concurrency` and `checksum-table-
   ↪ concurrency` will not take effect.
#### See https://docs.pingcap.com/tidb/stable/statistics#control-analyze-
   ↪ concurrency
#### for the meaning of each setting
build-stats-concurrency = 20
distsql-scan-concurrency = 100
index-serial-scan-concurrency = 20
checksum-table-concurrency = 16

#### The default SQL mode used to parse and execute the SQL statements.
sql-mode = "ONLY_FULL_GROUP_BY,NO_AUTO_CREATE_USER"
#### Sets maximum packet size allowed for SQL connections.
#### Set this to 0 to automatically fetch the `max_allowed_packet` variable
   ↪ from server on every connection.
max-allowed-packet = 67_108_864

#### Whether to use TLS for SQL connections. Valid values are:
#### - "": if configuration items in the [tidb.security] section are
   ↪ configured, TiDB Lightning requires TLS for SQL connections (same
   ↪ behavior as "cluster"). Otherwise, it uses an unencrypted connection.
#### - "false": same behavior as "".
#### - "cluster": requires TLS and verifies the server's certificate with
   ↪ the CA specified in the [tidb.security] section.
```

```
#### - "skip-verify": requires TLS but does not verify the server's
   ↪ certificate (insecure). If the server does not support TLS, the
   ↪ connection falls back to an unencrypted state.
#### - "preferred": same behavior as "skip-verify".
#### tls = ""

#### Specifies certificates and keys for TLS-enabled MySQL connections.
#### Defaults to a copy of the [security] section.
#### [tidb.security]
#### Public certificate of the CA. Set to empty string to disable TLS for
   ↪ SQL.
#### ca-path = "/path/to/ca.pem"
#### Public certificate of this service. Default to copy of `security.cert-
   ↪ path`
#### cert-path = "/path/to/lightning.pem"
#### Private key of this service. Default to copy of `security.key-path`
#### key-path = "/path/to/lightning.key"

#### In the physical import mode, when data importing is complete, TiDB
   ↪ Lightning can
#### automatically perform the Checksum and Analyze operations. It is
   ↪ recommended
#### to leave these as true in the production environment.
#### The execution order: Checksum -> Analyze.
#### Note that in the logical import mode, Checksum and Analyze is not
   ↪ needed, and they are always
#### skipped in the actual operation.
[post-restore]
#### Specifies whether to perform `ADMIN CHECKSUM TABLE <table>` for each
   ↪ table to verify data integrity after importing.
#### The following options are available:
#### - "required" (default value): Perform admin checksum. If checksum fails
   ↪ , TiDB Lightning will exit with failure.
#### - "optional": Perform admin checksum. If checksum fails, TiDB Lightning
   ↪  will report a WARN log but ignore any error.
#### - "off": Do not perform checksum.
#### Note that since v4.0.8, the default value has changed from "true" to "
   ↪ required".
#### Note:
#### 1. Checksum failure usually means import exception (data loss or
   ↪ inconsistency). It is recommended to always enable checksum.
#### 2. For backward compatibility, bool values "true" and "false" are also
   ↪ allowed for this field.
#### "true" is equivalent to "required" and "false" is equivalent to "off".
checksum = "required"
```

```
#### Specifies whether the ADMIN CHECKSUM TABLE <table> operation is
    ↪ executed via TiDB.
#### The default value is "false", which means that the ADMIN CHECKSUM TABLE
    ↪  <table> command is sent to TiKV for execution via TiDB Lightning.
#### It is recommended that you set this value to "true" to make it easier
    ↪ to locate the problem if checksum fails.
#### Meanwhile, if you want to adjust concurrency when this value is "true",
    ↪  you need to set the `tidb_checksum_table_concurrency` variable in
    ↪ TiDB (https://docs.pingcap.com/tidb/stable/system-variables#
    ↪ tidb_checksum_table_concurrency).
checksum-via-sql = "false"
#### Specifies whether to perform `ANALYZE TABLE <table>` for each table
    ↪ after checksum is done.
#### Options available for this field are the same as `checksum`. However,
    ↪ the default value for this field is "optional".
analyze = "optional"

#### Configures the background periodic actions.
#### Supported units: h (hour), m (minute), s (second).
[cron]
#### Duration between which TiDB Lightning automatically refreshes the
    ↪ import mode
#### status. Should be shorter than the corresponding TiKV setting.
switch-mode = "5m"
#### Duration between which an import progress is printed to the log.
log-progress = "5m"
```

13.7.13.5.2 Command line parameters

Usage of `tidb-lightning`

Parameter	Explanation	Corresponding setting
–config *file*	Reads global configuration from *file*. If not specified, the default configuration would be used.	
-V	Prints program version	
-d *directory*	Directory or external storage URL of the data dump to read from	`mydumper` ↪ `.` ↪ `data` ↪ `-` ↪ `source` ↪ `-` ↪ `dir` ↪

Parameter	Explanation	Corresponding setting
-L *level*	Log level: debug, info, warn, error, fatal (default = info)	`lightning.log.level`
-f *rule*	Table filter rules (can be specified multiple times)	`mydumper.filter`
–backend *backend*	Select an import mode. `local` refers to the physical import mode; `tidb` refers to the logical import mode.	`local`

Parameter	Explanation	Corresponding setting
–log-file *file*	Log file path. By default, it is `/tmp/lightning.log.{timestamp}`. If set to '-', it means that the log files will be output to stdout.	`lightning.log-file`
–status-addr *ip:port*	Listening address of the TiDB Lightning server	`lightning.status-port`
–importer *host:port*	Address of TiKV Importer	`tikv-importer.addr`

Parameter	Explanation	Corresponding setting
–pd-urls *host:port*	PD endpoint address	`tidb.pd.addr`
–tidb-host *host*	TiDB server host	`tidb.host`
–tidb-port *port*	TiDB server port (default = 4000)	`tidb.port`
–tidb-status *port*	TiDB status port (default = 10080)	`tidb.status.port`
–tidb-user *user*	User name to connect to TiDB	`tidb.user`

Parameter	Explanation	Corresponding setting
–tidb-password *password*	Password to connect to TiDB. The password can either be plaintext or Base64 encoded.	`tidb.password`
–enable-checkpoint *bool*	Whether to enable checkpoints (default = true)	`checkpoint.enable`

Parameter	Explanation	Corresponding setting
–analyze *level*	Analyze tables after importing. Available values are “required”, “optional” (default value), and “off”	`post-restore.analyze`
–checksum *level*	Compare checksum after importing. Available values are “required” (default value), “optional”, and “off”	`post-restore.checksum`

Parameter	Explanation	Corresponding setting
–check-requirements *bool*	Check cluster version compatibility before starting the task, and check whether TiKV has more than 10% free space left during running time. (default = true)	`lightning.check-requirements`

Parameter	Explanation	Corresponding setting
–ca *file*	CA certificate path for TLS connection	`security.ca-path`
–cert *file*	Certificate path for TLS connection	`security.cert-path`
–key *file*	Private key path for TLS connection	`security.key-path`
–server-mode	Start TiDB Lightning in server mode	`lightning.server-mode`

If a command line parameter and the corresponding setting in the configuration file are both provided, the command line parameter will be used. For example, running `./tidb-lightning -L debug --config cfg.toml` would always set the log level to "debug" regardless of the content of `cfg.toml`.

13.7.13.5.3 Usage of `tidb-lightning-ctl`

This tool can execute various actions given one of the following parameters:

Parameter	Explanation
–compact	Performs a full compaction
–switch-mode *mode*	Switches every TiKV store to the given mode: normal, import
–fetch-mode	Prints the current mode of every TiKV store
–import-engine *uuid*	Imports the closed engine file from TiKV Importer into the TiKV cluster

Parameter	Explanation
–cleanup-engine *uuid*	Deletes the engine file from TiKV Importer
–checkpoint-dump *folder*	Dumps current checkpoint as CSVs into the folder
–checkpoint-error-destroy *table-name*	Removes the checkpoint and drops the table if it caused error
–checkpoint-error-ignore *table-name*	Ignores any error recorded in the checkpoint involving the given table

Parameter	Explanation
–checkpoint-remove *tablename*	Unconditionally removes the checkpoint of the table

The *tablename* must either be a qualified table name in the form `` `db`.`tbl` `` (including the backquotes), or the keyword "all".

Additionally, all parameters of `tidb-lightning` described in the section above are valid in `tidb-lightning-ctl`.

13.7.13.6 TiDB Lightning Command Line Flags

You can configure TiDB Lightning either using the configuration file or in command line. This document describes the command line flags of TiDB Lightning.

13.7.13.6.1 Command line flags

`tidb-lightning`

You can configure the following parameters using `tidb-lightning`:

Parameter	Description	Corresponding configuration item
`--config <file>`	Read the global configuration from the file. If this parameter is not specified, TiDB Lightning uses the default configuration.	
`-V`	Print the program version.	

Parameter	Description	Corresponding configuration item
`-d <directory>`	Local directory or external storage URL of data files.	`mydumper.data-source-dir`
`-L <level>`	Log level: `debug`, `info`, `warn`, `error`, or `fatal`. `info` by default.	`lightning.level`
`-f <rule>`	Table filter rules. Can be specified multiple times.	`mydumper.filter`

Parameter	Description	Corresponding configuration item
`--backend <backend>`	Select an import mode. `local` refers to physical import mode; `tidb` refers to logical import mode.	`tikv-importer.backend`

Parameter	Description	Corresponding configuration item
`--log-file <file>`	Log file path. By default, it is `/tmp/lightning.log.{timestamp}`. If set to '-', it means that the log files will be output to stdout.	`lightning.log-file`
`--status-addr <ip:port>`	Listening address of the TiDB Lightning server	`lightning.status-port`

Parameter	Description	Corresponding configuration item
`--importer <host>:<port>`	Address of TiKV Importer	`tikv-importer.addr`
`--pd-urls <host>:<port>`	PD endpoint address	`tidb.pd-addr`
`--tidb-host <host>`	TiDB server host	`tidb.host`
`--tidb-port <port>`	TiDB server port (default = 4000)	`tidb.port`
`--tidb-status <port>`	TiDB status port (default = 10080)	`tidb.status-port`

Parameter	Description	Corresponding configuration item
`--tidb-user` <user>	User name to connect to TiDB	`tidb.user`
`--tidb-password` <password>	Password to connect to TiDB. The password can either be plaintext or Base64 encoded.	`tidb.password`
`--enable-checkpoint` <bool>	Whether to enable checkpoints (default = true)	`checkpoint.enable`

Parameter	Description	Corresponding configuration item
`--analyze <level>`	Analyze tables after importing. Available values are "required", "optional" (default value), and "off".	`post-restore.analyze`
`--checksum <level>`	Compare checksum after importing. Available values are "required" (default value), "optional", and "off".	`post-restore.checksum`

Parameter	Description	Corresponding configuration item
`--check-requirements <bool>`	Check cluster version compatibility before starting the task, and check whether TiKV has more than 10% free space left during running time. (default = true)	`lightning.check-requirements`

Parameter	Description	Corresponding configuration item
`--ca < file >`	CA certificate path for TLS connection	`security . ca - path`
`--cert < file >`	Certificate path for TLS connection	`security . cert - path`
`--key < file >`	Private key path for TLS connection	`security . key - path`
`--server-mode`	Start TiDB Lightning in server mode	`lightning . server - mode`

If you specify both a command line parameter and the corresponding setting in the configuration file, the command line parameter takes precedence. For example, running `./tidb -lightning -L debug --config cfg.toml` would always set the log level to "debug" regardless of the content of `cfg.toml`.

13.7.13.6.2 `tidb-lightning-ctl`

All parameters of `tidb-lightning` apply to `tidb-lightning-ctl`. In addition, you can also configure the following parameters using `tidb-lightning-ctl`:

Parameter	Description
`--compact`	Perform a full compaction.
`--switch-mode <mode>`	Switch every TiKV store to the given mode: normal or import.
`--fetch-mode`	Print the current mode of every TiKV store.
`--import-engine <uuid>`	Import the closed engine file from TiKV Importer into the TiKV cluster.

Parameter	Description
`--cleanup-engine`	Delete the engine file from TiKV Importer.
`--checkpoint-dump <folder>`	Dump the current checkpoint as CSVs into the folder.
`--checkpoint-error-destroy`	Remove the checkpoint, if it contains an error, and drop the table.
`--checkpoint-error-ignore`	Ignore the checkpoint error recorded in the checkpoint involving the given table.

Parameter	Description
`--checkpoint-remove <table_name>`	Unconditionally remove the checkpoint name of the table.

The `<table_name>` must either be a qualified table name in the form `` `db`.`tbl` `` (including the backquotes), or the keyword `all`.

13.7.13.7 TiDB Lightning Monitoring

`tidb-lightning` supports metrics collection via Prometheus. This document introduces the monitor configuration and monitoring metrics of TiDB Lightning.

13.7.13.7.1 Monitor configuration

If TiDB Lightning is manually installed, follow the instructions below.

The metrics of `tidb-lightning` can be gathered directly by Prometheus as long as it is discovered. You can set the metrics port in `tidb-lightning.toml`:

```
[lightning]
#### HTTP port for debugging and Prometheus metrics pulling (0 to disable)
pprof-port = 8289

...
```

and in `tikv-importer.toml`:

```
#### Listening address of the status server.
status-server-address = '0.0.0.0:8286'
```

You need to configure Prometheus to make it discover the servers. For instance, you can directly add the server address to the `scrape_configs` section:

```
...
scrape_configs:
  - job_name: 'tidb-lightning'
    static_configs:
      - targets: ['192.168.20.10:8289']
  - job_name: 'tikv-importer'
    static_configs:
      - targets: ['192.168.20.9:8286']
```

13.7.13.7.2 Grafana dashboard

Grafana is a web interface to visualize Prometheus metrics as dashboards.

Row 1: Speed

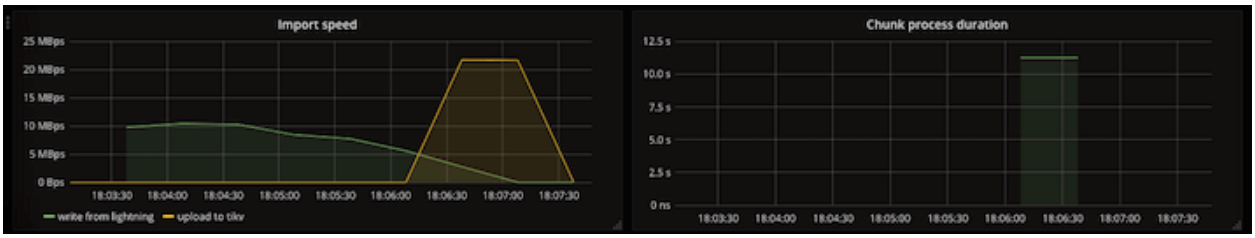


Figure 259: Panels in first row

Panel	Series	Description
Import speed	write from TiDB Lightning	Speed of sending KVs from TiDB Lightning to TiKV Importer, which depends on each table's complexity

Panel	Series	Description
Import speed	upload to tikv	Total upload speed from TiKV Importer to all TiKV replicas
Chunk process duration		Average time needed to completely encode one single data file

Sometimes the import speed will drop to zero allowing other parts to catch up. This is normal.

Row 2: Progress

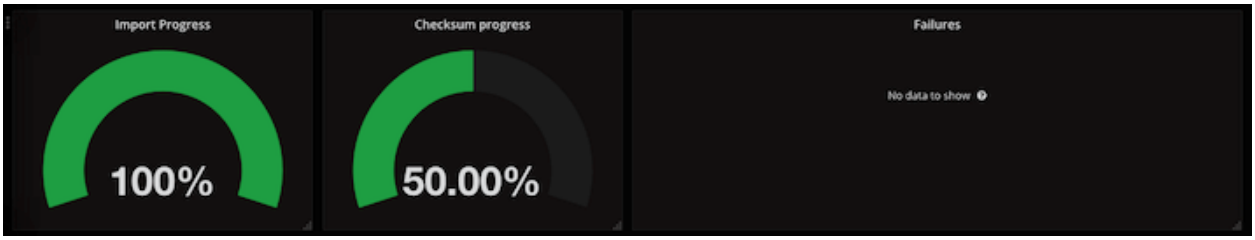


Figure 260: Panels in second row

Panel	Description
Import progress	Percentage of data files encoded so far
Checksum progress	Percentage of tables are verified to be imported successfully
Failures	Number of failed tables and their point of failure, normally empty

Row 3: Resource

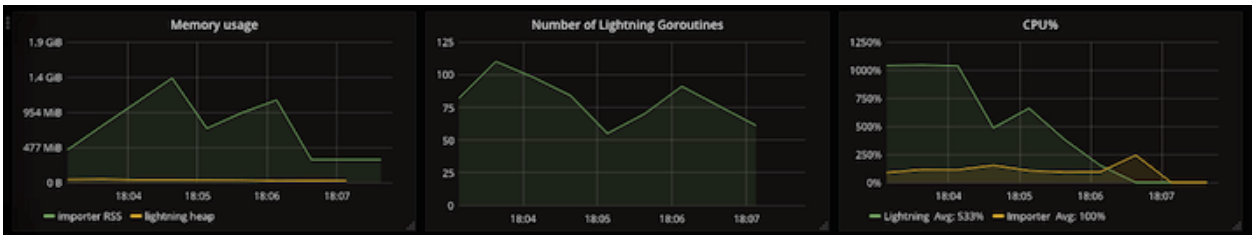


Figure 261: Panels in third row

Panel	Description
Memory usage	Amount of memory occupied by each service
Number of TiDB Lightning Goroutines	Number of running goroutines used by TiDB Lightning
CPU%	Number of logical CPU cores utilized by each service

Row 4: Quota

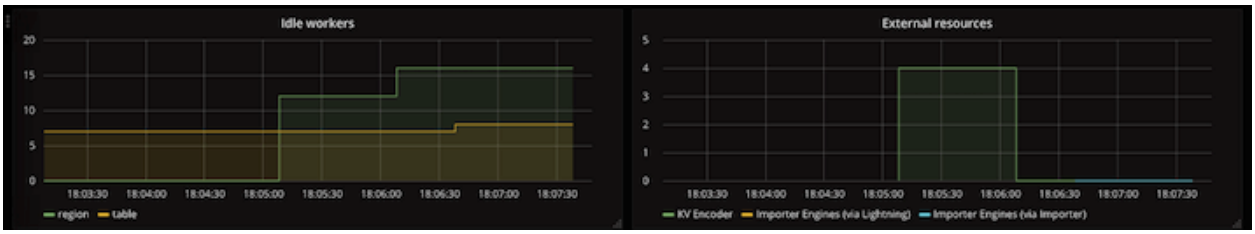


Figure 262: Panels in fourth row

Panel	Series	Description
Idle workers	io	Number of unused `io-concurrency`, normally close to configured value (default 5), and close to 0 means the disk is too slow

Panel	Series	Description
Idle workers	closed-engine	Number of engines which is closed but not yet cleaned up, normally close to index + table-concurrency (default 8), and close to 0 means TiDB Lightning is faster than TiKV Importer, which will cause TiDB Lightning to stall

Panel	Series	Description
Idle workers	table	Number of unused `table-concurrency`, normally 0 until the end of process
Idle workers	index	Number of unused `index-concurrency`, normally 0 until the end of process
Idle workers	region	Number of unused `region-concurrency`, normally 0 until the end of process

Panel	Series	Description
External resources	KV Encoder	Counts active KV encoders, normally the same as `region` ↪ `-` ↪ `concurrency` ↪ until the end of process
External resources	Importer Engines	Counts opened engine files, should never exceed the `max-` ↪ `open` ↪ `-` ↪ `engines` ↪ setting

Row 5: Read speed

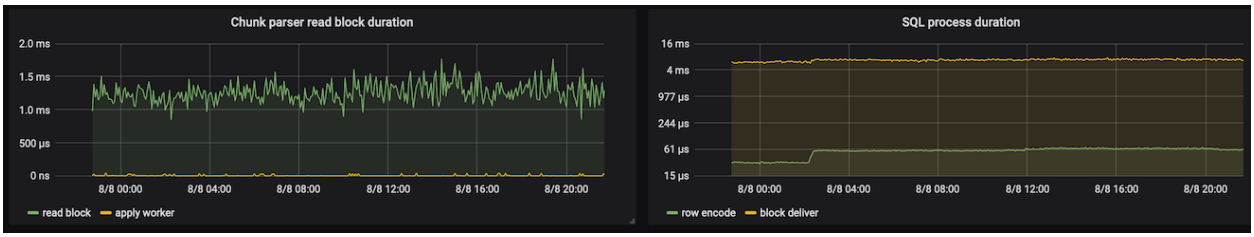


Figure 263: Panels in fifth row

Panel	Series	Description
Chunk parser read block duration	read block	Time taken to read one block of bytes to prepare for parsing
Chunk parser read block duration	apply worker	Time elapsed to wait for an idle io-concurrency
SQL process duration	row encode	Time taken to parse and encode a single row
SQL process duration	block deliver	Time taken to send a block of KV pairs to TiKV Importer

If any of the duration is too high, it indicates that the disk used by TiDB Lightning is too slow or busy with I/O.

Row 6: Storage

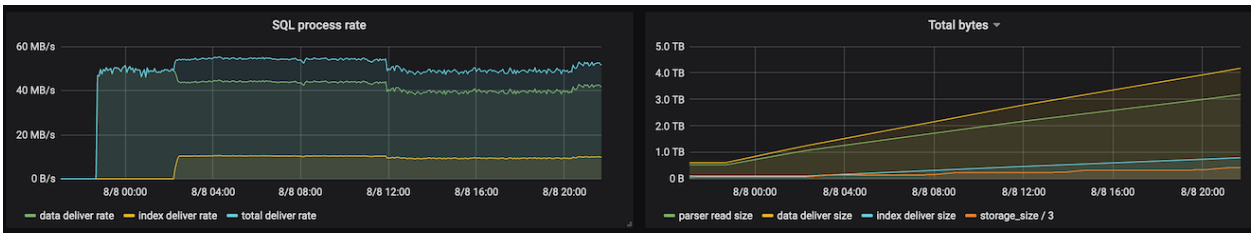


Figure 264: Panels in sixth row

Panel	Series	Description
SQL process rate	data deliver rate	Speed of delivery of data KV pairs to TiKV Importer
SQL process rate	index deliver rate	Speed of delivery of index KV pairs to TiKV Importer
SQL process rate	total deliver rate	The sum of two rates above

Panel	Series	Description
Total bytes	parser read size	Number of bytes being read by TiDB Lightning
Total bytes	data deliver size	Number of bytes of data KV pairs already delivered to TiKV Importer
Total bytes	index deliver size	Number of bytes of index KV pairs already delivered to TiKV Importer

Panel	Series	Description
Total bytes	storage_size / 3	Total size occupied by the TiKV cluster, divided by 3 (the default number of replicas)

Row 7: Import speed



Figure 265: Panels in seventh row

Panel	Series	Description
Delivery duration	Range delivery	Time taken to upload a range of KV pairs to the TiKV cluster

Panel	Series	Description
Delivery duration	SST delivery	Time taken to upload an SST file to the TiKV cluster
SST process duration	Split SST	Time taken to split the stream of KV pairs into SST files
SST process duration	SST upload	Time taken to upload an SST file
SST process duration	SST ingest	Time taken to ingest an uploaded SST file
SST process duration	SST size	File size of an SST file

13.7.13.7.3 Monitoring metrics

This section explains the monitoring metrics of `tikv-importer` and `tidb-lightning`, if you need to monitor other metrics not covered by the default Grafana dashboard.

`tikv-importer`

Metrics provided by `tikv-importer` are listed under the namespace `tikv_import_*`.

- **`tikv_import_rpc_duration`** (Histogram)

  Bucketed histogram for the duration of an RPC action. Labels:

  - **request**: what kind of RPC is executed
    - `switch_mode`: switched a TiKV node to import/normal mode
    - `open_engine`: opened an engine file
    - `write_engine`: received data and written into an engine
    - `close_engine`: closed an engine file
    - `import_engine`: imported an engine file into the TiKV cluster
    - `cleanup_engine`: deleted an engine file
    - `compact_cluster`: explicitly compacted the TiKV cluster
    - `upload`: uploaded an SST file
    - `ingest`: ingested an SST file
    - `compact`: explicitly compacted a TiKV node
  - **result**: the execution result of the RPC
    - `ok`
    - `error`

- **`tikv_import_write_chunk_bytes`** (Histogram)

  Bucketed histogram for the uncompressed size of a block of KV pairs received from TiDB Lightning.

- **`tikv_import_write_chunk_duration`** (Histogram)

  Bucketed histogram for the time needed to receive a block of KV pairs from TiDB Lightning.

- **`tikv_import_upload_chunk_bytes`** (Histogram)

  Bucketed histogram for the compressed size of a chunk of SST file uploaded to TiKV.

- **`tikv_import_upload_chunk_duration`** (Histogram)

  Bucketed histogram for the time needed to upload a chunk of SST file to TiKV.

- **`tikv_import_range_delivery_duration`** (Histogram)

  Bucketed histogram for the time needed to deliver a range of KV pairs into a `dispatch -job`.

- **`tikv_import_split_sst_duration`** (Histogram)

  Bucketed histogram for the time needed to split off a range from the engine file into a single SST file.

- **`tikv_import_sst_delivery_duration`** (Histogram)

  Bucketed histogram for the time needed to deliver an SST file from a `dispatch-job` to an `ImportSSTJob`.

- **`tikv_import_sst_recv_duration`** (Histogram)

  Bucketed histogram for the time needed to receive an SST file from a `dispatch-job` in an `ImportSSTJob`.

- **`tikv_import_sst_upload_duration`** (Histogram)

  Bucketed histogram for the time needed to upload an SST file from an `ImportSSTJob` to a TiKV node.

- **`tikv_import_sst_chunk_bytes`** (Histogram)

  Bucketed histogram for the compressed size of the SST file uploaded to a TiKV node.

- **`tikv_import_sst_ingest_duration`** (Histogram)

  Bucketed histogram for the time needed to ingest an SST file into TiKV.

- **`tikv_import_each_phase`** (Gauge)

  Indicates the running phase. Possible values are 1, meaning running inside the phase, and 0, meaning outside the phase. Labels:

  - **phase**: `prepare/import`

- **`tikv_import_wait_store_available_count`** (Counter)

  Counts the number of times a TiKV node is found to have insufficient space when uploading SST files. Labels:

  - **store_id**: The TiKV store ID.

`tidb-lightning`

Metrics provided by `tidb-lightning` are listed under the namespace `lightning_*`.

- **`lightning_importer_engine`** (Counter)

  Counts open and closed engine files. Labels:

  - **type**:
    - `open`
    - `closed`

- **`lightning_idle_workers`** (Gauge)

  Counts idle workers. Labels:

    - **name**:
        - `table`: the remainder of `table-concurrency`, normally 0 until the end of the process
        - `index`: the remainder of `index-concurrency`, normally 0 until the end of the process
        - `region`: the remainder of `region-concurrency`, normally 0 until the end of the process
        - `io`: the remainder of `io-concurrency`, normally close to configured value (default 5), and close to 0 means the disk is too slow
        - `closed-engine`: number of engines which have been closed but not yet cleaned up, normally close to index + table-concurrency (default 8). A value close to 0 means TiDB Lightning is faster than TiKV Importer, which might cause TiDB Lightning to stall

- **`lightning_kv_encoder`** (Counter)

  Counts open and closed KV encoders. KV encoders are in-memory TiDB instances that convert SQL `INSERT` statements into KV pairs. The net values need to be bounded in a healthy situation. Labels:

    - **type**:
        - `open`
        - `closed`

- **`lightning_tables`** (Counter)

  Counts processed tables and their statuses. Labels:

    - **state**: the status of the table, indicating which phase should be completed
        - `pending`: not yet processed
        - `written`: all data encoded and sent
        - `closed`: all corresponding engine files closed
        - `imported`: all engine files have been imported into the target cluster
        - `altered_auto_inc`: AUTO_INCREMENT ID altered
        - `checksum`: checksum performed
        - `analyzed`: statistics analysis performed
        - `completed`: the table has been fully imported and verified
    - **result**: the result of the current phase
        - `success`: the phase completed successfully
        - `failure`: the phase failed (did not complete)

- **`lightning_engines`** (Counter)

  Counts number of engine files processed and their status. Labels:

    - **state**: the status of the engine, indicating which phase should be completed

    - `pending`: not yet processed
    - `written`: all data encoded and sent
    - `closed`: engine file closed
    - `imported`: the engine file has been imported into the target cluster
    - `completed`: the engine has been fully imported
  - **result**: the result of the current phase
    - `success`: the phase completed successfully
    - `failure`: the phase failed (did not complete)

- **`lightning_chunks`** (Counter)

  Counts number of chunks processed and their status. Labels:

  - **state**: a chunk's status, indicating which phase the chunk is in
    - `estimated`: (not a state) this value gives total number of chunks in current task
    - `pending`: loaded but not yet processed
    - `running`: data are being encoded and sent
    - `finished`: the entire chunk has been processed
    - `failed`: errors happened during processing

- **`lightning_import_seconds`** (Histogram)

  Bucketed histogram for the time needed to import a table.

- **`lightning_row_read_bytes`** (Histogram)

  Bucketed histogram for the size of a single SQL row.

- **`lightning_row_encode_seconds`** (Histogram)

  Bucketed histogram for the time needed to encode a single SQL row into KV pairs.

- **`lightning_row_kv_deliver_seconds`** (Histogram)

  Bucketed histogram for the time needed to deliver a set of KV pairs corresponding to one single SQL row.

- **`lightning_block_deliver_seconds`** (Histogram)

  Bucketed histogram for the time needed to deliver a block of KV pairs to Importer.

- **`lightning_block_deliver_bytes`** (Histogram)

  Bucketed histogram for the uncompressed size of a block of KV pairs delivered to Importer.

- **`lightning_chunk_parser_read_block_seconds`** (Histogram)

  Bucketed histogram for the time needed by the data file parser to read a block.

- **`lightning_checksum_seconds`** (Histogram)

  Bucketed histogram for the time needed to compute the checksum of a table.

- **`lightning_apply_worker_seconds`** (Histogram)

  Bucketed histogram for the time needed to acquire an idle worker (see also the `lightning_idle_workers` gauge). Labels:

  - **name**:
    - `table`
    - `index`
    - `region`
    - `io`
    - `closed-engine`

13.7.13.8 TiDB Lightning Web Interface

TiDB Lightning provides a webpage for viewing the import progress and performing some simple task management. This is called the *server mode*.

To enable server mode, either start `tidb-lightning` with the `--server-mode` flag

```
tiup tidb-lightning --server-mode --status-addr :8289
```

or set the `lightning.server-mode` setting in the configuration file.

```
[lightning]
server-mode = true
status-addr = ':8289'
```

After TiDB Lightning is launched, visit `http://127.0.0.1:8289` to control the program (the actual URL depends on the `status-addr` setting).

In server mode, TiDB Lightning does not start running immediately. Rather, users submit (multiple) *tasks* via the web interface to import data.

13.7.13.8.1 Front page

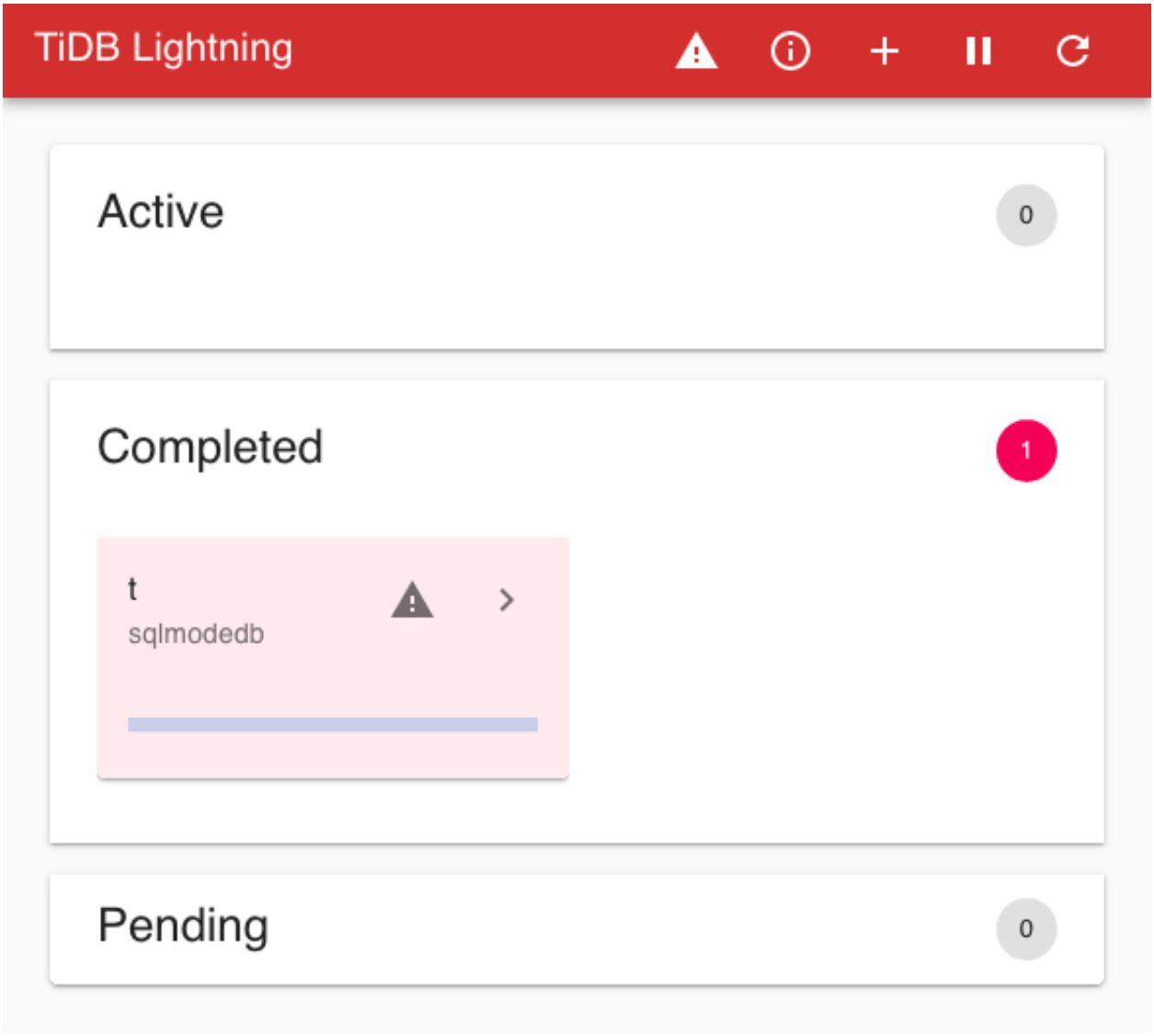


Figure 266: Front page of the web interface

Functions of the title bar, from left to right:

Icon	Function
"TiDB Lightning"	Click to go back to the front page

Icon	Function
	Display any error message from *previous* task
	List current and queued tasks; a badge may appear here to indicate number of queued tasks
+	Submit a task
/	Pause/resume current execution
	Configure auto-refresh of the web page

Three panels below the title bar show all tables in different states:

- Active: these tables are currently being imported
- Completed: these tables have been imported successfully or failed
- Pending: these tables are not yet processed

Each panel contains cards describing the status of the table.

13.7.13.8.2 Submit task

Click the + button on the title bar to submit a task.

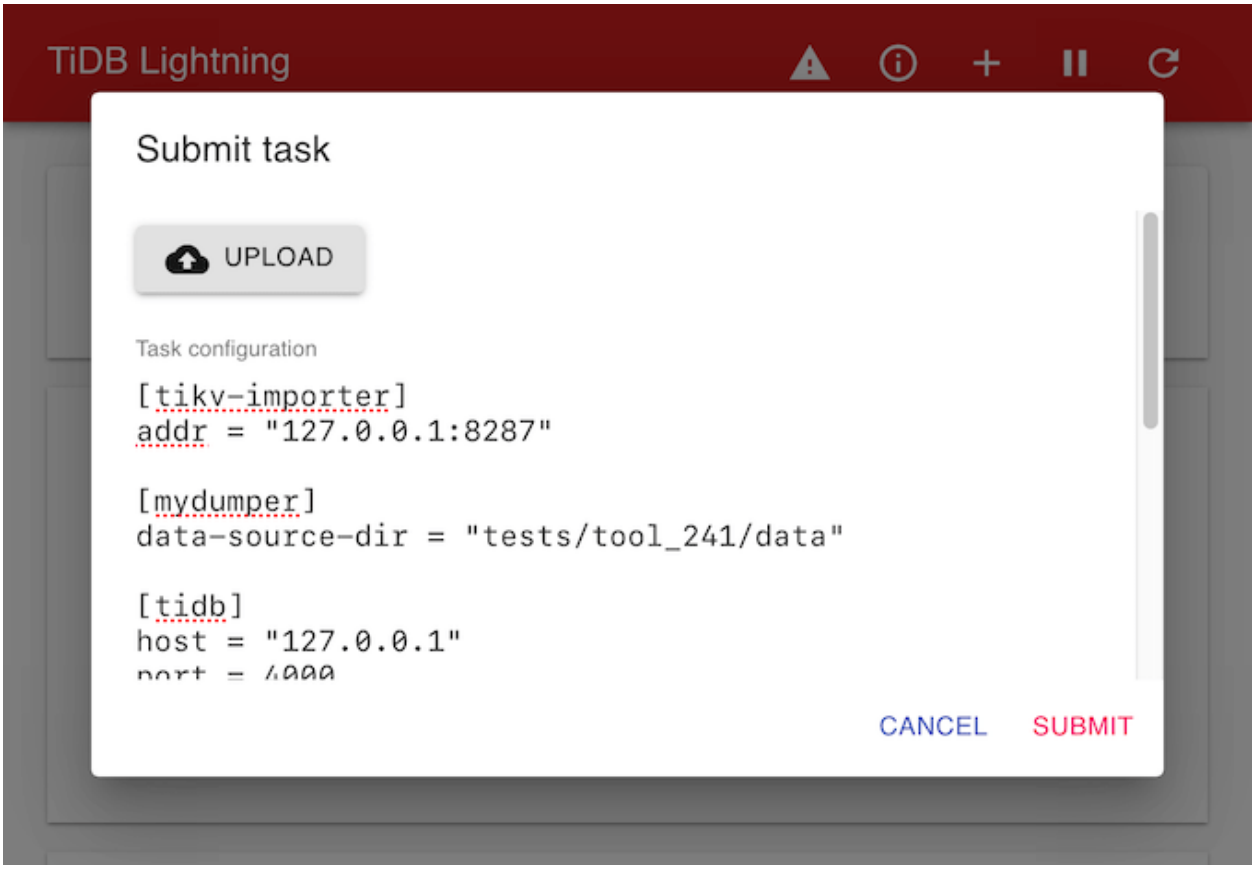


Figure 267: Submit task dialog

Tasks are TOML files described as task configurations. One could also open a local TOML file by clicking **UPLOAD**.

Click **SUBMIT** to run the task. If a task is already running, the new task will be queued and executed after the current task succeeds.

13.7.13.8.3 Table progress

Click the > button of a table card on the front page to view the detailed progress of a table.



Figure 268: Table progress

The page shows the import progress of every engine and data files associated with the table.

Click **TiDB Lightning** on the title bar to go back to the front page.

13.7.13.8.4 Task management

Click the button on the title bar to manage the current and queued tasks.



Figure 269: Task management page

Each task is labeled by the time it was submitted. Clicking the task would show the configuration formatted as JSON.

Manage tasks by clicking the button next to a task. You can stop a task immediately, or reorder queued tasks.

13.7.13.9 TiDB Lightning FAQs

This document lists the frequently asked questions (FAQs) and answers about TiDB Lightning.

13.7.13.9.1 What is the minimum TiDB/TiKV/PD cluster version supported by TiDB Lightning?

The version of TiDB Lightning should be the same as the cluster. If you use the Local-backend mode, the earliest available version is 4.0.0. If you use the Importer-backend mode or the TiDB-backend mode, the earliest available version is 2.0.9, but it is recommended to use the 3.0 stable version.

13.7.13.9.2 Does TiDB Lightning support importing multiple schemas (databases)?

Yes.

13.7.13.9.3 What are the privilege requirements for the target database?

For details about the permissions, see Prerequisites for using TiDB Lightning.

13.7.13.9.4 TiDB Lightning encountered an error when importing one table. Will it affect other tables? Will the process be terminated?

If only one table has an error encountered, the rest will still be processed normally.

13.7.13.9.5 How to properly restart TiDB Lightning?

If you are using Importer-backend, depending on the status of `tikv-importer`, the basic sequence of restarting TiDB Lightning is like this:

If `tikv-importer` is still running:

1. Stop `tidb-lightning`.
2. Perform the intended modifications, such as fixing the source data, changing settings, replacing hardware etc.
3. If the modification previously has changed any table, remove the corresponding checkpoint too.
4. Start `tidb-lightning`.

If `tikv-importer` needs to be restarted:

1. Stop `tidb-lightning`.
2. Stop `tikv-importer`.
3. Perform the intended modifications, such as fixing the source data, changing settings, replacing hardware etc.
4. Start `tikv-importer`.
5. Start `tidb-lightning` *and wait until the program fails with CHECKSUM error, if any.*
    - Restarting `tikv-importer` would destroy all engine files still being written, but `tidb-lightning` did not know about it. As of v3.0 the simplest way is to let `tidb-lightning` go on and retry.
6. Destroy the failed tables and checkpoints
7. Start `tidb-lightning` again.

If you are using Local-backend or TiDB-backend, the operations are the same as those of using Importer-backend when the `tikv-importer` is still running.

13.7.13.9.6 How to ensure the integrity of the imported data?

TiDB Lightning by default performs checksum on the local data source and the imported tables. If there is checksum mismatch, the process would be aborted. These checksum information can be read from the log.

You could also execute the `ADMIN CHECKSUM TABLE` SQL command on the target table to recompute the checksum of the imported data.

```
ADMIN CHECKSUM TABLE `schema`.`table`;
```

```
+--------+------------+--------------------+-----------+-------------+
| Db_name | Table_name | Checksum_crc64_xor | Total_kvs | Total_bytes |
+--------+------------+--------------------+-----------+-------------+
| schema | table     | 5505282386844578743 |     3 |         96 |
+--------+------------+--------------------+-----------+-------------+
1 row in set (0.01 sec)
```

13.7.13.9.7 What kinds of data source formats are supported by TiDB Lightning?

TiDB Lightning supports:

- Importing files exported by Dumpling, CSV files, and Apache Parquet files generated by Amazon Aurora.
- Reading data from a local disk or from the Amazon S3 storage. For details, see External Storages.

13.7.13.9.8 Could TiDB Lightning skip creating schema and tables?

Starting from v5.1, TiDB Lightning can automatically recognize the schema and tables in the downstream. If you use TiDB Lightning earlier than v5.1, you need to set `no-schema = true` in the `[mydumper]` section in `tidb-lightning.toml`. This makes TiDB Lightning skip the `CREATE TABLE` invocations and fetch the metadata directly from the target database. TiDB Lightning will exit with error if a table is actually missing.

13.7.13.9.9 Can the Strict SQL Mode be disabled to allow importing invalid data?

Yes. By default, the `sql_mode` used by TiDB Lightning is `"STRICT_TRANS_TABLES, NO_ENGINE_SUBSTITUTION"`, which disallows invalid data such as the date `1970-00-00`. The mode can be changed by modifying the `sql-mode` setting in the `[tidb]` section in `tidb-lightning.toml`.

```
...
[tidb]
sql-mode = ""
...
```

13.7.13.9.10 Can one `tikv-importer` serve multiple `tidb-lightning` instances?

Yes, as long as every `tidb-lightning` instance operates on different tables.

13.7.13.9.11 How to stop the `tikv-importer` process?

To stop the `tikv-importer` process, you can choose the corresponding operation according to your deployment method.

- For manual deployment: if `tikv-importer` is running in foreground, press Ctrl+C to exit. Otherwise, obtain the process ID using the `ps aux | grep tikv-importer` command and then terminate the process using the `kill ${PID}` command.

13.7.13.9.12 How to stop the `tidb-lightning` process?

To stop the `tidb-lightning` process, you can choose the corresponding operation according to your deployment method.

- For manual deployment: if `tidb-lightning` is running in foreground, press Ctrl+C to exit. Otherwise, obtain the process ID using the `ps aux | grep tidb-lightning` command and then terminate the process using the `kill -2 ${PID}` command.

13.7.13.9.13 Can TiDB Lightning be used with 1-Gigabit network card?

TiDB Lightning is best used with a 10-Gigabit network card.

1-Gigabit network cards can only provide a total bandwidth of 120 MB/s, which has to be shared among all target TiKV stores. TiDB Lightning can easily saturate all bandwidth of the 1-Gigabit network in physical import mode and bring down the cluster because PD is unable to be contacted anymore.

13.7.13.9.14 Why TiDB Lightning requires so much free space in the target TiKV cluster?

With the default settings of 3 replicas, the space requirement of the target TiKV cluster is 6 times the size of data source. The extra multiple of "2" is a conservative estimation because the following factors are not reflected in the data source:

- The space occupied by indices
- Space amplification in RocksDB

13.7.13.9.15 Can TiKV Importer be restarted while TiDB Lightning is running?

No. TiKV Importer stores some information of engines in memory. If `tikv-importer` is restarted, `tidb-lightning` will be stopped due to lost connection. At this point, you need to destroy the failed checkpoints as those TiKV Importer-specific information is lost. You can restart TiDB Lightning afterwards.

See also How to properly restart TiDB Lightning? for the correct sequence.

13.7.13.9.16 How to completely destroy all intermediate data associated with TiDB Lightning?

1. Delete the checkpoint file.

```
tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-
    ↪ remove=all
```

   If, for some reason, you cannot run this command, try manually deleting the file `/tmp ↪ /tidb_lightning_checkpoint.pb`.

2. If you are using Local-backend, delete the `sorted-kv-dir` directory in the configuration. If you are using Importer-backend, delete the entire `import` directory on the machine hosting `tikv-importer`.

3. Delete all tables and databases created on the TiDB cluster, if needed.

4. Clean up the residual metadata. You need to clean up the metadata schema manually if either of the following conditions exist.

   - For TiDB Lightning v5.1.x and v5.2.x versions, the `tidb-lightning-ctl` command does not clean up the metadata schema in the target cluster. You need to clean it up manually.
   - If you have deleted the checkpoint files manually, you need to clean up the downstream metadata schema manually; otherwise, the correctness of subsequent imports might be affected.

   Use the following command to clean up the metadata:

```
DROP DATABASE IF EXISTS `lightning_metadata`;
```

13.7.13.9.17 How to get the runtime goroutine information of TiDB Lightning

1. If `status-port` has been specified in the configuration file of TiDB Lightning, skip this step. Otherwise, you need to send the USR1 signal to TiDB Lightning to enable `status-port`.

Get the process ID (PID) of TiDB Lightning using commands like `ps`, and then run the following command:

```
kill -USR1 <lightning-pid>
```

Check the log of TiDB Lightning. The log of `starting HTTP server` / `start HTTP server` / `started HTTP server` shows the newly enabled `status-port`.

2. Access `http://<lightning-ip>:<status-port>/debug/pprof/goroutine?debug=2` to get the goroutine information.

13.7.13.9.18 Why is TiDB Lightning not compatible with Placement Rules in SQL?

TiDB Lightning is not compatible with Placement Rules in SQL. When TiDB Lightning imports data that contains placement policies, TiDB Lightning reports an error.

The reason is explained as follows:

The purpose of placement rule in SQL is to control the data location of certain TiKV nodes at the table or partition level. TiDB Lightning imports data in text files into the target TiDB cluster. If the data files is exported with the definition of placement rules, during the import process, TiDB Lightning must create the corresponding placement rule policy in the target cluster based on the definition. When the source cluster and the target cluster have different topology, this might cause problems.

Suppose the source cluster has the following topology:

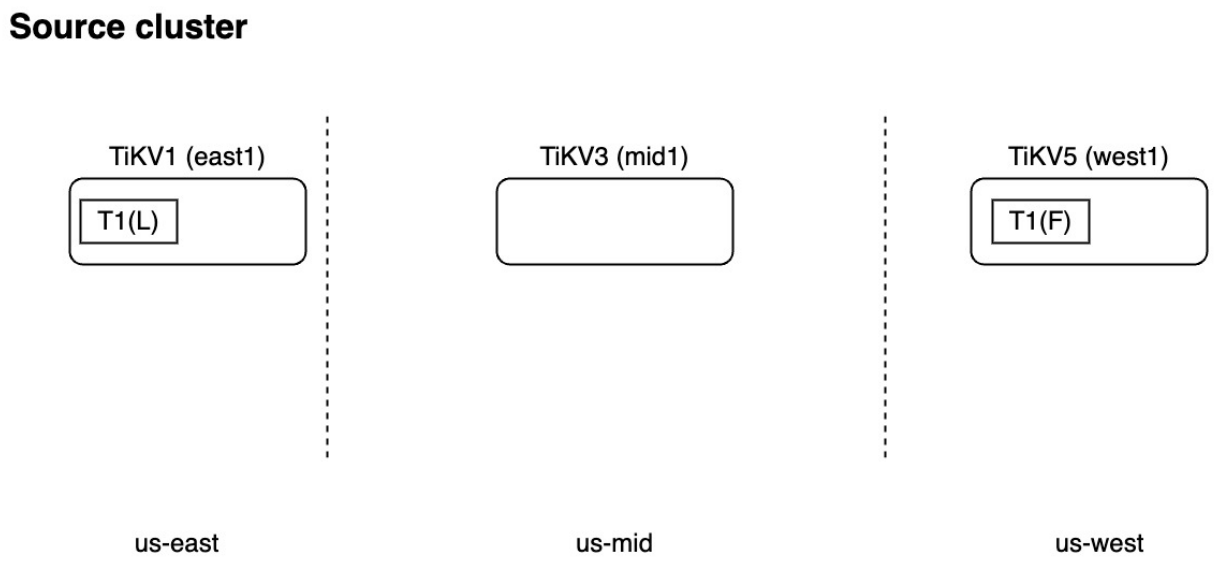


Figure 270: TiDB Lightning FAQ - source cluster topology

The source cluster has the following placement policy:

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east" REGIONS="us-east,us-
    ↪ west";
```

Situation 1: The target cluster has 3 replicas, and the topology is different from the source cluster. In such cases, when TiDB Lightning creates the placement policy in the target cluster, it will not report an error. However, the semantics in the target cluster is wrong.

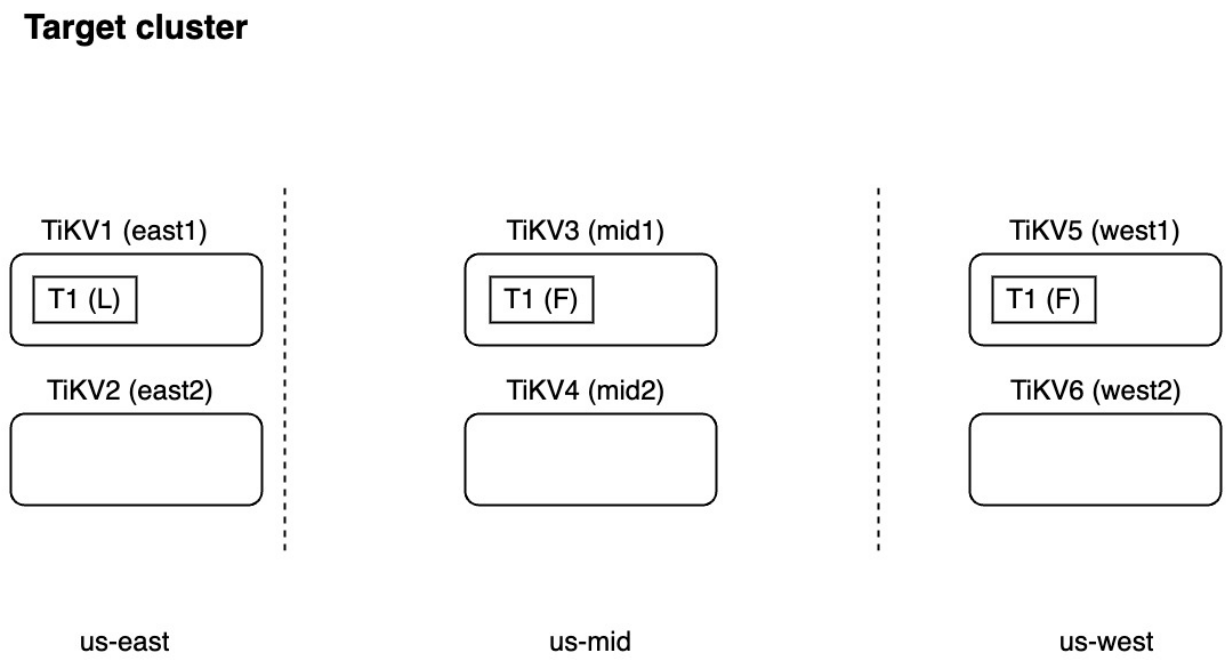


Figure 271: TiDB Lightning FAQ - situation 1

Situation 2: The target cluster locates the follower replica in another TiKV node in region "us-mid" and does not have the region "us-west" in the topology. In such cases, when creating the placement policy in the target cluster, TiDB Lightning will report an error.

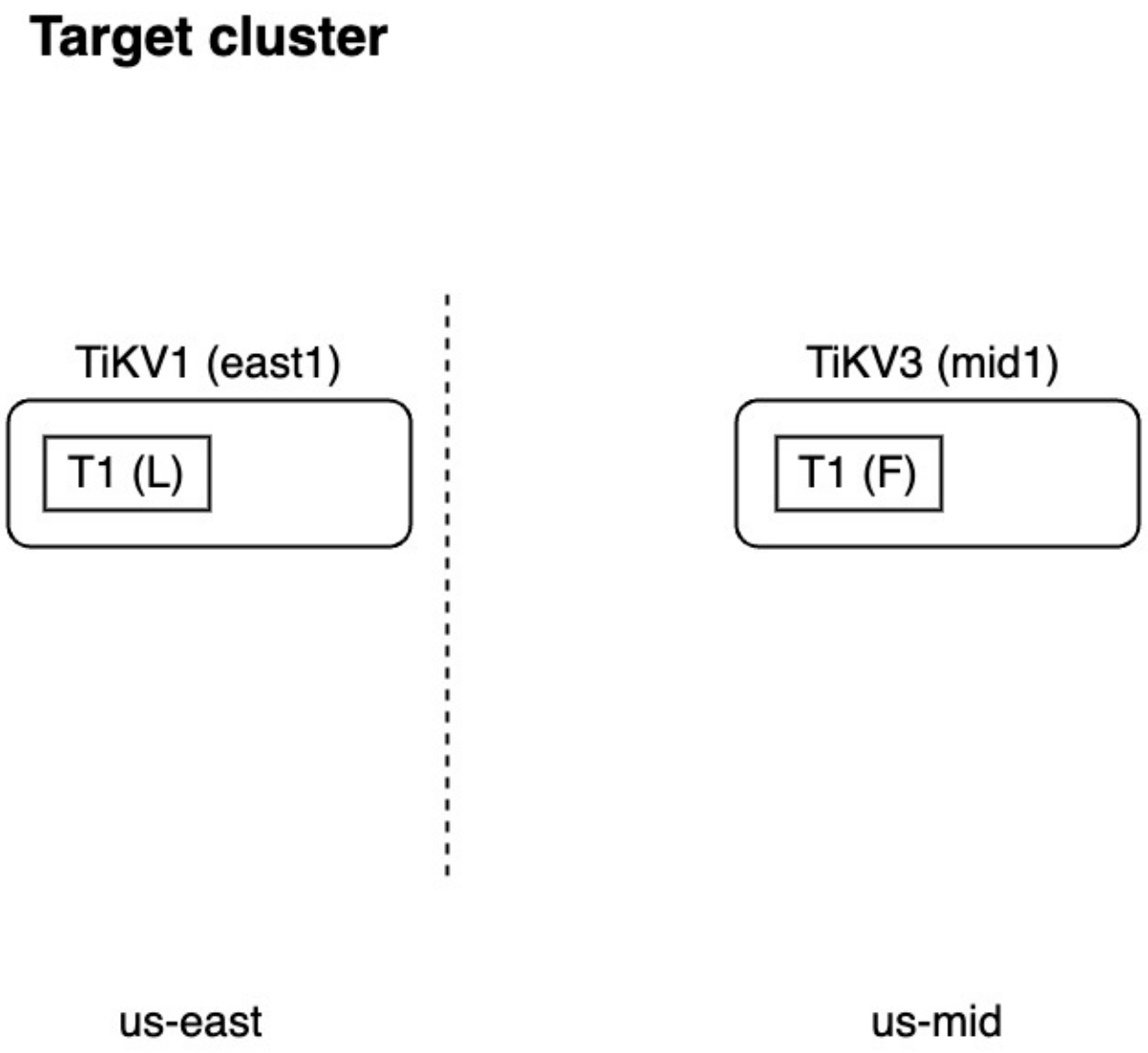


Figure 272: TiDB Lightning FAQ - situation 2

Workaround:

To use placement rules in SQL with TiDB Lightning, you need to make sure that the related labels and objects have been created in the target TiDB cluster **before** you import data into the target table. Because the placement rules in SQL acts at the PD and TiKV layer, TiDB Lightning can get the necessary information to find out which TiKV should be used to store the imported data. In this way, this placement rule in SQL is transparent to TiDB Lightning.

The steps are as follows:

1. Plan the data distribution topology.

2. Configure the required labels for TiKV and PD.
3. Create the placement rule policy and apply the created policy to the target table.
4. Use TiDB Lightning to import data into the target table.

13.7.13.10 TiDB Lightning Glossary

This page explains the special terms used in TiDB Lightning's logs, monitoring, configurations, and documentation.

13.7.13.10.1 A

Analyze

An operation to rebuild the statistics information of a TiDB table, which is running the `ANALYZE TABLE` statement.

Because TiDB Lightning imports data without going through TiDB, the statistics information is not automatically updated. Therefore, TiDB Lightning explicitly analyzes every table after importing. This step can be omitted by setting the `post-restore.analyze` configuration to `false`.

`AUTO_INCREMENT_ID`

Every table has an associated `AUTO_INCREMENT_ID` counter to provide the default value of an auto-incrementing column. In TiDB, this counter is additionally used to assign row IDs.

Because TiDB Lightning imports data without going through TiDB, the `AUTO_INCREMENT_ID` counter is not automatically updated. Therefore, TiDB Lightning explicitly alters `AUTO_INCREMENT_ID` to a valid value. This step is always performed, even if the table has no `AUTO_INCREMENT` columns.

13.7.13.10.2 B

Back end

Back end is the destination where TiDB Lightning sends the parsed result. Also spelled as "backend".

See TiDB Lightning architecture for details.

13.7.13.10.3 C

Checkpoint

TiDB Lightning continuously saves its progress into a local file or a remote database while importing. This allows it to resume from an intermediate state should it crashes in the process. See the Checkpoints section for details.

Checksum

In TiDB Lightning, the checksum of a table is a set of 3 numbers calculated from the content of each KV pair in that table. These numbers are respectively:

- the number of KV pairs,
- total length of all KV pairs, and
- the bitwise-XOR of CRC-64-ECMA value each pair.

TiDB Lightning validates the imported data by comparing the local and remote checksums of every table. The program would stop if any pair does not match. You can skip this check by setting the `post-restore.checksum` configuration to `false`.

See also the FAQs for how to properly handle checksum mismatch.

Chunk

A continuous range of source data, normally equivalent to a single file in the data source.

When a file is too large, TiDB Lightning might split a file into multiple chunks.

Compaction

An operation that merges multiple small SST files into one large SST file, and cleans up the deleted entries. TiKV automatically compacts data in background while TiDB Lightning is importing.

> **Note:**
>
> For legacy reasons, you can still configure TiDB Lightning to explicitly trigger a compaction every time a table is imported. However, this is not recommended and the corresponding settings are disabled by default.

See RocksDB's wiki page on Compaction for its technical details.

13.7.13.10.4 D

Data engine

An engine for sorting actual row data.

When a table is very large, its data is placed into multiple data engines to improve task pipelining and save space of TiKV Importer. By default, a new data engine is opened for every 100 GB of SQL data, which can be configured through the `mydumper.batch-size` setting.

TiDB Lightning processes multiple data engines concurrently. This is controlled by the `lightning.table-concurrency` setting.

13.7.13.10.5 E

Engine

In TiKV Importer, an engine is a RocksDB instance for sorting KV pairs.

TiDB Lightning transfers data to TiKV Importer through engines. It first opens an engine, sends KV pairs to it (with no particular order), and finally closes the engine. The engine sorts the received KV pairs after it is closed. These closed engines can then be further uploaded to the TiKV stores for ingestion.

Engines use TiKV Importer's `import-dir` as temporary storage, which are sometimes referred to as "engine files".

See also data engine and index engine.

13.7.13.10.6 F

Filter

A configuration list that specifies which tables to be imported or excluded.

See Table Filter for details.

13.7.13.10.7 I

Import mode

A configuration that optimizes TiKV for writing at the cost of degraded read speed and space usage.

TiDB Lightning automatically switches to and off the import mode while running. However, if TiKV gets stuck in import mode, you can use `tidb-lightning-ctl` to force revert to normal mode.

Index engine

An engine for sorting indices.

Regardless of number of indices, every table is associated with exactly one index engine.

TiDB Lightning processes multiple index engines concurrently. This is controlled by the `lightning.index-concurrency` setting. Since every table has exactly one index engine, this also configures the maximum number of tables to process at the same time.

Ingest

An operation which inserts the entire content of an SST file into the RocksDB (TiKV) store.

Ingestion is a very fast operation compared with inserting KV pairs one by one. This operation is the determinant factor for the performance of TiDB Lightning.

See RocksDB's wiki page on Creating and Ingesting SST files for its technical details.

13.7.13.10.8 K

KV pair

Abbreviation of "key-value pair".

KV encoder

A routine which parses SQL or CSV rows to KV pairs. Multiple KV encoders run in parallel to speed up processing.

13.7.13.10.9 L

Local checksum

The checksum of a table calculated by TiDB Lightning itself before sending the KV pairs to TiKV Importer.

13.7.13.10.10 N

Normal mode

The mode where import mode is disabled.

13.7.13.10.11 P

Post-processing

The period of time after the entire data source is parsed and sent to TiKV Importer. TiDB Lightning is waiting for TiKV Importer to upload and ingest the SST files.

13.7.13.10.12 R

Remote checksum

The checksum of a table calculated by TiDB after it has been imported.

13.7.13.10.13 S

Scattering

An operation that randomly reassigns the leader and the peers of a Region. Scattering ensures that the imported data are distributed evenly among TiKV stores. This reduces stress on PD.

Splitting

An engine is typically very large (around 100 GB), which is not friendly to TiKV if treated as a single region. TiKV Importer splits an engine into multiple small SST files (configurable by TiKV Importer's `import.region-split-size` setting) before uploading.

SST file

SST is the abbreviation of "sorted string table". An SST file is RocksDB's (and thus TiKV's) native storage format of a collection of KV pairs.

TiKV Importer produces SST files from a closed engine. These SST files are uploaded and then ingested into TiKV stores.

13.8 Use Dumpling to Export Data

This document introduces the data export tool - Dumpling. Dumpling exports data stored in TiDB/MySQL as SQL or CSV data files and can be used to make a logical full backup or export. Dumpling also supports exporting data to Amazon S3.

You can get Dumpling using TiUP by running `tiup install dumpling`. Afterwards, you can use `tiup dumpling ...` to run Dumpling.

The Dumpling installation package is included in the TiDB Toolkit. To download the TiDB Toolkit, see Download TiDB Tools.

For detailed usage of Dumpling, use the `--help` option or refer to Option list of Dumpling.

When using Dumpling, you need to execute the export command on a running cluster.

TiDB also provides other tools that you can choose to use as needed.

- For backups of SST files (key-value pairs) or backups of incremental data that are not sensitive to latency, refer to BR.
- For real-time backups of incremental data, refer to TiCDC.
- All exported data can be imported back to TiDB using TiDB Lightning.

> **Note:**
>
> PingCAP previously maintained a fork of the mydumper project with enhancements specific to TiDB. For more information on Mydumper, refer to v4.0 Mydumper documentation. Starting from v7.5.0, Mydumper is deprecated and most of its features have been replaced by Dumpling. It is strongly recommended that you use Dumpling instead of Mydumper.

Compared to Mydumper, Dumpling has the following improvements:

- Support exporting data in multiple formats, including SQL and CSV.
- Support the table-filter feature, which makes it easier to filter data.
- Support exporting data to Amazon S3 cloud storage.
- More optimizations are made for TiDB:

- Support configuring the memory limit of a single TiDB SQL statement.
- If Dumpling can access the PD address and the `INFORMATION_SCHEMA.CLUSTER_INFO` table of the TiDB cluster, Dumpling supports automatically adjusting the GC safe point time to block GC for TiDB v4.0.0 and later versions.
- Use TiDB's hidden column `_tidb_rowid` to optimize the performance of concurrent data export from a single table.
- For TiDB, you can set the value of `tidb_snapshot` to specify the time point of the data backup. This ensures the consistency of the backup, instead of using `FLUSH TABLES WITH READ LOCK` to ensure the consistency.

Note:

Dumpling cannot connect to PD in the following scenarios:

- The TiDB cluster is running on Kubernetes (unless Dumpling itself is run inside the Kubernetes environment).
- The TiDB cluster is running on TiDB Cloud.

In such cases, you need to manually adjust the TiDB GC time to avoid export failure.

13.8.1 Export data from TiDB or MySQL

13.8.1.1 Required privileges

- SELECT
- RELOAD
- LOCK TABLES
- REPLICATION CLIENT
- PROCESS

13.8.1.2 Export to SQL files

This document assumes that there is a TiDB instance on the 127.0.0.1:4000 host and that this TiDB instance has a root user without a password.

Dumpling exports data to SQL files by default. You can also export data to SQL files by adding the `--filetype sql` flag:

```
dumpling -u root -P 4000 -h 127.0.0.1 --filetype sql -t 8 -o /tmp/test -r
    ↪ 200000 -F 256MiB
```

In the command above:

- The `-h`, `-P`, and `-u` option respectively mean the address, the port, and the user. If a password is required for authentication, you can use `-p $YOUR_SECRET_PASSWORD` to pass the password to Dumpling.

- The `-o` option specifies the export directory of the storage, which supports a local file path or a URL of an external storage.

- The `-t` option specifies the number of threads for the export. Increasing the number of threads improves the concurrency of Dumpling and the export speed, and also increases the database's memory consumption. Therefore, it is not recommended to set the number too large. Usually, it's less than 64.
- The `-r` option enables the in-table concurrency to speed up the export. The default value is `0`, which means disabled. A value greater than 0 means it is enabled, and the value is of `INT` type. When the source database is TiDB, a `-r` value greater than 0 indicates that the TiDB region information is used for splitting, and reduces the memory usage. The specific `-r` value does not affect the split algorithm. When the source database is MySQL and the primary key is of the `INT` type, specifying `-r` can also enable the in-table concurrency.
- The `-F` option is used to specify the maximum size of a single file (the unit here is `MiB`; inputs like `5GiB` or `8KB` are also acceptable). It is recommended to keep its value to 256 MiB or less if you plan to use TiDB Lightning to load this file into a TiDB instance.

> **Note:**
>
> If the size of a single exported table exceeds 10 GB, it is **strongly recommended to use** the `-r` and `-F` options.

13.8.1.3 Export to CSV files

You can export data to CSV files by adding the `--filetype csv` argument.

When you export data to CSV files, you can use `--sql <SQL>` to filter the records with the SQL statements. For example, you can export all records that match `id < 100` in `test.sbtest1` using the following command:

```
./dumpling -u root -P 4000 -h 127.0.0.1 -o /tmp/test --filetype csv --sql '
    ↪ select * from `test`.`sbtest1` where id < 100' -F 100MiB --output-
    ↪ filename-template 'test.sbtest1.{{.Index}}'
```

In the command above:

- The `--sql` option can be used only for exporting to CSV files. The command above executes the `SELECT * FROM <table-name> WHERE id <100` statement on all tables to be exported. If a table does not have the specified field, the export fails.

- When you use the `--sql` option, Dumpling cannot obtain the exported table and schema information. You can specify the file name format of the CSV files using the `--output-filename-template` option, which facilitates the subsequent use of TiDB Lightning to import the data file. For example, `--output-filename-template='test.sbtest1.{{.Index}}'` specifies that the exported CSV files are named as `test.sbtest1.000000000` or `test.sbtest1.000000001`.

- You can use options like `--csv-separator` and `--csv-delimiter` to configure the CSV file format. For details, refer to the Dumpling option list.

> **Note:**
>
> *Strings* and *keywords* are not distinguished by Dumpling. If the imported data is the Boolean type, the value of `true` is converted to `1` and the value of `false` is converted to `0`.

13.8.1.4 Format of exported files

- `metadata`: The start time of the exported files and the position of the master binary log.

```
cat metadata
```

```
Started dump at: 2020-11-10 10:40:19
SHOW MASTER STATUS:
       Log: tidb-binlog
       Pos: 420747102018863124
Finished dump at: 2020-11-10 10:40:20
```

- `{schema}-schema-create.sql`: The SQL file used to create the schema

```
cat test-schema-create.sql
```

```
CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4 */;
```

- `{schema}.{table}-schema.sql`: The SQL file used to create the table

```
cat test.t1-schema.sql
```

```
CREATE TABLE `t1` (
  `id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

- `{schema}.{table}.{0001}.{sql|csv}`: The date source file

```
cat test.t1.0.sql
```

```
/*!40101 SET NAMES binary*/;
INSERT INTO `t1` VALUES
(1);
```

- `*-schema-view.sql`, `*-schema-trigger.sql`, `*-schema-post.sql`: Other exported files

13.8.1.5 Export data to Amazon S3 cloud storage

Starting from v4.0.8, Dumpling supports exporting data to cloud storages. If you need to back up data to Amazon S3, you need to specify the Amazon S3 storage path in the `-o` parameter.

You need to create an Amazon S3 bucket in the specified region (see the Amazon documentation - How do I create an S3 Bucket). If you also need to create a folder in the bucket, see the Amazon documentation - Creating a folder.

Pass `SecretKey` and `AccessKey` of the account with the permission to access the Amazon S3 backend storage to the Dumpling node as environment variables.

```
export AWS_ACCESS_KEY_ID=${AccessKey}
export AWS_SECRET_ACCESS_KEY=${SecretKey}
```

Dumpling also supports reading credential files from `~/.aws/credentials`. For more Dumpling configuration, see the configuration of External storages.

When you back up data using Dumpling, explicitly specify the `--s3.region` parameter, which means the region of the S3 storage (for example, `ap-northeast-1`):

```
./dumpling -u root -P 4000 -h 127.0.0.1 -r 200000 -o "s3://${Bucket}/${
    ↪ Folder}" --s3.region "${region}"
```

13.8.1.6 Filter the exported data

13.8.1.6.1 Use the `--where` option to filter data

By default, Dumpling exports all databases except system databases (including `mysql`, `sys`, `INFORMATION_SCHEMA`, `PERFORMANCE_SCHEMA`, `METRICS_SCHEMA`, and `INSPECTION_SCHEMA`). You can use `--where <SQL where expression>` to select the records to be exported.

```
./dumpling -u root -P 4000 -h 127.0.0.1 -o /tmp/test --where "id < 100"
```

The above command exports the data that matches `id < 100` from each table. Note that you cannot use the `--where` parameter together with `--sql`.

13.8.1.6.2 Use the `--filter` option to filter data

Dumpling can filter specific databases or tables by specifying the table filter with the `--filter` option. The syntax of table filters is similar to that of `.gitignore`. For details, see Table Filter.

```
./dumpling -u root -P 4000 -h 127.0.0.1 -o /tmp/test -r 200000 --filter "
    ↪ employees.*" --filter "*.WorkOrder"
```

The above command exports all the tables in the `employees` database and the `WorkOrder` tables in all databases.

13.8.1.6.3 Use the `-B` or `-T` option to filter data

Dumpling can also export specific databases with the `-B` option or specific tables with the `-T` option.

Note:

- The `--filter` option and the `-T` option cannot be used at the same time.
- The `-T` option can only accept a complete form of inputs like `database ↪ -name.table-name`, and inputs with only the table name are not accepted. Example: Dumpling cannot recognize `-T WorkOrder`.

Examples:

- `-B employees` exports the `employees` database.
- `-T employees.WorkOrder` exports the `employees.WorkOrder` table.

13.8.1.7 Improve export efficiency through concurrency

The exported file is stored in the `./export-<current local time>` directory by default. Commonly used options are as follows:

- The `-t` option specifies the number of threads for the export. Increasing the number of threads improves the concurrency of Dumpling and the export speed, and also increases the database's memory consumption. Therefore, it is not recommended to set the number too large.
- The `-r` option enables the in-table concurrency to speed up the export. The default value is `0`, which means disabled. A value greater than 0 means it is enabled, and the value is of `INT` type. When the source database is TiDB, a `-r` value greater than

0 indicates that the TiDB region information is used for splitting, and reduces the memory usage. The specific `-r` value does not affect the split algorithm. When the source database is MySQL and the primary key is of the `INT` type, specifying `-r` can also enable the in-table concurrency.
- The `--compress gzip` option can be used to compress the dump. This can help to speed up dumping of data if storage is the bottleneck or if storage capacity is a concern. The drawback of this is an increase in CPU usage. Each file is compressed individually.

With the above options specified, Dumpling can have a quicker speed of data export.

13.8.1.8 Adjust Dumpling's data consistency options

> **Note:**
>
> The default value is `auto` for the data consistency option. In most scenarios, you do not need to adjust the default data consistency options of Dumpling.

Dumpling uses the `--consistency <consistency level>` option to control the way in which data is exported for "consistency assurance". When using snapshot for consistency, you can use the `--snapshot` option to specify the timestamp to be backed up. You can also use the following levels of consistency:

- `flush`: Use `FLUSH TABLES WITH READ LOCK` to temporarily interrupt the DML and DDL operations of the replica database, to ensure the global consistency of the backup connection, and to record the binlog position (POS) information. The lock is released after all backup connections start transactions. It is recommended to perform full backups during off-peak hours or on the MySQL replica database.
- `snapshot`: Get a consistent snapshot of the specified timestamp and export it.
- `lock`: Add read locks on all tables to be exported.
- `none`: No guarantee for consistency.
- `auto`: Use `flush` for MySQL and `snapshot` for TiDB.

After everything is done, you can see the exported file in `/tmp/test`:

```
ls -lh /tmp/test | awk '{print $5 "\t" $9}'
```

```
140B metadata
66B  test-schema-create.sql
300B test.sbtest1-schema.sql
190K test.sbtest1.0.sql
300B test.sbtest2-schema.sql
```

```
190K test.sbtest2.0.sql
300B test.sbtest3-schema.sql
190K test.sbtest3.0.sql
```

13.8.1.9 Export historical data snapshots of TiDB

Dumpling can export the data of a certain tidb_snapshot with the `--snapshot` option specified.

The `--snapshot` option can be set to a TSO (the `Position` field output by the `SHOW MASTER STATUS` command) or a valid time of the `datetime` data type (in the form of `YYYY-MM-DD hh:mm:ss`), for example:

```
./dumpling --snapshot 417773951312461825
./dumpling --snapshot "2020-07-02 17:12:45"
```

The TiDB historical data snapshots when the TSO is `417773951312461825` and the time is `2020-07-02 17:12:45` are exported.

13.8.1.10 Control the memory usage of exporting large tables

When Dumpling is exporting a large single table from TiDB, Out of Memory (OOM) might occur because the exported data size is too large, which causes connection abort and export failure. You can use the following parameters to reduce the memory usage of TiDB:

- Setting `-r` to split the data to be exported into chunks. This reduces the memory overhead of TiDB's data scan and enables concurrent table data dump to improve export efficiency. When the upstream database is TiDB v3.0 or later versions, a `-r` value greater than 0 indicates that the TiDB region information is used for splitting and the specific `-r` value does not affect the split algorithm.
- Reduce the value of `--tidb-mem-quota-query` to `8589934592` (8 GB) or lower. `--tidb-mem-quota-query` controls the memory usage of a single query statement in TiDB.
- Adjust the `--params "tidb_distsql_scan_concurrency=5"` parameter. tidb_distsql_scan_conc is a session variable which controls the concurrency of the scan operations in TiDB.

13.8.1.11 Manually set the TiDB GC time

When exporting data from TiDB (less than 1 TB), if the TiDB version is v4.0.0 or later and Dumpling can access the PD address and the `INFORMATION_SCHEMA.CLUSTER_INFO` table of the TiDB cluster, Dumpling automatically adjusts the GC safe point to block GC without affecting the original cluster.

However, in either of the following scenarios, Dumpling cannot automatically adjust the GC time:

- The data size is very large (more than 1 TB).
- Dumpling cannot connect directly to PD, for example, if the TiDB cluster is on TiDB Cloud or on Kubernetes that is separated from Dumpling.

In such scenarios, you must manually extend the GC time in advance to avoid export failure due to GC during the export process.

To manually adjust the GC time, use the following SQL statement:

```
SET GLOBAL tidb_gc_life_time = '720h';
```

After Dumpling exits, regardless of whether the export is successful or not, you must set the GC time back to its original value (the default value is `10m`).

```
SET GLOBAL tidb_gc_life_time = '10m';
```

13.8.2 Option list of Dumpling

Options	Usage	Default value
`-V` or `--version`	Output the Dumpling version and exit directly	
`-B` or `--database`	Export specified databases	
`-T` or `--tables-list`	Export specified tables	

Options	Usage	Default value
`-f` or `--filter`	Export tables that match the filter pattern. For the filter syntax, see table-filter.	`[\*.\*,!/^(mysql|sys|INFORMATION_SCHEMA|PERFORMANCE_SCHEMA|METRICS_SCHEMA|INSPECTION_SCHEMA)$/.\*]` (export all databases or tables excluding system schemas)
`--case-sensitive`	whether table-filter is case-sensitive	false (case-insensitive)
`-h` or `--host`	The IP address of the connected database host	“127.0.0.1”
`-t` or `--threads`	The number of concurrent backup threads	4

Options	Usage	Default value
`-r` or `--rows`	Enable the in-table concurrency to speed up the export. The default value is `0`, which means disabled. A value greater than 0 means it is enabled, and the value is of `INT` type. When the source database is TiDB, a `-r` value greater than 0 indicates that the TiDB region information is used for splitting, and reduces the memory usage. The specific `-r` value does not affect the split algorithm. When the source database is MySQL and the primary key is of the `INT` type, specifying `-r` can also enable the in-table concurrency.	
`-L` or `--logfile`	Log output address. If it is empty, the log will be output to the console	“”
`--loglevel`	Log level {debug,info,warn,error,dpanic,panic,fatal}	“info”
`--logfmt`	Log output format {text,json}	“text”
`-d` or `--no-data`	Do not export data (suitable for scenarios where only the schema is exported)	
`--no-header`	Export CSV files of the tables without generating header	
`-W` or `--no-views`	Do not export the views	true
`-m` or `--no-schemas`	Do not export the schema with only the data exported	
`-s` or `--statement-size`	Control the size of the `INSERT` statements; the unit is bytes	

Options	Usage	Default value
-F or --filesize	The file size of the divided tables. The unit must be specified such as 128B, 64KiB, 32MiB, and 1.5GiB.	
--filetype	Exported file type (csv/sql)	“sql”
-o or --output	The path of exported local files or the URL of the external storage	“./export-${time}”
-S or --sql	Export data according to the specified SQL statement. This command does not support concurrent export.	
--consistency	flush: use FTWRL before the dump snapshot: dump the TiDB data at a specific snapshot of a TSO lock: execute lock tables read on all tables to be dumped none: dump without adding locks, which cannot guarantee consistency auto: use –consistency flush for MySQL; use –consistency snapshot for TiDB	“auto”
--snapshot	Snapshot TSO; valid only when consistency=snapshot	
--where	Specify the scope of the table backup through the where condition	
-p or --password	The password of the connected database host	
-P or --port	The port of the connected database host	4000
-u or --user	The username of the connected database host	“root”

Options	Usage	Default value
--dump-empty-database	Export the CREATE DATABASE statements of the empty databases	true
--ca	The address of the certificate authority file for TLS connection	
--cert	The address of the client certificate file for TLS connection	
--key	The address of the client private key file for TLS connection	
--csv-delimiter	Delimiter of character type variables in CSV files	'"'
--csv-separator	Separator of each value in CSV files. It is not recommended to use the default ','. It is recommended to use '|+|' or other uncommon character combinations	','
--csv-null-value	Representation of null values in CSV files	"\N"
--escape-backslash	Use backslash (\) to escape special characters in the export file	true
--output-filename-template	The filename templates represented in the format of golang template Support the {{.DB}}, {{.Table}}, and {{.Index}} arguments The three arguments represent the database name, table name, and chunk ID of the data file	'{{.DB}}.{{.Table}}.{{.

Options	Usage	Default value
--status-addr	Dumpling's service address, including the address for Prometheus to pull metrics and pprof debugging	":8281"
--tidb-mem-quota-query	The memory limit of exporting SQL statements by a single line of Dumpling command, and the unit is byte. For v4.0.10 or later versions, if you do not set this parameter, TiDB uses the value of the `mem-quota-query` configuration item as the memory limit value by default. For versions earlier than v4.0.10, the parameter value defaults to 32 GB.	34359738368
--params	Specifies the session variable for the connection of the database to be exported. The required format is `"character_set_client=latin1,character_set_connection=latin1"`	

13.9 TiCDC

13.9.1 TiCDC Overview

TiCDC is a tool used for replicating incremental data of TiDB. Specifically, TiCDC pulls TiKV change logs, sorts captured data, and exports row-based incremental data to downstream databases.

13.9.1.1 Usage scenarios

- Database disaster recovery: TiCDC can be used for disaster recovery between homogeneous databases to ensure eventual data consistency of primary and secondary databases after a disaster event. This function works only with TiDB primary and secondary clusters.
- Data integration: TiCDC provides TiCDC Canal-JSON Protocol, which allows other systems to subscribe data changes from TiCDC. In this way, TiCDC provides data sources for various scenarios such as monitoring, caching, global indexing, data analysis, and primary-secondary replication between heterogeneous databases.

13.9.1.2 TiCDC architecture

When TiCDC is running, it is a stateless node that achieves high availability through etcd in PD. The TiCDC cluster supports creating multiple replication tasks to replicate data to multiple different downstream platforms.

The architecture of TiCDC is shown in the following figure:

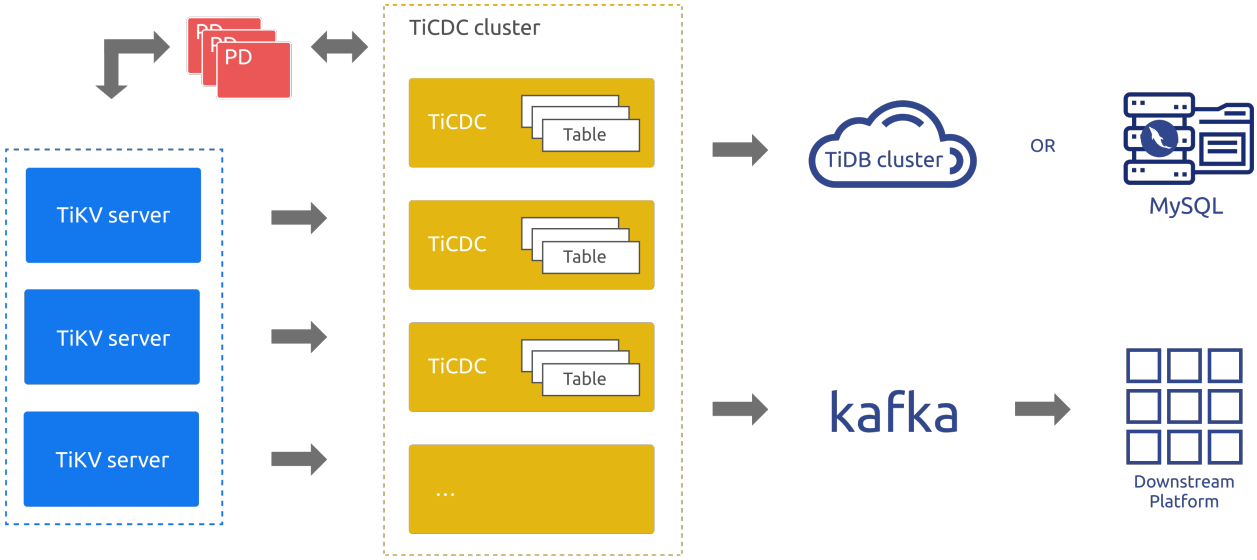


Figure 273: TiCDC architecture

13.9.1.2.1 System roles

- TiKV CDC component: Only outputs key-value (KV) change logs.
    - Assembles KV change logs in the internal logic.
    - Provides the interface to output KV change logs. The data sent includes real-time change logs and incremental scan change logs.
- `capture`: The operating process of TiCDC. Multiple `captures` form a TiCDC cluster that replicates KV change logs.
    - Each `capture` pulls a part of KV change logs.
    - Sorts the pulled KV change log(s).
    - Restores the transaction to downstream or outputs the log based on the TiCDC open protocol.

13.9.1.3 Replication features

This section introduces the replication features of TiCDC.

13.9.1.3.1 Sink support

Currently, the TiCDC sink component supports replicating data to the following downstream platforms:

- Databases compatible with MySQL protocol. The sink component provides the final consistency support.
- Kafka based on the TiCDC Open Protocol. The sink component ensures the row-level order, final consistency or strict transactional consistency.

13.9.1.3.2 Ensure replication order and consistency

Replication order

- For all DDL or DML statements, TiCDC outputs them **at least once**.
- When the TiKV or TiCDC cluster encounters failure, TiCDC might send the same DDL/DML statement repeatedly. For duplicated DDL/DML statements:
    - MySQL sink can execute DDL statements repeatedly. For DDL statements that can be executed repeatedly in the downstream, such as `truncate table`, the statement is executed successfully. For those that cannot be executed repeatedly, such as `create table`, the execution fails, and TiCDC ignores the error and continues the replication.
    - Kafka sink sends messages repeatedly, but the duplicate messages do not affect the constraints of `Resolved Ts`. Users can filter the duplicated messages from Kafka consumers.

Replication consistency

- MySQL sink
    - TiCDC does not split single-table transactions and **ensures** the atomicity of single-table transactions.
    - TiCDC does **not ensure** that the execution order of downstream transactions is the same as that of upstream transactions.
    - TiCDC splits cross-table transactions in the unit of table and does **not ensure** the atomicity of cross-table transactions.
    - TiCDC **ensures** that the order of single-row updates is consistent with that in the upstream.

> **Note:**
>
> Since v6.1.1, you can use the sink uri parameter `transaction-atomicity` to control whether to split single-table transactions. Splitting single-table transactions can greatly reduce the latency and memory consumption of replicating large transactions.

- Kafka sink

- TiCDC provides different strategies for data distribution. You can distribute data to different Kafka partitions based on the table, primary key, or timestamp.
- For different distribution strategies, the different consumer implementations can achieve different levels of consistency, including row-level consistency, eventual consistency, or cross-table transactional consistency.
- TiCDC does not have an implementation of Kafka consumers, but only provides TiCDC Open Protocol. You can implement the Kafka consumer according to this protocol.

13.9.1.4 Restrictions

There are some restrictions when using TiCDC.

13.9.1.4.1 Requirements for valid index

TiCDC only replicates the table that has at least one **valid index**. A **valid index** is defined as follows:

- The primary key (`PRIMARY KEY`) is a valid index.
- The unique index (`UNIQUE INDEX`) that meets the following conditions at the same time is a valid index:
  - Every column of the index is explicitly defined as non-nullable (`NOT NULL`).
  - The index does not have the virtual generated column (`VIRTUAL GENERATED COLUMNS`).

13.9.1.4.2 Unsupported scenarios

Currently, the following scenarios are not supported:

- The TiKV cluster that uses RawKV alone.
- The DDL operation `CREATE SEQUENCE` and the SEQUENCE function in TiDB. When the upstream TiDB uses `SEQUENCE`, TiCDC ignores `SEQUENCE` DDL operations/functions performed upstream. However, DML operations using `SEQUENCE` functions can be correctly replicated.
- Currently, performing BR data recovery and TiDB Lightning imports on tables and databases that are being replicated by TiCDC is not supported. For more information, see Why does replication using TiCDC stall or even stop after data restore using TiDB Lightning and BR from upstream.

TiCDC only provides partial support for scenarios of large transactions in the upstream. For details, refer to Does TiCDC support replicating large transactions? Is there any risk?.

Note:

Since v5.3.0, TiCDC no longer supports the cyclic replication feature.

13.9.1.5 Install and deploy TiCDC

You can either deploy TiCDC along with a new TiDB cluster or add the TiCDC component to an existing TiDB cluster. For details, see Deploy TiCDC.

13.9.1.6 Manage TiCDC Cluster and Replication Tasks

Currently, you can use the `cdc cli` tool to manage the status of a TiCDC cluster and data replication tasks. For details, see:

- Use `cdc cli` to manage cluster status and data replication task
- Use OpenAPI to manage cluster status and data replication task

13.9.1.7 TiCDC Open Protocol

TiCDC Open Protocol is a row-level data change notification protocol that provides data sources for monitoring, caching, full-text indexing, analysis engines, and primary-secondary replication between different databases. TiCDC complies with TiCDC Open Protocol and replicates data changes of TiDB to third-party data medium such as MQ (Message Queue). For more information, see TiCDC Open Protocol.

13.9.1.8 Compatibility notes

13.9.1.8.1 Incompatibility issue caused by using the TiCDC v5.0.0-rc `cdc cli` tool to operate a v4.0.x cluster

When using the `cdc cli` tool of TiCDC v5.0.0-rc to operate a v4.0.x TiCDC cluster, you might encounter the following abnormal situations:

- If the TiCDC cluster is v4.0.8 or an earlier version, using the v5.0.0-rc `cdc cli` tool to create a replication task might cause cluster anomalies and get the replication task stuck.
- If the TiCDC cluster is v4.0.9 or a later version, using the v5.0.0-rc `cdc cli` tool to create a replication task will cause the old value and unified sorter features to be unexpectedly enabled by default.

Solutions:

Use the `cdc` executable file corresponding to the TiCDC cluster version to perform the following operations:

1. Delete the changefeed created using the v5.0.0-rc `cdc cli` tool. For example, run the `tiup cdc:v4.0.9 cli changefeed remove -c xxxx --pd=xxxxx --force` command.
2. If the replication task is stuck, restart the TiCDC cluster. For example, run the `tiup cluster restart <cluster_name> -R cdc` command.
3. Re-create the changefeed. For example, run the `tiup cdc:v4.0.9 cli changefeed create --sink-uri=xxxx --pd=xxx` command.

> **Note:**
>
> The above issue exists only when `cdc cli` is v5.0.0-rc. Other v5.0.x `cdc cli` tool can be compatible with v4.0.x clusters.

13.9.1.8.2 Compatibility notes for `sort-dir` and `data-dir`

The `sort-dir` configuration is used to specify the temporary file directory for the TiCDC sorter. Its functionalities might vary in different versions. The following table lists `sort-dir`'s compatibility changes across versions.

	sort ↪ - ↪ engine ↪ functionality	Note	Recommendation
Version			
v4.0.11 or an earlier v4.0 version, v5.0.0-rc	It is a changefeed configuration item and specifies temporary file directory for the file sorter and unified ↪ sorter.	In these versions, file ↪ sorter and unified ↪ sorter are **experimental features** and **NOT** recommended for the production environment. 2211	It is not recommended to use unified ↪ sorter in the production environment. If mul-

	sort ↪ - ↪ engine ↪ functionality	Note	Recommendation
v4.0.12, v4.0.13, v5.0.0, and v5.0.1	It is a configuration item of changefeed or of cdc ↪ server ↪ .	By default, the sort ↪ - ↪ dir ↪ configuration item of a changefeed does not take effect, and the sort ↪ - ↪ dir ↪ configuration of cdc ↪ ↪ server ↪ defaults to /	You need to configure sort ↪ - ↪ dir ↪ using the cdc ↪ ↪ server ↪ command-line parameter (or TiUP).
	2212		

Version	sort ↪ - ↪ engine ↪ functionality	Note	Recommendation
v4.0.14 and later v4.0 versions, v5.0.3 and later v5.0 versions, later TiDB versions	sort ↪ - ↪ dir ↪ is deprecated. It is recommended to configure data ↪ - ↪ dir ↪ .	You can configure data ↪ - ↪ dir ↪ using the latest version of TiUP. In these TiDB versions, unified sorter is enabled by default. Make sure that data ↪ - ↪ dir ↪ has been con-	You need to configure data ↪ - ↪ dir ↪ using the cdc ↪ server ↪ command-line parameter (or TiUP).
2213			

Version	sort - engine func-tion-ality	Note	Recommendation

13.9.1.8.3 Compatibility with temporary tables

Since v5.3.0, TiCDC supports global temporary tables. Replicating global temporary tables to the downstream using TiCDC of a version earlier than v5.3.0 causes table definition error.

If the upstream cluster contains a global temporary table, the downstream TiDB cluster is expected to be v5.3.0 or a later version. Otherwise, an error occurs during the replication process.

13.9.1.9 TiCDC FAQs and troubleshooting

- To learn the FAQs of TiCDC, see TiCDC FAQs.
- To learn how to troubleshoot TiCDC, see Troubleshoot TiCDC.

13.9.2 Deploy TiCDC

This document describes how to deploy a TiCDC cluster and the hardware and software recommendations for deploying and running it. You can either deploy TiCDC along with a new TiDB cluster or add the TiCDC component to an existing TiDB cluster. Generally, it is recommended that you deploy TiCDC using TiUP. In addition, you can also deploy it using binary as needed.

13.9.2.1 Software and hardware recommendations

In production environments, the recommendations of software and hardware for TiCDC are as follows:

Linux OS	Version
Red Hat Enterprise Linux	7.3 or later versions
CentOS	7.3 or later versions

CPU	Memory	Disk type	Network	Number of TiCDC cluster instances (minimum requirements for production environment)
16 core+	64 GB+	SSD	10 Gigabit network card (2 preferred)	2

For more information, see Software and Hardware Recommendations.

13.9.2.2 Deploy a new TiDB cluster that includes TiCDC using TiUP

When you deploy a new TiDB cluster using TiUP, you can also deploy TiCDC at the same time. You only need to add the `cdc_servers` section in the initialization configuration file that TiUP uses to start the TiDB cluster. For detailed operations, see Edit the initialization configuration file. For detailed configurable fields, see Configure `cdc_servers` using TiUP.

13.9.2.3 Add TiCDC to an existing TiDB cluster using TiUP

You can also use TiUP to add the TiCDC component to an existing TiDB cluster. Take the following procedures:

1. Make sure that the current TiDB version supports TiCDC; otherwise, you need to upgrade the TiDB cluster to `v4.0.0-rc.1` or later versions. Since v4.0.6, TiCDC has become a feature for general availability (GA). It is recommended that you use v4.0.6 or later versions.
2. To deploy TiCDC, refer to Scale out a TiCDC cluster.

13.9.2.4 Add TiCDC to an existing TiDB cluster using binary (not recommended)

Suppose that the PD cluster has a PD node (the client URL is `10.0.10.25:2379`) that can provide services. If you want to deploy three TiCDC nodes, start the TiCDC cluster by executing the following commands. You only need to specify the same PD address, and the newly started nodes automatically join the TiCDC cluster.

```
cdc server --pd=http://10.0.10.25:2379 --log-file=ticdc_1.log --addr
    ↪ =0.0.0.0:8301 --advertise-addr=127.0.0.1:8301
cdc server --pd=http://10.0.10.25:2379 --log-file=ticdc_2.log --addr
    ↪ =0.0.0.0:8302 --advertise-addr=127.0.0.1:8302
cdc server --pd=http://10.0.10.25:2379 --log-file=ticdc_3.log --addr
    ↪ =0.0.0.0:8303 --advertise-addr=127.0.0.1:8303
```

13.9.2.5 Description of TiCDC `cdc server` command-line parameters

The following are descriptions of options available in the `cdc server` command:

- `addr`: The listening address of TiCDC, the HTTP API address, and the Prometheus address of the TiCDC service. The default value is `127.0.0.1:8300`.
- `advertise-addr`: The advertised address via which clients access TiCDC. If unspecified, the value is the same as that of `addr`.
- `pd`: A comma-separated list of PD endpoints.
- `config`: The address of the configuration file that TiCDC uses (optional). This option is supported since TiCDC v5.0.0. This option can be used in the TiCDC deployment since TiUP v1.4.0.
- `data-dir`: Specifies the directory that TiCDC uses when it needs to use disks to store files. Unified Sorter uses this directory to store temporary files. It is recommended to ensure that the free disk space for this directory is greater than or equal to 500 GiB. For more details, see Unified Sorter. If you are using TiUP, you can configure `data_dir` in the `cdc_servers` section, or directly use the default `data_dir` path in `global`.

- `gc-ttl`: The TTL (Time To Live) of the service level `GC safepoint` in PD set by TiCDC, and the duration that the replication task can suspend, in seconds. The default value is `86400`, which means 24 hours. Note: Suspending of the TiCDC replication task affects the progress of TiCDC GC safepoint, which means that it affects the progress of upstream TiDB GC, as detailed in Complete Behavior of TiCDC GC safepoint.
- `log-file`: The path to which logs are output when the TiCDC process is running. If this parameter is not specified, logs are written to the standard output (stdout).
- `log-level`: The log level when the TiCDC process is running. The default value is `"info"`.
- `ca`: Specifies the path of the CA certificate file in PEM format for TLS connection (optional).
- `cert`: Specifies the path of the certificate file in PEM format for TLS connection (optional).
- `cert-allowed-cn`: Specifies the path of the common name in PEM format for TLS connection (optional).
- `key`: Specifies the path of the private key file in PEM format for TLS connection (optional).
- `tz`: Time zone used by the TiCDC service. TiCDC uses this time zone when it internally converts time data types such as `TIMESTAMP` or when it replicates data to the downstream. The default is the local time zone in which the process runs. If you specify `time-zone` (in `sink-uri`) and `tz` at the time, the internal TiCDC processes use the time zone specified by `tz`, and the sink uses the time zone specified by `time-zone` for replicating data to the downstream.

13.9.3 Manage TiCDC Cluster and Replication Tasks

This document describes how to upgrade TiCDC cluster and modify the configuration of TiCDC cluster using TiUP, and how to manage the TiCDC cluster and replication tasks using the command-line tool `cdc cli`.

You can also use the HTTP interface (the TiCDC OpenAPI feature) to manage the TiCDC cluster and replication tasks. For details, see TiCDC OpenAPI.

13.9.3.1 Upgrade TiCDC using TiUP

This section introduces how to upgrade the TiCDC cluster using TiUP. In the following example, assume that you need to upgrade TiCDC and the entire TiDB cluster to v6.1.7.

```
tiup update --self && \
tiup update --all && \
tiup cluster upgrade <cluster-name> v6.1.7
```

13.9.3.1.1 Notes for upgrade

- The `changefeed` configuration has changed in TiCDC v4.0.2. See Compatibility notes for the configuration file for details.

- If you encounter any issues, see Upgrade TiDB using TiUP - FAQ.

13.9.3.2 Modify TiCDC configuration using TiUP

This section introduces how to modify the configuration of TiCDC cluster using the `tiup cluster edit-config` command of TiUP. The following example changes the value of `gc-ttl` from the default `86400` to `3600`, namely, one hour.

First, run the following command. You need to replace `<cluster-name>` with your actual cluster name.

```
tiup cluster edit-config <cluster-name>
```

Then, enter the vi editor page and modify the `cdc` configuraion under `server-configs`. The configuration is shown below:

```
server_configs:
 tidb: {}
 tikv: {}
 pd: {}
 tiflash: {}
 tiflash-learner: {}
 pump: {}
 drainer: {}
 cdc:
   gc-ttl: 3600
```

After the modification, run the `tiup cluster reload -R cdc` command to reload the configuration.

13.9.3.3 Use TLS

For details about using encrypted data transmission (TLS), see Enable TLS Between TiDB Components.

13.9.3.4 Use `cdc cli` to manage cluster status and data replication task

This section introduces how to use `cdc cli` to manage a TiCDC cluster and data replication tasks. `cdc cli` is the `cli` sub-command executed using the `cdc` binary. The following description assumes that:

- `cli` commands are executed directly using the `cdc` binary;
- PD listens on `10.0.10.25` and the port is `2379`.

Note:

The IP address and port that PD listens on correspond to the `advertise-client-urls` parameter specified during the `pd-server` startup. Multiple `pd-servers` have multiple `advertise-client-urls` parameters and you can specify one or multiple parameters. For example, `--pd=http://10.0.10.25:2379` or `--pd=http://10.0.10.25:2379,http://10.0.10.26:2379,http://10.0.10.27:2379`.

If you deploy TiCDC using TiUP, replace `cdc cli` in the following commands with `tiup ctl:<cluster-version> cdc`.

13.9.3.4.1 Manage TiCDC service progress (`capture`)

- Query the `capture` list:

```
cdc cli capture list --pd=http://10.0.10.25:2379
```

```
[
  {
    "id": "806e3a1b-0e31-477f-9dd6-f3f2c570abdd",
    "is-owner": true,
    "address": "127.0.0.1:8300"
  },
  {
    "id": "ea2a4203-56fe-43a6-b442-7b295f458ebc",
    "is-owner": false,
    "address": "127.0.0.1:8301"
  }
]
```

  - `id`: The ID of the service process.
  - `is-owner`: Indicates whether the service process is the owner node.
  - `address`: The address via which the service process provides interface to the outside.

13.9.3.4.2 Manage replication tasks (`changefeed`)

State transfer of replication tasks

The state of a replication task represents the running status of the replication task. During the running of TiCDC, replication tasks might fail with errors, be manually paused, resumed, or reach the specified `TargetTs`. These behaviors can lead to the change of the

replication task state. This section describes the states of TiCDC replication tasks and the transfer relationships between states.

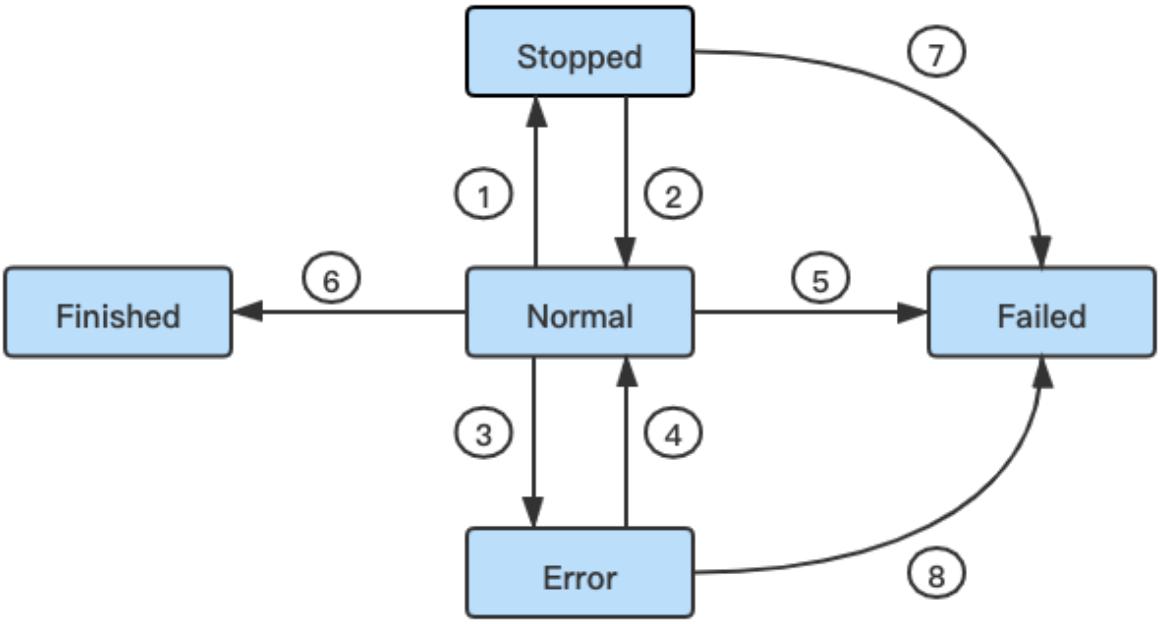


Figure 274: TiCDC state transfer

The states in the above state transfer diagram are described as follows:

- `Normal`: The replication task runs normally and the checkpoint-ts proceeds normally.
- `Stopped`: The replication task is stopped, because the user manually pauses the changefeed. The changefeed in this state blocks GC operations.
- `Error`: The replication task returns an error. The replication cannot continue due to some recoverable errors. The changefeed in this state keeps trying to resume until the state transfers to `Normal`. The changefeed in this state blocks GC operations.
- `Finished`: The replication task is finished and has reached the preset `TargetTs`. The changefeed in this state does not block GC operations.
- `Failed`: The replication task fails. Due to some unrecoverable errors, the replication task cannot resume and cannot be recovered. The changefeed in this state does not block GC operations.

The numbers in the above state transfer diagram are described as follows.

- Run the `changefeed pause` command.
- Run the `changefeed resume` command to resume the replication task.

- Recoverable errors occur during the `changefeed` operation, and the operation is resumed automatically.
- Run the `changefeed resume` command to resume the replication task.
- Unrecoverable errors occur during the `changefeed` operation.
- `changefeed` has reached the preset `TargetTs`, and the replication is automatically stopped.
- `changefeed` suspended longer than the duration specified by `gc-ttl`, and cannot be resumed.
- `changefeed` experienced an unrecoverable error when trying to execute automatic recovery.

Create a replication task

Run the following commands to create a replication task:

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
    ↪ root:123456@127.0.0.1:3306/" --changefeed-id="simple-replication-task
    ↪ " --sort-engine="unified"
```

```
Create changefeed successfully!
ID: simple-replication-task
Info: {"sink-uri":"mysql://root:123456@127.0.0.1:3306/","opts":{},"create-
    ↪ time":"2020-03-12T22:04:08.103600025+08:00","start-ts
    ↪ ":415241823337054209,"target-ts":0,"admin-job-type":0,"sort-engine":"
    ↪ unified","sort-dir":".","config":{"case-sensitive":true,"filter":{"
    ↪ rules":["*.*"],"ignore-txn-start-ts":null,"ddl-allow-list":null},"
    ↪ mounter":{"worker-num":16},"sink":{"dispatchers":null},"scheduler":{"
    ↪ type":"table-number","polling-time":-1}},"state":"normal","history":
    ↪ null,"error":null}
```

- `--changefeed-id`: The ID of the replication task. The format must match the `^[ ↪ a-zA-Z0-9]+(\-[a-zA-Z0-9]+)*$` regular expression. If this ID is not specified, TiCDC automatically generates a UUID (the version 4 format) as the ID.

- `--sink-uri`: The downstream address of the replication task. Configure `--sink-uri` according to the following format. Currently, the scheme supports `mysql/tidb/kafka ↪ /pulsar/s3/local`.

  ```
  [scheme]://[userinfo@][host]:[port][/path]?[query_parameters]
  ```

  When a URI contains special characters, you need to process these special characters using URL encoding.

- `--start-ts`: Specifies the starting TSO of the `changefeed`. From this TSO, the TiCDC cluster starts pulling data. The default value is the current time.

- `--target-ts`: Specifies the ending TSO of the `changefeed`. To this TSO, the TiCDC cluster stops pulling data. The default value is empty, which means that TiCDC does not automatically stop pulling data.
- `--sort-engine`: Specifies the sorting engine for the `changefeed`. Because TiDB and TiKV adopt distributed architectures, TiCDC must sort the data changes before writing them to the sink. This option supports `unified` (by default)/`memory`/`file`.
    - `unified`: When `unified` is used, TiCDC prefers data sorting in memory. If the memory is insufficient, TiCDC automatically uses the disk to store the temporary data. This is the default value of `--sort-engine`.
    - `memory`: Sorts data changes in memory. It is **NOT recommended** to use this sorting engine, because OOM is easily triggered when you replicate a large amount of data.
    - `file`: Entirely uses the disk to store the temporary data. This feature is **deprecated**. It is **NOT recommended** to use it in **any** situation.
- `--config`: Specifies the configuration file of the `changefeed`.
- `sort-dir`: Specifies the temporary file directory used by the sorting engine. **Note that this option is not supported since TiDB v4.0.13, v5.0.3 and v5.1.0. Do not use it any more**.

Configure sink URI with `mysql/tidb`

Sample configuration:

```
--sink-uri="mysql://root:123456@127.0.0.1:3306/?worker-count=16&max-txn-row
    ↪ =5000&transaction-atomicity=table"
```

The following are descriptions of parameters and parameter values that can be configured for the sink URI with `mysql/tidb`:

Parameter/Parameter Value	Description
`root`	The username of the downstream database
`123456`	The password of the downstream database
`127.0.0.1`	The IP address of the downstream database
`3306`	The port for the downstream data
`worker-count`	The number of SQL statements that can be concurrently executed to the downstream (optional, `16` by default)
`max-txn-row`	The size of a transaction batch that can be executed to the downstream (optional, `256` by default)
`ssl-ca`	The path of the CA certificate file needed to connect to the downstream MySQL instance (optional)

Parameter/Parameter Value	Description
`ssl-cert`	The path of the certificate file needed to connect to the downstream MySQL instance (optional)
`ssl-key`	The path of the certificate key file needed to connect to the downstream MySQL instance (optional)
`time-zone`	The time zone used when connecting to the downstream MySQL instance, which is effective since v4.0.8. This is an optional parameter. If this parameter is not specified, the time zone of TiCDC service processes is used. If this parameter is set to an empty value, no time zone is specified when TiCDC connects to the downstream MySQL instance and the default time zone of the downstream is used.
`transaction-atomicity`	The atomicity level of a transaction. This is an optional parameter, with the default value of `none`. When the value is `table`, TiCDC ensures the atomicity of a single-table transaction. When the value is `none`, TiCDC splits the single-table transaction.
`safe-mode`	Specifies how TiCDC handles `INSERT` and `UPDATE` statements when replicating data to the downstream. When it is `true`, TiCDC converts all upstream `INSERT` statements to `REPLACE INTO` statements, and all `UPDATE` statements to `DELETE` + `REPLACE INTO` statements. Before v6.1.3, the default value of this parameter is `true`. Starting from v6.1.3, the default value is changed to `false`. When TiCDC starts, it obtains a current timestamp `ThresholdTs`. For `INSERT` and `UPDATE` statements with `CommitTs` less than `ThresholdTs`, TiCDC converts them to `REPLACE INTO` statements and `DELETE` + `REPLACE INTO` statements respectively. For `INSERT` and `UPDATE` statements with `CommitTs` greater than or equal to `ThresholdTs`, TiCDC directly replicates them to the downstream.

Configure sink URI with `kafka`

Sample configuration:

```
--sink-uri="kafka://127.0.0.1:9092/topic-name?kafka-version=2.4.0&partition-
    ↪ num=6&max-message-bytes=67108864&replication-factor=1"
```

The following are descriptions of parameters and parameter values that can be configured

for the sink URI with `kafka`:

Parameter/Parameter Value	Description
`127.0.0.1`	The IP address of the downstream Kafka services
`9092`	The port for the downstream Kafka
`topic-name`	Variable. The name of the Kafka topic
`kafka-version`	The version of the downstream Kafka (optional, `2.4.0` by default. Currently, the earliest supported Kafka version is `0.11.0.2` and the latest one is `3.1.0`. This value needs to be consistent with the actual version of the downstream Kafka)
`kafka-client-id`	Specifies the Kafka client ID of the replication task (optional. `TiCDC_sarama_producer_replication ID` by default)
`partition-num`	The number of the downstream Kafka partitions (optional. The value must be **no greater than** the actual number of partitions; otherwise, the replication task cannot be created successfully. `3` by default)
`max-message-bytes`	The maximum size of data that is sent to Kafka broker each time (optional, `10MB` by default). From v5.0.6 and v4.0.6, the default value has changed from 64MB and 256MB to 10MB.
`replication-factor`	The number of Kafka message replicas that can be saved (optional, `1` by default)
`compression`	The compression algorithm used when sending messages (value options are `none`, `lz4`, `gzip`, `snappy`, and `zstd`; `none` by default).
`protocol`	The protocol with which messages are output to Kafka. The value options are `canal-json`, `open-protocol`, and `avro`.
`auto-create-topic`	Determines whether TiCDC creates the topic automatically when the `topic-name` passed in does not exist in the Kafka cluster (optional, `true` by default)
`enable-tidb-extension`	Optional. `false` by default. When the output protocol is `canal-json`, if the value is `true`, TiCDC sends Resolved events and adds the TiDB extension field to the Kafka message. From v6.1.0, this parameter is also applicable to the `avro` protocol. If the value is `true`, TiCDC adds three TiDB extension fields to the Kafka message.
`max-batch-size`	New in v4.0.9. If the message protocol supports outputting multiple data changes to one Kafka message, this parameter specifies the maximum number of data changes in one Kafka message. It currently takes effect only when Kafka's `protocol` is `open-protocol`. (optional, `16` by default)
`enable-tls`	Whether to use TLS to connect to the downstream Kafka instance (optional, `false` by default)
`ca`	The path of the CA certificate file needed to connect to the downstream Kafka instance (optional)

Parameter/Parameter Value	Description
`cert`	The path of the certificate file needed to connect to the downstream Kafka instance (optional)
`key`	The path of the certificate key file needed to connect to the downstream Kafka instance (optional)
`sasl-user`	The identity (authcid) of SASL/PLAIN or SASL/SCRAM authentication needed to connect to the downstream Kafka instance (optional)
`sasl-password`	The password of SASL/PLAIN or SASL/SCRAM authentication needed to connect to the downstream Kafka instance (optional)
`sasl-mechanism`	The name of SASL authentication needed to connect to the downstream Kafka instance. The value can be `plain`, `scram-sha-256`, `scram-sha-512`, or `gssapi`.
`sasl-gssapi-auth-type`	The gssapi authentication type. Values can be `user` or `keytab` (optional)
`sasl-gssapi-keytab-path`	The gssapi keytab path (optional)
`sasl-gssapi-kerberos-config-path`	The gssapi kerberos configuration path (optional)
`sasl-gssapi-service-name`	The gssapi service name (optional)
`sasl-gssapi-user`	The user name of gssapi authentication (optional)
`sasl-gssapi-password`	The password of gssapi authentication (optional)
`sasl-gssapi-realm`	The gssapi realm name (optional)
`sasl-gssapi-disable-pafxfast`	Whether to disable the gssapi PA-FX-FAST (optional)
`dial-timeout`	The timeout in establishing a connection with the downstream Kafka. The default value is `10s`
`read-timeout`	The timeout in getting a response returned by the downstream Kafka. The default value is `10s`
`write-timeout`	The timeout in sending a request to the downstream Kafka. The default value is `10s`
`avro-decimal-handling-mode`	Only effective with the `avro` protocol. Determines how Avro handles the DECIMAL field. The value can be `string` or `precise`, indicating either mapping the DECIMAL field to a string or a precise floating number.
`avro-bigint-unsigned-handling-mode`	Only effective with the `avro` protocol. Determines how Avro handles the BIGINT UNSIGNED field. The value can be `string` or `long`, indicating either mapping the BIGINT UNSIGNED field to a 64-bit signed number or a string.

Best practices:

- It is recommended that you create your own Kafka Topic. At a minimum, you need to set the maximum amount of data of each message that the Topic can send to the Kafka broker, and the number of downstream Kafka partitions. When you create a changefeed, these two settings correspond to `max-message-bytes` and `partition-num`, respectively.
- If you create a changefeed with a Topic that does not yet exist, TiCDC will try to create the Topic using the `partition-num` and `replication-factor` parameters. It is recommended that you specify these parameters explicitly.
- In most cases, it is recommended to use the `canal-json` protocol.

> **Note:**
>
> When `protocol` is `open-protocol`, TiCDC tries to avoid generating messages that exceed `max-message-bytes` in length. However, if a row is so large that a single change alone exceeds `max-message-bytes` in length, to avoid silent failure, TiCDC tries to output this message and prints a warning in the log.

TiCDC uses the authentication and authorization of Kafka

The following are examples when using Kafka SASL authentication:

- SASL/PLAIN

```
--sink-uri="kafka://127.0.0.1:9092/topic-name?kafka-version=2.4.0&sasl-
    ↪ user=alice-user&sasl-password=alice-secret&sasl-mechanism=plain"
```

- SASL/SCRAM

  SCRAM-SHA-256 and SCRAM-SHA-512 are similar to the PLAIN method. You just need to specify `sasl-mechanism` as the corresponding authentication method.

- SASL/GSSAPI

  SASL/GSSAPI `user` authentication:

```
--sink-uri="kafka://127.0.0.1:9092/topic-name?kafka-version=2.4.0&sasl-
    ↪ mechanism=gssapi&sasl-gssapi-auth-type=user&sasl-gssapi-kerberos-
    ↪ config-path=/etc/krb5.conf&sasl-gssapi-service-name=kafka&sasl-
    ↪ gssapi-user=alice/for-kafka&sasl-gssapi-password=alice-secret&
    ↪ sasl-gssapi-realm=example.com"
```

Values of `sasl-gssapi-user` and `sasl-gssapi-realm` are related to the principle specified in kerberos. For example, if the principle is set as `alice/for-kafka@example.com`, then `sasl-gssapi-user` and `sasl-gssapi-realm` are specified as `alice/for-kafka` and `example.com` respectively.

SASL/GSSAPI `keytab` authentication:

```
--sink-uri="kafka://127.0.0.1:9092/topic-name?kafka-version=2.4.0&sasl-mechanism=gssapi&sasl-gssapi-auth-type=keytab&sasl-gssapi-kerberos-config-path=/etc/krb5.conf&sasl-gssapi-service-name=kafka&sasl-gssapi-user=alice/for-kafka&sasl-gssapi-keytab-path=/var/lib/secret/alice.key&sasl-gssapi-realm=example.com"
```

For more information about SASL/GSSAPI authentication methods, see Configuring GSSAPI.

- TLS/SSL encryption

  If the Kafka broker has TLS/SSL encryption enabled, you need to add the `-enable-tls=true` parameter to `--sink-uri`. If you want to use self-signed certificates, you also need to specify `ca`, `cert` and `key` in `--sink-uri`.

- ACL authorization

  The minimum set of permissions required for TiCDC to function properly is as follows.

  - The `Create` and `Write` permissions for the Topic resource type.
  - The `DescribeConfigs` permission for the Cluster resource type.

Integrate TiCDC with Kafka Connect (Confluent Platform)

To use the data connectors provided by Confluent to stream data to relational or non-relational databases, you need to use the `avro` protocol and provide a URL for Confluent Schema Registry in `schema-registry`.

Sample configuration:

```
--sink-uri="kafka://127.0.0.1:9092/topic-name?&protocol=avro&replication-factor=3" --schema-registry="http://127.0.0.1:8081" --config changefeed_config.toml
```

```
[sink]
dispatchers = [
 {matcher = ['*.*'], topic = "tidb_{schema}_{table}"},
]
```

For detailed integration guide, see Quick Start Guide on Integrating TiDB with Confluent Platform.

Configure sink URI with `pulsar`

Warning:

This is still an experimental feature. Do **NOT** use it in a production environment.

Sample configuration:

```
--sink-uri="pulsar://127.0.0.1:6650/topic-name?connectionTimeout=2s"
```

The following are descriptions of parameters that can be configured for the sink URI with `pulsar`:

Parameter	Description
`connectionTimeout`	The timeout for establishing a connection to the downstream Pulsar, which is optional and defaults to 30 (seconds)
`operationTimeout`	The timeout for performing an operation on the downstream Pulsar, which is optional and defaults to 30 (seconds)
`tlsTrustCertsFilePath` ↪	The path of the CA certificate file needed to connect to the downstream Pulsar instance (optional)
`tlsAllowInsecureConnection` ↪	Determines whether to allow unencrypted connection after TLS is enabled (optional)
`tlsValidateHostname` ↪	Determines whether to verify the host name of the certificate from the downstream Pulsar (optional)
`maxConnectionsPerBroker` ↪	The maximum number of connections allowed to a single downstream Pulsar broker, which is optional and defaults to 1
`auth.tls`	Uses the TLS mode to verify the downstream Pulsar (optional). For example, `auth=tls&auth.tlsCertFile=/path/to/cert&` ↪ `auth.tlsKeyFile=/path/to/key`.
`auth.token`	Uses the token mode to verify the downstream Pulsar (optional). For example, `auth=token&auth.token=secret-token` or `auth=token&auth.file=path/to/secret-token-file`.
`name`	The name of Pulsar producer in TiCDC (optional)
`protocol`	The protocol with which messages are output to Pulsar. The value options are `canal-json`, `open-protocol`, and `avro`.
`maxPendingMessages` ↪	Sets the maximum size of the pending message queue, which is optional and defaults to 1000. For example, pending for the confirmation message from Pulsar.
`disableBatching`	Disables automatically sending messages in batches (optional)
`batchingMaxPublishDelay` ↪	Sets the duration within which the messages sent are batched (default: 10ms)
`compressionType`	Sets the compression algorithm used for sending messages (optional). The value options are `NONE`, `LZ4`, `ZLIB`, and `ZSTD`. (`NONE` by default)

Parameter	Description
`hashingScheme`	The hash algorithm used for choosing the partition to which a message is sent (optional). The value options are `JavaStringHash` (default) and `Murmur3`.
`properties.*`	The customized properties added to the Pulsar producer in TiCDC (optional). For example, `properties.location=Hangzhou`.

For more parameters of Pulsar, see pulsar-client-go ClientOptions and pulsar-client-go ProducerOptions.

Use the task configuration file

For more replication configuration (for example, specify replicating a single table), see Task configuration file.

You can use a configuration file to create a replication task in the following way:

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
    ↪ root:123456@127.0.0.1:3306/" --config changefeed.toml
```

In the command above, `changefeed.toml` is the configuration file for the replication task.

Query the replication task list

Run the following command to query the replication task list:

```
cdc cli changefeed list --pd=http://10.0.10.25:2379
```

```
[{
    "id": "simple-replication-task",
    "summary": {
      "state": "normal",
      "tso": 417886179132964865,
      "checkpoint": "2020-07-07 16:07:44.881",
      "error": null
    }
}]
```

- `checkpoint` indicates that TiCDC has already replicated data before this time point to the downstream.
- `state` indicates the state of the replication task.
    - `normal`: The replication task runs normally.
    - `stopped`: The replication task is stopped (manually paused).
    - `error`: The replication task is stopped (by an error).

- `removed`: The replication task is removed. Tasks of this state are displayed only when you have specified the `--all` option. To see these tasks when this option is not specified, run the `changefeed query` command.
- `finished`: The replication task is finished (data is replicated to the `target-ts`). Tasks of this state are displayed only when you have specified the `--all` option. To see these tasks when this option is not specified, run the `changefeed query` command.

Query a specific replication task

To query a specific replication task, run the `changefeed query` command. The query result includes the task information and the task state. You can specify the `--simple` or `-s` argument to simplify the query result that will only include the basic replication state and the checkpoint information. If you do not specify this argument, detailed task configuration, replication states, and replication table information are output.

```
cdc cli changefeed query -s --pd=http://10.0.10.25:2379 --changefeed-id=
    ↪ simple-replication-task
```

```
{
 "state": "normal",
 "tso": 419035700154597378,
 "checkpoint": "2020-08-27 10:12:19.579",
 "error": null
}
```

In the command and result above:

- `state` is the replication state of the current `changefeed`. Each state must be consistent with the state in `changefeed list`.
- `tso` represents the largest transaction TSO in the current `changefeed` that has been successfully replicated to the downstream.
- `checkpoint` represents the corresponding time of the largest transaction TSO in the current `changefeed` that has been successfully replicated to the downstream.
- `error` records whether an error has occurred in the current `changefeed`.

```
cdc cli changefeed query --pd=http://10.0.10.25:2379 --changefeed-id=simple-
    ↪ replication-task
```

```
{
  "info": {
    "sink-uri": "mysql://127.0.0.1:3306/?max-txn-row=20\u0026worker-count
        ↪ =4",
    "opts": {},
    "create-time": "2020-08-27T10:33:41.687983832+08:00",
```

```
  "start-ts": 419036036249681921,
  "target-ts": 0,
  "admin-job-type": 0,
  "sort-engine": "unified",
  "sort-dir": ".",
  "config": {
    "case-sensitive": true,
    "enable-old-value": false,
    "filter": {
      "rules": [
        "*.*"
      ],
      "ignore-txn-start-ts": null,
      "ddl-allow-list": null
    },
    "mounter": {
      "worker-num": 16
    },
    "sink": {
      "dispatchers": null,
    },
    "scheduler": {
      "type": "table-number",
      "polling-time": -1
    }
  },
  "state": "normal",
  "history": null,
  "error": null
},
"status": {
  "resolved-ts": 419036036249681921,
  "checkpoint-ts": 419036036249681921,
  "admin-job-type": 0
},
"count": 0,
"task-status": [
  {
    "capture-id": "97173367-75dc-490c-ae2d-4e990f90da0f",
    "status": {
      "tables": {
        "47": {
          "start-ts": 419036036249681921
        }
      },
```

```
      "operation": null,
      "admin-job-type": 0
    }
  }
 ]
}
```

In the command and result above:

- `info` is the replication configuration of the queried `changefeed`.
- `status` is the replication state of the queried `changefeed`.
  - `resolved-ts`: The largest transaction `TS` in the current `changefeed`. Note that this `TS` has been successfully sent from TiKV to TiCDC.
  - `checkpoint-ts`: The largest transaction `TS` in the current `changefeed`. Note that this `TS` has been successfully written to the downstream.
  - `admin-job-type`: The status of a `changefeed`:
    - `0`: The state is normal.
    - `1`: The task is paused. When the task is paused, all replicated `processors` exit. The configuration and the replication status of the task are retained, so you can resume the task from `checkpiont-ts`.
    - `2`: The task is resumed. The replication task resumes from `checkpoint-ts`.
    - `3`: The task is removed. When the task is removed, all replicated `processors` are ended, and the configuration information of the replication task is cleared up. Only the replication status is retained for later queries.
- `task-status` indicates the state of each replication sub-task in the queried `changefeed` ↪ .

Pause a replication task

Run the following command to pause a replication task:

```
cdc cli changefeed pause --pd=http://10.0.10.25:2379 --changefeed-id simple-
    ↪ replication-task
```

In the above command:

- `--changefeed-id=uuid` represents the ID of the `changefeed` that corresponds to the replication task you want to pause.

Resume a replication task

Run the following command to resume a paused replication task:

```
cdc cli changefeed resume --pd=http://10.0.10.25:2379 --changefeed-id simple
    ↪ -replication-task
```

In the above command:

- `--changefeed-id=uuid` represents the ID of the `changefeed` that corresponds to the replication task you want to resume.

Remove a replication task

Run the following command to remove a replication task:

```
cdc cli changefeed remove --pd=http://10.0.10.25:2379 --changefeed-id simple
    ↪ -replication-task
```

In the above command:

- `--changefeed-id=uuid` represents the ID of the `changefeed` that corresponds to the replication task you want to remove.

13.9.3.4.3 Update task configuration

Starting from v4.0.4, TiCDC supports modifying the configuration of the replication task (not dynamically). To modify the `changefeed` configuration, pause the task, modify the configuration, and then resume the task.

```
cdc cli changefeed pause -c test-cf --pd=http://10.0.10.25:2379
cdc cli changefeed update -c test-cf --pd=http://10.0.10.25:2379 --sink-uri
    ↪ ="mysql://127.0.0.1:3306/?max-txn-row=20&worker-count=8" --config=
    ↪ changefeed.toml
cdc cli changefeed resume -c test-cf --pd=http://10.0.10.25:2379
```

Currently, you can modify the following configuration items:

- `sink-uri` of the `changefeed`.
- The `changefeed` configuration file and all configuration items in the file.
- Whether to use the file sorting feature and the sorting directory.
- The `target-ts` of the `changefeed`.

13.9.3.4.4 Manage processing units of replication sub-tasks (`processor`)

- Query the `processor` list:

```
cdc cli processor list --pd=http://10.0.10.25:2379
```

```
[
        {
                "id": "9f84ff74-abf9-407f-a6e2-56aa35b33888",
                "capture-id": "b293999a-4168-4988-a4f4-35d9589b226b",
                "changefeed-id": "simple-replication-task"
        }
]
```

- Query a specific `changefeed` which corresponds to the status of a specific replication task:

```
cdc cli processor query --pd=http://10.0.10.25:2379 --changefeed-id=
    ↪ simple-replication-task --capture-id=b293999a-4168-4988-a4f4-35
    ↪ d9589b226b
```

```
{
  "status": {
    "tables": {
      "56": {   # ID of the replication table, corresponding to
          ↪ tidb_table_id of a table in TiDB
        "start-ts": 417474117955485702
      }
    },
    "operation": null,
    "admin-job-type": 0
  },
  "position": {
    "checkpoint-ts": 417474143881789441,
    "resolved-ts": 417474143881789441,
    "count": 0
  }
}
```

In the command above:

- `status.tables`: Each key number represents the ID of the replication table, corresponding to `tidb_table_id` of a table in TiDB.
- `resolved-ts`: The largest TSO among the sorted data in the current processor.
- `checkpoint-ts`: The largest TSO that has been successfully written to the downstream in the current processor.

13.9.3.5 Task configuration file

This section introduces the configuration of a replication task.

```
### Specifies whether the database names and tables in the configuration
    ↪ file are case-sensitive.
### The default value is true.
### This configuration item affects configurations related to filter and
    ↪ sink.
case-sensitive = true

### Specifies whether to output the old value. New in v4.0.5. Since v5.0,
    ↪ the default value is `true`.
enable-old-value = true

[filter]
### Ignores the transaction of specified start_ts.
### ignore-txn-start-ts = [1, 2]

### Filter rules.
### Filter syntax: https://docs.pingcap.com/tidb/stable/table-filter#syntax.
rules = ['*.*', '!test.*']

[mounter]
### mounter thread counts, which is used to decode the TiKV output data.
worker-num = 16

[sink]
### For the sink of MQ type, you can use dispatchers to configure the event
    ↪ dispatcher.
### Since v6.1, TiDB supports two types of event dispatchers: partition and
    ↪ topic. For more information, see the following section.
### The matching syntax of matcher is the same as the filter rule syntax.
    ↪ For details about the matcher rules, see the following section.

### dispatchers = [
###   {matcher = ['test1.*', 'test2.*'], topic = "Topic expression 1",
    ↪ partition = "ts" },
###   {matcher = ['test3.*', 'test4.*'], topic = "Topic expression 2",
    ↪ partition = "index-value" },
###   {matcher = ['test1.*', 'test5.*'], topic = "Topic expression 3",
    ↪ partition = "table"},
###   {matcher = ['test6.*'], partition = "ts"}
### ]
### For the sink of MQ type, you can specify the protocol format of the
    ↪ message.
### Currently the following protocols are supported: canal-json, open-
    ↪ protocol, and avro.
```

```
### protocol = "canal-json"
```

13.9.3.5.1 Notes for compatibility

- In TiCDC v4.0.0, `ignore-txn-commit-ts` is removed and `ignore-txn-start-ts` is added, which uses start_ts to filter transactions.
- In TiCDC v4.0.2, `db-dbs/db-tables/ignore-dbs/ignore-tables` are removed and `rules` is added, which uses new filter rules for databases and tables. For detailed filter syntax, see Table Filter.

13.9.3.6 Customize the rules for Topic and Partition dispatchers of Kafka Sink

13.9.3.6.1 Matcher rules

In the example of the previous section:

- For the tables that match the matcher rule, they are dispatched according to the policy specified by the corresponding topic expression. For example, the `test3.aa` table is dispatched according to "Topic expression 2"; the `test5.aa` table is dispatched according to "Topic expression 3".
- For a table that matches multiple matcher rules, it is dispatched according to the first matching topic expression. For example, the `test1.aa` table is distributed according to "Topic expression 1".
- For tables that do not match any matcher rule, the corresponding data change events are sent to the default topic specified in `--sink-uri`. For example, the `test10.aa` table is sent to the default topic.
- For tables that match the matcher rule but do not specify a topic dispatcher, the corresponding data changes are sent to the default topic specified in `--sink-uri`. For example, the `test6.aa` table is sent to the default topic.

13.9.3.6.2 Topic dispatchers

You can use topic = "xxx" to specify a Topic dispatcher and use topic expressions to implement flexible topic dispatching policies. It is recommended that the total number of topics be less than 1000.

The format of the Topic expression is `[prefix]{schema}[middle][{table}][suffix]`.

- `prefix`: optional. Indicates the prefix of the Topic Name.
- `{schema}`: required. Used to match the schema name.
- `middle`: optional. Indicates the delimiter between schema name and table name.
- `{table}`: optional. Used to match the table name.
- `suffix`: optional. Indicates the suffix of the Topic Name.

`prefix`, `middle` and `suffix` can only include the following characters: `a-z`, `A-Z`, `0-9`, `.`, `_` and `-`. `{schema}` and `{table}` are both lowercase. Placeholders such as `{Schema}` and `{TABLE}` are invalid.

Some examples:

- `matcher = ['test1.table1', 'test2.table2'], topic = "hello_{schema}_{table}"`
  - The data change events corresponding to `test1.table1` are sent to the topic named `hello_test1_table1`.
  - The data change events corresponding to `test2.table2` are sent to the topic named `hello_test2_table2`.
- `matcher = ['test3.*', 'test4.*'], topic = "hello_{schema}_world"`
  - The data change events corresponding to all tables in `test3` are sent to the topic named `hello_test3_world`.
  - The data change events corresponding to all tables in `test4` are sent to the topic named `hello_test4_world`.
- `matcher = ['*.*'], topic = "{schema}_{table}"`
  - All tables listened by TiCDC are dispatched to separate topics according to the "schema_table" rule. For example, for the `test.account` table, TiCDC dispatches its data change log to a Topic named `test_account`.

13.9.3.6.3 Dispatch DDL events

Schema-level DDLs

DDLs that are not related to a specific table are called schema-level DDLs, such as `create database` and `drop database`. The events corresponding to schema-level DDLs are sent to the default topic specified in `--sink-uri`.

Table-level DDLs

DDLs that are related to a specific table are called table-level DDLs, such as `alter table` and `create table`. The events corresponding to table-level DDLs are sent to the corresponding topic according to dispatcher configurations.

For example, for a dispatcher like `matcher = ['test.*'], topic = {schema}_{table}`, DDL events are dispatched as follows:

- If a single table is involved in the DDL event, the DDL event is sent to the corresponding topic as is. For example, for the DDL event `drop table test.table1`, the event is sent to the topic named `test_table1`.

- If multiple tables are involved in the DDL event (`rename table` / `drop table` / `drop view` may involve multiple tables), the DDL event is split into multiple events and sent to the corresponding topics. For example, for the DDL event `rename table test.table1 to test.table10, test.table2 to test.table20`, the event `rename table test.table1 to test.table10` is sent to the topic named `test_table1` and the event `rename table test.table2 to test.table20` is sent to the topic named `test.table2`.

13.9.3.6.4 Partition dispatchers

You can use `partition = "xxx"` to specify a partition dispatcher. It supports four dispatchers: default, ts, index-value, and table. The dispatcher rules are as follows:

- default: When multiple unique indexes (including the primary key) exist or the Old Value feature is enabled, events are dispatched in the table mode. When only one unique index (or the primary key) exists, events are dispatched in the index-value mode.
- ts: Use the commitTs of the row change to hash and dispatch events.
- index-value: Use the value of the primary key or the unique index of the table to hash and dispatch events.
- table: Use the schema name of the table and the table name to hash and dispatch events.

Note:

Since v6.1, to clarify the meaning of the configuration, the configuration used to specify the partition dispatcher has been changed from `dispatcher` to `partition`, with `partition` being an alias for `dispatcher`. For example, the following two rules are exactly equivalent.

```
[sink]
dispatchers = [
   {matcher = ['*.*'], dispatcher = "ts"},
   {matcher = ['*.*'], partition = "ts"},
]
```

However, `dispatcher` and `partition` cannot appear in the same rule. For example, the following rule is invalid.

```
{matcher = ['*.*'], dispatcher = "ts", partition = "table"},
```

13.9.3.7 Output the historical value of a Row Changed Event New in v4.0.5

In the default configuration, the Row Changed Event of TiCDC Open Protocol output in a replication task only contains the changed value, not the value before the change. Therefore, the output value cannot be used by the consumer ends of TiCDC Open Protocol as the historical value of a Row Changed Event.

Starting from v4.0.5, TiCDC supports outputting the historical value of a Row Changed Event. To enable this feature, specify the following configuration in the `changefeed` configuration file at the root level:

```
enable-old-value = true
```

This feature is enabled by default since v5.0. To learn the output format of the TiCDC Open Protocol after this feature is enabled, see TiCDC Open Protocol - Row Changed Event.

13.9.3.8 Replicate tables with the new framework for collations enabled

Starting from v4.0.15, v5.0.4, v5.1.1 and v5.2.0, TiCDC supports tables that have enabled new framework for collations.

13.9.3.9 Replicate tables without a valid index

Since v4.0.8, TiCDC supports replicating tables that have no valid index by modifying the task configuration. To enable this feature, configure in the `changefeed` configuration file as follows:

```
enable-old-value = true
force-replicate = true
```

> **Warning:**
>
> For tables without a valid index, operations such as `INSERT` and `REPLACE` are not reentrant, so there is a risk of data redundancy. TiCDC guarantees that data is distributed only at least once during the replication process. Therefore, enabling this feature to replicate tables without a valid index will definitely cause data redundancy. If you do not accept data redundancy, it is recommended to add an effective index, such as adding a primary key column with the `AUTO RANDOM` attribute.

13.9.3.10 Unified Sorter

Unified sorter is the sorting engine in TiCDC. It can mitigate OOM problems caused by the following scenarios:

- The data replication task in TiCDC is paused for a long time, during which a large amount of incremental data is accumulated and needs to be replicated.

- The data replication task is started from an early timestamp so it becomes necessary to replicate a large amount of incremental data.

For the changefeeds created using `cdc cli` after v4.0.13, Unified Sorter is enabled by default; for the changefeeds that have existed before v4.0.13, the previous configuration is used.

To check whether or not the Unified Sorter feature is enabled on a changefeed, you can run the following example command (assuming the IP address of the PD instance is `http://10.0.10.25:2379`):

```
cdc cli --pd="http://10.0.10.25:2379" changefeed query --changefeed-id=
    ↪ simple-replication-task | grep 'sort-engine'
```

In the output of the above command, if the value of `sort-engine` is "unified", it means that Unified Sorter is enabled on the changefeed.

Note:

- If your servers use mechanical hard drives or other storage devices that have high latency or limited bandwidth, use the unified sorter with caution.
- By default, Unified Sorter uses `data_dir` to store temporary files. It is recommended to ensure that the free disk space is greater than or equal to 500 GiB. For production environments, it is recommended to ensure that the free disk space on each node is greater than (the maximum `checkpoint-ts` delay allowed by the business) * (upstream write traffic at business peak hours). In addition, if you plan to replicate a large amount of historical data after `changefeed` is created, make sure that the free space on each node is greater than the amount of replicated data.
- Unified sorter is enabled by default. If your servers do not match the above requirements and you want to disable the unified sorter, you need to manually set `sort-engine` to `memory` for the changefeed.
- To enable Unified Sorter on an existing changefeed that uses `memory` to sort, see the methods provided in How do I handle the OOM that occurs after TiCDC is restarted after a task interruption?.

13.9.3.11 Eventually consistent replication in disaster scenarios

Starting from v6.1.1, this feature becomes GA. Starting from v5.3.0, TiCDC supports backing up incremental data from an upstream TiDB cluster to an object storage or an NFS of the downstream cluster. When the upstream cluster encounters a disaster and becomes

unavailable, TiCDC can restore the downstream data to the recent eventually consistent state. This is the eventually consistent replication capability provided by TiCDC. With this capability, you can switch applications to the downstream cluster quickly, avoiding long-time downtime and improving service continuity.

Currently, TiCDC can replicate incremental data from a TiDB cluster to another TiDB cluster or a MySQL-compatible database system (including Aurora, MySQL, and MariaDB). In case the upstream cluster crashes, TiCDC can restore data in the downstream cluster within 5 minutes, given the conditions that TiCDC replicates data normally before the crash, and the replication lag is small. It allows data loss of 10s at most, that is, RTO <= 5 min, and P95 RPO <= 10s.

TiCDC replication lag increases in the following scenarios:

- The TPS increases significantly in a short time
- Large or long transactions occur in the upstream
- The TiKV or TiCDC cluster in the upstream is reloaded or upgraded
- Time-consuming DDL statements, such as `add index`, are executed in the upstream
- The PD is configured with aggressive scheduling strategies, resulting in frequent transfer of Region leaders, or frequent Region merge or Region split

> **Note:**
>
> Starting from v6.1.1, the eventually consistent replication feature of TiCDC supports Amazon S3-compatible object storage. Starting from v6.1.4, this feature supports GCS- and Azure-compatible object storage.

13.9.3.11.1 Prerequisites

- Prepare a highly available object storage or NFS for storing TiCDC's real-time incremental data backup files. These files can be accessed in case of a disaster in the upstream.
- Enable this feature for changefeeds that need to have eventual consistency in disaster scenarios. To enable it, you can add the following configuration to the changefeed configuration file.

```
[consistent]
### Consistency level. Options include:
### - none: the default value. In a non-disaster scenario, eventual
   ↪ consistency is only guaranteed if and only if finished-ts is
   ↪ specified.
```

```
### - eventual: Uses redo log to guarantee eventual consistency in case of
    ↪ the primary cluster disasters.
level = "eventual"

### Individual redo log file size, in MiB. By default, it's 64. It is
    ↪ recommended to be no more than 128.
max-log-size = 64

### The interval for flushing or uploading redo logs to Amazon S3, in
    ↪ milliseconds. It is recommended that this configuration be equal to
    ↪ or greater than 2000.
flush-interval = 2000

### # The path under which redo log backup is stored. The scheme can be nfs
    ↪ (NFS directory), or Amazon S3, GCS, and Azure (uploaded to object
    ↪ storage).
storage = "$SCHEME://logbucket/test-changefeed?endpoint=http://$ENDPOINT/"
```

13.9.3.11.2 Disaster recovery

When a disaster happens in the primary cluster, you need to recover manually in the secondary cluster by running the `cdc redo` command. The recovery process is as follows.

1. Ensure that all the TiCDC processes have exited. This is to prevent the primary cluster from resuming service during data recovery and prevent TiCDC from restarting data synchronization.
2. Use cdc binary for data recovery. Run the following command:

```
cdc redo apply --tmp-dir="/tmp/cdc/redo/apply" \
    --storage="s3://logbucket/test-changefeed?endpoint=http
        ↪ ://10.0.10.25:24927/" \
    --sink-uri="mysql://normal:123456@10.0.10.55:3306/"
```

In this command:

- `tmp-dir`: Specifies the temporary directory for downloading TiCDC incremental data backup files.
- `storage`: Specifies the address for storing the TiCDC incremental data backup files, either an URI of object storage or an NFS directory.
- `sink-uri`: Specifies the secondary cluster address to restore the data to. Scheme can only be `mysql`.

13.9.4 Monitor and Alert

13.9.4.1 Key Monitoring Metrics of TiCDC

If you use TiUP to deploy the TiDB cluster, you can see a sub-dashboard for TiCDC in the monitoring system which is deployed at the same time. You can get an overview of TiCDC's current status from the TiCDC dashboard, where the key metrics are displayed. This document provides a detailed description of these key metrics.

The metric description in this document is based on the following replication task example, which replicates data to MySQL using the default configuration.

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
    ↪ root:123456@127.0.0.1:3306/" --changefeed-id="simple-replication-task
    ↪ "
```

The TiCDC dashboard contains four monitoring panels. See the following screenshot:



Figure 275: TiCDC Dashboard - Overview

The description of each panel is as follows:

- **Server**: The summary information of TiKV nodes and TiCDC nodes in the TiDB cluster
- **Changefeed**: The detailed information of TiCDC replication tasks
- **Events**: The detail information about the data flow within the TiCDC cluster
- **TiKV**: TiKV information related to TiCDC

13.9.4.1.1 Server

The following is an example of the **Server** panel:



Figure 276: TiCDC Dashboard - Server metrics

The description of each metric in the **Server** panel is as follows:

- Uptime: The time for which TiKV nodes and TiCDC nodes have been running
- Goroutine count: The number of goroutines of a TiCDC node
- Open FD count: The number of file handles opened by TiCDC nodes
- Ownership: The current status of nodes in the TiCDC cluster
- Ownership history: The ownership history of the TiCDC cluster
- CPU usage: The CPU usage of TiCDC nodes
- Memory usage: The memory usage of TiCDC nodes

13.9.4.1.2 Changefeed

The following is an example of the **Changefeed** panel:





The description of each metric in the **Changefeed** panel is as follows:

- Changefeed table count: The number of tables that each TiCDC node needs to replicate in the replication task
- Processor resolved ts: The timestamps that have been resolved in the TiCDC cluster
- Table resolved ts: The replication progress of each table in the replication task
- Changefeed checkpoint: The progress of replicating data to the downstream. Normally, the green bars are connected to the yellow line
- PD etcd requests/s: The number of requests that a TiCDC node sends to PD per second
- Exit error count: The number of errors that interrupt the replication task per minute
- Changefeed checkpoint lag: The progress lag of data replication (the unit is second) between the upstream and the downstream
- Changefeed resolved ts lag: The progress lag of data replication (the unit is second) between the upstream and TiCDC nodes
- Flush sink duration: The histogram of the time spent by TiCDC asynchronously flushing data to the downstream
- Flush sink duration percentile: The time (P95, P99, and P999) spent by TiCDC asynchronously flushing data to the downstream within one second
- Sink write duration: The histogram of the time spent by TiCDC writing a transaction change to the downstream
- Sink write duration percentile: The time (P95, P99, and P999) spent by TiCDC writing a transaction change to the downstream within one second
- MySQL sink conflict detect duration: The histogram of the time spent on detecting

MySQL sink conflicts

- MySQL sink conflict detect duration percentile: The time (P95, P99, and P999) spent on detecting MySQL sink conflicts within one second
- MySQL sink worker load: The workload of MySQL sink workers of TiCDC nodes

13.9.4.1.3 Events

The following is an example of the **Events** panel:



The description of each metric in the **Events** panel is as follows:

- Eventfeed count: The number of Eventfeed RPC requests of TiCDC nodes
- Event size percentile: The event size (P95, P99, and P999) which TiCDC receives from TiKV within one second
- Eventfeed error/m: The number of errors reported by Eventfeed RPC requests of TiCDC nodes per minute
- KV client receive events/s: The number of events that the KV client module of TiCDC nodes receives from TiKV per second
- Puller receive events/s: The number of events that the Puller module of TiCDC nodes receives from the KV client per second
- Puller output events/s: The number of events that the Puller module of TiCDC nodes sends to the Sorter module per second
- Sink flush rows/s: The number of events that TiCDC nodes write to the downstream per second
- Puller buffer size: The number of events that TiCDC nodes cache in the Puller module
- Entry sorter buffer size: The number of events that TiCDC nodes cache in the Sorter module
- Processor/Mounter buffer size: The number of events that TiCDC nodes cache in the Processor module and the Mounter module
- Sink row buffer size: The number of events that TiCDC nodes cache in the Sink module
- Entry sorter sort duration: The histogram of the time spent by TiCDC nodes sorting events
- Entry sorter sort duration percentile: The time (P95, P99, and P999) spent by TiCDC sorting events within one second
- Entry sorter merge duration: The histogram of the time spent by TiCDC nodes merging sorted events
- Entry sorter merge duration percentile: The time (P95, P99, and P999) spent by TiCDC merging sorted events within one second
- Mounter unmarshal duration: The histogram of the time spent by TiCDC nodes unmarshaling events
- Mounter unmarshal duration percentile: The time (P95, P99, and P999) spent by TiCDC unmarshaling events within one second
- KV client dispatch events/s: The number of events that the KV client module dispatches among the TiCDC nodes
- KV client batch resolved size: The batch size of resolved timestamp messages that TiKV sends to TiCDC

13.9.4.1.4 TiKV

The following is an example of the **TiKV** panel:



The description of each metric in the **TiKV** panel is as follows:

- CDC endpoint CPU: The CPU usage of the CDC endpoint threads on TiKV nodes
- CDC worker CPU: The CPU usage of the CDC worker threads on TiKV nodes
- Min resolved ts: The minimum resolved timestamp on TiKV nodes
- Min resolved region: The Region ID of the minimum resolved timestamp on TiKV nodes
- Resolved ts lag duration percentile: The lag between the minimum resolved timestamp on TiKV nodes and the current time
- Initial scan duration: The histogram of the time spent on incremental scan when TiKV nodes connect to TiCDC nodes
- Initial scan duration percentile: The time (P95, P99, and P999) spent on the incremental scan of TiKV nodes within one second
- Memory without block cache: The memory usage of TiKV nodes excluding the RocksDB block cache
- CDC pending bytes in memory: The memory usage of CDC module on TiKV nodes
- Captured region count: The number of event-capturing Regions on TiKV nodes

13.9.4.2 TiCDC Alert Rules

This document describes the TiCDC alert rules and the corresponding solutions. In descending order, the severity levels are: **Critical**, **Warning**.

13.9.4.2.1 Critical alerts

This section introduces critical alerts and solutions.

`cdc_checkpoint_high_delay`

For critical alerts, you need to pay close attention to abnormal monitoring metrics.

- Alert rule:

  `(time()- ticdc_processor_checkpoint_ts / 1000)> 600`

- Description:

  A replication task is delayed more than 10 minutes.

- Solution:

  See TiCDC Handles Replication Interruption.

13.9.4.2.2 `cdc_resolvedts_high_delay`

- Alert rule:

  `(time()- ticdc_processor_resolved_ts / 1000)> 300`

- Description:

  The Resolved TS of a replication task is delayed more than 5 minutes.

- Solution:

  See TiCDC Handles Replication Interruption.

`ticdc_changefeed_failed`

- Alert rule:

  `(max_over_time(ticdc_owner_status[1m])== 2)> 0`

- Description:

  A replication task encounters an unrecoverable error and enters the failed state.

- Solution:

  This alert is similar to replication interruption. See TiCDC Handles Replication Interruption.

`ticdc_processor_exit_with_error_count`

- Alert rule:

  `changes(ticdc_processor_exit_with_error_count[1m])> 0`

- Description:

  A replication task reports an error and exits.

- Solution:

  See TiCDC Handles Replication Interruption.

13.9.4.2.3 Warning alerts

Warning alerts are a reminder for an issue or error.

`cdc_multiple_owners`

- Alert rule:

  `sum(rate(ticdc_owner_ownership_counter[30s]))>= 2`

- Description:

  There are multiple owners in the TiCDC cluster.

- Solution:

  Collect TiCDC logs to locate the root cause.

`ticdc_mounter_unmarshal_and_mount_time_more_than_1s`

- Alert rule:

```
histogram_quantile(0.9, rate(ticdc_mounter_unmarshal_and_mount_bucket[1m
↪ ]))* 1000 > 1000
```

- Description:

  It takes a replication task more than 1 second to unmarshal the data changes.

- Solution:

  Collect TiCDC logs to locate the root cause.

`cdc_sink_execute_duration_time_more_than_10s`

- Alert rule:

```
histogram_quantile(0.9, rate(ticdc_sink_txn_exec_duration_bucket[1m]))>
↪  10
```

- Description:

  It takes a replication task more than 10 seconds to write data to the downstream database.

- Solution:

  Check whether there are problems in the downstream database.

`cdc_processor_checkpoint_tso_no_change_for_1m`

- Alert rule:

  `changes(ticdc_processor_checkpoint_ts[1m])< 1`

- Description:

  A replication task has not advanced for more than 1 minute.

- Solution:

  See TiCDC Handles Replication Interruption.

`ticdc_puller_entry_sorter_sort_bucket`

- Alert rule:

  `histogram_quantile(0.9, rate(ticdc_puller_entry_sorter_sort_bucket{}[1m ↪ ]))> 1`

- Description:

  The delay of TiCDC puller entry sorter is too high.

- Solution:

  Collect TiCDC logs to locate the root cause.

`ticdc_puller_entry_sorter_merge_bucket`

- Alert rule:

  `histogram_quantile(0.9, rate(ticdc_puller_entry_sorter_merge_bucket{}[1 ↪ m]))> 1`

- Description:

  The delay of TiCDC puller entry sorter merge is too high.

- Solution:

  Collect TiCDC logs to locate the root cause.

`tikv_cdc_min_resolved_ts_no_change_for_1m`

- Alert rule:

```
changes(tikv_cdc_min_resolved_ts[1m])< 1 and ON (instance)tikv_cdc_region_resolve_s
↪ {status="resolved"} > 0 and ON (instance)tikv_cdc_captured_region_total
↪  > 0
```

- Description:

  The minimum Resolved TS 1 of TiKV CDC has not advanced for 1 minute.

- Solution:

  Collect TiKV logs to locate the root cause.

`tikv_cdc_scan_duration_seconds_more_than_10min`

- Alert rule:

```
histogram_quantile(0.9, rate(tikv_cdc_scan_duration_seconds_bucket{}[1m
↪ ]))> 600
```

- Description:

  The TiKV CDC module has scanned for incremental replication for more than 10 minutes.

- Solution:

  Collect TiCDC monitoring metrics and TiKV logs to locate the root cause.

`ticdc_sink_mysql_execution_error`

- Alert rule:

```
changes(ticdc_sink_mysql_execution_error[1m])> 0
```

- Description:

  An error occurs when a replication task writes data to the downstream MySQL.

- Solution:

  There are many possible root causes. See Troubleshoot TiCDC.

`ticdc_memory_abnormal`

- Alert rule:

```
go_memstats_heap_alloc_bytes{job="ticdc"} > 1e+10
```

- Description:

  The TiCDC heap memory usage exceeds 10 GiB.

- Solution:

  Collect TiCDC logs to locate the root cause.

13.9.5 Troubleshoot TiCDC

This document introduces the common errors you might encounter when using TiCDC, and the corresponding maintenance and troubleshooting methods.

> **Note:**
>
> In this document, the PD address specified in `cdc cli` commands is `--pd =http://10.0.10.25:2379`. When you use the command, replace the address with your actual PD address.

13.9.5.1 TiCDC replication interruptions

13.9.5.1.1 How do I know whether a TiCDC replication task is interrupted?

- Check the `changefeed checkpoint` monitoring metric of the replication task (choose the right `changefeed id`) in the Grafana dashboard. If the metric value stays unchanged, or the `checkpoint lag` metric keeps increasing, the replication task might be interrupted.
- Check the `exit error count` monitoring metric. If the metric value is greater than `0`, an error has occurred in the replication task.
- Execute `cdc cli changefeed list` and `cdc cli changefeed query` to check the status of the replication task. `stopped` means the task has stopped, and the `error` item provides the detailed error message. After the error occurs, you can search `error on running processor` in the TiCDC server log to see the error stack for troubleshooting.
- In some extreme cases, the TiCDC service is restarted. You can search the `FATAL` level log in the TiCDC server log for troubleshooting.

13.9.5.1.2 How do I know whether the replication task is stopped manually?

You can know whether the replication task is stopped manually by executing `cdc cli`. For example:

```
cdc cli changefeed query --pd=http://10.0.10.25:2379 --changefeed-id 28
    ↪ c43ffc-2316-4f4f-a70b-d1a7c59ba79f
```

In the output of the above command, `admin-job-type` shows the state of this replication task:

- 0: In progress, which means that the task is not stopped manually.
- 1: Paused. When the task is paused, all replicated `processor`s exit. The configuration and the replication status of the task are retained, so you can resume the task from `checkpiont-ts`.
- 2: Resumed. The replication task resumes from `checkpoint-ts`.
- 3: Removed. When the task is removed, all replicated `processor`s are ended, and the configuration information of the replication task is cleared up. The replication status is retained only for later queries.

13.9.5.1.3 How do I handle replication interruptions?

A replication task might be interrupted in the following known scenarios:

- The downstream continues to be abnormal, and TiCDC still fails after many retries.
    - In this scenario, TiCDC saves the task information. Because TiCDC has set the service GC safepoint in PD, the data after the task checkpoint is not cleaned by TiKV GC within the valid period of `gc-ttl`.
    - Handling method: You can resume the replication task via the HTTP interface after the downstream is back to normal.
- Replication cannot continue because of incompatible SQL statement(s) in the downstream.
    - In this scenario, TiCDC saves the task information. Because TiCDC has set the service GC safepoint in PD, the data after the task checkpoint is not cleaned by TiKV GC within the valid period of `gc-ttl`.
    - Handling procedures:
        1. Query the status information of the replication task using the `cdc cli changefeed query` command and record the value of `checkpoint-ts`.
        2. Use the new task configuration file and add the `ignore-txn-start-ts` parameter to skip the transaction corresponding to the specified `start-ts`.
        3. Stop the old replication task via HTTP API. Execute `cdc cli changefeed create` to create a new task and specify the new task configuration file. Specify `checkpoint-ts` recorded in step 1 as the `start-ts` and start a new task to resume the replication.
- In TiCDC v4.0.13 and earlier versions, when TiCDC replicates the partitioned table, it might encounter an error that leads to replication interruption.
    - In this scenario, TiCDC saves the task information. Because TiCDC has set the service GC safepoint in PD, the data after the task checkpoint is not cleaned by TiKV GC within the valid period of `gc-ttl`.
    - Handling procedures:

1. Pause the replication task by executing `cdc cli changefeed pause -c <changefeed-id>`.
2. Wait for about one munite, and then resume the replication task by executing `cdc cli changefeed resume -c <changefeed-id>`.

13.9.5.1.4 What should I do to handle the OOM that occurs after TiCDC is restarted after a task interruption?

- Update your TiDB cluster and TiCDC cluster to the latest versions. The OOM problem has already been resolved in **v4.0.14 and later v4.0 versions, v5.0.2 and later v5.0 versions, and the latest versions**.

13.9.5.2 How do I handle the `Error 1298: Unknown or incorrect time zone: 'UTC'` error when creating the replication task or replicating data to MySQL?

This error is returned when the downstream MySQL does not load the time zone. You can load the time zone by running `mysql_tzinfo_to_sql`. After loading the time zone, you can create tasks and replicate data normally.

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql -p
```

If the output of the command above is similar to the following one, the import is successful:

```
Enter password:
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone.
    ↪ Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leap-seconds.list' as time zone
    ↪ . Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone.
    ↪ Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone.
    ↪ Skipping it.
```

If the downstream is a special MySQL environment (a public cloud RDS or some MySQL derivative versions) and importing the time zone using the above method fails, you need to specify the MySQL time zone of the downstream using the `time-zone` parameter in `sink-uri`. You can first query the time zone used by MySQL:

1. Query the time zone used by MySQL:

```
show variables like '%time_zone%';
```

```
+------------------+--------+
| Variable_name  | Value |
+------------------+--------+
```

```
| system_time_zone | CST |
| time_zone        | SYSTEM |
+------------------+--------+
```

2. Specify the time zone when you create the replication task and create the TiCDC service:

```
cdc cli changefeed create --sink-uri="mysql://root@127.0.0.1:3306/?time
    ↪ -zone=CST" --pd=http://10.0.10.25:2379
```

> **Note:**
>
> CST might be an abbreviation for the following four different time zones:
>
> - Central Standard Time (USA) UT-6:00
> - Central Standard Time (Australia) UT+9:30
> - China Standard Time UT+8:00
> - Cuba Standard Time UT-4:00
>
> In China, CST usually stands for China Standard Time.

13.9.5.3 How do I handle the incompatibility issue of configuration files caused by TiCDC upgrade?

Refer to Notes for compatibility.

13.9.5.4 The `start-ts` timestamp of the TiCDC task is quite different from the current time. During the execution of this task, replication is interrupted and an error `[CDC:ErrBufferReachLimit]` occurs. What should I do?

Since v4.0.9, you can try to enable the unified sorter feature in your replication task, or use the BR tool for an incremental backup and restore, and then start the TiCDC replication task from a new time.

13.9.5.5 When the downstream of a changefeed is a database similar to MySQL and TiCDC executes a time-consuming DDL statement, all other changefeeds are blocked. What should I do?

1. Pause the execution of the changefeed that contains the time-consuming DDL statement. Then you can see that other changefeeds are no longer blocked.
2. Search for the `apply job` field in the TiCDC log and confirm the `start-ts` of the time-consuming DDL statement.
3. Manually execute the DDL statement in the downstream. After the execution finishes, go on performing the following operations.

4. Modify the changefeed configuration and add the above `start-ts` to the `ignore-txn-start-ts` configuration item.
5. Resume the paused changefeed.

13.9.5.6 After I upgrade the TiCDC cluster to v4.0.8, the `[CDC:ErrKafkaInvalidConfig]Canal requires old value to be enabled` error is reported when I execute a changefeed. What should I do?

Since v4.0.8, if the `canal-json` protocol is used for output in a changefeed, TiCDC enables the old value feature automatically. However, if you have upgraded TiCDC from an earlier version to v4.0.8 or later, when the changefeed uses the `canal-json` protocol and the old value feature is disabled, this error is reported.

To fix the error, take the following steps:

1. Set the value of `enable-old-value` in the changefeed configuration file to `true`.
2. Execute `cdc cli changefeed pause` to pause the replication task.

```
cdc cli changefeed pause -c test-cf --pd=http://10.0.10.25:2379
```

3. Execute `cdc cli changefeed update` to update the original changefeed configuration.

```
cdc cli changefeed update -c test-cf --pd=http://10.0.10.25:2379 --sink-uri="mysql://127.0.0.1:3306/?max-txn-row=20&worker-count=8" --config=changefeed.toml
```

4. Execute `cdc cli changfeed resume` to resume the replication task.

```
cdc cli changefeed resume -c test-cf --pd=http://10.0.10.25:2379
```

13.9.5.7 The `[tikv:9006]GC life time is shorter than transaction duration, transaction starts at xx, GC safe point is yy` error is reported when I use TiCDC to create a changefeed. What should I do?

You need to run the `pd-ctl service-gc-safepoint --pd <pd-addrs>` command to query the current GC safepoint and service GC safepoint. If the GC safepoint is smaller than the `start-ts` of the TiCDC replication task (changefeed), you can directly add the `--disable-gc-check` option to the `cdc cli create changefeed` command to create a changefeed.

If the result of `pd-ctl service-gc-safepoint --pd <pd-addrs>` does not have `gc_worker service_id`:

- If your PD version is v4.0.8 or earlier, refer to PD issue #3128 for details.
- If your PD is upgraded from v4.0.8 or an earlier version to a later version, refer to PD issue #3366 for details.

13.9.5.8 When I use TiCDC to replicate messages to Kafka, Kafka returns the `Message was too large` error. Why?

For TiCDC v4.0.8 or earlier versions, you cannot effectively control the size of the message output to Kafka only by configuring the `max-message-bytes` setting for Kafka in the Sink URI. To control the message size, you also need to increase the limit on the bytes of messages to be received by Kafka. To add such a limit, add the following configuration to the Kafka server configuration.

```
### The maximum byte number of a message that the broker receives
message.max.bytes=2147483648
### The maximum byte number of a message that the broker copies
replica.fetch.max.bytes=2147483648
### The maximum message byte number that the consumer side reads
fetch.message.max.bytes=2147483648
```

13.9.5.9 How can I find out whether a DDL statement fails to execute in downstream during TiCDC replication? How to resume the replication?

If a DDL statement fails to execute, the replication task (changefeed) automatically stops. The checkpoint-ts is the DDL statement's finish-ts minus one. If you want TiCDC to retry executing this statement in the downstream, use `cdc cli changefeed resume` to resume the replication task. For example:

```
cdc cli changefeed resume -c test-cf --pd=http://10.0.10.25:2379
```

If you want to skip this DDL statement that goes wrong, set the start-ts of the changefeed to the checkpoint-ts (the timestamp at which the DDL statement goes wrong) plus one, and then run the `cdc cli changefeed create` command to create a new changefeed task. For example, if the checkpoint-ts at which the DDL statement goes wrong is `415241823337054209`, run the following commands to skip this DDL statement:

```
cdc cli changefeed remove --pd=http://10.0.10.25:2379 --changefeed-id simple
    ↪ -replication-task
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
    ↪ root:123456@127.0.0.1:3306/" --changefeed-id="simple-replication-task
    ↪ " --sort-engine="unified" --start-ts 415241823337054210
```

13.9.5.10 TiCDC OpenAPI

TiCDC provides the OpenAPI feature for querying and operating the TiCDC cluster, which is similar to the feature of `cdc cli` tool.

You can use the APIs to perform the following maintenance operations on the TiCDC cluster:

- Get the status information of a TiCDC node

- Check the health status of a TiCDC cluster
- Create a replication task
- Remove a replication task
- Update the replication configuration
- Query the replication task list
- Query a specific replication task
- Pause a replication task
- Resume a replication task
- Query the replication subtask list
- Query a specific replication subtask
- Query the TiCDC service process list
- Evict an owner node
- Manually trigger the load balancing of all tables in a replication task
- Manually schedule a table to another node
- Dynamically adjust the log level of the TiCDC server

The request body and returned value of all APIs are in JSON format. The following sections describe the specific usage of the APIs.

In the following examples, the listening IP address of the TiCDC server is `127.0.0.1` and the port is `8300`. You can bind a specified IP and port via `--addr=ip:port` when starting the TiCDC server.

13.9.5.10.1 API error message template

After sending an API request, if an error occurs, the returned error message is in the following format:

```
{
    "error_msg": "",
    "error_code": ""
}
```

From the above JSON output, `error_msg` describes the error message and `error_code` is the corresponding error code.

13.9.5.10.2 Get the status information of a TiCDC node

This API is a synchronous interface. If the request is successful, the status information of the corresponding node is returned.

Request URI

`GET /api/v1/status`

Example

The following request gets the status information of the TiCDC node whose IP address is `127.0.0.1` and port number is `8300`.

```
curl -X GET http://127.0.0.1:8300/api/v1/status
```

```
{
    "version": "v5.2.0-master-dirty",
    "git_hash": "f191cd00c53fdf7a2b1c9308a355092f9bf8824e",
    "id": "c6a43c16-0717-45af-afd6-8b3e01e44f5d",
    "pid": 25432,
    "is_owner": true
}
```

The fields of the above output are described as follows:

- version: The current TiCDC version number.
- git_hash: The Git hash value.
- id: The capture ID of the node.
- pid: The capture process PID of the node.
- is_owner: Indicates whether the node is an owner.

13.9.5.10.3 Check the health status of a TiCDC cluster

This API is a synchronous interface. If the cluster is healthy, `200 OK` is returned.

Request URI

`GET /api/v1/health`

Example

```
curl -X GET http://127.0.0.1:8300/api/v1/health
```

13.9.5.10.4 Create a replication task

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

Request URI

`POST /api/v1/changefeeds`

Parameter description

Compared to the optional parameters for creating a replication task using the `cli` command, the optional parameters for creating such task using the API are not as complete. This API supports the following parameters.

Parameters for the request body

Parameter name | Description |
:———————— | :————————- ————————————- |

`changefeed_id` | `STRING` type. The ID of the replication task. (Optional) |
`start_ts` | `UINT64` type. Specifies the start TSO of the changefeed. (Optional) |
`target_ts` | `UINT64` type. Specifies the target TSO of the changefeed. (Optional) |
`sink_uri` | `STRING` type. The downstream address of the replication task. (**Required**) |
`force_replicate` | `BOOLEAN` type. Determines whether to forcibly replicate the tables without unique indexes. (Optional) |
`ignore_ineligible_table` | `BOOLEAN` type. Determines whether to ignore the tables that cannot be replicated. (Optional) |
`filter_rules` | `STRING` type array. The rules for table schema filtering. (Optional) |
`ignore_txn_start_ts` | `UINT64` type array. Ignores the transaction of a specified start_ts. (Optional) |
`mounter_worker_num` | `INT` type. The mounter thread number. (Optional) |
`sink_config` | The configuration parameters of sink. (Optional) |

The meaning and format of `changefeed_id`, `start_ts`, `target_ts`, and `sink_uri` are the same as those described in the Use `cdc cli` to create a replication task document. For the detailed description of these parameters, see this document. Note that when you specify the certificate path in `sink_uri`, make sure you have uploaded the corresponding certificate to the corresponding TiCDC server.

Some other parameters in the above table are described further as follows.

`force_replicate`: This parameter defaults to `false`. When it is specified as `true`, TiCDC tries to forcibly replicate tables that do not have a unique index.

`ignore_ineligible_table`: This parameter defaults to `false`. When it is specified as `true`, TiCDC ignores tables that cannot be replicated.

`filter_rules`: The rules for table schema filtering, such as `filter_rules = ['foo *.*','bar*.*']`. For details, see the Table Filter document.

`ignore_txn_start_ts`: When this parameter is specified, the specified start_ts is ignored. For example, `ignore-txn-start-ts = [1, 2]`.

`mounter_worker_num`: The thread number of mounter. Mounter is used to decode the data output from TiKV. The default value is `16`.

The configuration parameters of sink are as follows:

```
{
  "dispatchers":[
    {"matcher":["test1.*", "test2.*"], "dispatcher":"ts"},
    {"matcher":["test3.*", "test4.*"], "dispatcher":"rowid"}
  ],
  "protocal":"canal-json"
}
```

`dispatchers`: For the sink of MQ type, you can use dispatchers to configure the event dispatcher. Four dispatchers are supported: `default`, `ts`, `rowid`, and `table`. The dispatcher rules are as follows:

- `default`: When multiple unique indexes (including the primary key) exist or the Old Value feature is enabled, events are dispatched in the `table` mode. When only one unique index (or the primary key) exists, events are dispatched in the `rowid` mode.
- `ts`: Uses the commitTs of the row change to create the hash value and dispatch events.
- `rowid`: Uses the name and value of the selected HandleKey column to create the hash value and dispatch events.
- `table`: Uses the schema name of the table and the table name to create the hash value and dispatch events.

`matcher`: The matching syntax of matcher is the same as the filter rule syntax.

`protocol`: For the sink of MQ type, you can specify the protocol format of the message. Currently the following protocols are supported: `canal-json`, `open-protocol`, and `avro`.

Example

The following request creates a replication task with an ID of `test5` and a `sink_uri` of `blackhome://`.

```
curl -X POST -H "'Content-type':'application/json'" http://127.0.0.1:8300/
    ↪ api/v1/changefeeds -d '{"changefeed_id":"test5","sink_uri":"blackhole
    ↪ ://"}'
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.5 Remove a replication task

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

Request URI

`DELETE /api/v1/changefeeds/{changefeed_id}`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The ID of the replication task (changefeed) to be removed.

Example

The following request removes the replication task with the ID `test1`.

```
curl -X DELETE http://127.0.0.1:8300/api/v1/changefeeds/test1
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.6 Update the replication configuration

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

To modify the changefeed configuration, follow the steps of `pause the replication task -> modify the configuration -> resume the replication task`.

Request URI

`PUT /api/v1/changefeeds/{changefeed_id}`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The ID of the replication task (changefeed) to be updated.

Parameters for the request body

Currently, only the following configuration can be modified via the API.

Parameter name | Description |
:——————— | :——————— ——————— |
`target_ts` | `UINT64` type. Specifies the target TSO of the changefeed. (Optional) |
`sink_uri` | `STRING` type. The downstream address of the replication task. (Optional) |
`filter_rules` | `STRING` type array. The rules for table schema filtering. (Optional) |
`ignore_txn_start_ts` | `UINT64` type array. Ignores the transaction of a specified start_ts. (Optional) |
`mounter_worker_num` | `INT` type. The mounter thread number. (Optional) |
`sink_config` | The configuration parameters of sink. (Optional) |

The meanings of the above parameters are the same as those in the Create a replication task section. See that section for details.

Example

The following request updates the `mounter_worker_num` of the replication task with the ID `test1` to 32.

```
curl -X PUT -H "'Content-type':'application/json'" http://127.0.0.1:8300/
    ↪ api/v1/changefeeds/test1 -d '{"mounter_worker_num":32}'
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.7 Query the replication task list

This API is a synchronous interface. If the request is successful, the basic information of all nodes in the TiCDC cluster is returned.

Request URI

`GET /api/v1/changefeeds`

Parameter description

Query parameters

Parameter name | Description |
:—— | :————————————- —— |
`state` | When this parameter is specified, the replication status information only of this state is returned.(Optional) |

The value options for `state` are `all`, `normal`, `stopped`, `error`, `failed`, and `finished`.

If this parameter is not specified, the basic information of replication tasks whose state is normal, stopped, or failed is returned by default.

Example

The following request queries the basic information of all replication tasks whose state is `normal`.

```
curl -X GET http://127.0.0.1:8300/api/v1/changefeeds?state=normal
```

```
[
    {
        "id": "test1",
        "state": "normal",
        "checkpoint_tso": 426921294362574849,
        "checkpoint_time": "2021-08-10 14:04:54.242",
        "error": null
    },
    {
        "id": "test2",
        "state": "normal",
        "checkpoint_tso": 426921294362574849,
        "checkpoint_time": "2021-08-10 14:04:54.242",
        "error": null
    }
]
```

The fields in the returned result above are described as follows:

- id: The ID of the replication task.
- state: The current state of the replication task.

- checkpoint_tso: The TSO representation of the current checkpoint of the replication task.
- checkpoint_time: The formatted time representation of the current checkpoint of the replication task.
- error: The error information of the replication task.

13.9.5.10.8 Query a specific replication task

This API is a synchronous interface. If the request is successful, the detailed information of the specified replication task is returned.

Request URI

`GET /api/v1/changefeeds/{changefeed_id}`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The ID of the replication task (changefeed) to be queried.

Example

The following request queries the detailed information of the replication task with the ID `test1`.

```
curl -X GET http://127.0.0.1:8300/api/v1/changefeeds/test1
```


```
{
    "id": "test1",
    "sink_uri": "blackhole://",
    "create_time": "2021-08-10 11:41:30.642",
    "start_ts": 426919038970232833,
    "target_ts": 0,
    "checkpoint_tso": 426921014615867393,
    "checkpoint_time": "2021-08-10 13:47:07.093",
    "sort_engine": "unified",
    "state": "normal",
    "error": null,
    "error_history": null,
    "creator_version": "",
    "task_status": [
        {
            "capture_id": "d8924259-f52f-4dfb-97a9-c48d26395945",
            "table_ids": [
                63,
                65
```

```
            ],
            "table_operations": {}
        }
    ]
}
```

13.9.5.10.9 Pause a replication task

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

Request URI

`POST /api/v1/changefeeds/{changefeed_id}/pause`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The ID of the replication task (changefeed) to be paused.

Example

The following request pauses the replication task with the ID `test1`.

```
curl -X POST http://127.0.0.1:8300/api/v1/changefeeds/test1/pause
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.10 Resume a replication task

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

Request URI

`POST /api/v1/changefeeds/{changefeed_id}/resume`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The ID of the replication task (changefeed) to be resumed.

Example

The following request resumes the replication task with the ID `test1`.

```
curl -X POST http://127.0.0.1:8300/api/v1/changefeeds/test1/resume
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.11 Query the replication subtask list

This API is a synchronous interface. If the request is successful, the basic information of all replication subtasks (`processor`) is returned.

Request URI

`GET /api/v1/processors`

Example

```
curl -X GET http://127.0.0.1:8300/api/v1/processors
```

```
[
    {
        "changefeed_id": "test1",
        "capture_id": "561c3784-77f0-4863-ad52-65a3436db6af"
    }
]
```

13.9.5.10.12 Query a specific replication subtask

This API is a synchronous interface. If the request is successful, the detailed information of the specified replication subtask (`processor`) is returned.

Request URI

`GET /api/v1/processors/{changefeed_id}/{capture_id}`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The changefeed ID of the replication subtask to be queried.
`capture_id`	The capture ID of the replication subtask to be queried.

Example

The following request queries the detailed information of a subtask whose `changefeed_id` is `test` and `capture_id` is `561c3784-77f0-4863-ad52-65a3436db6af`. A subtask can be

indentifed by `changefeed_id` and `capture_id`.

```
curl -X GET http://127.0.0.1:8300/api/v1/processors/test1/561c3784-77f0
    ↪ -4863-ad52-65a3436db6af
```

```
{
    "checkpoint_ts": 426919123303006208,
    "resolved_ts": 426919123369066496,
    "table_ids": [
        63,
        65
    ],
    "error": null
}
```

13.9.5.10.13 Query the TiCDC service process list

This API is a synchronous interface. If the request is successful, the basic information of all replication processes (`capture`) is returned.

Request URI

`GET /api/v1/captures`

Example

```
curl -X GET http://127.0.0.1:8300/api/v1/captures
```

```
[
    {
        "id": "561c3784-77f0-4863-ad52-65a3436db6af",
        "is_owner": true,
        "address": "127.0.0.1:8300"
    }
]
```

13.9.5.10.14 Evict an owner node

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

Request URI

`POST /api/v1/owner/resign`

Example

The following request evicts the current owner node of TiCDC and triggers a new round of elections to generate a new owner node.

```
curl -X POST http://127.0.0.1:8300/api/v1/owner/resign
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.15 Manually trigger the load balancing of all tables in a replication task

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

Request URI

`POST /api/v1/changefeeds/{changefeed_id}/tables/rebalance_table`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The ID of the replication task (changefeed) to be scheduled.

Example

The following request triggers the load balancing of all tables in the changefeed with the ID `test1`.

```
curl -X POST http://127.0.0.1:8300/api/v1/changefeeds/test1/tables/
    ↪ rebalance_table
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.16 Manually schedule a table to another node

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

Request URI

`POST /api/v1/changefeeds/{changefeed_id}/tables/move_table`

Parameter description

Path parameters

Parameter name	Description
`changefeed_id`	The ID of the replication task (changefeed) to be scheduled.

Parameters for the request body

Parameter name	Description
`target_capture_id`	The ID of the target capture.
`table_id`	The ID of the table to be scheduled.

Example

The following request schedules the table with the ID `49` in the changefeed with the ID `test1` to the capture with the ID `6f19a6d9-0f8c-4dc9-b299-3ba7c0f216f5`.

```
curl -X POST -H "'Content-type':'application/json'" http://127.0.0.1:8300/
    ↪ api/v1/changefeeds/changefeed-test1/tables/move_table -d '{"
    ↪ capture_id":"6f19a6d9-0f8c-4dc9-b299-3ba7c0f216f5","table_id":49}'
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

13.9.5.10.17 Dynamically adjust the log level of the TiCDC server

This API is a synchronous interface. If the request is successful, `202 OK` is returned.

Request URI

`POST /api/v1/log`

Request parameters

Parameters for the request body

Parameter name	Description
`log_level`	The log level you want to set.

`log_level` supports the log levels provided by zap: “debug”, “info”, “warn”, “error”, “dpanic” , “panic”, and “fatal”.

Example

```
curl -X POST -H "'Content-type':'application/json'" http://127.0.0.1:8300/
    ↪ api/v1/log -d '{"log_level":"debug"}'
```

If the request is successful, `202 OK` is returned. If the request fails, an error message and error code are returned.

13.9.5.11 TiCDC Open Protocol

TiCDC Open Protocol is a row-level data change notification protocol that provides data sources for monitoring, caching, full-text indexing, analysis engines, and primary-secondary replication between different databases. TiCDC complies with TiCDC Open Protocol and replicates data changes of TiDB to third-party data medium such as MQ (Message Queue).

TiCDC Open Protocol uses Event as the basic unit to replicate data change events to the downstream. The Event is divided into three categories:

- Row Changed Event: Represents the data change in a row. When a row is changed, this Event is sent and contains information about the changed row.
- DDL Event: Represents the DDL change. This Event is sent after a DDL statement is successfully executed in the upstream. The DDL Event is broadcasted to every MQ Partition.
- Resolved Event: Represents a special time point before which the Event received is complete.

13.9.5.11.1 Restrictions

- In most cases, the Row Changed Event of a version is sent only once, but in special situations such as node failure and network partition, the Row Changed Event of the same version might be sent multiple times.
- On the same table, the Row Changed Events of each version which is first sent are incremented in the order of timestamps (TS) in the Event stream.
- Resolved Events are periodically broadcasted to each MQ Partition. The Resolved Event means that any Event with a TS earlier than Resolved Event TS has been sent to the downstream.
- DDL Events are broadcasted to each MQ Partition.
- Multiple Row Changed Events of a row are sent to the same MQ Partition.

13.9.5.11.2 Message format

A Message contains one or more Events, arranged in the following format:

Key:

Offset(Byte)	0~7	8~15	16~(15+length1)	...	...
Parameter	Protocol version	Length1	Event Key1	LengthN	Event KeyN

Value:

Offset(Byte)	0~7	8~(7+length1)	...	...
Parameter	Length1	Event Value1	LengthN	Event ValueN

- `LengthN` represents the length of the `N`th key/value.
- The length and protocol version are the big-endian `int64` type.
- The version of the current protocol is `1`.

13.9.5.11.3 Event format

This section introduces the formats of Row Changed Event, DDL Event, and Resolved Event.

Row Changed Event

- **Key:**

```
{
    "ts":<TS>,
    "scm":<Schema Name>,
    "tbl":<Table Name>,
    "t":1
}
```

Parameter	Type	Description
TS	Number	The timestamp of the transaction that causes the row change.
Schema Name	String	The name of the schema where the row is in.
Table Name	String	The name of the table where the row is in.

- **Value:**

`Insert` event. The newly added row data is output.

```
{
    "u":{
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        },
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        }
    }
}
```

`Update` event. The newly added row data ("u") and the row data before the update ("p") are output. The latter ("p") is output only when the old value feature is enabled.

```
{
    "u":{
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        },
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        }
    },
    "p":{
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        },
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        }
    }
}
```

`Delete` event. The deleted row data is output. When the old value feature is enabled, the `Delete` event includes all the columns of the deleted row data; when this feature is disabled, the `Delete` event only includes the HandleKey column.

```
{
    "d":{
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        },
```

```
        <Column Name>:{
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        }
    }
}
```

Parameter	Type	Description
Column Name	String	The column name.
Column Type	Number	The column type. For details, see Column Type Code.
Where Handle	Boolean	Determines whether this column can be the filter condition of the `Where` clause. When this column is unique on the table, `Where Handle` is `true`.
Flag	Number	The bit flags of columns. For details, see Bit flags of columns.
Column Value	Any	The Column value.

DDL Event

- **Key:**

```
{
    "ts":<TS>,
    "scm":<Schema Name>,
    "tbl":<Table Name>,
    "t":2
}
```

Parameter	Type	Description
TS	Number	The timestamp of the transaction that performs the DDL change.
Schema Name	String	The schema name of the DDL change, which might be an empty string.
Table Name	String	The table name of the DDL change, which might be am empty string.

- **Value:**

```
{
    "q":<DDL Query>,
    "t":<DDL Type>
}
```

Parameter	Type	Description
DDL Query	String	DDL Query SQL
DDL Type	String	The DDL type. For details, see DDL Type Code.

Resolved Event

- **Key:**

```
{
    "ts":<TS>,
    "t":3
}
```

Parameter	Type	Description
TS	Number	The Resolved timestamp. Any TS earlier than this Event has been sent.

- **Value:** None

13.9.5.11.4 Examples of the Event stream output

This section shows and displays the output logs of the Event stream.

Suppose that you execute the following SQL statement in the upstream and the MQ Partition number is 2:

```
CREATE TABLE test.t1(id int primary key, val varchar(16));
```

From the following Log 1 and Log 3, you can see that the DDL Event is broadcasted to all MQ Partitions, and that the Resolved Event is periodically broadcasted to each MQ Partition.

```
1. [partition=0] [key="{\"ts\":415508856908021766,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":2}"] [value="{\"q\":\"CREATE TABLE test.t1(id int
   ↪ primary key, val varchar(16))\",\"t\":3}"]
2. [partition=0] [key="{\"ts\":415508856908021766,\"t\":3}"] [value=]
3. [partition=1] [key="{\"ts\":415508856908021766,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":2}"] [value="{\"q\":\"CREATE TABLE test.t1(id int
   ↪ primary key, val varchar(16))\",\"t\":3}"]
4. [partition=1] [key="{\"ts\":415508856908021766,\"t\":3}"] [value=]
```

Execute the following SQL statements in the upstream:

```
BEGIN;
INSERT INTO test.t1(id, val) VALUES (1, 'aa');
INSERT INTO test.t1(id, val) VALUES (2, 'aa');
UPDATE test.t1 SET val = 'bb' WHERE id = 2;
INSERT INTO test.t1(id, val) VALUES (3, 'cc');
COMMIT;
```

- From the following Log 5 and Log 6, you can see that Row Changed Events on the same table might be sent to different partitions based on the primary key, but changes to the same row are sent to the same partition so that the downstream can easily process the Event concurrently.
- From Log 6, multiple changes to the same row in a transaction are only sent in one Row Changed Event.
- Log 8 is a repeated event of Log 7. Row Changed Event might be repeated, but the first Event of each version is sent orderly.

```
5. [partition=0] [key="{\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":1},\"val\":{\"t\":15,\"v\":\"YWE=\"}}}"]
6. [partition=1] [key="{\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":2},\"val\":{\"t\":15,\"v\":\"YmI=\"}}}"]
7. [partition=0] [key="{\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":3},\"val\":{\"t\":15,\"v\":\"Y2M=\"}}}"]
8. [partition=0] [key="{\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":3},\"val\":{\"t\":15,\"v\":\"Y2M=\"}}}"]
```

Execute the following SQL statements in the upstream:

```
BEGIN;
DELETE FROM test.t1 WHERE id = 1;
UPDATE test.t1 SET val = 'dd' WHERE id = 3;
UPDATE test.t1 SET id = 4, val = 'ee' WHERE id = 2;
COMMIT;
```

- Log 9 is the Row Changed Event of the `Delete` type. This type of Event only contains primary key columns or unique index columns.
- Log 13 and Log 14 are Resolved Events. The Resolved Event means that in this Partition, any events smaller than the Resolved TS (including Row Changed Event and DDL Event) have been sent.

```
9. [partition=0] [key="{\"ts\":415508881418485761,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"d\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":1}}}"]
10. [partition=1] [key="{\"ts\":415508881418485761,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"d\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":2}}}"]
11. [partition=0] [key="{\"ts\":415508881418485761,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":3},\"val\":{\"t\":15,\"v\":\"ZGQ=\"}}}"]
12. [partition=0] [key="{\"ts\":415508881418485761,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":4},\"val\":{\"t\":15,\"v\":\"ZWU=\"}}}"]
13. [partition=0] [key="{\"ts\":415508881038376963,\"t\":3}"] [value=]
14. [partition=1] [key="{\"ts\":415508881038376963,\"t\":3}"] [value=]
```

13.9.5.11.5 Protocol parsing for consumers

Currently, TiCDC does not provide the standard parsing library for TiCDC Open Protocol, but the Golang version and Java version of parsing examples are provided. You can refer to the data format provided in this document and the following examples to implement the protocol parsing for consumers.

- Golang examples
- Java examples

13.9.5.11.6 Column type code

`Column Type Code` represents the column data type of the Row Changed Event.

Type	Code	Output Example	Description
TINYINT/BOOLEAN	1	{“t”:1,“v”:1}	
SMALLINT	2	{“t”:2,“v”:1}	
INT	3	{“t”:3,“v”:123}	
FLOAT	4	{“t”:4,“v”:153.123}	
DOUBLE	5	{“t”:5,“v”:153.123}	
NULL	6	{“t”:6,“v”:null}	
TIMESTAMP	7	{“t”:7,“v”:“1973-12-30 15:30:00”}	
BIGINT	8	{“t”:8,“v”:123}	
MEDIUMINT	9	{“t”:9,“v”:123}	
DATE	10/14	{“t”:10,“v”:“2000-01-01”}	
TIME	11	{“t”:11,“v”:“23:59:59”}	
DATETIME	12	{“t”:12,“v”:“2015-12-20 23:58:58”}	
YEAR	13	{“t”:13,“v”:1970}	

Type	Code	Output Example	Description
VARCHAR/VARBINARY	15/253	{“t”:15,“v”:“test”} / {“t”:15,“v”:“\\x89PNG\\r\\n\\x1a\\n”}	The value is encoded in UTF-8. When the upstream type is VARBINARY, invisible characters are escaped.
BIT	16	{“t”:16,“v”:81}	
JSON	245	{“t”:245,“v”:“{\”key1\“:\”value1\“}”}	
DECIMAL	246	{“t”:246,“v”:“129012.1230000”}	
ENUM	247	{“t”:247,“v”:1}	
SET	248	{“t”:248,“v”:3}	
TINYTEXT/TINYBLOB	249	{“t”:249,“v”:“Tl65rWL6K+VdGV4dA==”}	The value is encoded in Base64.

Type	Code	Output Example	Description
MEDIUMTEXT/MEDIUMBLOB	250	{“t”:250,“v”:“Th5rWL6K+VdGV4dA==”}	The value is encoded in Base64.
LONGTEXT/LONGBLOB	251	{“t”:251,“v”:“Th5rWL6K+VdGV4dA==”}	The value is encoded in Base64.
TEXT/BLOB	252	{“t”:252,“v”:“Th5rWL6K+VdGV4dA==”}	The value is encoded in Base64.

Type	Code	Output Example	Description
CHAR/BINARY	254	{“t”:254,“v”:“Test”} / {“t”:254,“v”:“\\x89PNG\\r\\n\\x1a\\n”}	The value encoded in UTF-8. When the upstream type is BINARY, invisible characters are escaped.
GEOMETRY	255		Unsupported

13.9.5.11.7 DDL Type Code

`DDL Type Code` represents the DDL statement type of the DDL Event.

Type	Code
Create Schema	1
Drop Schema	2
Create Table	3
Drop Table	4
Add Column	5
Drop Column	6
Add Index	7
Drop Index	8
Add Foreign Key	9

Type	Code
Drop Foreign Key	10
Truncate Table	11
Modify Column	12
Rebase Auto ID	13
Rename Table	14
Set Default Value	15
Shard RowID	16
Modify Table Comment	17
Rename Index	18
Add Table Partition	19
Drop Table Partition	20
Create View	21
Modify Table Charset And Collate	22
Truncate Table Partition	23
Drop View	24
Recover Table	25
Modify Schema Charset And Collate	26
Lock Table	27
Unlock Table	28
Repair Table	29
Set TiFlash Replica	30
Update TiFlash Replica Status	31
Add Primary Key	32
Drop Primary Key	33
Create Sequence	34
Alter Sequence	35
Drop Sequence	36

13.9.5.11.8 Bit flags of columns

The bit flags represent specific attributes of columns.

Bit	Value	Name	Description
1	0x01	BinaryFlag	Whether the column is a binary-encoded column.
2	0x02	HandleKeyFlag	Whether the column is a Handle index column.
3	0x04	GeneratedColumnFlag	Whether the column is a generated column.
4	0x08	PrimaryKeyFlag	Whether the column is a primary key column.
5	0x10	UniqueKeyFlag	Whether the column is a unique index column.
6	0x20	MultipleKeyFlag	Whether the column is a composite index column.
7	0x40	NullableFlag	Whether the column is a nullable column.
8	0x80	UnsignedFlag	Whether the column is an unsigned column.

Example:

If the value of a column flag is `85`, the column is a nullable column, a unique index column, a generated column, and a binary-encoded column.

```
85 == 0b_101_0101
   == NullableFlag | UniqueKeyFlag | GeneratedColumnFlag | BinaryFlag
```

If the value of a column is `46`, the column is a composite index column, a primary key column, a generated column, and a Handle key column.

```
46 == 0b_010_1110
   == MultipleKeyFlag | PrimaryKeyFlag | GeneratedColumnFlag | HandleKeyFlag
```

Note:

- `BinaryFlag` is meaningful only when the column type is BLOB/TEXT (including TINYBLOB/TINYTEXT and BINARY/CHAR). When the upstream column is the BLOB type, the `BinaryFlag` value is set to `1`. When the upstream column is the TEXT type, the `BinaryFlag` value is set to `0`.
- To replicate a table from the upstream, TiCDC selects a valid index as the Handle index. The `HandleKeyFlag` value of the Handle index column is set to `1`.

13.9.5.12 TiCDC Avro Protocol

Avro is a data exchange format protocol defined by Apache Avro™ and chosen by Confluent Platform as the default data exchange format. This document describes the implementation of the Avro data format in TiCDC, including TiDB extension fields, definition of the Avro data format, and the interaction between Avro and Confluent Schema Registry.

13.9.5.12.1 Use Avro

When using Message Queue (MQ) as a downstream sink, you can specify Avro in `sink-uri`. TiCDC captures TiDB DML events, creates Avro messages from these events, and sends the messages downstream. When Avro detects a schema change, it registers the latest schema with Schema Registry.

The following is a configuration example using Avro:

```
cdc cli changefeed create --pd=http://127.0.0.1:2379 --changefeed-id="kafka-
    ↪ avro" --sink-uri="kafka://127.0.0.1:9092/topic-name?protocol=avro" --
    ↪ schema-registry=http://127.0.0.1:8081 --config changefeed_config.toml
```

```
[sink]
dispatchers = [
 {matcher = ['*.*'], topic = "tidb_{schema}_{table}"},
]
```

The value of `--schema-registry` supports the `https` protocol and `username:password` authentication, for example, `--schema-registry=https://username:password@schema-registry-uri.com`. The username and password must be URL-encoded.

13.9.5.12.2 TiDB extension fields

By default, Avro only collects data of changed rows in DML events and does not collect the type of data changes or TiDB-specific CommitTS (the unique identifiers of transactions). To address this issue, TiCDC introduces the following three TiDB extension fields to the Avro protocol message. When `enable-tidb-extension` is set to `true` (`false` by default) in `sink-uri`, TiCDC adds these three fields to the Avro messages during message generation.

- `_tidb_op`: The DML type. "c" indicates insert and "u" indicates updates.
- `_tidb_commit_ts`: The unique identifier of a transaction.
- `_tidb_commit_physical_time`: The physical timestamp in a transaction identifier.

The following is a configuration example:

```
cdc cli changefeed create --pd=http://127.0.0.1:2379 --changefeed-id="kafka-avro-enable-extension" --sink-uri="kafka://127.0.0.1:9092/topic-name?protocol=avro&enable-tidb-extension=true" --schema-registry=http://127.0.0.1:8081 --config changefeed_config.toml
```

```
[sink]
dispatchers = [
 {matcher = ['*.*'], topic = "tidb_{schema}_{table}"},
]
```

13.9.5.12.3 Definition of the data format

TiCDC converts a DML event into a Kafka event, and the Key and Value of an event are encoded according to the Avro protocol.

Key data format

```
{
    "name":"{{TableName}}",
    "namespace":"{{Namespace}}",
    "type":"record",
    "fields":[
```

```
        {{ColumnValueBlock}},
        {{ColumnValueBlock}},
    ]
}
```

- `{{TableName}}` indicates the name of the table where the event occurs.
- `{{Namespace}}` is the namespace of Avro.
- `{{ColumnValueBlock}}` defines the format of each column of data.

The `fields` in the key contains only primary key columns or unique index columns.

Value data format

```
{
    "name":"{{TableName}}",
    "namespace":"{{Namespace}}",
    "type":"record",
    "fields":[
        {{ColumnValueBlock}},
        {{ColumnValueBlock}},
    ]
}
```

The data format of Value is the same as that of Key, by default. However, `fields` in the Value contains all columns, not just the primary key columns.

Note:

The Avro protocol encodes DML events as follows:

- For Delete events, Avro only encodes the Key part. The Value part is empty.
- For Insert events, Avro encodes all column data to the Value part.
- For Update events, Avro encodes only all column data that is updated to the Value part.

The Avro protocol does not encode the old values for Update and Delete events. Additionally, to be compatible with most Confluent sink connectors that rely on `null` records to identify deletions (`delete.on.null`), Delete events do not include extension information, such as `_tidb_commit_ts`, even when `enable-tidb-extension` is enabled. If you need these features, consider using other protocols such as Canal-JSON or Debezium.

After you enable `enable-tidb-extension`, the data format of the Value will be as follows:

```
{
    "name":"{{TableName}}",
    "namespace":"{{Namespace}}",
    "type":"record",
    "fields":[
        {{ColumnValueBlock}},
        {{ColumnValueBlock}},
        {
            "name":"_tidb_op",
            "type":"string"
        },
        {
            "name":"_tidb_commit_ts",
            "type":"long"
        },
        {
            "name":"_tidb_commit_physical_time",
            "type":"long"
        }
    ]
}
```

Compared with the Value data format with `enable-tidb-extension` disabled, three new fields are added: `_tidb_op`, `_tidb_commit_ts`, and `_tidb_commit_physical_time`.

Column data format

The Column data is the `{{ColumnValueBlock}}` part of the Key/Value data format. TiCDC generates the Column data format based on the SQL Type. The basic Column data format is as follows:

```
{
    "name":"{{ColumnName}}",
    "type":{
        "connect.parameters":{
            "tidb_type":"{{TIDB_TYPE}}"
        },
        "type":"{{AVRO_TYPE}}"
    }
}
```

If one column can be NULL, the Column data format can be:

```
{
    "default":null,
```

```
    "name":"{{ColumnName}}",
    "type":[
        "null",
        {
            "connect.parameters":{
                "tidb_type":"{{TIDB_TYPE}}"
            },
            "type":"{{AVRO_TYPE}}"
        }
    ]
}
```

- `{{ColumnName}}` indicates the column name.
- `{{TIDB_TYPE}}` indicates the type in TiDB, which is not a one-to-one mapping with the SQL type.
- `{{AVRO_TYPE}}` indicates the type in avro spec.

SQL TYPE	TIDB_TYPE	AVRO_TYPE	Description
BOOL	INT	int	
TINYINT	INT	int	When it is unsigned, TIDB_TYPE is INT UNSIGNED.
SMALLINT	INT	int	When it is unsigned, TIDB_TYPE is INT UNSIGNED.
MEDIUMINT	INT	int	When it is unsigned, TIDB_TYPE is INT UNSIGNED.
INT	INT	int	When it is unsigned, TIDB_TYPE is INT UNSIGNED and AVRO_TYPE is long.
BIGINT	BIGINT	long	When it is unsigned, TIDB_TYPE is BIGINT UNSIGNED. If `avro-bigint-unsigned-handling-mode` is string, AVRO_TYPE is string.
TINYBLOB	BLOB	bytes	
BLOB	BLOB	bytes	
MEDIUMBLOB	BLOB	bytes	
LONGBLOB	BLOB	bytes	
BINARY	BLOB	bytes	
VARBINARY	BLOB	bytes	
TINYTEXT	TEXT	string	
TEXT	TEXT	string	
MEDIUMTEXT	TEXT	string	
LONGTEXT	TEXT	string	
CHAR	TEXT	string	
VARCHAR	TEXT	string	
FLOAT	FLOAT	double	
DOUBLE	DOUBLE	double	
DATE	DATE	string	

SQL TYPE	TIDB_TYPE	AVRO_TYPE	Description
DATETIME	DATETIME	string	
TIMESTAMP	TIMESTAMP	string	
TIME	TIME	string	
YEAR	YEAR	int	
BIT	BIT	bytes	
JSON	JSON	string	
ENUM	ENUM	string	
SET	SET	string	
DECIMAL	DECIMAL	bytes	When `avro-decimal-handling-mode` is string, AVRO_TYPE is string.

In the Avro protocol, two other `sink-uri` parameters might affect the Column data format as well: `avro-decimal-handling-mode` and `avro-bigint-unsigned-handling-mode`.

- `avro-decimal-handling-mode` controls how Avro handles decimal fields, including:
  - string: Avro handles decimal fields as strings.
  - precise: Avro handles decimal fields as bytes.
- `avro-bigint-unsigned-handling-mode` controls how Avro handles BIGINT UNSIGNED fields, including:
  - string: Avro handles BIGINT UNSIGNED fields as strings.
  - long: Avro handles BIGINT UNSIGNED fields as 64-bit signed integers. When the value is greater than 9223372036854775807, overflow will occur.

The following is a configuration example:

```
cdc cli changefeed create --pd=http://127.0.0.1:2379 --changefeed-id="kafka-
    ↪ avro-string-option" --sink-uri="kafka://127.0.0.1:9092/topic-name?
    ↪ protocol=avro&avro-decimal-handling-mode=string&avro-bigint-unsigned-
    ↪ handling-mode=string" --schema-registry=http://127.0.0.1:8081 --
    ↪ config changefeed_config.toml
```

```
[sink]
dispatchers = [
 {matcher = ['*.*'], topic = "tidb_{schema}_{table}"},
]
```

Most SQL types are mapped to the base Column data format. Some other SQL types extend the base data format to provide more information.

BIT(64)

```
{
    "name":"{{ColumnName}}",
    "type":{
        "connect.parameters":{
            "tidb_type":"BIT",
            "length":"64"
        },
        "type":"bytes"
    }
}
```

ENUM/SET(a,b,c)

```
{
    "name":"{{ColumnName}}",
    "type":{
        "connect.parameters":{
            "tidb_type":"ENUM/SET",
            "allowed":"a,b,c"
        },
        "type":"string"
    }
}
```

DECIMAL(10, 4)

```
{
    "name":"{{ColumnName}}",
    "type":{
        "connect.parameters":{
            "tidb_type":"DECIMAL",
        },
        "logicalType":"decimal",
        "precision":10,
        "scale":4,
        "type":"bytes"
    }
}
```

13.9.5.12.4 DDL events and schema changes

Avro does not generate DDL events downstream. It checks whether a schema changes each time a DML event occurs. If a schema changes, Avro generates a new schema and registers it with the Schema Registry. If the schema change does not pass the compatibility check, the registration fails. TiCDC does not resolve any schema compatibility issues.

Note that, even if a schema change passes the compatibility check and a new version is registered, the data producers and consumers still need to perform an upgrade to ensure normal running of the system.

Assume that the default compatibility policy of Confluent Schema Registry is `BACKWARD` and add a non-empty column to the source table. In this situation, Avro generates a new schema but fails to register it with Schema Registry due to compatibility issues. At this time, the changefeed enters an error state.

For more information about schemas, refer to Schema Registry related documents.

13.9.5.12.5 Topic distribution

Schema Registry supports three Subject Name Strategies: TopicNameStrategy, RecordNameStrategy, and TopicRecordNameStrategy. Currently, TiCDC Avro only supports TopicNameStrategy, which means that a Kafka topic can only receive data in one data format. Therefore, TiCDC Avro prohibits mapping multiple tables to the same topic. When you create a changefeed, an error will be reported if the topic rule does not include the `{schema}` and `{table}` placeholders in the configured distribution rule.

13.9.5.12.6 Compatibility

When upgrading the TiCDC cluster to v6.1.6 or a later v6.1.x version, if a table replicated using Avro contains the `FLOAT` data type, you need to manually adjust the compatibility policy of Confluent Schema Registry to `None` before upgrading so that the changefeed can successfully update the schema. Otherwise, after upgrading, the changefeed will be unable to update the schema and enter an error state. For more information, see #8490.

13.9.5.13 TiCDC Canal-JSON Protocol

Canal-JSON is a data exchange format protocol defined by Alibaba Canal. In this document, you can learn how Canal-JSON data formats are implemented in TiCDC, including the TiDB extension field, the definitions of the Canal-JSON data formats, and comparison with the official Canal.

13.9.5.13.1 Use Canal-JSON

When using Message Queue (MQ) as the downstream Sink, you can specify Canal-JSON in `sink-uri`. TiCDC wraps and constructs Canal-JSON messages with Event as the basic unit, and sends TiDB data change Events to the downstream.

There are three types of Events:

- DDL Event: Represents a DDL change record. It is sent after an upstream DDL statement is successfully executed. The DDL Event is sent to the MQ Partition with the index being 0.

- DML Event: Represents a row data change record. This type of Event is sent when a row change occurs. It contains the information about the row after the change occurs.
- WATERMARK Event: Represents a special time point. It indicates that the Events received before this point is complete. It applies only to the TiDB extension field and takes effect when you set `enable-tidb-extension` to `true` in `sink-uri`.

The following is an example of using `Canal-JSON`:

```
cdc cli changefeed create --pd=http://127.0.0.1:2379 --changefeed-id="kafka-
    ↪ canal-json" --sink-uri="kafka://127.0.0.1:9092/topic-name?kafka-
    ↪ version=2.4.0&protocol=canal-json"
```

13.9.5.13.2 TiDB extension field

The Canal-JSON protocol is originally designed for MySQL. It does not contain important fields such as the TiDB-specific unique identifier for the CommitTS transaction. To solve this problem, TiCDC appends a TiDB extension field to the Canal-JSON protocol format. After you set `enable-tidb-extension` to `true` (`false` by default) in `sink-uri`, TiCDC behaves as follows when generating Canal-JSON messages:

- TiCDC sends DML Event and DDL Event messages that contain a field named `_tidb`.
- TiCDC sends WATERMARK Event messages.

The following is an example:

```
cdc cli changefeed create --pd=http://127.0.0.1:2379 --changefeed-id="kafka-
    ↪ canal-json-enable-tidb-extension" --sink-uri="kafka://127.0.0.1:9092/
    ↪ topic-name?kafka-version=2.4.0&protocol=canal-json&enable-tidb-
    ↪ extension=true"
```

13.9.5.13.3 Definitions of message formats

This section describes the formats of DDL Event, DML Event and WATERMARK Event, and how the data is resolved on the consumer side.

DDL Event

TiCDC encodes a DDL Event into the following Canal-JSON format.

```
{
    "id": 0,
    "database": "test",
    "table": "",
    "pkNames": null,
    "isDdl": true,
    "type": "QUERY",
```

```
    "es": 1639633094670,
    "ts": 1639633095489,
    "sql": "drop database if exists test",
    "sqlType": null,
    "mysqlType": null,
    "data": null,
    "old": null,
    "_tidb": {   // TiDB extension field
        "commitTs": 429918007904436226 // A TiDB TSO timestamp
    }
}
```

The fields are explained as follows.

Field	Type	Description
id	Number	The default value is 0 in TiCDC.
database	String	The name of the database where the row is located
table	String	The name of the table where the row is located
pkNames	Array	The names of all the columns that make up the primary key
isDdl	Bool	Whether the message is a DDL event
type	String	Event types defined by Canal-JSON
es	Number	13-bit (millisecond) timestamp when the event that generated the message happened
ts	Number	13-bit (millisecond) timestamp when TiCDC generated the message
sql	String	When isDdl is `true`, records the corresponding DDL statement
sqlType	Object	When isDdl is `false`, records how the data type of each column is represented in Java
mysqlType	object	When isDdl is `false`, records how the data type of each column is represented in MySQL
data	Object	When isDdl is `false`, records the name of each column and its data value
old	Object	Only if the message is generated by an update Event, records the name of each column and the data value before the update
_tidb	Object	TiDB extension field. It exists only if you set `enable-tidb-extension` to `true`. The value of `commitTs` is the TSO of the transaction that caused the row to change.

DML Event

TiCDC encodes a row of DML data change event as follows:

```
{
    "id": 0,
    "database": "test",
```

```
    "table": "tp_int",
    "pkNames": [
        "id"
    ],
    "isDdl": false,
    "type": "INSERT",
    "es": 1639633141221,
    "ts": 1639633142960,
    "sql": "",
    "sqlType": {
        "c_bigint": -5,
        "c_int": 4,
        "c_mediumint": 4,
        "c_smallint": 5,
        "c_tinyint": -6,
        "id": 4
    },
    "mysqlType": {
        "c_bigint": "bigint",
        "c_int": "int",
        "c_mediumint": "mediumint",
        "c_smallint": "smallint",
        "c_tinyint": "tinyint",
        "id": "int"
    },
    "data": [
        {
            "c_bigint": "9223372036854775807",
            "c_int": "2147483647",
            "c_mediumint": "8388607",
            "c_smallint": "32767",
            "c_tinyint": "127",
            "id": "2"
        }
    ],
    "old": null,
    "_tidb": {   // TiDB extension field
        "commitTs": 429918007904436226 // A TiDB TSO timestamp
    }
}
```

WATERMARK Event

TiCDC sends a WATERMARK Event only when you set `enable-tidb-extension` to `true`. The value of the `type` field is `TIDB_WATERMARK`. The Event contains the `_tidb` field,

and the field contains only one parameter `watermarkTs`. The value of `watermarkTs` is the TSO recorded when the Event is sent.

When you receive an Event of this type, all Events with `commitTs` less than `watermarkTs` have been sent. Because TiCDC provides the "At Least Once" semantics, data might be sent repeatedly. If a subsequent Event with `commitTs` less than `watermarkTs` is received, you can safely ignore this Event.

The following is an example of the WATERMARK Event.

```
{
    "id": 0,
    "database": "",
    "table": "",
    "pkNames": null,
    "isDdl": false,
    "type": "TIDB_WATERMARK",
    "es": 1640007049196,
    "ts": 1640007050284,
    "sql": "",
    "sqlType": null,
    "mysqlType": null,
    "data": null,
    "old": null,
    "_tidb": {   // TiDB extension field
        "watermarkTs": 429918007904436226 // A TiDB TSO timestamp
    }
}
```

Data resolution on the consumer side

As you can see from the example above, Canal-JSON has a uniform data format, with different field filling rules for different Event types. You can use a uniform method to resolve this JSON format data, and then determine the Event type by checking the field values.

- When `isDdl` is `true`, the message contains a DDL Event.
- When `isDdl` is `false`, you need to further check the `type` field. If `type` is `TIDB_WATERMARK`, it is a WATERMARK Event; otherwise, it is a DML Event.

13.9.5.13.4 Field descriptions

The Canal-JSON format records the corresponding data type in the `mysqlType` field and the `sqlType` field.

MySQL Type field

In the `mysqlType` field, the Canal-JSON format records the string of MySQL Type in each column. For more information, see TiDB Data Types.

SQL Type field

In the `sqlType` field, the Canal-JSON format records Java SQL Type of each column, which is the data type corresponding to the data in JDBC. Its value can be calculated by MySQL Type and the specific data value. The mapping is as follows:

MySQL Type	Java SQL Type Code
Boolean	-6
Float	7
Double	8
Decimal	3
Char	1
Varchar	12
Binary	2004
Varbinary	2004
Tinytext	2005
Text	2005
Mediumtext	2005
Longtext	2005
Tinyblob	2004
Blob	2004
Mediumblob	2004
Longblob	2004
Date	91
Datetime	93
Timestamp	93
Time	92
Year	12
Enum	4
Set	-7
Bit	-7
JSON	12

13.9.5.13.5 Integer types

You need to consider whether integer types have the `Unsigned` constraint and the value size, which corresponds to different Java SQL Type Codes respectively, as shown in the following table.

MySQL Type String	Value Range	Java SQL Type Code
tinyint	[-128, 127]	-6
tinyint unsigned	[0, 127]	-6
tinyint unsigned	[128, 255]	5
smallint	[-32768, 32767]	5
smallint unsigned	[0, 32767]	5

MySQL Type String	Value Range	Java SQL Type Code
smallint unsigned	[32768, 65535]	4
mediumint	[-8388608, 8388607]	4
mediumint unsigned	[0, 8388607]	4
mediumint unsigned	[8388608, 16777215]	4
int	[-2147483648, 2147483647]	4
int unsigned	[0, 2147483647]	4
int unsigned	[2147483648, 4294967295]	-5
bigint	[-9223372036854775808, 9223372036854775807]	-5
bigint unsigned	[0, 9223372036854775807]	-5
bigint unsigned	[9223372036854775808, 18446744073709551615]	3

The following table shows the mapping relationships between Java SQL Types in TiCDC and their codes.

Java SQL Type	Java SQL Type Code
CHAR	1
DECIMAL	3
INTEGER	4
SMALLINT	5
REAL	7
DOUBLE	8
VARCHAR	12
DATE	91
TIME	92
TIMESTAMP	93
BLOB	2004
CLOB	2005
BIGINT	-5
TINYINT	-6
Bit	-7

For more information about Java SQL Types, see Java SQL Class Types.

13.9.5.13.6 Comparison of TiCDC Canal-JSON and the official Canal

The way that TiCDC implements the Canal-JSON data format, including the `Update` Event and the `mysqlType` field, differs from the official Canal. The following table shows the main differences.

Item	TiCDC Canal-JSON	Canal
Event of `Update` Type	The `old` field contains all the column data	The `old` field contains only the modified column data

Item	TiCDC Canal-JSON	Canal
`mysqlType` field	For types with parameters, it does not contain the information of the type parameter	For types with parameters, it contains the full information of the type parameter

Event of `Update` Type

For an Event of `Update` Type:

- In TiCDC, the `old` field contains all the column data
- In the official Canal, the `old` field contains only the modified column data

Assume that the following SQL statements are executed sequentially in the upstream TiDB:

```
create table tp_int
(
    id         int auto_increment,
    c_tinyint  tinyint  null,
    c_smallint smallint null,
    c_mediumint mediumint null,
    c_int      int      null,
    c_bigint   bigint   null,
    constraint pk
        primary key (id)
);

insert into tp_int(c_tinyint, c_smallint, c_mediumint, c_int, c_bigint)
values (127, 32767, 8388607, 2147483647, 9223372036854775807);

update tp_int set c_int = 0, c_tinyint = 0 where c_smallint = 32767;
```

For the `update` statement, TiCDC outputs an Event message with `type` as `UPDATE`, as shown below. The `update` statement only modifies the `c_int` and `c_tinyint` columns. The `old` field in the output event message contains all the column data.

```
{
    "id": 0,
    ...
    "type": "UPDATE",
    ...
    "sqlType": {
        ...
    },
```

```
    "mysqlType": {
        ...
    },
    "data": [
        {
            "c_bigint": "9223372036854775807",
            "c_int": "0",
            "c_mediumint": "8388607",
            "c_smallint": "32767",
            "c_tinyint": "0",
            "id": "2"
        }
    ],
    "old": [                          // In TiCDC, this field contains all
        ↪ the column data.
        {
            "c_bigint": "9223372036854775807",
            "c_int": "2147483647",    // Modified column
            "c_mediumint": "8388607",
            "c_smallint": "32767",
            "c_tinyint": "127",       // Modified column
            "id": "2"
        }
    ]
}
```

For the official Canal, the `old` field in the output event message contains only the modified column data, as shown below.

```
{
    "id": 0,
    ...
    "type": "UPDATE",
    ...
    "sqlType": {
        ...
    },
    "mysqlType": {
        ...
    },
    "data": [
        {
            "c_bigint": "9223372036854775807",
            "c_int": "0",
            "c_mediumint": "8388607",
```

```
            "c_smallint": "32767",
            "c_tinyint": "0",
            "id": "2"
        }
    ],
    "old": [                        // In Canal, this field contains only
        ↪ the modified column data.
        {
            "c_int": "2147483647",    // Modified column
            "c_tinyint": "127",       // Modified column
        }
    ]
}
```

`mysqlType` field

For the `mysqlType` field, if a type contains parameters, the official Canal contains the full information of the type parameter. TiCDC does not contain such information.

In the following example, the table-defining SQL statement contains a parameter for each column, such as the ones for `decimal`, `char`, `varchar` and `enum`. By comparing the Canal-JSON formats generated by TiCDC and the official Canal, you can see that TiCDC only contains the basic MySQL information in the `mysqlType` field. If you need the full information of the type parameter, you need to implement it by other means.

Assume that the following SQL statements are executed sequentially in the upstream TiDB:

```
create table t (
    id    int auto_increment,
    c_decimal  decimal(10, 4) null,
    c_char     char(16)    null,
    c_varchar  varchar(16) null,
    c_binary   binary(16)  null,
    c_varbinary varbinary(16) null,
    c_enum enum('a','b','c') null,
    c_set set('a','b','c') null,
    c_bit bit(64)          null,
    constraint pk
        primary key (id)
);

insert into t (c_decimal, c_char, c_varchar, c_binary, c_varbinary, c_enum,
    ↪ c_set, c_bit)
values (123.456, "abc", "abc", "abc", "abc", 'a', 'a,b', b'1000001');
```

The output of TiCDC is as follows:

```
{
    "id": 0,
    ...
    "isDdl": false,
    "sqlType": {
        ...
    },
    "mysqlType": {
        "c_binary": "binary",
        "c_bit": "bit",
        "c_char": "char",
        "c_decimal": "decimal",
        "c_enum": "enum",
        "c_set": "set",
        "c_varbinary": "varbinary",
        "c_varchar": "varchar",
        "id": "int"
    },
    "data": [
        {
            ...
        }
    ],
    "old": null,
}
```

The output of the official Canal is as follows:

```
{
    "id": 0,
    ...
    "isDdl": false,
    "sqlType": {
        ...
    },
    "mysqlType": {
        "c_binary": "binary(16)",
        "c_bit": "bit(64)",
        "c_char": "char(16)",
        "c_decimal": "decimal(10, 4)",
        "c_enum": "enum('a','b','c')",
        "c_set": "set('a','b','c')",
        "c_varbinary": "varbinary(16)",
        "c_varchar": "varchar(16)",
        "id": "int"
```

```
    },
    "data": [
        {
            ...
        }
    ],
    "old": null,
}
```

13.9.5.13.7 Changes in TiCDC Canal-JSON

Changes in the `Old` field of the `Delete` events

From v5.4.0, the `old` field of the `Delete` events has changed.

The following is a `Delete` event message. Before v5.4.0, the `old` field contains the same content as the "data" field. In v5.4.0 and later versions, the `old` field is set to null. You can get the deleted data by using the "data" field.

```
{
    "id": 0,
    "database": "test",
    ...
    "type": "DELETE",
    ...
    "sqlType": {
        ...
    },
    "mysqlType": {
        ...
    },
    "data": [
        {
            "c_bigint": "9223372036854775807",
            "c_int": "0",
            "c_mediumint": "8388607",
            "c_smallint": "32767",
            "c_tinyint": "0",
            "id": "2"
        }
    ],
    "old": null,
    // The following is an example before v5.4.0. The `old` field contains
        ↪ the same content as the "data" field.
    "old": [
        {
```

```
            "c_bigint": "9223372036854775807",
            "c_int": "0",
            "c_mediumint": "8388607",
            "c_smallint": "32767",
            "c_tinyint": "0",
            "id": "2"
        }
    ]
}
```

13.9.6 TiCDC FAQs

This document introduces the common questions that you might encounter when using TiCDC.

> **Note:**
>
> In this document, the PD address specified in `cdc cli` commands is `--pd =http://10.0.10.25:2379`. When you use the command, replace the address with your actual PD address.

13.9.6.1 How do I choose `start-ts` when creating a task in TiCDC?

The `start-ts` of a replication task corresponds to a Timestamp Oracle (TSO) in the upstream TiDB cluster. TiCDC requests data from this TSO in a replication task. Therefore, the `start-ts` of the replication task must meet the following requirements:

- The value of `start-ts` is larger than the `tikv_gc_safe_point` value of the current TiDB cluster. Otherwise, an error occurs when you create a task.
- Before starting a task, ensure that the downstream has all data before `start-ts`. For scenarios such as replicating data to message queues, if the data consistency between upstream and downstream is not required, you can relax this requirement according to your application need.

If you do not specify `start-ts`, or specify `start-ts` as `0`, when a replication task is started, TiCDC gets a current TSO and starts the task from this TSO.

13.9.6.2 Why can't some tables be replicated when I create a task in TiCDC?

When you execute `cdc cli changefeed create` to create a replication task, TiCDC checks whether the upstream tables meet the replication restrictions. If some tables do not

meet the restrictions, `some tables are not eligible to replicate` is returned with a list of ineligible tables. You can choose `Y` or `y` to continue creating the task, and all updates on these tables are automatically ignored during the replication. If you choose an input other than `Y` or `y`, the replication task is not created.

13.9.6.3 How do I view the state of TiCDC replication tasks?

To view the status of TiCDC replication tasks, use `cdc cli`. For example:

```
cdc cli changefeed list --pd=http://10.0.10.25:2379
```

The expected output is as follows:

```
[{
    "id": "4e24dde6-53c1-40b6-badf-63620e4940dc",
    "summary": {
      "state": "normal",
      "tso": 417886179132964865,
      "checkpoint": "2020-07-07 16:07:44.881",
      "error": null
    }
}]
```

- `checkpoint`: TiCDC has replicated all data before this timestamp to downstream.
- `state`: The state of this replication task:
    - `normal`: The task runs normally.
    - `stopped`: The task is stopped manually or encounters an error.
    - `removed`: The task is removed.

> **Note:**
>
> This feature is introduced in TiCDC 4.0.3.

13.9.6.4 What is `gc-ttl` in TiCDC?

Since v4.0.0-rc.1, PD supports external services in setting the service-level GC safepoint. Any service can register and update its GC safepoint. PD ensures that the key-value data later than this GC safepoint is not cleaned by GC.

When the replication task is unavailable or interrupted, this feature ensures that the data to be consumed by TiCDC is retained in TiKV without being cleaned by GC.

When starting the TiCDC server, you can specify the Time To Live (TTL) duration of GC safepoint by configuring `gc-ttl`. You can also use TiUP to modify `gc-ttl`. The default value is 24 hours. In TiCDC, this value means:

- The maximum time the GC safepoint is retained at the PD after the TiCDC service is stopped.
- The maximum time a replication task can be suspended after the task is interrupted or manually stopped. If the time for a suspended replication task is longer than the value set by `gc-ttl`, the replication task enters the `failed` status, cannot be resumed, and cannot continue to affect the progress of the GC safepoint.

The second behavior above is introduced in TiCDC v4.0.13 and later versions. The purpose is to prevent a replication task in TiCDC from suspending for too long, causing the GC safepoint of the upstream TiKV cluster not to continue for a long time and retaining too many outdated data versions, thus affecting the performance of the upstream cluster.

> **Note:**
>
> In some scenarios, for example, when you use TiCDC for incremental replication after full replication with Dumpling/BR, the default 24 hours of `gc-ttl` may not be sufficient. You need to specify an appropriate value for `gc-ttl` when you start the TiCDC server.

13.9.6.5 What is the complete behavior of TiCDC garbage collection (GC) safepoint?

If a replication task starts after the TiCDC service starts, the TiCDC owner updates the PD service GC safepoint with the smallest value of `checkpoint-ts` among all replication tasks. The service GC safepoint ensures that TiCDC does not delete data generated at that time and after that time. If the replication task is interrupted, or manually stopped, the `checkpoint-ts` of this task does not change. Meanwhile, PD's corresponding service GC safepoint is not updated either.

If the replication task is suspended longer than the time specified by `gc-ttl`, the replication task enters the `failed` status and cannot be resumed. The PD corresponding service GC safepoint will continue.

The Time-To-Live (TTL) that TiCDC sets for a service GC safepoint is 24 hours, which means that the GC mechanism does not delete any data if the TiCDC service can be recovered within 24 hours after it is interrupted.

13.9.6.6 How to understand the relationship between the TiCDC time zone and the time zones of the upstream/downstream databases?

Configuration method	Upstream TiDB time zone	TiCDC Server Time Zone Support	TiCDC Configured using the `--tz` parameter when you start the TiCDC server	Downstream time zone	Configured using the `time-zone` parameter in `sink-uri`

	Upstream time zone	TiCDC time zone	Downstream time zone
Description	The time zone of the upstream TiDB, which affects DML operations of the timestamp type and DDL operations related to timestamp type columns.	TiCDC assumes that the upstream TiDB's time zone is the same as the TiCDC time zone configuration, and performs related operations on the timestamp column.	The downstream MySQL processes the timestamp in the DML and DDL operations according to the downstream time zone setting.

Note:

Be careful when you set the time zone of the TiCDC server, because this time zone is used for converting the time type. Keep the upstream time zone, TiCDC time zone, and the downstream time zone consistent. The TiCDC server chooses its time zone in the following priority:

- TiCDC first uses the time zone specified using `--tz`.
- When `--tz` is not available, TiCDC tries to read the time zone set using the `TZ` environment variable.
- When the `TZ` environment variable is not available, TiCDC uses the default time zone of the machine.

13.9.6.7 What is the default behavior of TiCDC if I create a replication task without specifying the configuration file in `--config`?

If you use the `cdc cli changefeed create` command without specifying the `-config` parameter, TiCDC creates the replication task in the following default behaviors:

- Replicates all tables except system tables
- Enables the Old Value feature
- Skips replicating tables that do not contain valid indexes

13.9.6.8 Does TiCDC support outputting data changes in the Canal protocol?

Yes. Note that for the Canal protocol, TiCDC only supports the JSON output format, while the protobuf format is not officially supported yet. To enable Canal output, specify `protocol` as `canal-json` in the `--sink-uri` configuration. For example:

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="kafka
    ↪ ://127.0.0.1:9092/cdc-test?kafka-version=2.4.0&protocol=canal-json"
    ↪ --config changefeed.toml
```

Note:

- This feature is introduced in TiCDC 4.0.2.
- TiCDC currently supports outputting data changes in the Canal-JSON format only to MQ sinks such as Kafka and Pulsar.

For more information, refer to Create a replication task.

13.9.6.9 Why does the latency from TiCDC to Kafka become higher and higher?

- Check how do I view the state of TiCDC replication tasks.
- Adjust the following parameters of Kafka:
    - Increase the `message.max.bytes` value in `server.properties` to `1073741824` (1 GB).
    - Increase the `replica.fetch.max.bytes` value in `server.properties` to `1073741824` (1 GB).
    - Increase the `fetch.message.max.bytes` value in `consumer.properties` to make it larger than the `message.max.bytes` value.

13.9.6.10 When TiCDC replicates data to Kafka, can I control the maximum size of a single message in TiDB?

When `protocol` is set to `avro` or `canal-json`, messages are sent per row change. A single Kafka message contains only one row change and is generally no larger than Kafka's limit. Therefore, there is no need to limit the size of a single message. If the size of a single Kafka message does exceed Kakfa's limit, refer to Why does the latency from TiCDC to Kafka become higher and higher?.

When `protocol` is set to `open-protocol`, messages are sent in batches. Therefore, one Kafka message might be excessively large. To avoid this situation, you can configure the `max-message-bytes` parameter to control the maximum size of data sent to the Kafka broker each time (optional, `10MB` by default). You can also configure the `max-batch-size` parameter (optional, `16` by default) to specify the maximum number of change records in each Kafka message.

13.9.6.11 If I modify a row multiple times in a transaction, will TiCDC output multiple row change events?

No. When you modify the same row in one transaction multiple times, TiDB only sends the latest modification to TiKV. Therefore, TiCDC can only obtain the result of the latest modification.

13.9.6.12 When TiCDC replicates data to Kafka, does a message contain multiple types of data changes?

Yes. A single message might contain multiple `update`s or `delete`s, and `update` and `delete` might co-exist.

13.9.6.13 When TiCDC replicates data to Kafka, how do I view the timestamp, table name, and schema name in the output of TiCDC Open Protocol?

The information is included in the key of Kafka messages. For example:

```
{
    "ts":<TS>,
    "scm":<Schema Name>,
    "tbl":<Table Name>,
    "t":1
}
```

For more information, refer to TiCDC Open Protocol event format.

13.9.6.14 When TiCDC replicates data to Kafka, how do I know the timestamp of the data changes in a message?

You can get the unix timestamp by moving `ts` in the key of the Kafka message by 18 bits to the right.

13.9.6.15 How does TiCDC Open Protocol represent `null`?

In TiCDC Open Protocol, the type code `6` represents `null`.

Type	Code	Output Example	Note
Null	6	`{"t":6,"v":null}`	

For more information, refer to TiCDC Open Protocol column type code.

13.9.6.16 How can I tell if a Row Changed Event of TiCDC Open Protocol is an `INSERT` event or an `UPDATE` event?

If the Old Value feature is not enabled, you cannot tell whether a Row Changed Event of TiCDC Open Protocol is an `INSERT` event or an `UPDATE` event. If the feature is enabled, you can determine the event type by the fields it contains:

- `UPDATE` event contains both `"p"` and `"u"` fields
- `INSERT` event only contains the `"u"` field
- `DELETE` event only contains the `"d"` field

For more information, refer to Open protocol Row Changed Event format.

13.9.6.17 How much PD storage does TiCDC use?

When using TiCDC, you might encounter the `etcdserver: mvcc: database space exceeded` error, which is primarily related to the mechanism that TiCDC uses etcd in PD to store metadata.

etcd uses Multi-Version Concurrency Control (MVCC) to store data, and the default compaction interval in PD is 1 hour. This means that etcd retains multiple versions of all data for 1 hour before compaction.

Before v6.0.0, TiCDC uses etcd in PD to store and update metadata for all tables in a changefeed. Therefore, the PD storage space used by TiCDC is proportional to the number of tables being replicated by the changefeed. When TiCDC is replicating a large number of tables, the etcd storage space could fill up quickly, increasing the probability of the `etcdserver: mvcc: database space exceeded` error.

If you encounter this error, refer to etcd maintenance space-quota to clean up the etcd storage space.

Starting from v6.0.0, TiCDC optimizes its metadata storage mechanism, effectively avoiding the etcd storage space issues caused by the preceding reasons. If your TiCDC version is earlier than v6.0.0, it is recommended to upgrade to v6.0.0 or later versions.

13.9.6.18 Does TiCDC support replicating large transactions? Is there any risk?

TiCDC provides partial support for large transactions (more than 5 GB in size). Depending on different scenarios, the following risks might exist:

- The latency of primary-secondary replication might greatly increase.
- When TiCDC's internal processing capacity is insufficient, the replication task error `ErrBufferReachLimit` might occur.
- When TiCDC's internal processing capacity is insufficient or the throughput capacity of TiCDC's downstream is insufficient, out of memory (OOM) might occur.

Since v6.1.1, TiCDC supports splitting a single-table transaction into multiple transactions. This can greatly reduce the latency and memory consumption of replicating large transactions. Therefore, if your application does not have a high requirement on transaction atomicity, it is recommended to enable the splitting of large transactions to avoid possible replication latency and OOM. To enable the splitting, set the value of the sink uri parameter `transaction-atomicity` to `none`.

If you still encounter an error above, it is recommended to use BR to restore the incremental data of large transactions. The detailed operations are as follows:

1. Record the `checkpoint-ts` of the changefeed that is terminated due to large transactions, use this TSO as the `--lastbackupts` of the BR incremental backup, and execute incremental data backup.

2. After backing up the incremental data, you can find a log record similar to `["Full backup Failed summary : total backup ranges: 0, total success: 0, total failed: 0"] [BackupTS=421758868510212097]` in the BR log output. Record the `BackupTS` in this log.
3. Restore the incremental data.
4. Create a new changefeed and start the replication task from `BackupTS`.
5. Delete the old changefeed.

13.9.6.19 The default value of the time type field is inconsistent when replicating a DDL statement to the downstream MySQL 5.7. What can I do?

Suppose that the `create table test (id int primary key, ts timestamp)` statement is executed in the upstream TiDB. When TiCDC replicates this statement to the downstream MySQL 5.7, MySQL uses the default configuration. The table schema after the replication is as follows. The default value of the `timestamp` field becomes `CURRENT_TIMESTAMP`:

```
mysql root@127.0.0.1:test> show create table test;
+--
   ↪ -----+-----------------------------------------------------------------------
   ↪
| Table | Create Table
   ↪                                                                    |
+--
   ↪ -----+-----------------------------------------------------------------------
   ↪
| test | CREATE TABLE `test` (
   ↪                                                          |
|      |   `id` int(11) NOT NULL,
   ↪                                                        |
|      |   `ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
   ↪ CURRENT_TIMESTAMP, |
|      |   PRIMARY KEY (`id`)
   ↪                                                          |
|      | ) ENGINE=InnoDB DEFAULT CHARSET=latin1
   ↪                                    |
+--
   ↪ -----+-----------------------------------------------------------------------
   ↪
1 row in set
```

From the result, you can see that the table schema before and after the replication is inconsistent. This is because the default value of `explicit_defaults_for_timestamp` in TiDB is different from that in MySQL. See MySQL Compatibility for details.

Since v5.0.1 or v4.0.13, for each replication to MySQL, TiCDC automatically sets `explicit_defaults_for_timestamp = ON` to ensure that the time type is consistent

between the upstream and downstream. For versions earlier than v5.0.1 or v4.0.13, pay attention to the compatibility issue caused by the inconsistent `explicit_defaults_for_timestamp` ↪ value when using TiCDC to replicate the time type data.

13.9.6.20 Why do `INSERT/UPDATE` statements from the upstream become `REPLACE INTO` after being replicated to the downstream if I set `safe-mode` to `true` when I create a TiCDC replication task?

When a changefeed is created in TiCDC, the `safe-mode` setting defaults to `true`, which generates the `REPLACE INTO` statement to execute for the upstream `INSERT/UPDATE` statements.

Currently, users cannot modify the `safe-mode` setting, so this issue currently has no solution.

13.9.6.21 When the sink of the replication downstream is TiDB or MySQL, what permissions do users of the downstream database need?

When the sink is TiDB or MySQL, the users of the downstream database need the following permissions:

- `Select`
- `Index`
- `Insert`
- `Update`
- `Delete`
- `Create`
- `Drop`
- `Alter`
- `Create View`

If you need to replicate `recover table` to the downstream TiDB, you should have the `Super` permission.

13.9.6.22 Why does TiCDC use disks? When does TiCDC write to disks? Does TiCDC use memory buffer to improve replication performance?

When upstream write traffic is at peak hours, the downstream may fail to consume all data in a timely manner, resulting in data pile-up. TiCDC uses disks to process the data that is piled up. TiCDC needs to write data to disks during normal operation. However, this is not usually the bottleneck for replication throughput and replication latency, given that writing to disks only results in latency within a hundred milliseconds. TiCDC also uses memory to accelerate reading data from disks to improve replication performance.

13.9.6.23 Why does replication using TiCDC stall or even stop after data restore using TiDB Lightning and BR from upstream?

Currently, TiCDC is not yet fully compatible with TiDB Lightning and BR. Therefore, avoid using TiDB Lightning and BR on tables that are replicated by TiCDC. Otherwise, unknown errors might occur, such as TiCDC replication getting stuck, a significant spike in replication latency, or data loss.

If you need to use TiDB Lightning or BR to restore data for some tables replicated by TiCDC, take these steps:

1. Remove the TiCDC replication task related to these tables.
2. Use TiDB Lightning or BR to restore data separately in the upstream and downstream clusters of TiCDC.
3. After the restoration is complete and data consistency between the upstream and downstream clusters is verified, create a new TiCDC replication task for incremental replication, with the timestamp (TSO) from the upstream backup as the `start-ts` for the task. For example, assuming the snapshot timestamp of the BR backup in the upstream cluster is `431434047157698561`, you can create a new TiCDC replication task using the following command:

```
cdc cli changefeed create -c "upstream-to-downstream-some-tables" --
    ↪ start-ts=431434047157698561 --sink-uri="mysql://root@127
    ↪ .0.0.1:4000? time-zone="
```

13.9.6.24 After a changefeed resumes from pause, its replication latency gets higher and higher and returns to normal only after a few minutes. Why?

When a changefeed is resumed, TiCDC needs to scan the historical versions of data in TiKV to catch up with the incremental data logs generated during the pause. The replication process proceeds only after the scan is completed. The scan process might take several to tens of minutes.

13.9.6.25 Why can't I use the `cdc cli` command to operate a TiCDC cluster deployed by TiDB Operator?

This is because the default port number of the TiCDC cluster deployed by TiDB Operator is `8301`, while the default port number of the `cdc cli` command to connect to the TiCDC server is `8300`. When using the `cdc cli` command to operate the TiCDC cluster deployed by TiDB Operator, you need to explicitly specify the `--server` parameter, as follows:

```
./cdc cli changefeed list --server "127.0.0.1:8301"
[
  {
    "id": "4k-table",
```

```
    "namespace": "default",
    "summary": {
      "state": "stopped",
      "tso": 441832628003799353,
      "checkpoint": "2023-05-30 22:41:57.910",
      "error": null
    }
  },
  {
    "id": "big-table",
    "namespace": "default",
    "summary": {
      "state": "normal",
      "tso": 441872834546892882,
      "checkpoint": "2023-06-01 17:18:13.700",
      "error": null
    }
  }
]
```

13.9.6.26 How do I resolve frequent `CDC:ErrMySQLDuplicateEntryCDC` errors?

When using TiCDC to replicate data to TiDB or MySQL, you might encounter the following error if SQL statements in the upstream are executed in a specific pattern:

```
CDC:ErrMySQLDuplicateEntryCDC
```

The cause of the error: TiDB combines `DELETE + INSERT` operations on the same row within the same transaction into a single `UPDATE` row change. When TiCDC replicates these changes as updates to the downstream, the `UPDATE` operations attempting to swap unique key values might result in conflicts.

Taking the following table as an example:

```
CREATE TABLE data_table (
    id BIGINT(20) NOT NULL PRIMARY KEY,
    value BINARY(16) NOT NULL,
    UNIQUE KEY value_index (value)
) CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

If the upstream attempts to swap the `value` field of the two rows in the table:

```
DELETE FROM data_table WHERE id = 1;
DELETE FROM data_table WHERE id = 2;
INSERT INTO data_table (id, value) VALUES (1, 'v3');
INSERT INTO data_table (id, value) VALUES (2, 'v1');
```

TiDB generates two `UPDATE` row changes, so TiCDC converts them into two `UPDATE` statements for replication to the downstream:

```
UPDATE data_table SET value = 'v3' WHERE id = 1;
UPDATE data_table SET value = 'v1' WHERE id = 2;
```

If the downstream table still contains `v1` when executing the second `UPDATE` statement, it violates the unique key constraint on the `value` column, resulting in the `CDC:ErrMySQLDuplicateEntryCDC` error.

If the `CDC:ErrMySQLDuplicateEntryCDC` error occurs frequently, you can enable TiCDC safe mode by setting the `safe-mode=true` parameter in the `sink-uri` configuration:

```
mysql://user:password@host:port/?safe-mode=true
```

In safe mode, TiCDC splits the `UPDATE` operation into `DELETE + REPLACE INTO` for execution, thus avoiding the unique key conflict error.

13.9.7 TiCDC Glossary

This glossary provides TiCDC-related terms and definitions. These terms appears in TiCDC logs, monitoring metrics, configurations, and documents.

For TiDB-related terms and definitions, refer to TiDB glossary.

13.9.7.1 C

13.9.7.1.1 Capture

A single TiCDC instance on which the replication task of the cluster runs. Multiple captures form a TiCDC cluster.

13.9.7.1.2 Changed data

The data to be written to TiCDC from the upstream TiDB cluster, including the DML-caused data changes and the DDL-caused table schema changes.

13.9.7.1.3 Changefeed

An incremental replication task in TiCDC, which outputs the data change logs of several tables in a TiDB cluster to the designated downstream.

13.9.7.2 O

13.9.7.2.1 Owner

A capture of a special role that manages the TiCDC cluster and schedules replication tasks of the cluster. An owner is elected by captures and there is at most one owner at any time.

13.9.7.3 P

13.9.7.3.1 Processor

TiCDC replication tasks allocate data tables on TiCDC instances, and the processor refers to the replication processing unit of these tables. Processor tasks include pulling, sorting, restoring, and distributing changed data.

13.10 Backup & Restore (BR)

13.10.1 BR Overview

BR (Backup & Restore) is a command-line tool for **distributed backup and restoration** of the TiDB cluster data. In addition to regular backup and restoration, you can also use BR for large-scale data migration as long as compatibility is ensured.

This document describes BR's architecture, features, and usage tips.

13.10.1.1 BR architecture

BR sends a backup or restoration command to each TiKV node. After receiving the command, TiKV performs the corresponding backup or restoration operation.

Each TiKV node has a path in which the backup files generated in the backup operation are stored and from which the stored backup files are read during the restoration.

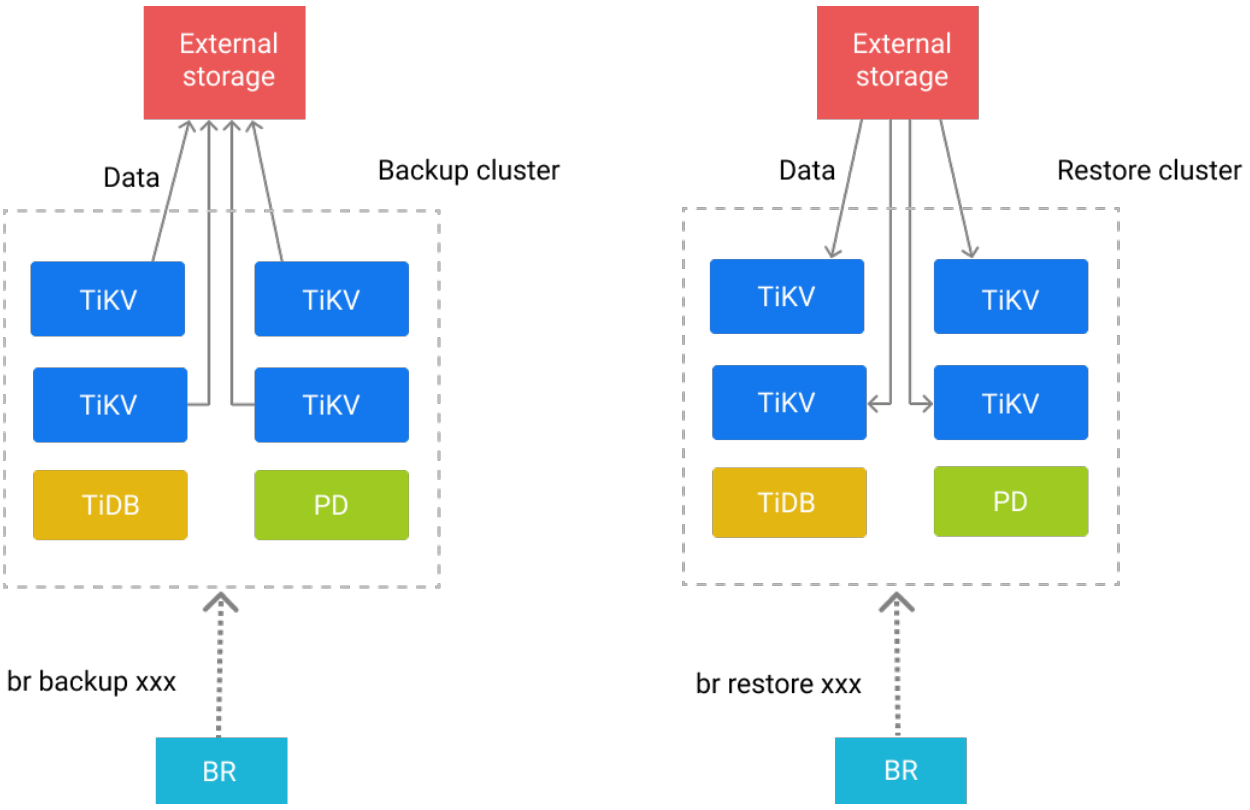


Figure 277: br-arch

For detailed information about the BR design, see BR Design Principles.

13.10.1.2 BR features

This section describes BR features and the performance impact.

13.10.1.2.1 Back up TiDB cluster data

- **Back up cluster snapshots**: A snapshot of a TiDB cluster contains transactionally consistent data at a specific time. You can back up snapshot data of a TiDB cluster using BR. For details, see Back up TiDB cluster snapshots.
- **Back up incremental data**: The incremental data of a TiDB cluster represents changes between the latest snapshot and the previous snapshot. Incremental data is smaller in size compared with full data, and can be used together with snapshot backup, which reduces the volume of backup data. For details, see Back up incremental data.
- **Back up a database or table**: On top of snapshot and incremental data backup, BR supports backing up a specific database or table and filtering out unnecessary data. For details, see Back up a database or table.

- **Encrypt backup data**: BR supports backup data encryption and Amazon S3 server-side encryption. You can select an encryption method as needed. For details, see Encrypt backup data.

Impact on performance

The impact of backup on a TiDB cluster is kept below 20%, and this value can be reduced to 10% or less with the proper configuration of the TiDB cluster. The backup speed of a TiKV node is scalable and ranges from 50 MB/s to 100 MB/s. For more information, see Backup performance and impact.

Storage types of backup data

BR supports backing up data to Amazon S3, Google Cloud Storage, Azure Blob Storage, NFS, and other S3-compatible file storage services. For details, see Back up data to external storages.

13.10.1.2.2 Restore TiDB cluster data

- **Restore snapshot backup**: You can restore snapshot backup data to a new cluster. For details, see Restore TiDB cluster snapshots.
- **Restore incremental backup**: You can restore the incremental backup data to a cluster. For details, see Restore incremental backup.
- **Restore a database or a table from backup**: You can restore part of a specific database or table. During the process, BR will filter out unnecessary data. For details, see Restore a database or a table.

Impact on performance

Data restoration is performed at a scalable speed. Generally, the speed is 100 MB/s per TiKV node. BR only supports restoring data to a new cluster and uses the resources of the target cluster as much as possible. For more details, see Restoration performance and impact.

13.10.1.3 Before you use BR

Before you use BR, pay attention to its usage restrictions, compatibility, and other considerations.

13.10.1.3.1 Usage restrictions

This section describes usage restrictions of BR.

Unsupported scenarios

When BR restores data to the upstream cluster of TiCDC or TiDB Binlog, TiCDC or TiDB Binlog cannot replicate the restored data to the downstream cluster.

Compatibility

The compatibility issues of BR and a TiDB cluster are as follows:

- There is a cross-version compatibility issue:

  Before v5.4.0, BR cannot restore tables with `charset=GBK`. At the same time, no version of BR supports restoring `charset=GBK` tables to a TiDB cluster earlier than v5.4.0.

- The KV format might change when some features are enabled or disabled. If these features are not consistently enabled or disabled during backup and restoration, compatibility issues might occur.

These features are as follows:

Feature	Issue	Solution
Feature	Issue	Solution
Clustered index	#565	Make sure that the value of the `tidb_enable_clustered_index` global variable during restoration is consistent with that during backup. Otherwise, data inconsistency might occur, such as `default not found` and inconsistent data index.

Feature	Issue	Solution
New collation	#352	Make sure that the value of the `new_collations_enabled_on_first_bootstrap` variable during restoration is consistent with that during backup. Otherwise, inconsistent data index might occur and checksum might fail to pass.

Feature	Issue	Solution
Global temporary tables		Make sure that you are using BR v5.3.0 or a later version to back up and restore data. Otherwise, an error occurs in the definition of the backed global temporary tables.

However, even after you have ensured that the preceding features are consistently enabled or disabled during backup and restoration, compatibility issues might still occur due to the inconsistent internal versions or inconsistent interfaces between BR and TiKV/TiDB/PD. To avoid such cases, BR provides a built-in version check.

Version check

Before performing backup and restoration, BR compares and checks the TiDB cluster

version and the BR version. If there is a major-version mismatch (for example, BR v4.x and TiDB v5.x), BR prompts a reminder to exit. To forcibly skip the version check, you can set `--check-requirements=false`.

Note that skipping the version check might introduce incompatibility. The version compatibility mapping between BR and TiDB versions are as follows:

Backup version (vertical)	Restoration version (horizontal)		
	Use BR v6.1 to restore TiDB v6.1	Use BR v5.0 to restore TiDB v5.0	Use BR v4.0 to restore TiDB v4.0
Use BR v6.1 to back up TiDB v6.1			(If a table with the primary key of the non-integer clustered index type is restored to a TiDB v4.0 cluster, BR will cause data error without warning.)

Backup version (vertical)	Restoration version (horizontal)		
	Use BR v6.1 to restore TiDB v6.1	Use BR v5.0 to restore TiDB v5.0	Use BR v4.0 to restore TiDB v4.0
Use BR v5.0 to back up TiDB v5.0			(If a table with the primary key of the non-integer clustered index type is restored to a TiDB v4.0 cluster, BR will cause data error without warning.)

Backup version (vertical)	Restoration version (horizontal)		
	Use BR v6.1 to restore TiDB v6.1	Use BR v5.0 to restore TiDB v5.0	Use BR v4.0 to restore TiDB v4.0
Use BR v4.0 to back up TiDB v4.0			(If TiKV >= v4.0.0-rc.1, and if BR contains the #233 bug fix and TiKV does not contain the #7241 bug fix, BR will cause the TiKV node to restart.)

Backup version (vertical) / Restoration version (horizontal)	Use BR v6.1 to restore TiDB v6.1	Use BR v5.0 to restore TiDB v5.0	Use BR v4.0 to restore TiDB v4.0
Use BR v6.1 or v5.0 to back up TiDB v4.0	(If the TiDB version is earlier than v4.0.9, the #609 issue might occur.)	(If the TiDB version is earlier than v4.0.9, the #609 issue might occur.)	(If the TiDB version is earlier than v4.0.9, the #609 issue might occur.)

Some tips

The following are some recommended operations for using BR:

- It is recommended that you perform the backup operation during off-peak hours to minimize the impact on applications.
- BR only supports restoring data to a new cluster and uses resources of the target cluster as much as possible. Therefore, it is not recommended that you restore data to a production cluster. Otherwise, services might be affected.
- It is recommended that you execute multiple backup or restoration operations one by one. Running backup or restoration operations in parallel reduces performance and also affects online applications. Worse still, lack of collaboration between multiple tasks might result in task failures and affect cluster performance.
- Amazon S3, Google Cloud Storage, and Azure Blob Storage are recommended to store backup data.

- Make sure that the BR and TiKV nodes, and the backup storage system have sufficient network bandwidth to ensure sound write/read performance. Insufficient storage capacity might be the bottleneck for a backup or restoration operation.

13.10.1.3.2 See also

- Back up Data to S3-Compatible Storage Using BR
- Restore Data from S3-Compatible Storage Using BR
- Back up Data to GCS Using BR
- Restore Data from GCS Using BR
- Back up Data to PV
- Restore Data from PV

13.10.2 Deploy and Use BR

This document describes the recommended deployment of Backup & Restore (BR) and how to use BR to back up and restore data.

13.10.2.1 Deploy BR

Recommended practices when deploying BR:

- In production environments, deploy BR on a node with at least 8 cores CPU and 16 GB memory. Select an appropriate OS version by following OS and platform requirements.
- Save backup data to Amazon S3, GCS or Azure Blob Storage.
- Allocate sufficient resources for backup and restoration:
    - BR, TiKV nodes, and the backup storage system should provide network bandwidth that is greater than the backup speed. If the target cluster is particularly large, the threshold of backup and restoration speed is limited by the bandwidth of the backup network.
    - The backup storage system should also provide sufficient write/read performance (IOPS). Otherwise, the IOPS might become a performance bottleneck during backup or restoration.
    - TiKV nodes need to have at least two additional CPU cores and high performance disks for backups. Otherwise, the backup might have an impact on the services running on the cluster.

Note:

- If no Network File System (NFS) is mounted to a BR or TiKV node, or if you use external storage that supports Amazon S3, GCS, or Azure Blob Storage protocols, the data backed up by BR is generated at each TiKV node. Because BR only backs up the leader replica, you need to estimate the space reserved on each node based on the leader size. Because TiDB uses the leader count for load balancing by default, leaders can greatly differ in size. This might result in the issue that the backup data is unevenly distributed on each node.
- **Note that this is not the recommended way to deploy BR**, because the backup data are scattered in the local file system of each node. Collecting the backup data might result in data redundancy and operation and maintenance problems. Meanwhile, if you restore data directly before collecting the backup data, you will encounter the `SST file not found` error.

13.10.2.2 Use BR

Currently, the following methods are supported to run the BR tool:

13.10.2.2.1 Use SQL statements

TiDB supports both `BACKUP` and `RESTORE` SQL statements. You can monitor the progress of these operations using the statement `SHOW BACKUPS|RESTORES`.

13.10.2.2.2 Use the command-line tool

For details, see Use BR Command-line for Backup and Restoration.

13.10.2.2.3 Use BR in the Kubernetes environment

In a Kubernetes environment, you can use TiDB Operator to back up TiDB cluster data to Amazon S3, GCS or persistent volumes, and restore data from the backup data in such systems. For details, see Back Up and Restore Data Using TiDB Operator.

13.10.3 Use BR to Back Up Cluster Data

This document describes how to back up TiDB cluster data in the following scenarios:

- Back up TiDB cluster snapshots
- Back up a database
- Back up a table
- Back up multiple tables with table filter
- Back up data to external storage

- Back up incremental data
- Encrypt backup data

If you are not familiar with the backup and restore tools, it is recommended that you read the following documents to fully understand usage principles and methods of these tools:

- BR Overview
- Use BR Command-line for Backup and Restoration

If you need to back up a small amount of data (for example, less than 50 GB) and do not require high backup speed, you can use Dumpling to export data to implement backup. For detailed backup operations, see Use Dumpling to back up full data.

13.10.3.1 Back up TiDB cluster snapshots

A snapshot of a TiDB cluster contains only the latest and transactionally consistent data at a specific time. You can back up the latest or specified snapshot data of a TiDB cluster by running the `br backup full` command. To get help on this command, run the `br backup full --help` command.

Example: Back up the snapshot generated at `2022-01-30 07:42:23` to the `2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup full \
    --pd "${PDIP}:2379" \
    --backupts '2022-01-30 07:42:23' \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file backupfull.log
```

In the preceding command:

- `--backupts`: The physical time of the snapshot. If data of this snapshot is processed by Garbage Collection (GC), the `br backup` command will exit with an error. If you leave this parameter unspecified, BR picks the snapshot corresponding to the backup start time.
- `--ratelimit`: The maximum speed **per TiKV** performing backup tasks (in MiB/s).
- `--log-file`: The target file for BR logging.

During backup, a progress bar is displayed in the terminal, as shown below. When the progress bar advances to 100%, the backup is complete.

```
br backup full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file backupfull.log
Full Backup <---------/.....................................................>
    ↪ 17.12%.
```

After the backup is completed, BR compares the checksum of the backup data with the admin checksum table of the cluster to ensure data correctness and security.

13.10.3.2 Back up a database or a table

BR supports backing up partial data of a specified database or table from a cluster snapshot or incremental data backup. This feature allows you to filter out unwanted data from snapshot backup and incremental data backup, and back up only business-critical data.

13.10.3.2.1 Back up a database

To back up a database in a cluster, run the `br backup db` command. To get help on this command, run the `br backup db --help` command.

Example: Back up the `test` database to the `db-test/2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup db \
    --pd "${PDIP}:2379" \
    --db test \
    --storage "s3://backup-data/db-test/2022-01-30/" \
    --ratelimit 128 \
    --log-file backuptable.log
```

In the preceding command, `--db` specifies the database name, and other parameters are the same as those in Back up TiDB cluster snapshots.

13.10.3.2.2 Back up a table

To back up a table in a cluster, run the `br backup table` command. To get help on this command, run the `br backup table --help` command.

Example: Back up `test.usertable` to the `table-db-usertable/2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup table \
    --pd "${PDIP}:2379" \
    --db test \
    --table usertable \
```

```
    --storage "s3://backup-data/table-db-usertable/2022-01-30/" \
    --ratelimit 128 \
    --log-file backuptable.log
```

In the preceding command, `--db` and `--table` specify the database name and table name respectively, and other parameters are the same as those in Back up TiDB cluster snapshots.

13.10.3.2.3 Back up multiple tables with table filter

To back up multiple tables with more criteria, run the `br backup full` command and specify the table filters with `--filter` or `-f`.

Example: Back up `db*.tbl*` data of a table to the `table-filter/2022-01-30/` directory in the `backup-data` bucket of Amazon S3.

```
br backup full \
    --pd "${PDIP}:2379" \
    --filter 'db*.tbl*' \
    --storage "s3://backup-data/table-filter/2022-01-30/" \
    --ratelimit 128 \
    --log-file backupfull.log
```

13.10.3.3 Back up data to external storage

BR supports backing up data to Amazon S3, Google Cloud Storage (GCS), Azure Blob Storage, NFS, or other S3-compatible file storage services. For details, see the following documents:

- Back up data on Amazon S3 using BR
- Back up data on Google Cloud Storage using BR
- Back up data on Azure Blob Storage using BR

13.10.3.4 Back up incremental data

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

Incremental data of a TiDB cluster is differentiated data between the snapshot of a starting point and that of an end point. Compared with snapshot data, incremental data is smaller and therefore it is a supplementary to snapshot backup, which reduces the volume of backup data.

To back up incremental data, run the `br backup` command with **the last backup timestamp** `--lastbackupts` specified. To get `--lastbackupts`, run the `validate` command. The following is an example:

```
LAST_BACKUP_TS=`br validate decode --field="end-version" -s s3://backup-data
    ↪ /2022-01-30/ | tail -n1`
```

Note:

- You need to save the incremental backup data under a different path from the previous snapshot backup.
- GC safepoint must be prior to `lastbackupts`. The defalt GC lifetime is 10 minutes in TiDB, which means that TiDB only backs up incremental data generated in the last 10 minutes. To back up earlier incremental data, you need to adjust TiDB GC Lifetime setting.

```
br backup full\
    --pd ${PDIP}:2379 \
    --ratelimit 128 \
    --storage "s3://backup-data/2022-01-30/incr" \
    --lastbackupts ${LAST_BACKUP_TS}
```

The preceding command backs up the incremental data between `(LAST_BACKUP_TS, current PD timestamp]` and the DDLs generated during this time period. When restoring incremental data, BR restores all DDLs first, and then restores data.

13.10.3.5 Encrypt backup data

Warning:

This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

BR supports encrypting backup data at the backup end and at the storage end when backing up to Amazon S3. You can choose either encryption method as required.

13.10.3.5.1 Encrypt backup data at the backup end

Since TiDB v5.3.0, you can encrypt backup data by configuring the following parameters:

- `--crypter.method`: Encryption algorithm, which can be `aes128-ctr`, `aes192-ctr`, or `aes256-ctr`. The default value is `plaintext`, indicating that data is not encrypted.
- `--crypter.key`: Encryption key in hexadecimal string format. It is a 128-bit (16 bytes) key for the algorithm `aes128-ctr`, 24-byte key for the algorithm `aes192-ctr`, and 32-byte key for the algorithm `aes256-ctr`.
- `--crypter.key-file`: The key file. You can directly pass in the file path where the key is stored as a parameter without passing in "crypter.key".

Example: Encrypt backup data at the backup end.

```
br backup full\
    --pd ${PDIP}:2379 \
    --storage "s3://backup-data/2022-01-30/" \
    --crypter.method aes128-ctr \
    --crypter.key 0123456789abcdef0123456789abcdef
```

Note:

- If the key is lost, the backup data cannot be restored to the cluster.
- The encryption feature needs to be used on BR tools and TiDB clusters v5.3.0 or later versions. The encrypted backup data cannot be restored on clusters earlier than v5.3.0.

13.10.3.5.2 Encrypt backup data when backing up to Amazon S3

BR supports server-side encryption (SSE) when backing up data to S3. In this scenario, you can use AWS KMS keys you have created to encrypt data. For details, see BR S3 server-side encryption.

13.10.3.6 Validate backup data

After you back up data using BR, you can validate the backup data, including checking its integrity and viewing the metadata (such as TSO) by decoding the `backupmeta` file.

13.10.3.6.1 Check the integrity of backup data

To check the integrity of backup data, you can run the `tiup br debug checksum` command to calculate the checksum of the backup data.

Example: Calculate the checksum of the backup data in the `${prefix}` directory in the `backup-data` bucket on Amazon S3.

```
br debug checksum \
    --storage 's3://backup-data/${prefix}' \
    --s3.endpoint '${S3-endpoint-URL}' \
    --log-file checksum.log
```

13.10.3.6.2 Decode `backupmeta` to a readable JSON file

After a backup is complete, you can run the `tiup br debug decode` command to decode the `backupmeta` file into a readable JSON file, through which you can view the metadata (such as TSO) of the snapshot.

Example: Decode the `backupmeta` file in the `${prefix}` directory in the `backup-data` bucket on Amazon S3 into a JSON file `backupmeta.json`. The decoded file is stored in `s3://backup-data/${prefix}/backupmeta.json`.

```
br debug decode \
    --storage 's3://backup-data/${prefix}' \
    --s3.endpoint '${S3-endpoint-URL}' \
    --log-file decode-backupmeta.log
```

Open the `backupmeta.json` file and search for `end_version` to view the TSO of the snapshot.

If necessary, you can also encode the JSON format `backupmeta` file back to the original state. Specifically, run the `tiup br debug encode` command to generate the file named `backupmeta_from_json`.

Example: Encode the `backupmeta.json` file in the `${prefix}` directory in the `backup-data` bucket on Amazon S3 into a `backupmeta` file. The encoded file is stored in `s3://backup-data/${prefix}/backupmeta_from_json`.

```
br debug encode \
    --storage 's3://backup-data/${prefix}' \
    --s3.endpoint '${S3-endpoint-URL}' \
    --log-file encode-backupmeta.log
```

13.10.3.7 Backup performance and impact

The backup feature has some impact on cluster performance (transaction latency and QPS). However, you can mitigate the impact by adjusting the number of backup threads `backup.num-threads` or by adding more clusters.

To illustrate the impact of backup, this document lists the test conclusions of several snapshot backup tests:

- (5.3.0 and earlier) When the backup threads of BR on a TiKV node takes up 75% of the total CPU of the node, the QPS is reduced by 30% of the original QPS.
- (5.4.0 and later) When there are no more than `8` threads of BR on a TiKV node and the cluster's total CPU utilization does not exceed 80%, the impact of BR tasks on the cluster (write and read) is 20% at most.
- (5.4.0 and later) When there are no more than `8` threads of BR on a TiKV node and the cluster's total CPU utilization does not exceed 75%, the impact of BR tasks on the cluster (write and read) is 10% at most.
- (5.4.0 and later) When there are no more than `8` threads of BR on a TiKV node and the cluster's total CPU utilization does not exceed 60%, BR tasks has little impact on the cluster (write and read).

You can mitigate impact on cluster performance by reducing the number of backup threads. However, this might cause backup performance to deteriorate. Based on the preceding test results: (On a single TiKV node) the backup speed is proportional to the number of backup threads. When the number of threads is small, the backup speed is about 20 MB/thread. For example, a single node with 5 backup threads can deliver a backup speed of 100 MB/s.

> **Note:**
>
> The impact and speed of backup depends much on cluster configuration, deployment, and running services. The preceding test conclusions, based on simulation tests in many scenarios and verified in some customer sites, are worthy of reference. However, the exact impact and performance cap may vary depending on the scenarios. Therefore, you should always run the test and verify the test results.

Since v5.3.0, BR introduces the auto tunning feature (enabled by default) to adjust the number of backup threads. It can maintain the CPU utilization of the cluster below 80% during backup tasks. For details, see BR Auto-Tune.

13.10.4 Use BR to Restore Cluster Data

This document describes how to restore TiDB cluster data in the following scenarios:

- Restore TiDB cluster snapshots
- Restore a database
- Restore a table
- Restore multiple tables with table filter
- Restore backup data from external storage

- Restore incremental data
- Restore encrypted backup data
- Restore tables created in the `mysql` schema

If you are not familiar with backup and restore tools, it is recommended that you read the following documents to fully understand usage principles and methods of these tools:

- BR Overview
- Use BR Command-line for Backup and Restoration

If you need to restore data exported by Dumpling, CSV files, or Apache Parquet files generated by Amazon Aurora, you can use TiDB Lightning to import data to implement restore. For details, see Use TiDB Lightning to restore full data.

13.10.4.1 Restore TiDB cluster snapshots

BR supports restoring snapshot backup on an empty cluster to restore the target cluster to the latest state when the snapshot is backed up.

Example: Restore the snapshot generated at `2022-01-30 07:42:23` from the `2022-01-30/` directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file restorefull.log
```

In the preceding command,

- `--ratelimit`: The maximum speed for **each TiKV** to perform a restoration task (unit: MiB/s)
- `--log-file` The target file for BR logging

During restoration, a progress bar is displayed in the terminal, as shown below. When the progress bar advances to 100%, the restoration is complete. To ensure data security, BR performs a check on the restored data.

```
br restore full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/" \
    --ratelimit 128 \
    --log-file restorefull.log
Full Restore <---------/.....................................>
    ↪ 17.12%.
```

13.10.4.2 Restore a database or a table

BR supports restoring partial data of a specified database or table from backup data. This feature allows you to filter out unwanted data and back up only a specific database or table.

13.10.4.2.1 Restore a database

To restore a database to the cluster, run the `br restore db` command. To get help on this command, run the `br restore db --help` command.

Example: Restore the `test` database from the `db-test/2022-01-30/` directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore db \
    --pd "${PDIP}:2379" \
    --db "test" \
    --ratelimit 128 \
    --storage "s3://backup-data/db-test/2022-01-30/" \
    --log-file restore_db.log
```

In the preceding command, `--db` specifies the name of the database to be restored, and other parameters are the same as those in Restore TiDB cluster snapshots.

> **Note:**
>
> When you restore the backup data, the database name specified by `--db` must be the same as the one specified by `-- db` in the backup command. Otherwise, the restoration fails. This is because the metafile of the backup data (`backupmeta` file) records the database name, and you can only restore data to the database with the same name. The recommended method is to restore the backup data to the database with the same name in another cluster.

13.10.4.2.2 Restore a table

To restore a single table to the cluster, run the `br restore table` command. To get help on this command, run the `br restore table --help` command.

Example: Restore `test.usertable` from the `table-db-usertable/2022-01-30/` directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore table \
    --pd "${PDIP}:2379" \
    --db "test" \
```

```
    --table "usertable" \
    --ratelimit 128 \
    --storage "s3://backup-data/table-db-usertable/2022-01-30/" \
    --log-file restore_table.log
```

In the preceding command, `--table` specifies the name of the table to be restored, and other parameters are the same as those in Restore TiDB cluster snapshots.

13.10.4.2.3 Restore multiple tables with table filter

To restore multiple tables with more criteria, run the `br restore full` command and specify the table filters with `--filter` or `-f`.

Example: Restore data matching the `db*.tbl*` table from the `table-filter` ↪ `/2022-01-30/` directory in the `backup-data` bucket of Amazon S3 to the target cluster.

```
br restore full \
    --pd "${PDIP}:2379" \
    --filter 'db*.tbl*' \
    --storage "s3://backup-data/table-filter/2022-01-30/" \
    --log-file restorefull.log
```

13.10.4.3 Restore backup data from external storage

BR supports restoring data to Amazon S3, Google Cloud Storage (GCS), Azure Blob Storage, NFS, or other S3-compatible file storage services. For details, see the following documents:

- Restore data on Amazon S3 using BR
- Restore data on Google Cloud Storage using BR
- Restore data on Azure Blob Storage using BR

13.10.4.4 Restore incremental data

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

Restoring incremental data is similar to restoring full data using BR. When restoring incremental data, make sure that all the data backed up before `last backup ts` has been

restored to the target cluster. Also, because incremental restoration updates ts data, you need to ensure that there are no other writes during the restoration. Otherwise, conflicts might occur.

```
br restore full \
    --pd "${PDIP}:2379" \
    --storage "s3://backup-data/2022-01-30/incr" \
    --ratelimit 128 \
    --log-file restorefull.log
```

13.10.4.5 Restore encrypted backup data

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

After encrypting the backup data, you need to pass in the corresponding decryption parameters to restore the data. Ensure that the decryption algorithm and key are correct. If the decryption algorithm or key is incorrect, the data cannot be restored.

```
br restore full\
    --pd ${PDIP}:2379 \
    --storage "s3://backup-data/2022-01-30/" \
    --crypter.method aes128-ctr \
    --crypter.key 0123456789abcdef0123456789abcdef
```

13.10.4.6 Restore tables created in the `mysql` schema

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

BR backs up tables created in the `mysql` schema by default. When you restore data using BR, the tables created in the `mysql` schema are not restored by default. To restore these tables, you can explicitly include them using the table filter. The following example restores `mysql.usertable` created in the `mysql` schema. The command restores `mysql.usertable` along with other data.

```
br restore full -f '*.*' -f '!mysql.*' -f 'mysql.usertable' -s
    ↪ $external_storage_url --ratelimit 128
```

In the preceding command,

- `-f '*.*'` is used to override the default rules
- `-f '!mysql.*'` instructs BR not to restore tables in `mysql` unless otherwise stated.
- `-f 'mysql.usertable'` indicates that `mysql.usertable` should be restored.

If you only need to restore `mysql.usertable`, run the following command:

```
br restore full -f 'mysql.usertable' -s $external_storage_url --ratelimit
    ↪ 128
```

Warning:

Although you can back up system tables (such as `mysql.tidb`) using BR, BR ignores the following system tables even if you use the –filter setting to perform the restoration:

- Statistical information tables (`mysql.stat_*`)
- System variable tables (`mysql.tidb`, `mysql.global_variables`)
- User information tables (such as `mysql.user` and `mysql.columns_priv`)
- Other system tables

Compatibility issues might occur when restoring system tables. Therefore, avoid restoring system tables in production environments.

13.10.4.7 Restoration performance and impact

- TiDB fully uses TiKV CPU, disk IO, network bandwidth, and other resources when restoring data. Therefore, it is recommended that you restore backup data on an empty cluster to avoid affecting running services.
- The restoration speed depends much on cluser configuration, deployment, and running services. Generally, the restoration speed can reach 100 MB/s (per TiKV node).

Note:

The preceding test conclusions, based on simulation tests in many scenarios and verified in some customer sites, are worthy of reference. However, the restoration speed may vary depending on the scenarios. Therefore, you should always run the test and verify the test results.

13.10.5 BR Use Cases

Backup & Restore (BR) is a tool for distributed backup and restoration of the TiDB cluster data.

This document describes common backup and restoration scenarios:

- Back up a single table to a network disk (recommended for production environments)
- Restore data from a network disk (recommended for production environments)
- Back up a single table to a local disk
- Restore data from a local disk

This document aims to help you achieve the following goals:

- Back up and restore data using a network disk or local disk correctly.
- Get the status of a backup or restoration operation through monitoring metrics.
- Learn how to tune performance during the backup or restoration operation.
- Troubleshoot the possible anomalies during the backup operation.

13.10.5.1 Audience

You are expected to have a basic understanding of TiDB and TiKV.

Before reading on, make sure you have read BR Overview, especially Usage Restrictions and Some tips.

13.10.5.2 Prerequisites

This section introduces the recommended method of deploying TiDB, cluster versions, the hardware information of the TiKV cluster, and the cluster configuration for the use case demonstrations.

You can estimate the performance of your backup or restoration operation based on your own hardware and configuration. It is recommended that you use a network disk to back up and restore data. This spares you from collecting backup files and greatly improves the backup efficiency especially when the TiKV cluster is in a large scale.

13.10.5.2.1 Deployment method

It is recommended that you deploy the TiDB cluster using TiUP and install BR using TiUP.

13.10.5.2.2 Cluster versions

- TiDB: v6.1.7
- TiKV: v6.1.7
- PD: v6.1.7
- BR: v6.1.7

> **Note:**
>
> It is recommended that you use the latest version of TiDB/TiKV/PD/BR and make sure that the BR version is **consistent with** the TiDB version.

13.10.5.2.3 TiKV hardware information

- Operating system: CentOS Linux release 7.6.1810 (Core)
- CPU: 16-Core Common KVM processor
- RAM: 32 GB
- Disk: 500 GB SSD * 2
- NIC: 10 Gigabit network card

13.10.5.2.4 Cluster configuration

BR directly sends commands to the TiKV cluster and are not dependent on the TiDB server, so you do not need to configure the TiDB server when using BR.

- TiKV: default configuration
- PD: default configuration

13.10.5.2.5 Others

In addition to the preceding prerequisites, you should also perform the following checks before performing the backup and restoration.

Check before backup

The BR tool already supports self-adapting to GC. It automatically registers `backupTS` ↪ (the latest PD timestamp by default) to PD's `safePoint` to ensure that TiDB's GC

Safe Point does not move forward during the backup, thus avoiding manually setting GC configurations.

Before running the `br backup` command, make sure that the target storage device has required space (no less than 1/3 of the disk space of the backup cluster).

Check before restoration

Before running the `br restore` command, check the target cluster to ensure that the table in this cluster does not have a duplicate name.

13.10.5.3 Back up a single table to a network disk (recommended for production environments)

Run the `br backup` command to back up the single table data `--db batchmark --table order_line` to the specified path `local:///br_data` in the network disk.

13.10.5.3.1 Backup prerequisites

- Check before backup
- Configure a high-performance SSD hard disk host as the NFS server to store data, and all BR nodes, TiKV nodes, and TiFlash nodes as NFS clients. Mount the same path (for example, `/br_data`) to the NFS server for NFS clients to access the server.
- The total transfer rate between the NFS server and all NFS clients must reach at least `the number of TiKV instances * 150MB/s`. Otherwise, the network I/O might become the performance bottleneck.

Note:

- During data backup, because only the data of leader replicas are backed up, even if there is a TiFlash replica in the cluster, BR can complete the backup without mounting TiFlash nodes.
- When restoring data, BR will restore the data of all replicas. Also, TiFlash nodes need access to the backup data for BR to complete the restore. Therefore, before the restore, you must mount TiFlash nodes to the NFS server.

13.10.5.3.2 Topology

The following diagram shows the typology of BR:

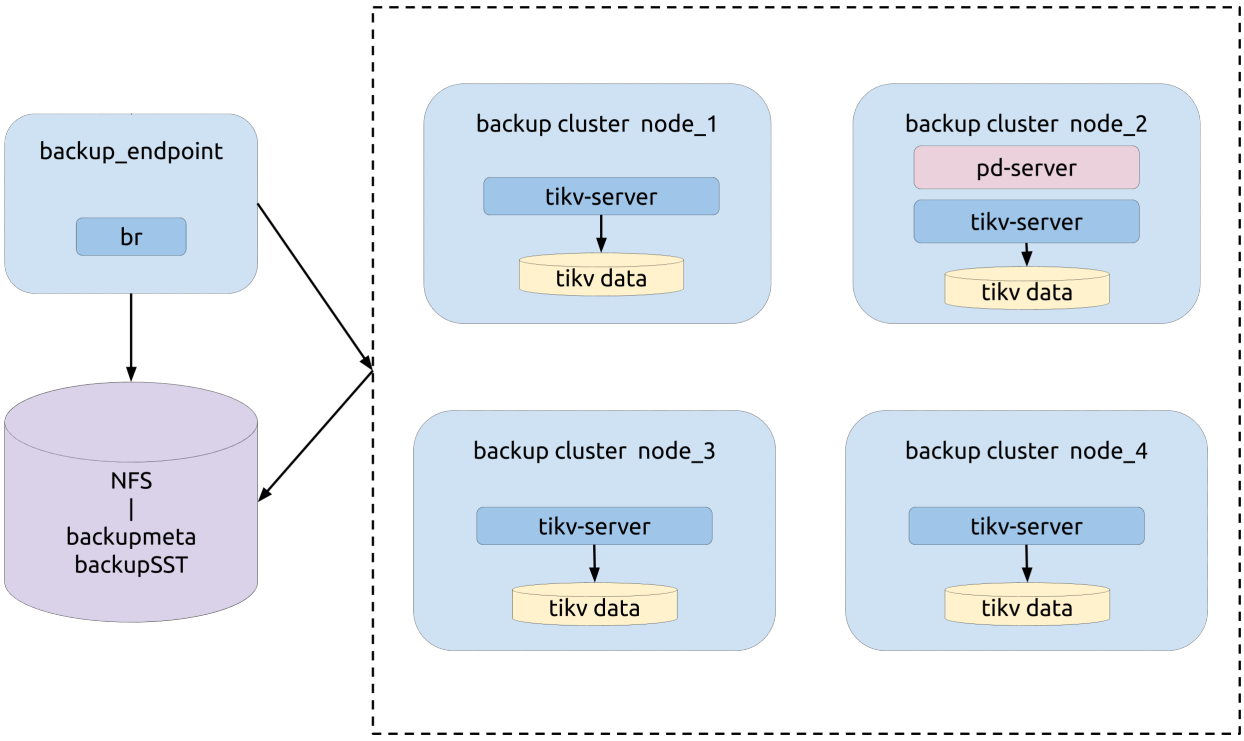


Figure 278: img

13.10.5.3.3 Backup operation

Run the `br backup` command:

```
bin/br backup table \
    --db batchmark \
    --table order_line \
    -s local:///br_data \
    --pd ${PD_ADDR}:2379 \
    --log-file backup-nfs.log
```

13.10.5.3.4 Monitoring metrics for the backup

During the backup process, pay attention to the following metrics on the monitoring panels to get the status of the backup process.

Backup CPU Utilization: the CPU usage rate of each working TiKV node in the backup operation (for example, backup-worker and backup-endpoint).

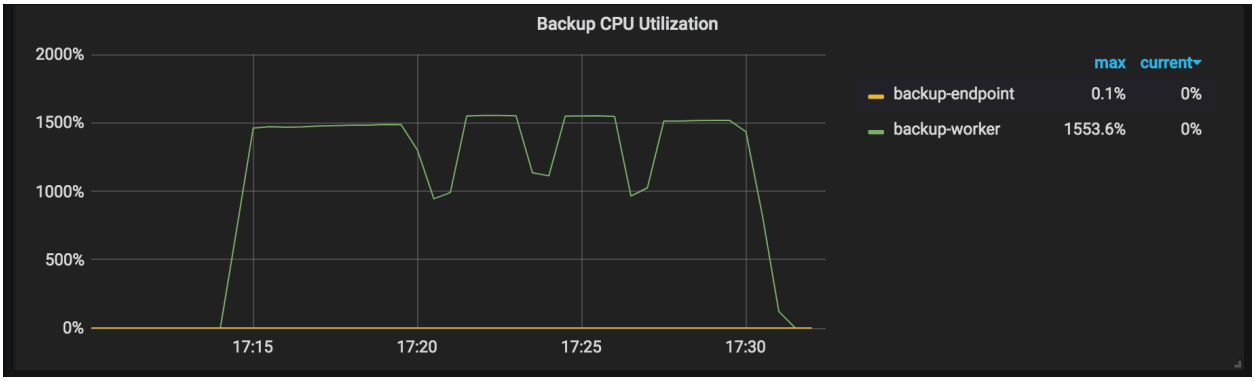


Figure 279: img

IO Utilization: the I/O usage rate of each working TiKV node in the backup operation.

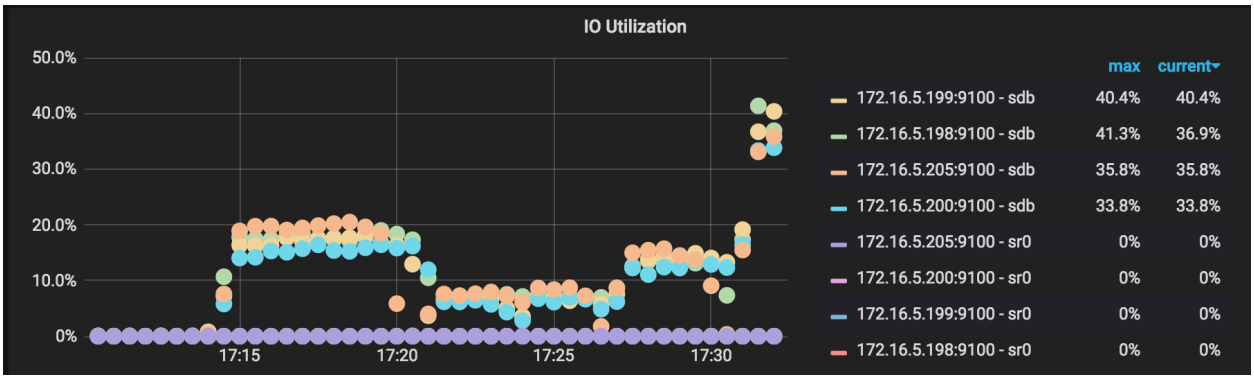


Figure 280: img

BackupSST Generation Throughput: the backupSST generation throughput of each working TiKV node in the backup operation, which is normally around 150 MB/s.

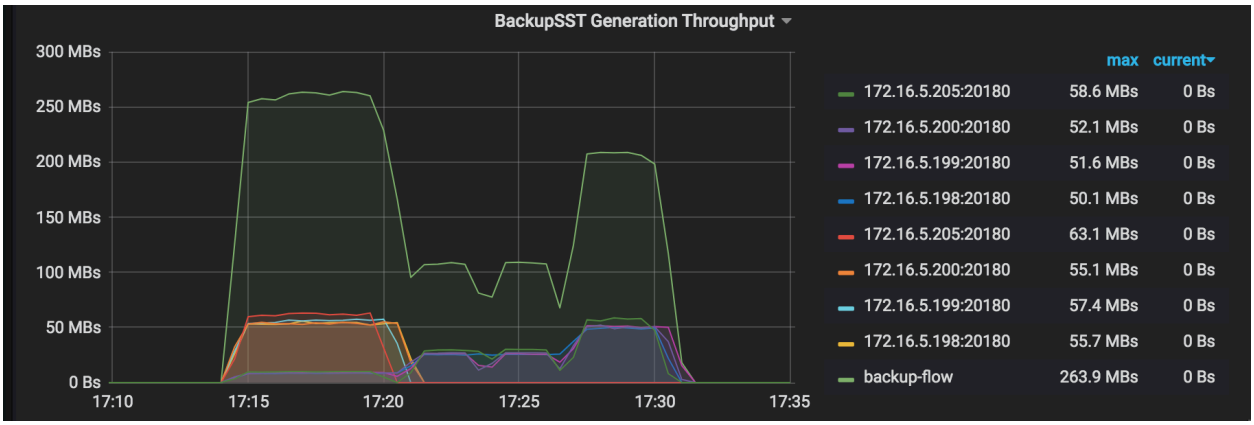


Figure 281: img

One Backup Range Duration: the duration of backing up a range, which is the total time cost of scanning KVs and storing the range as the backupSST file.

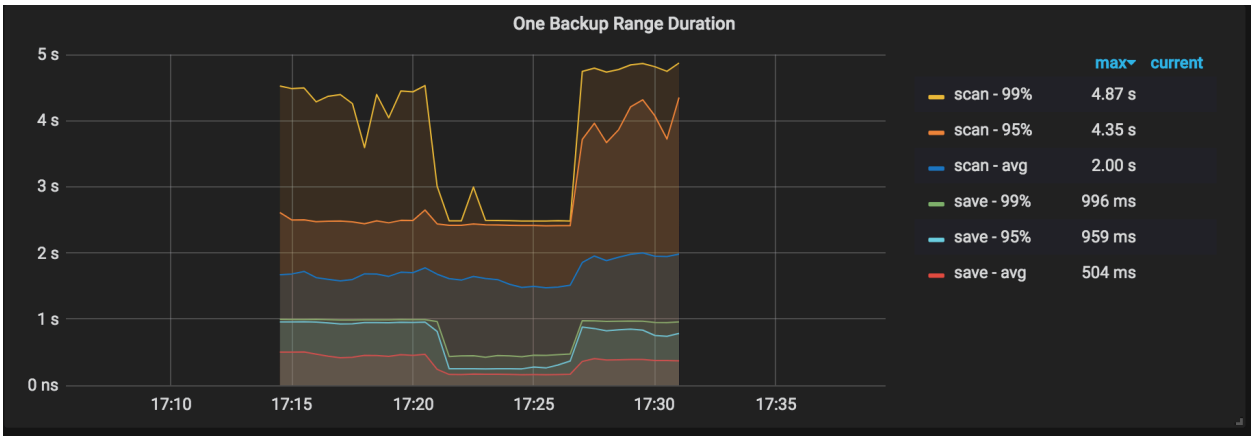


Figure 282: img

One Backup Subtask Duration: the duration of each sub-task into which a backup task is divided.

Note:

- In this task, the single table to be backed up has three indexes and the task is normally divided into four sub-tasks.
- The panel in the following image has 20 points on it, 10 blue and 10 yellow, indicating that there are 10 sub-tasks. Region scheduling might occur during the backup process, so a few retries is normal.

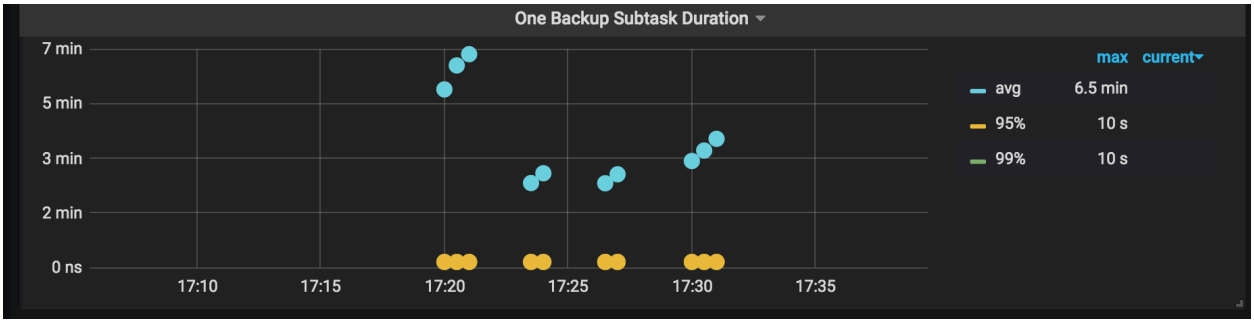


Figure 283: img

Backup Errors: the errors occurred during the backup process. No error occurs in normal situations. Even if a few errors occur, the backup operation has the retry mechanism which might increase the backup time but does not affect the operation correctness.

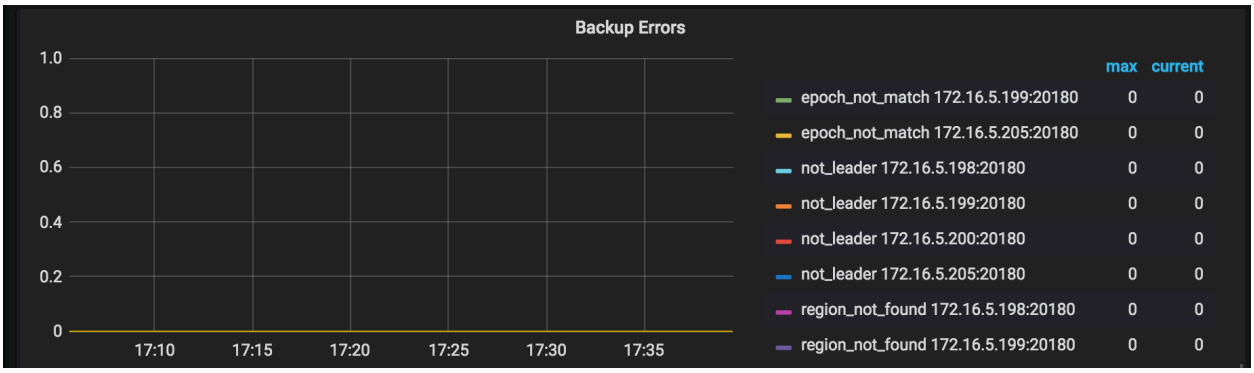


Figure 284: img

Checksum Request Duration: the duration of the admin checksum request in the backup cluster.

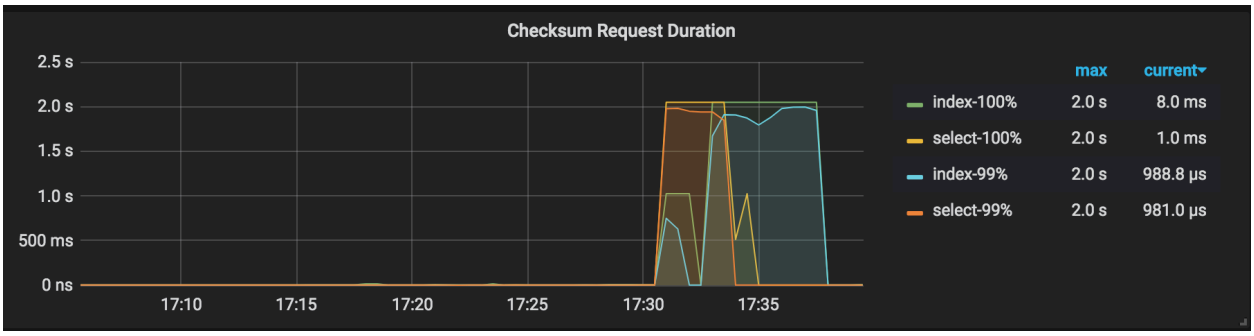


Figure 285: img

13.10.5.3.5 Backup results explanation

When finishing the backup, BR outputs the backup summary to the console.

In the log specified before running the backup command, you can get the statistical information of the backup operation from this log. Search "summary" in this log, you can see the following information:

```
["Full backup Success summary:
    total backup ranges: 2,
    total success: 2,
    total failed: 0,
    total take(Full backup time): 31.802912166s,
    total take(real time): 49.799662427s,
    total size(MB): 5997.49,
    avg speed(MB/s): 188.58,
    total kv: 120000000"]
    ["backup checksum"=17.907153678s]
    ["backup fast checksum"=349.333µs]
    ["backup total regions"=43]
    [BackupTS=422618409346269185]
    [Size=826765915]
```

The preceding log includes the following information:

- `total take(Full backup time)`: Backup duration
- `total take(real time)`: Total runtime of the application
- `total size(MB)`: The size of the backup data
- `avg speed(MB/s)`: Backup throughput
- `total kv`: The number of backed-up KV pairs
- `backup checksum`: Backup checksum duration
- `backup fast checksum`: The total duration of calculating the checksum, KV pairs, and bytes of each table
- `backup total regions`: The total number of backup Regions
- `BackupTS`: The snapshot timestamp of the backup data
- `Size`: The actual size of the backup data in the disk after compression

From the preceding information, the throughput of a single TiKV instance can be calculated: `avg speed(MB/s)/tikv_count` = `62.86`.

13.10.5.3.6 Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the backup process (for example, in the Monitoring metrics for the backup, the highest CPU usage rate of backup-worker is around `1500%` and the overall I/O usage rate is below `30%`), you can try to increase the value of `--concurrency` (`4` by default) to tune the performance. But this

performance tuning method is not suitable for the use cases of many small tables. See the following example:

```
bin/br backup table \
    --db batchmark \
    --table order_line \
    -s local:///br_data/ \
    --pd ${PD_ADDR}:2379 \
    --log-file backup-nfs.log \
    --concurrency 16
```

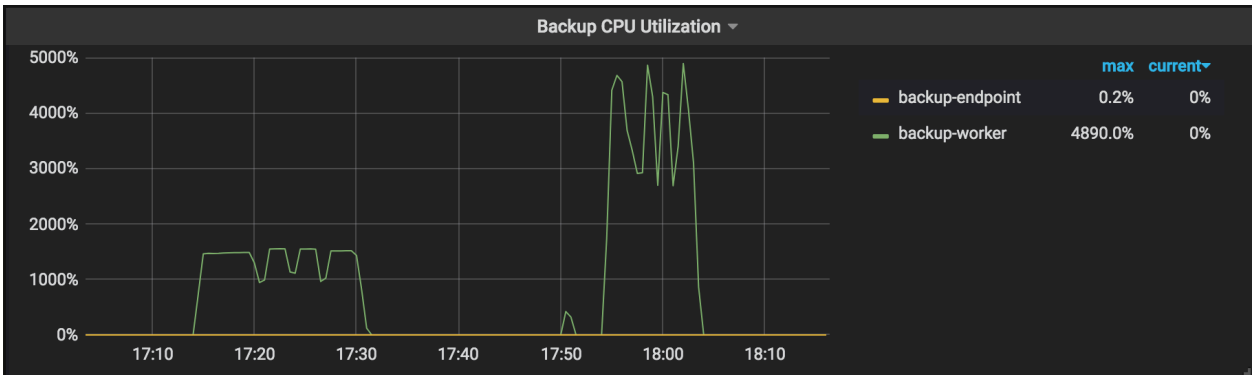


Figure 286: img

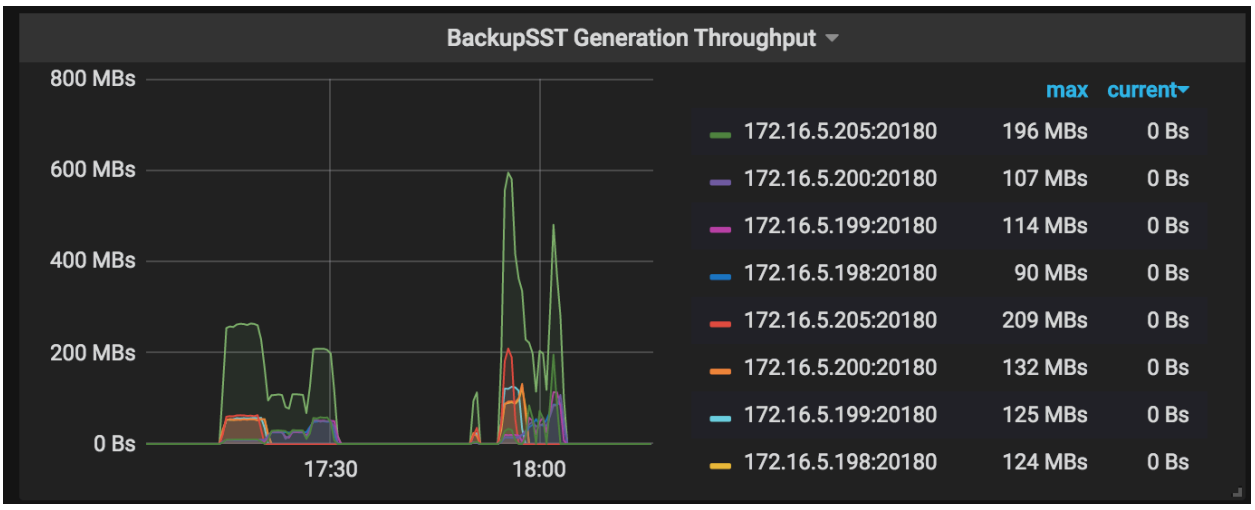


Figure 287: img

The tuned performance results are as follows (with the same data size):

- Backup duration (`total take(s)`): reduced from `986.43` to `535.53`

- Backup throughput (`avg speed(MB/s)`): increased from `358.09` to `659.59`
- Throughput of a single TiKV instance (`avg speed(MB/s)/tikv_count`): increased from `89` to `164.89`

13.10.5.4 Restore data from a network disk (recommended for production environments)

Use the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

13.10.5.4.1 Restoration prerequisites

- Check before restore

13.10.5.4.2 Topology

The following diagram shows the typology of BR:

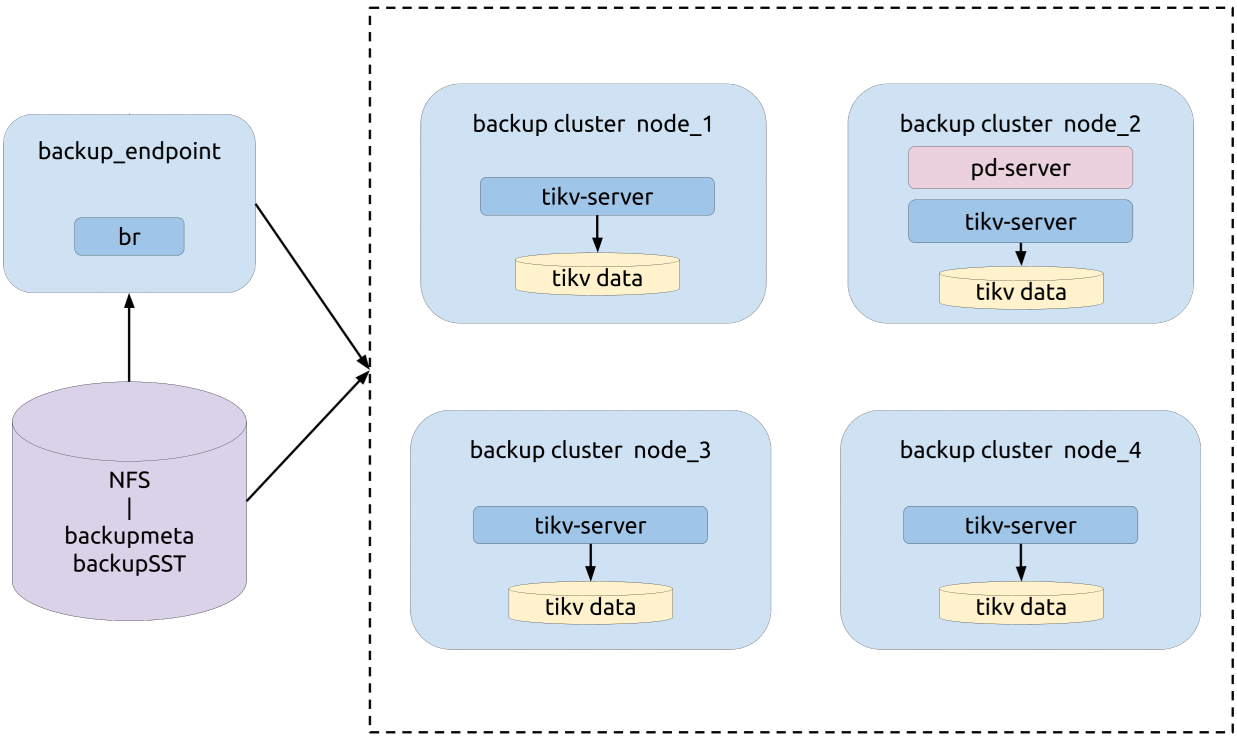


Figure 288: img

13.10.5.4.3 Restoration operation

Run the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data
    ↪ --pd 172.16.5.198:2379 --log-file restore-nfs.log
```

13.10.5.4.4 Monitoring metrics for the restoration

During the restoration process, pay attention to the following metrics on the monitoring panels to get the status of the restoration process.

CPU: the CPU usage rate of each working TiKV node in the restoration operation.

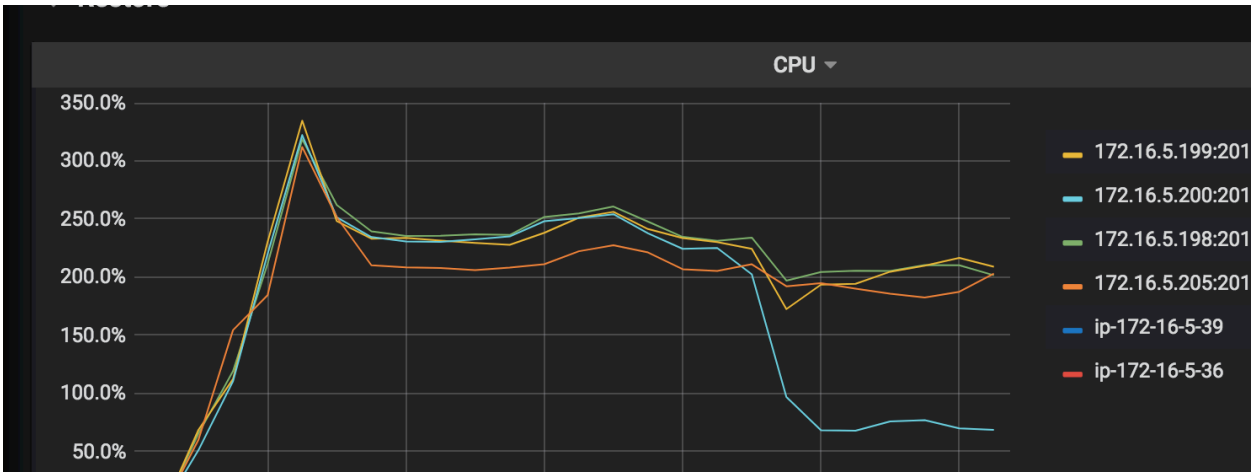


Figure 289: img

IO Utilization: the I/O usage rate of each working TiKV node in the restoration operation.

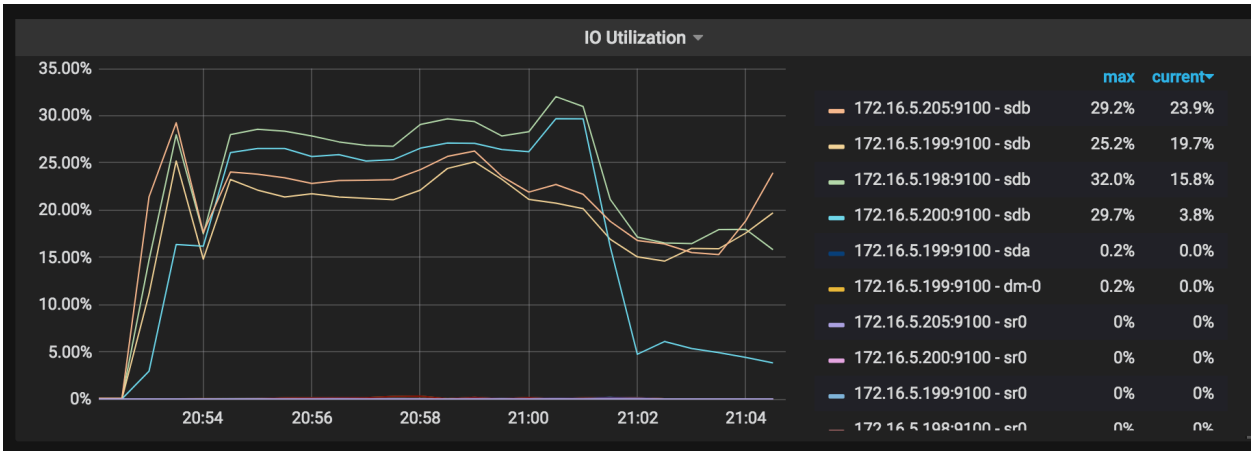


Figure 290: img

Region: the Region distribution. The more even Regions are distributed, the better the restoration resources are used.

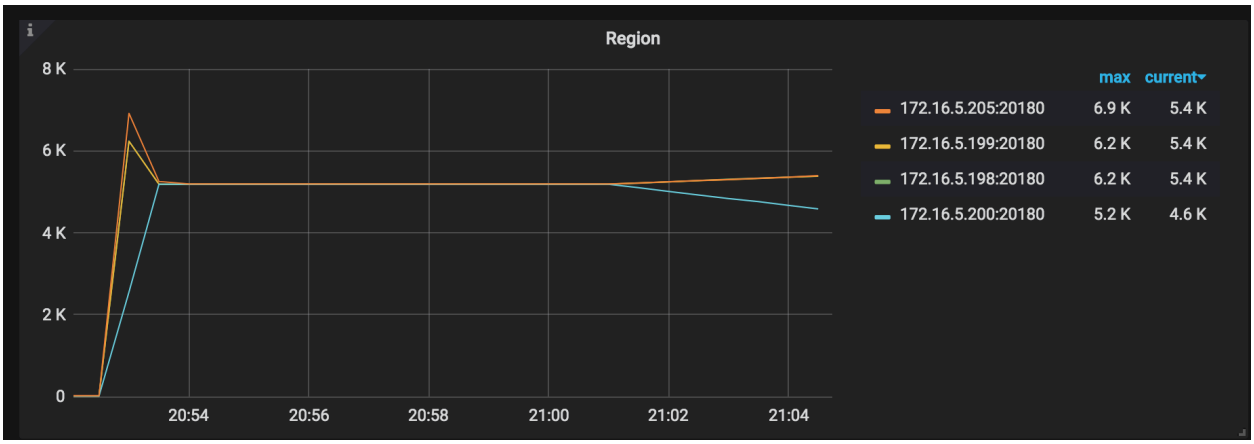


Figure 291: img

Process SST Duration: the delay of processing the SST files. When restoring a table, if `tableID` is changed, you need to rewrite `tableID`. Otherwise, `tableID` is renamed. Generally, the delay of rewriting is longer than that of renaming.

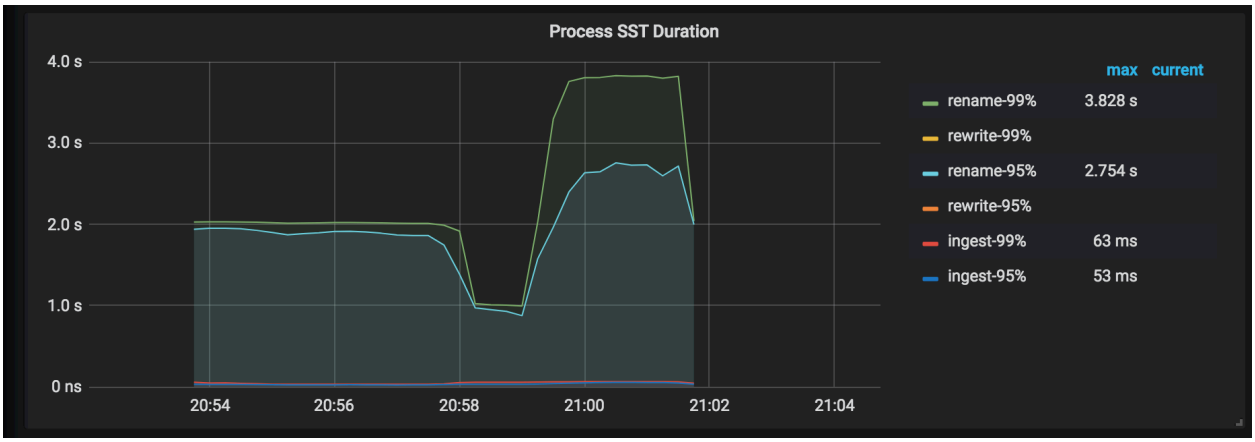


Figure 292: img

DownLoad SST Throughput: the throughput of downloading SST files from External Storage.

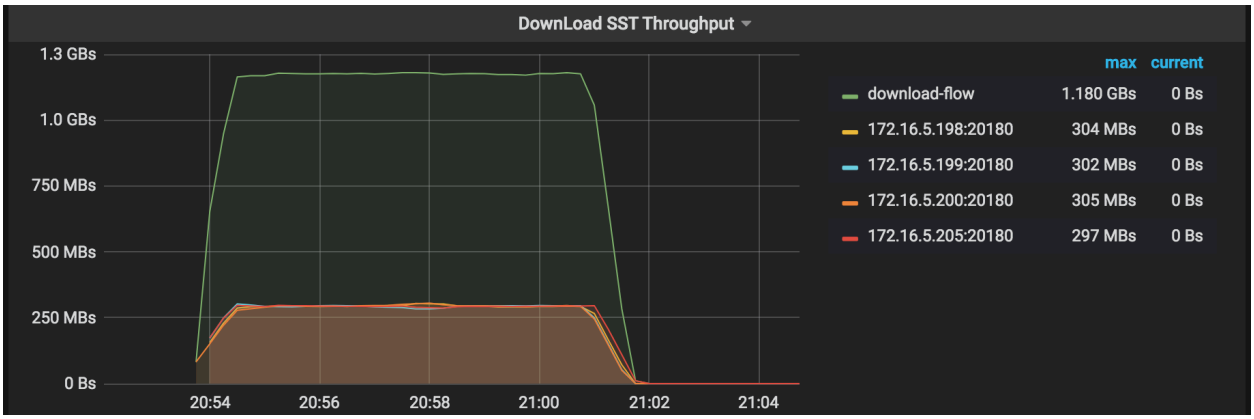


Figure 293: img

Restore Errors: the errors occurred during the restoration process.

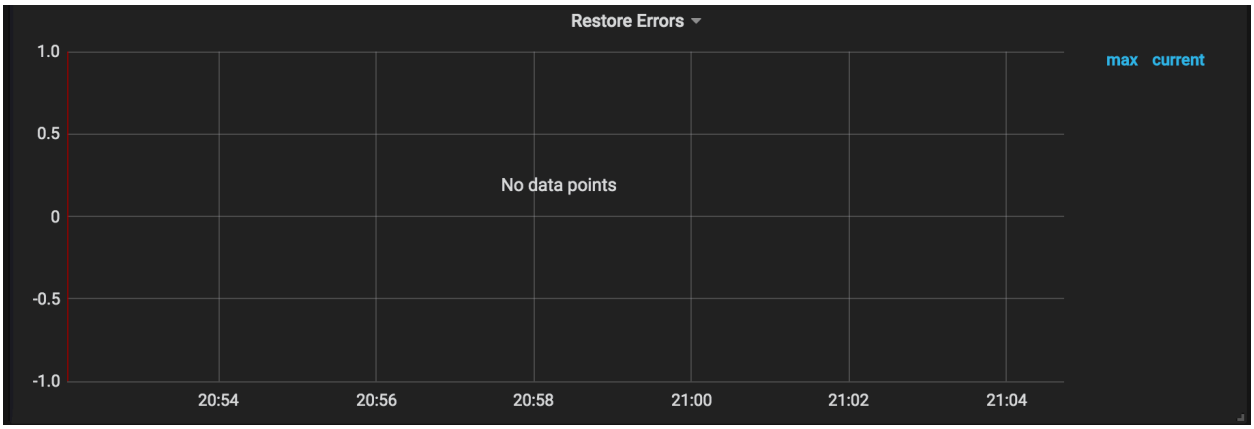


Figure 294: img

Checksum Request Duration: the duration of the admin checksum request. This duration for the restoration is longer than that for the backup.

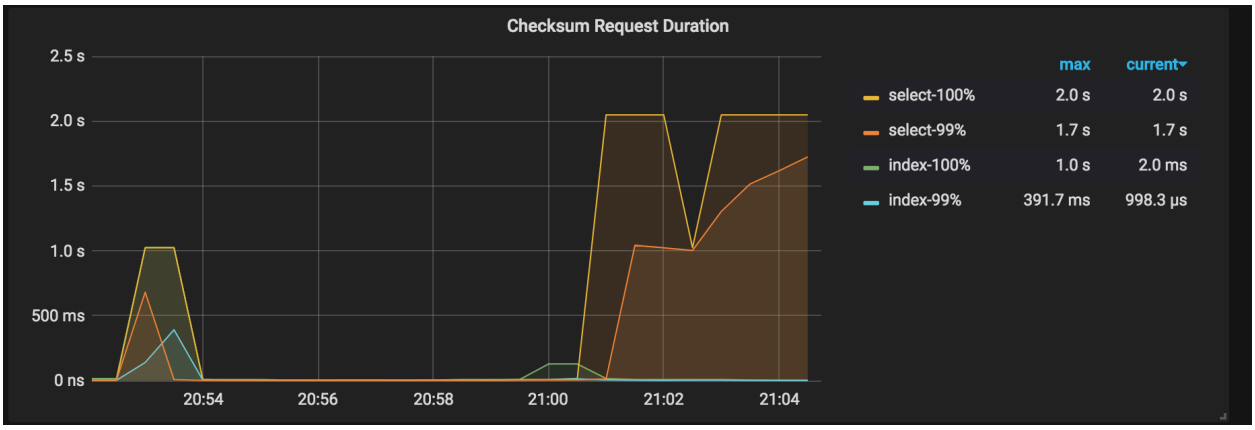


Figure 295: img

13.10.5.4.5 Restoration results explanation

In the log specified before running the restoration command, you can get the statistical information of the restoration operation from this log. Search "summary" in this log, you can see the following information:

```
["Table Restore summary:
    total restore tables: 1,
    total success: 1,
    total failed: 0,
    total take(Full restore time): 17m1.001611365s,
    total take(real time): 16m1.371611365s,
    total kv: 5659888624,
    total size(MB): 353227.18,
    avg speed(MB/s): 367.42"]
    ["restore files"=9263]
    ["restore ranges"=6888]
    ["split region"=49.049182743s]
    ["restore checksum"=6m34.879439498s]
    [Size=48693068713]
```

The preceding log includes the following information:

- `total take(Full restore time)`: The restoration duration
- `total take(real time)`: The total runtime of the application
- `total size(MB)`: The size of the data to be restored
- `total kv`: The number of restored KV pairs
- `avg speed(MB/s)`: The restoration throughput
- `split region`: The Region split duration
- `restore checksum`: The restoration checksum duration

- `Size`: The actual size of the restored data in the disk

From the preceding information, the following items can be calculated:

- The throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` = `91.8`
- The average restore speed of a single TiKV instance: `total size(MB)/(split time + restore time)/tikv_count` = `87.4`

Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the restore process, you can increase the value of `--concurrency` (defaults to `128`). See the following example:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data/
    ↪ --pd 172.16.5.198:2379 --log-file restore-concurrency.log --
    ↪ concurrency 1024
```

The tuned performance results are as follows (with the same data size):

- Restoration duration (`total take(s)`): reduced from `961.37` to `443.49`
- Restoration throughput (`avg speed(MB/s)`): increased from `367.42` to `796.47`
- Throughput of a single TiKV instance (`avg speed(MB/s)/tikv_count`): increased from `91.8` to `199.1`
- Average restore speed of a single TiKV instance (`total size(MB)/(split time + restore time)/tikv_count`): increased from `87.4` to `162.3`

13.10.5.5 Back up a single table to a local disk (recommended for testing environments)

Run the `br backup` command to back up a single table `--db batchmark --table order_line` to the specified path `local:///home/tidb/backup_local` in the local disk.

13.10.5.5.1 Backup prerequisites

- Check before backup
- Each TiKV node has a separate disk to store backupSST files.
- The `backup_endpoint` node has a separate disk to store `backupmeta` files.
- TiKV and the `backup_endpoint` node share the same directory (for example, `/home/tidb/backup_local`) for backup.

13.10.5.5.2 Topology

The following diagram shows the typology of BR:

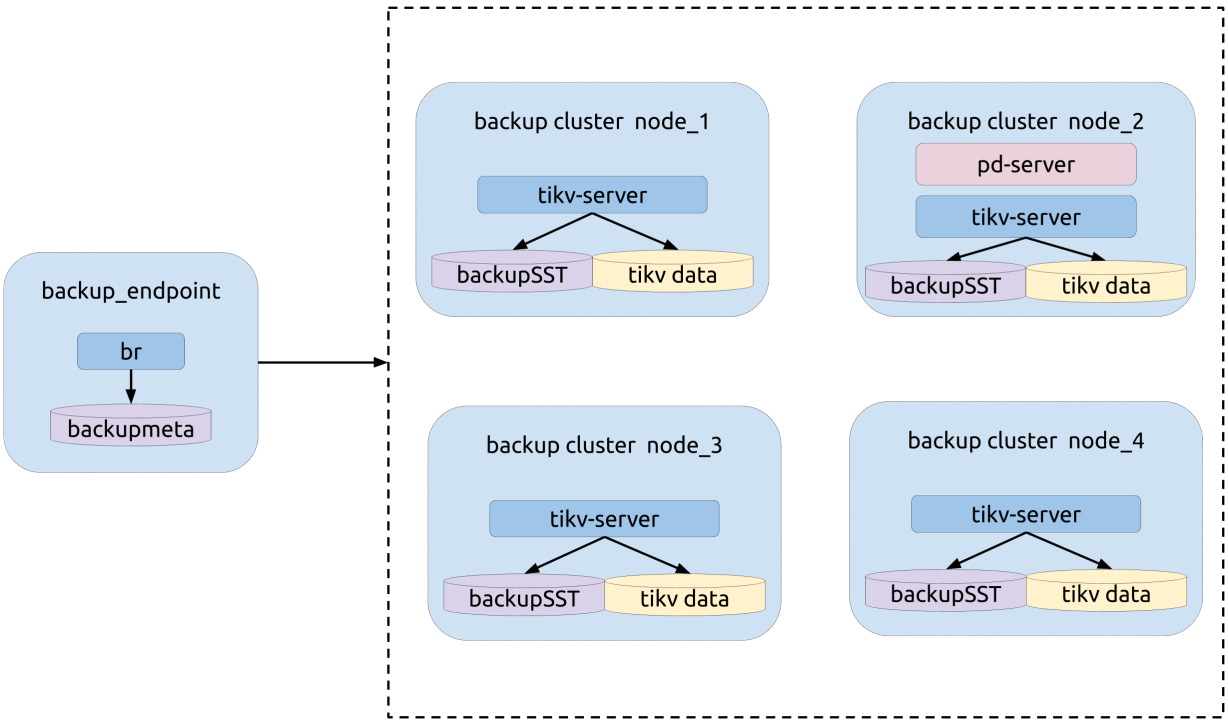


Figure 296: img

13.10.5.5.3 Backup operation

Run the `br backup` command:

```
bin/br backup table \
    --db batchmark \
    --table order_line \
    -s local:///home/tidb/backup_local/ \
    --pd ${PD_ADDR}:2379 \
    --log-file backup_local.log
```

During the backup process, pay attention to the metrics on the monitoring panels to get the status of the backup process. See Monitoring metrics for the backup for details.

Backup results explanation

In the log specified before running the backup command, you can get the statistical information of the restoration operation from this log. Search "summary" in this log, you can see the following information:

```
["Table backup summary:
```

```
total backup ranges: 4,
total success: 4,
total failed: 0,
total take(s): 551.31,
total kv: 5659888624,
total size(MB): 353227.18,
avg speed(MB/s): 640.71"]
["backup total regions"=6795]
["backup checksum"=6m33.962719217s]
["backup fast checksum"=22.995552ms]
```

The preceding log includes the following information:

- `total take(s)`: The backup duration
- `total size(MB)`: The data size
- `avg speed(MB/s)`: The backup throughput
- `backup checksum`: The backup checksum duration

From the preceding information, the throughput of a single TiKV instance can be calculated: `avg speed(MB/s)/tikv_count` = 160.

13.10.5.6 Restore data from a local disk (recommended for testing environments)

Run the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

13.10.5.6.1 Restoration prerequisites

- Check before restore
- The TiKV cluster and the backup data do not have a duplicate database or table. Currently, BR does not support table route.
- Each TiKV node has a separate disk to store backupSST files.
- The `restore_endpoint` node has a separate disk to store `backupmeta` files.
- TiKV and the `restore_endpoint` node share the same directory (for example, `/home/tidb/backup_local/`) for restoration.

Before the restoration, follow these steps:

1. Collect all backupSST files into the same directory.
2. Copy the collected backupSST files to all TiKV nodes of the cluster.
3. Copy the `backupmeta` files to the `restore endpoint` node.

13.10.5.6.2 Topology

The following diagram shows the typology of BR:

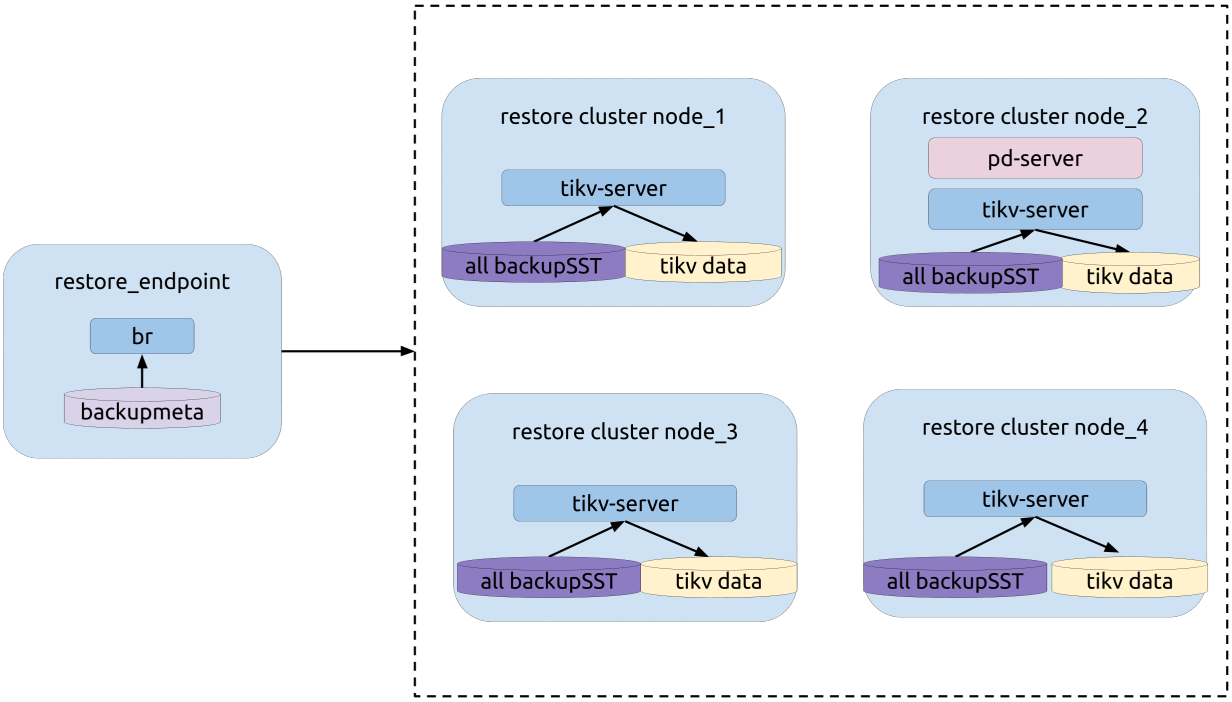


Figure 297: img

13.10.5.6.3 Restoration operation

Run the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///home/tidb
    ↪ /backup_local/ --pd 172.16.5.198:2379 --log-file restore_local.log
```

During the restoration process, pay attention to the metrics on the monitoring panels to get the status of the restoration process. See Monitoring metrics for the restoration for details.

13.10.5.6.4 Restoration results explanation

In the log specified before running the restoration command, you can get the statistical information of the restoration operation from this log. Search "summary" in this log, you can see the following information:

```
["Table Restore summary:
    total restore tables: 1,
    total success: 1,
    total failed: 0,
```

```
total take(s): 908.42,
total kv: 5659888624,
total size(MB): 353227.18,
avg speed(MB/s): 388.84"]
["restore files"=9263]
["restore ranges"=6888]
["split region"=58.7885518s]
["restore checksum"=6m19.349067937s]
```

The preceding log includes the following information:

- `total take(s)`: The restoration duration
- `total size(MB)`: The data size
- `avg speed(MB/s)`: The restoration throughput
- `split region`: The region split duration
- `restore checksum`: The restoration checksum duration

From the preceding information, the following items can be calculated:

- The throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` = 97.2
- The average restoration speed of a single TiKV instance: `total size(MB)/(split time + restore time)/tikv_count` = 92.4

13.10.5.7 Error handling during backup

This section introduces the common errors that might occur during the backup process.

13.10.5.7.1 `key locked Error` in the backup log

Error message in the log: `log - ["backup occur kv error"][error="{\"KvError \":{\"locked\":`

If a key is locked during the backup process, BR tries to resolve the lock. A small number of this error do not affect the correctness of the backup.

13.10.5.7.2 Backup failure

Error message in the log: `log - Error: msg:"Io(Custom { kind: AlreadyExists, error: \"[5_5359_42_123_default.sst] is already exists in /dir/backup_local /\" })"`

If the backup operation fails and the preceding message occurs, perform one of the following operations and then start the backup operation again:

- Change the directory for the backup. For example, change `/dir/backup-2020-01-01/` to `/dir/backup_local/`.
- Delete the backup directory of all TiKV nodes and BR nodes.

13.10.6 BR Features

13.10.6.1 BR Auto-Tune New in v5.4.0

Before TiDB v5.4.0, when you back up data using BR, the number of threads used for backup makes up 75% of the logical CPU cores. Without a speed limit, the backup process can consume a lot of cluster resources, which has a considerable impact on the performance of the online cluster. Although you can reduce the impact of backup by adjusting the size of the thread pool, it is a tedious task to observe the CPU load and manually adjust the thread pool size.

To reduce the impact of backup tasks on the cluster, starting from TiDB v5.4.0, BR introduces the auto-tune feature. When the cluster resource utilization is high, BR automatically limits the resources used by backup tasks and thereby reduces their impact on the cluster. The auto-tune feature is enabled by default.

13.10.6.1.1 User scenario

If you want to reduce the impact of backup tasks on the cluster, you can enable the auto-tune feature. With this feature enabled, BR performs backup tasks as fast as possible without excessively affecting the cluster.

Alternatively, you can limit the backup speed by using the TiKV configuration item `backup.num-threads` or using the parameter `--ratelimit`. When `--ratelimit` is set, to avoid too many tasks causing the speed limit to fail, the `concurrency` parameter of br is automatically adjusted to `1`.

13.10.6.1.2 Use auto-tune

The auto-tune feature is enabled by default, without additional configuration.

> **Note:**
>
> For clusters that upgrade from v5.3.x to v5.4.0 or later versions, the auto-tune feature is disabled by default. You need to manually enable it.

To manually enable the auto-tune feature, you need to set the TiKV configuration item `backup.enable-auto-tune` to `true`.

TiKV supports dynamically configuring the auto-tune feature. You can enable or disable the feature without restarting your cluster. To dynamically enable or disable the auto-tune feature, run the following command:

```
tikv-ctl modify-tikv-config -n backup.enable-auto-tune -v <true|false>
```

When you perform backup tasks on an offline cluster, to speed up the backup, you can modify the value of `backup.num-threads` to a larger number using `tikv-ctl`.

13.10.6.1.3 Limitations

Auto-tune is a coarse-grained solution for limiting backup speed. It reduces the need for manual tuning. However, because of the lack of fine-grained control, auto-tune might not be able to completely remove the impact of backup on the cluster.

The auto-tune feature has the following issues and corresponding solutions:

- Issue 1: For **write-heavy clusters**, auto-tune might put the workload and backup tasks into a "positive feedback loop": the backup tasks take up too many resources, which causes the cluster to use fewer resources; at this point, auto-tune might mistakenly assume that the cluster is not under heavy workload and thus allow BR to run faster. In such cases, auto-tune is ineffective.

    - Solution: Manually adjust `backup.num-threads` to a smaller number to limit the number of threads used by backup tasks. The working principle is as follows:

      The backup process includes lots of SST decoding, encoding, compression, and decompression, which consume CPU resources. In addition, previous test cases have shown that during the backup process, the CPU utilization of the thread pool used for backup is close to 100%. This means that the backup tasks take up a lot of CPU resources. By adjusting the number of threads used by the backup tasks, TiKV can limit the CPU cores used by backup tasks, thus reducing the impact of backup tasks on the cluster performance.

- Issue 2: For **clusters with hotspots**, backup tasks on the TiKV node that has hotspots might be excessively limited, which slows down the overall backup process.

    - Solution: Eliminate the hotspot node, or disable auto-tune on the hotspot node (this might reduce the cluster performance).

- Issue 3: For scenarios with **high traffic jitter**, because auto-tune adjusts the speed limit on a fixed interval (1 minute by default), it might not be able to handle high traffic jitter. For details, see `auto-tune-refresh-interval`.

    - Solution: Disable auto-tune.

13.10.6.1.4 Implementation

Auto-tune adjusts the size of the thread pool used by backup tasks using BR to ensure that the overall CPU utilization of the cluster does not exceed a specific threshold.

This feature has two related configuration items not listed in the TiKV configuration file. These two configuration items are only for internal tuning. You do **not** need to configure these two configuration items when you perform backup tasks.

- `backup.auto-tune-remain-threads`:

- Auto-tune controls the resources used by the backup tasks and ensures that at least `backup.auto-tune-remain-threads` cores are available for other tasks on the same node.
- Default value: `round(0.2 * vCPU)`

- `backup.auto-tune-refresh-interval`:
  - Every `backup.auto-tune-refresh-interval` minute(s), auto-tune refreshes the statistics and recalculates the maximum number of CPU cores that backup tasks can use.
  - Default value: `1m`

The following is an example of how auto-tune works. `*` denotes a CPU core used by backup tasks. `^` denotes a CPU core used by other tasks. `-` denotes an idle CPU core.

```
|--------| The server has 8 logical CPU cores.
|****----| By default, `backup.num-threads` is `4`. Note that auto-tune
    ↪ makes sure that the thread pool size is never larger than `backup.num
    ↪ -threads`.
|^^****--| By default, `auto-tune-remain-threads` = round(8 * 0.2) = 2. Auto
    ↪ -tune adjusts the size of the thread pool to `4`.
|^^^^**--| Because the cluster workload gets higher, auto-tune adjusts the
    ↪ size of the thread pool to `2`. After that, the cluster still has 2
    ↪ idle CPU cores.
```

In the **Backup CPU Utilization** panel, you can see the size of the thread pool adjusted by auto-tune:

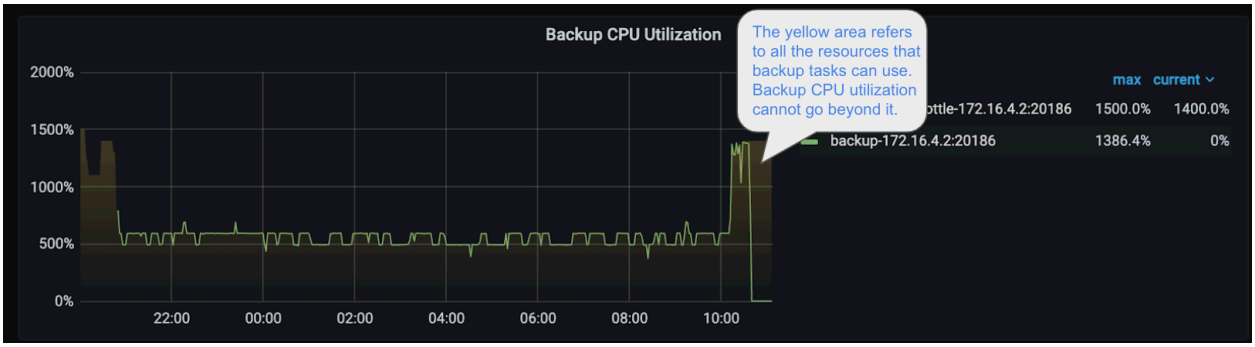


Figure 298: Grafana dashboard example of backup auto-tune metrics

In the image above, the yellow semi-transparent area represents the threads available for backup tasks. You can see the CPU utilization of backup tasks does not go beyond the yellow area.

13.10.6.2 Batch Create Table

When restoring data, Backup & Restore (BR) creates databases and tables in the target TiDB cluster and then restores the backup data to the tables. In versions earlier than TiDB v6.0.0, BR uses the serial execution implementation to create tables in the restore process. However, when BR restores data with a large number of tables (nearly 50000), this implementation takes much time on creating tables.

To speed up the table creation process and reduce the time for restoring data, the Batch Create Table feature is introduced in TiDB v6.0.0. This feature is enabled by default.

Note:

- To use the Batch Create Table feature, both TiDB and BR are expected to be of v6.0.0 or later. If either TiDB or BR is earlier than v6.0.0, BR uses the serial execution implementation.
- Suppose that you use a cluster management tool (for example, TiUP), and your TiDB and BR are of v6.0.0 or later versions, or your TiDB and BR are upgraded from a version earlier than v6.0.0 to v6.0.0 or later. In this case, BR enables the Batch Create Table feature by default.

13.10.6.2.1 Usage scenario

If you need to restore data with a massive amount of tables, for example, 50000 tables, you can use the Batch Create Table feature to speed up the restore process.

For the detailed effect, see Test for the Batch Create Table Feature.

13.10.6.2.2 Use the Batch Create Table feature

BR enables the Batch Create Table feature by default, with the default configuration of `--ddl-batch-size=128` in v6.0.0 or later to speed up the restore process. Therefore, you do not need to configure this parameter. `--ddl-batch-size=128` means that BR creates tables in batches, each batch with 128 tables.

To disable this feature, you can set `--ddl-batch-size` to `1`. See the following example command:

```
br restore full -s local:///br_data/ --pd 172.16.5.198:2379 --log-file
    ↪ restore.log --ddl-batch-size=1
```

After this feature is disabled, BR uses the serial execution implementation instead.

13.10.6.2.3 Implementation principles

- Serial execution implementation before v6.0.0:

  When restoring data, BR creates databases and tables in the target TiDB cluster and then restores the backup data to the tables. To create tables, BR calls TiDB internal API first, and then processes table creation tasks, which works similarly to executing the `Create Table` statement by BR. The TiDB DDL owner creates tables sequentially. Once the DDL owner creates a table, the DDL schema version changes correspondingly and each version change is synchronized to other TiDB DDL workers (including BR). Therefore, when BR restores a large number of tables, the serial execution implementation is time-consuming.

- Batch create table implementation since v6.0.0:

  By default, BR creates tables in multiple batches, and each batch has 128 tables. Using this implementation, when BR creates one batch of tables, the TiDB schema version only changes once. This implementation significantly increases the speed of table creation.

13.10.6.2.4 Test for the Batch Create Table feature

This section describes the test information about the Batch Create Table feature. The test environment is as follows:

- Cluster configurations:
  - 15 TiKV instances. Each TiKV instance is equipped with 16 CPU cores, 80 GB memory, and 16 threads to process RPC requests (`import.num-threads` = 16).
  - 3 TiDB instances. Each TiDB instance is equipped with 16 CPU cores, 32 GB memory.
  - 3 PD instances. Each PD instance is equipped with 16 CPU cores, 32 GB memory.
- The size of data to be restored: 16.16 TB

The test result is as follows:

```
'[2022/03/12 22:37:49.060 +08:00] [INFO] [collector.go:67] ["Full restore
   ↪ success summary"] [total-ranges=751760] [ranges-succeed=751760] [
   ↪ ranges-failed=0] [split-region=1h33m18.078448449s] [restore-ranges
   ↪ =542693] [total-take=1h41m35.471476438s] [restore-data-size(after-
   ↪ compressed)=8.337TB] [Size=8336694965072] [BackupTS
   ↪ =431773933856882690] [total-kv=148015861383] [total-kv-size=16.16TB]
   ↪ [average-speed=2.661GB/s]'
```

From the test result, you can see that the average speed of restoring one TiKV instance is as high as 181.65 MB/s (which equals to `average-speed/tikv_count`) .

13.10.7 References

13.10.7.1 BR Design Principles

This document describes the design principles of Backup & Restore (BR), including its architecture and backup files.

13.10.7.1.1 BR architecture

BR sends a backup or restoration command to each TiKV node. After receiving the command, TiKV performs the corresponding backup or restoration operation.

Each TiKV node has a path in which the backup files generated in the backup operation are stored and from which the stored backup files are read during the restoration.

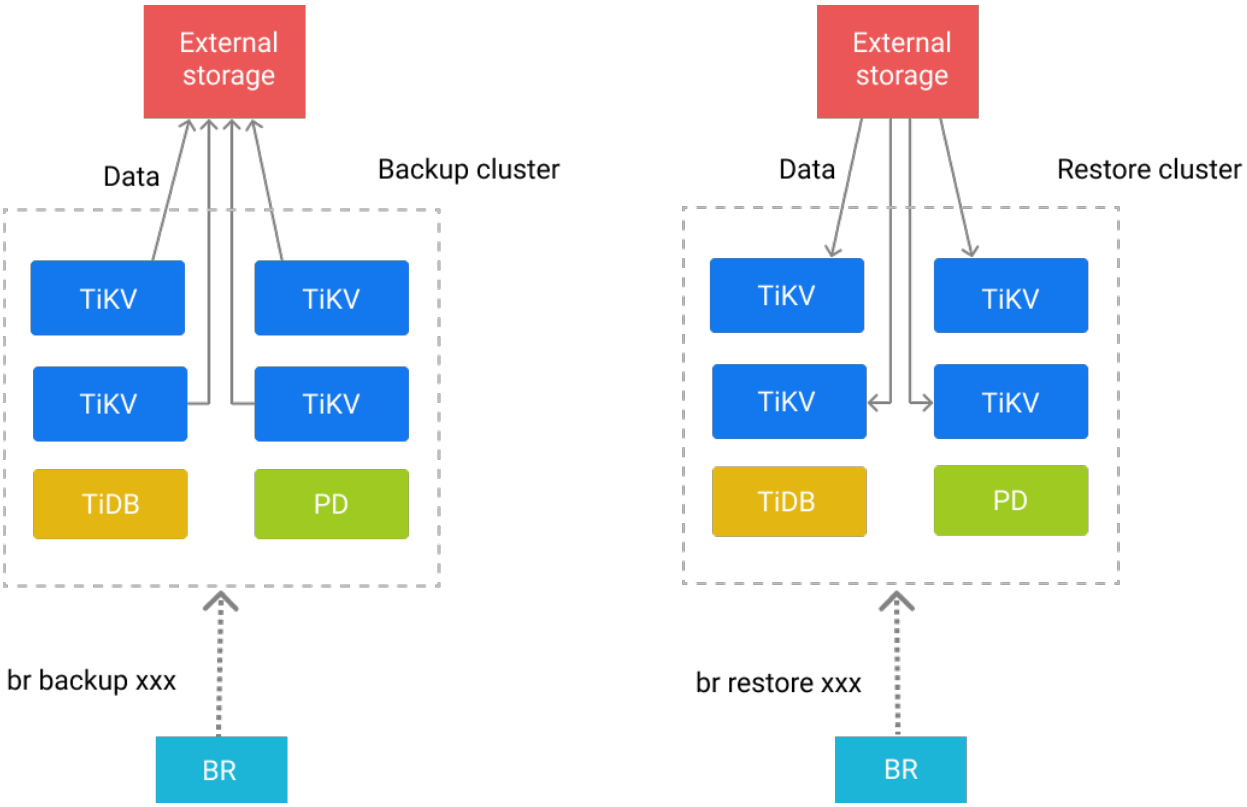


Figure 299: br-arch

13.10.7.1.2 Backup files

This section describes the design of backup files generated by BR.

Types of backup files

BR can generate the following types of backup files:

- `SST` file: stores the data that the TiKV node backs up.
- `backupmeta` file: stores the metadata of a backup operation, including the number, the key range, the size, and the Hash (sha256) value of the backup files.
- `backup.lock` file: prevents multiple backup operations from storing data to the same directory.

Naming format of SST files

When data is backed up to Google Cloud Storage or Azure Blob Storage, SST files are named in the format of `storeID_regionID_regionEpoch_keyHash_timestamp_cf`. The fields in the format are explained as follows:

- `storeID` is the TiKV node ID.
- `regionID` is the Region ID.
- `regionEpoch` is the version number of a Region.
- `keyHash` is the Hash (sha256) value of the startKey of a range, which ensures the uniqueness of a key.
- `timestamp` is the Unix timestamp of an SST file when it is generated at TiKV.
- `cf` indicates the Column Family of RocksDB (`default` or `write` by default).

When data is backed up to Amazon S3 or a network disk, the SST files are named in the format of `regionID_regionEpoch_keyHash_timestamp_cf`. The fields in the format are explained as follows:

- `regionID` is the Region ID.
- `regionEpoch` is the version number of a Region.
- `keyHash` is the Hash (sha256) value of the startKey of a range, which ensures the uniqueness of a key.
- `timestamp` is the Unix timestamp of an SST file when it is generated at TiKV.
- `cf` indicates the Column Family of RocksDB (`default` or `write` by default).

Storage format of SST files

- For details about the storage format of SST files, see Rocksdb BlockBasedTable Format.
- For details about the encoding format of backup data in SST files, see Mapping of table data to Key-Value.

Backup file structure

When you back up data to Google Cloud Storage or Azure Blob Storage, the SST files, backupmeta files, and backup.lock files are stored in the same directory in the following structure:

```
.
├── 20220621
│   ├── backupmeta
│   ├── backup.lock
│   ├── {storeID}-{regionID}-{regionEpoch}-{keyHash}-{timestamp}-{cf}.sst
│   ├── {storeID}-{regionID}-{regionEpoch}-{keyHash}-{timestamp}-{cf}.sst
│   └── {storeID}-{regionID}-{regionEpoch}-{keyHash}-{timestamp}-{cf}.sst
```

When you back up data to Amazon S3 or a network disk, the SST files are stored in sub-directories based on the storeID. The structure is as follows:

```
.
├── 20220621
│   ├── backupmeta
│   ├── backup.lock
│   ├── store1
│   │   └── {regionID}-{regionEpoch}-{keyHash}-{timestamp}-{cf}.sst
│   ├── store100
│   │   └── {regionID}-{regionEpoch}-{keyHash}-{timestamp}-{cf}.sst
│   ├── store2
│   │   └── {regionID}-{regionEpoch}-{keyHash}-{timestamp}-{cf}.sst
│   ├── store3
│   ├── store4
│   └── store5
```

13.10.7.2 Use BR Command-line for Backup and Restoration

This document describes how to back up and restore TiDB cluster data using the BR command line.

Make sure you have read BR Tool Overview, especially Usage restrictions and Some tips.

13.10.7.2.1 BR command-line description

A `br` command consists of sub-commands, options, and parameters.

- Sub-command: the characters without `-` or `--`.
- Option: the characters that start with `-` or `--`.
- Parameter: the characters that immediately follow behind and are passed to the sub-command or the option.

This is a complete `br` command:

```
`br backup full --pd "${PDIP}:2379" -s "s3://backup-data/2022-01-30/"`
```

Explanations for the above command are as follows:

- `backup`: the sub-command of `br`.
- `full`: the sub-command of `backup`.
- `-s` (or `--storage`): the option that specifies the path where the backup files are stored.
- `"s3://backup-data/2022-01-30/"`: the parameter of `-s`, indicating that backup data is stored to the `2022-01-30/` directory in the `backup-data` bucket of Amazon S3.
- `--pd`: the option that specifies the Placement Driver (PD) service address.
- `"${PDIP}:2379"`: the parameter of `--pd`.

Sub-commands

A `br` command consists of multiple layers of sub-commands. Currently, BR has the following sub-commands:

- `br backup`: used to back up the data of the TiDB cluster.
- `br restore`: used to restore the data of the TiDB cluster.

Each of the above sub-commands might still include the following sub-commands to specify the scope of an operation:

- `full`: used to back up or restore all the cluster data.
- `db`: used to back up or restore the specified database of the cluster.
- `table`: used to back up or restore a single table in the specified database of the cluster.

Common options

- `--pd`: used for connection, specifying the PD server address. For example, `"${PDIP}:2379"`.
- `-h` (or `--help`): used to get help on all sub-commands. For example, `br backup --help`.
- `-V` (or `--version`): used to check the version of BR.
- `--ca`: specifies the path to the trusted CA certificate in the PEM format.
- `--cert`: specifies the path to the SSL certificate in the PEM format.
- `--key`: specifies the path to the SSL certificate key in the PEM format.
- `--status-addr`: specifies the listening address through which BR provides statistics to Prometheus.

13.10.7.2.2 Examples of using BR command-line to back up cluster data

To back up cluster data, run the `br backup` command. You can add the `full` or `table` sub-command to specify the scope of your backup operation: the whole cluster or a single table.

- Back up TiDB cluster snapshots
- Back up a database
- Back up a table
- Back up multiple tables with table filter
- Back Up data on Amazon S3 using BR
- Back up data on Google Cloud Storage using BR
- Back up data on Azure Blob Storage using BR
- Back up incremental data
- Encrypt data during backup

13.10.7.2.3 Examples of using BR command-line to restore cluster data

To restore cluster data, run the `br restore` command. You can add the `full`, `db` or `table` sub-command to specify the scope of your restoration: the whole cluster, a database or a single table.

- Restore TiDB cluster snapshots
- Restore a database
- Restore a table
- Restore multiple tables with table filter
- Restore data on Amazon S3 using BR
- Restore data on Google Cloud Storage using BR
- Restore data on Azure Blob Storage using BR
- Restore incremental data
- Restore encrypted backup data

13.10.7.3 External Storages

Backup & Restore (BR), TiDB Lightning, and Dumpling support reading and writing data on the local filesystem and on Amazon S3. BR also supports reading and writing data on the Google Cloud Storage (GCS) and Azure Blob Storage (Azblob). These are distinguished by the URL scheme in the `--storage` parameter passed into BR, in the `-d` parameter passed into TiDB Lightning, and in the `--output` (`-o`) parameter passed into Dumpling.

13.10.7.3.1 Schemes

The following services are supported:

Service	Schemes	Example URL
Local filesystem, distributed on every node	local	`local:///path/to/dest/`
Amazon S3 and compatible services	s3	`s3://bucket-name/prefix/of/dest/`
Google Cloud Storage (GCS)	gcs, gs	`gcs://bucket-name/prefix/of/dest/`
Azure Blob Storage	azure, azblob	`azure://container-name/prefix/of/dest`
Write to nowhere (for benchmarking only)	noop	`noop://`

13.10.7.3.2 URL parameters

Cloud storages such as S3, GCS and Azblob sometimes require additional configuration for connection. You can specify parameters for such configuration. For example:

- Use Dumpling to export data to S3:

```
./dumpling -u root -h 127.0.0.1 -P 3306 -B mydb -F 256MiB \
   -o 's3://my-bucket/sql-backup?region=us-west-2'
```

- Use TiDB Lightning to import data from S3:

```
./tidb-lightning --tidb-port=4000 --pd-urls=127.0.0.1:2379 --backend=
   ↪ local --sorted-kv-dir=/tmp/sorted-kvs \
   -d 's3://my-bucket/sql-backup?region=us-west-2'
```

- Use TiDB Lightning to import data from S3 (using the path style in the request mode):

```
./tidb-lightning --tidb-port=4000 --pd-urls=127.0.0.1:2379 --backend=
   ↪ local --sorted-kv-dir=/tmp/sorted-kvs \
   -d 's3://my-bucket/sql-backup?force-path-style=true&endpoint=http
      ↪ ://10.154.10.132:8088'
```

- Use BR to back up data to GCS:

```
./br backup full -u 127.0.0.1:2379 \
   -s 'gcs://bucket-name/prefix'
```

- Use BR to back up data to Azblob:

```
./br backup full -u 127.0.0.1:2379 \
   -s 'azure://container-name/prefix'
```

S3 URL parameters

URL parameter	Description
`access-key`	The access key
`secret-` ↪ `access-` ↪ `key`	The secret access key
`region`	Service Region for Amazon S3 (default to `us-east-1`)

URL parameter	Description
`use-accelerate-endpoint`	Whether to use the accelerate endpoint on Amazon S3 (default to `false`)
`endpoint`	URL of custom endpoint for S3-compatible services (for example, `https://s3.example.com/`)
`force-path-style`	Use path style access rather than virtual hosted style access (default to `true`)
`storage-class`	Storage class of the uploaded objects (for example, `STANDARD`, `STANDARD_IA`)

URL parameter	Description
`sse`	Server-side encryption algorithm used to encrypt the upload (empty, `AES256` or `aws:kms`)
`sse-kms-key-id`	If `sse` is set to `aws:kms`, specifies the KMS ID
`acl`	Canned ACL of the uploaded objects (for example, `private`, `authenticated-read`)

Note:

It is not recommended to pass in the access key and secret access key directly in the storage URL, because these keys are logged in plain text.

If the access key and secret access key are not specified, the migration tools try to infer these keys from the environment in the following order:

1. `$AWS_ACCESS_KEY_ID` and `$AWS_SECRET_ACCESS_KEY` environment variables
2. `$AWS_ACCESS_KEY` and `$AWS_SECRET_KEY` environment variables
3. Shared credentials file on the tool node at the path specified by the `$AWS_SHARED_CREDENTIALS_FILE` environment variable
4. Shared credentials file on the tool node at `~/.aws/credentials`
5. Current IAM role of the Amazon EC2 container
6. Current IAM role of the Amazon ECS task

GCS URL parameters

URL parameter	Description
`credentials-file`	The path to the credentials JSON file on the tool node
`storage-class`	Storage class of the uploaded objects (for example, `STANDARD`, `COLDLINE`)
`predefined-acl`	Predefined ACL of the uploaded objects (for example, `private`, `project-private`)

When `credentials-file` is not specified, the migration tool will try to infer the credentials from the environment, in the following order:

1. Content of the file on the tool node at the path specified by the `$GOOGLE_APPLICATION_CREDENTIALS` environment variable
2. Content of the file on the tool node at `~/.config/gcloud/application_default_credentials.json`
3. When running in GCE or GAE, the credentials fetched from the metadata server.

Azblob URL parameters

URL parameter	Description
`account-name`	The account name of the storage

URL parameter	Description
`account-key`	The access key
`access-tier`	Access tier of the uploaded objects (for example, `Hot`, `Cool`, `Archive`). If `access-tier` is not set (the value is empty), the value is `Hot` by default.

To ensure that TiKV and BR use the same storage account, BR determines the value of `account-name`. That is, `send-credentials-to-tikv = true` is set by default. BR infers these keys from the environment in the following order:

1. If both `account-name` **and** `account-key` are specified, the key specified by this parameter is used.
2. If `account-key` is not specified, BR tries to read the related credentials from environment variables on the node of BR. BR reads `$AZURE_CLIENT_ID`, `$AZURE_TENANT_ID`, and `$AZURE_CLIENT_SECRET` first. At the same time, BR allows TiKV to read these three environment variables from the respective nodes and access the variables using Azure AD (Azure Active Directory).
3. If the preceding three environment variables are not configured in the BR node, BR tries to read `$AZURE_STORAGE_KEY` using an access key.

Note:

- When using Azure Blob Storage as the external storage, you should set `send-credentials-to-tikv = true` (which is set by default). Otherwise, the backup task will fail.
- `$AZURE_CLIENT_ID`, `$AZURE_TENANT_ID`, and `$AZURE_CLIENT_SECRET` ↪ respectively refer to the application ID `client_id`, the tenant ID `tenant_id`, and the client password `client_secret` of the Azure application. For details about how to confirm the presence of the three environment variables, or how to configure the environment variables as parameters, see Configure environment variables.

13.10.7.3.3 Command-line parameters

In addition to the URL parameters, BR and Dumpling also support specifying these configurations using command-line parameters. For example:

```
./dumpling -u root -h 127.0.0.1 -P 3306 -B mydb -F 256MiB \
    -o 's3://my-bucket/sql-backup' \
    --s3.region 'us-west-2'
```

If you have specified URL parameters and command-line parameters at the same time, the URL parameters are overwritten by the command-line parameters.

S3 command-line parameters

Command-line parameter	Description
`--s3.` ↪ `region`	Amazon S3's service region, which defaults to `us-` ↪ `east` ↪ `-1`.

Command-line parameter	Description
`--s3.` ↪ `endpoint` ↪	The URL of custom endpoint for S3-compatible services. For example, `https` ↪ `://` ↪ `s3.` ↪ `example` ↪ `.` ↪ `com` ↪ `/.`
`--s3.` ↪ `storage` ↪ `-class`	The storage class of the upload object. For example, `STANDARD` ↪ or `STANDARD_IA` ↪ .

Command-line parameter	Description
`--s3.sse`	The server-side encryption algorithm used to encrypt the upload. The value options are empty, `AES256` and `aws:kms`.
`--s3.sse-kms-key-id`	If `--s3.sse` is configured as `aws:kms`, this parameter is used to specify the KMS ID.

Command-line parameter	Description
`--s3.acl`	The canned ACL of the upload object. For example, `private` ↪ or `authenticated` ↪ - ↪ `read` ↪ .
`--s3.` ↪ `provider` ↪	The type of the S3-compatible service. The supported types are `aws`, `alibaba` ↪ , `ceph`, `netease` ↪ and `other`.

To export data to non-AWS S3 cloud storage, specify the cloud provider and whether to use `virtual-hosted style`. In the following examples, data is exported to the Alibaba Cloud OSS storage:

- Export data to Alibaba Cloud OSS using Dumpling:

```
./dumpling -h 127.0.0.1 -P 3306 -B mydb -F 256MiB \
  -o "s3://my-bucket/dumpling/" \
  --s3.endpoint="http://oss-cn-hangzhou-internal.aliyuncs.com" \
  --s3.provider="alibaba" \
```

```
   -r 200000 -F 256MiB
```

- Back up data to Alibaba Cloud OSS using BR:

```
./br backup full --pd "127.0.0.1:2379" \
    --storage "s3://my-bucket/full/" \
    --s3.endpoint="http://oss-cn-hangzhou-internal.aliyuncs.com" \
    --s3.provider="alibaba" \
    --send-credentials-to-tikv=true \
    --ratelimit 128 \
    --log-file backuptable.log
```

- Export data to Alibaba Cloud OSS using TiDB Lightning. You need to specify the following content in the YAML-formatted configuration file:

```
[mydumper]
data-source-dir = "s3://my-bucket/dumpling/?endpoint=http://oss-cn-
    ↪ hangzhou-internal.aliyuncs.com&provider=alibaba"
```

GCS command-line parameters

Command-line parameter	Description
`--gcs.credentials-file`	The path of the JSON-formatted credential on the tool node
`--gcs.storage-class`	The storage type of the upload objects (for example, `STANDARD` or `COLDLINE`)

Command-line parameter	Description
`--gcs.predefined-acl`	The predefined ACL of the upload objects (for example, `private` or `project-private`)

Azblob command-line parameters

Command-line parameter | Description |
`--azblob.account-name` | The account name of the storage |
`--azblob.account-key` | The access key |
`--azblob.access-tier` | Access tier of the uploaded objects (for example, `Hot`, `Cool`, `Archive`). If `access-tier` is not set (the value is empty), the value is `Hot` by default. |

13.10.7.3.4 BR sending credentials to TiKV

By default, when using S3, GCS, or Azblob destinations, BR will send the credentials to every TiKV node to reduce setup complexity.

However, this is unsuitable on cloud environment, where every node has their own role and permission. In such cases, you need to disable credentials sending with `--send-credentials-to-tikv=false` (or the short form `-c=0`):

```
./br backup full -c=0 -u pd-service:2379 -s 's3://bucket-name/prefix'
```

When using SQL statements to back up and restore data, you can add the `SEND_CREDENTIALS_TO_TIKV = FALSE` option:

```
BACKUP DATABASE * TO 's3://bucket-name/prefix' SEND_CREDENTIALS_TO_TIKV =
    FALSE;
```

This option is not supported in TiDB Lightning and Dumpling, because the two applications are currently standalone.

13.10.7.4 Back Up and Restore Data on Amazon S3 Using BR

The Backup & Restore (BR) tool supports using Amazon S3 or other Amazon S3-compatible file storages as the external storage for backing up and restoring data.

13.10.7.4.1 Application scenarios

By using Amazon S3, you can quickly back up the data of a TiDB cluster deployed on Amazon EC2 to Amazon S3, or quickly restore a TiDB cluster from the backup data in Amazon S3.

13.10.7.4.2 Configure privileges to access S3

Before performing backup or restoration using S3, you need to configure the privileges required to access S3.

Configure access to the S3 directory

Before backup, configure the following privileges to access the backup directory on S3.

- Minimum privileges for TiKV and BR to access the backup directories of `s3:ListBucket`, `s3:PutObject`, and `s3:AbortMultipartUpload` during backup
- Minimum privileges for TiKV and BR to access the backup directories of `s3:ListBucket` and `s3:GetObject` during restoration

If you have not yet created a backup directory, refer to AWS Official Document to create an S3 bucket in the specified region. If necessary, you can also create a folder in the bucket by referring to AWS official documentation - Create Folder.

Configure a user to access S3

It is recommended that you configure access to S3 using either of the following ways:

- Associate an IAM role that can access S3 with the EC2 instances where the TiKV and BR nodes run. After the association, BR can access the backup directories of S3.

```
br backup full --pd "${PDIP}:2379" --storage "s3://${Bucket}/${Folder}"
    ↪  --s3.region "${region}"
```

- Configure `access-key` and `secret-access-key` for accessing S3 in the `br` CLI, and set `--send-credentials-to-tikv=true` to pass the access key from BR to each TiKV.

```
br backup full --pd "${PDIP}:2379" --storage "s3://${Bucket}/${Folder}?
    ↪ access-key=${accessKey}&secret-access-key=${secretAccessKey}" --
    ↪ s3.region "${region}" --send-credentials-to-tikv=true
```

Because the access key in a command is vulnerable to leakage, you are recommended to associate an IAM role to EC2 instances to access S3.

13.10.7.4.3 Back up data to S3

```
br backup full \
    --pd "${PDIP}:2379" \
    --storage "s3://${Bucket}/${Folder}?access-key=${accessKey}&secret-
        ↪ access-key=${secretAccessKey}" \
    --s3.region "${region}" \
    --send-credentials-to-tikv=true \
    --ratelimit 128 \
    --log-file backuptable.log
```

In the preceding command:

- `--s3.region`: specifies the region of S3.
- `--send-credentials-to-tikv`: specifies that access key is passed to the TiKV nodes.

13.10.7.4.4 Restore data from S3

```
br restore full \
    --pd "${PDIP}:2379" \
    --storage "s3://${Bucket}/${Folder}?access-key=${accessKey}&secret-
        ↪ access-key=${secretAccessKey}" \
    --s3.region "${region}" \
    --ratelimit 128 \
    --send-credentials-to-tikv=true \
    --log-file restorefull.log
```

13.10.7.4.5 See also

To know more information about external storages supported by BR, see External storages.

13.10.7.5 Back up and Restore Data on Azure Blob Storage Using BR

The Backup & Restore (BR) tool supports using Azure Blob Storage as the external storage for backing up and restoring data.

13.10.7.5.1 User scenario

Azure virtual machines can quickly store large-scale data on Azure Blob Storage. If you are using Azure virtual machines to deploy your cluster, you can back up your data on Azure Blob Storage.

13.10.7.5.2 Usage

With BR, you can back up and restore data on Azure Blob Storage by the following two methods:

- Back up and restore data using Azure AD (Azure Active Directory)
- Back up and restore data using an access key

In common cases, to avoid exposing the key information (such as `account-key`) in command lines, it is recommended to use Azure AD.

The following is an example of backup and restoration operations on Azure Blob Storage using the preceding two methods. The purpose of the operations are as follows:

- Back up: Back up the `test` database to a space in the `container=test` container with `t1` as the path prefix in Azure Blob Storage.
- Restore: Restore data from a space in the `container=test` container with `t1` as the path prefix in Azure Blob Storage to the `test` database.

> **Note:**
>
> When backing up data to the Azure Blob Storage using Azure AD or an access key, you need to set `send-credentials-to-tikv = true` (which is `true` by default). Otherwise, the backup task will fail.

Method 1: Back up and restore data using Azure AD (recommended)

This section describes how to back up and restore data using Azure AD. Before performing backup or restoration, you need to configure environment variables.

Configure environment variables

In the operating environment of BR and TiKV, configure the environment variables `$AZURE_CLIENT_ID`, `$AZURE_TENANT_ID`, and `$AZURE_CLIENT_SECRET`.

- When you start a cluster using TiUP, TiKV uses the "systemd" service. The following example introduces how to configure the preceding three environment variables as parameters for TiKV:

> **Note:**
>
> You need to restart TiKV in Step 3. If your TiKV cannot be restarted, use Method 2 to back up and restore data.

1. Suppose that the TiKV port on this node is 24000, that is, the name of the "systemd" service is "tikv-24000":

```
systemctl edit tikv-24000
```

2. Fill in the environment variable information:

```
[Service]
Environment="AZURE_CLIENT_ID=aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
Environment="AZURE_TENANT_ID=aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
Environment="AZURE_CLIENT_SECRET=
    ↪ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
```

3. Reload the configuration and restart TiKV:

```
systemctl daemon-reload
systemctl restart tikv-24000
```

- To configure the Azure AD information for TiKV and BR started with command lines, you only need to check whether the environment variables `$AZURE_CLIENT_ID` ↪ , `$AZURE_TENANT_ID`, and `$AZURE_CLIENT_SECRET` are configured in the operating environment by running the following commands:

```
echo $AZURE_CLIENT_ID
echo $AZURE_TENANT_ID
echo $AZURE_CLIENT_SECRET
```

For more information about the environment variables, see Azblob URL parameters.

Back up

This section shows backing up data to `cool tier`, that is, the access tier of the uploaded object is `Cool`. You can specify `account-name` and `access-tier` in two ways. The backup operations differ depending on the way you choose:

- Specify `account-name` and `access-tier` as parameters in URL:

```
tiup br backup db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?
    ↪ account-name=devstoreaccount1&access-tier=Cool'
```

If `access-tier` is not set (the value is empty), the value is `Hot` by default.

- Specify `account-name` and `access-tier` as command-line parameters:

```
tiup br backup db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?' --
    ↪ azblob.account-name=devstoreaccount1 --azblob.access-tier=Cool
```

Restore

Similar to how `account-name` is specified in Back up, you can restore data either using URLs or command-line parameters:

- Specify `account-name` as a parameter in URL:

```
tiup br restore db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?
    ↪ account-name=devstoreaccount1'
```

- Specify `account-name` as a command-line parameter:

```
tiup br restore db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?' --
    ↪ azblob.account-name=devstoreaccount1
```

Method 2: Back up and restore using an access key (easy)

Compared with backing up and restoring data using Azure AD, backup and restoration using an access key is easier because you do not need to configure environment variables. Other steps are similar to those of using Azure AD.

Back up

- Specify `account-name`, `account-key`, and `access-tier` as parameters in URL:

```
tiup br backup db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?
    ↪ account-name=devstoreaccount1&account-key=
    ↪ Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/
    ↪ K1SZFPTOtr/KBHBeksoGMGw==&access-tier=Cool'
```

- Specify `account-name`, `account-key`, and `access-tier` as command-line parameters:

```
tiup br backup db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?' --
    ↪ azblob.account-name=devstoreaccount1 --azblob.account-key=
    ↪ Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/
    ↪ K1SZFPTOtr/KBHBeksoGMGw== --azblob.access-tier=Cool
```

Restore

- Specify `account-name` and `account-key` as parameters in URL:

```
tiup br restore db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?
    ↪ account-name=devstoreaccount1&account-key=
    ↪ Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/
    ↪ K1SZFPTOtr/KBHBeksoGMGw=='
```

- Specify `account-name` and `account-key` as command-line parameters:

```
tiup br restore db --db test -u 127.0.0.1:2379 -s 'azure://test/t1?' --
    ↪ azblob.account-name=devstoreaccount1 --azblob.account-key=
    ↪ Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/
    ↪ K1SZFPTOtr/KBHBeksoGMGw==
```

13.10.7.5.3 Compatibility

This feature is **only compatible** with v5.4.0 and later versions.

13.10.7.5.4 See also

- To learn other external storages supported by BR, see External storages.
- To learn more about the parameters, see the following documents:
    - Azblob URL parameters
    - Azblob command-line parameters

13.10.7.6 Back Up and Restore Data on Google Cloud Storage Using BR

The Backup & Restore (BR) tool supports using Google Cloud Storage (GCS) as the external storage for backing up and restoring data.

13.10.7.6.1 User scenario

You can quickly back up the data of a TiDB cluster deployed in Google Compute Engine (GCE) to GCS, or quickly restore a TiDB cluster from the backup data in GCS.

13.10.7.6.2 Back up data to GCS

```
br backup full --pd "${PDIP}:2379" --storage 'gcs://bucket-name/prefix?
    ↪ credentials-file=${credentials-file-path}' --send-credentials-to-tikv
    ↪ =true
```

When backing up data to GCS, you need to place a credential file in the node where BR is running. The credential file contains the account credentials for accessing GCS. If `--send-credentials-to-tikv` is displayed, it means the account access credentials of GCS will be passed to the TiKV node.

To obtain the credential files, refer to CREATE AND DOWNLOAD THE GCS CREDENTIALS FILE.

13.10.7.6.3 Restore data from GCS

```
br restore full --pd "${PDIP}:2379" --storage 'gcs://bucket-name/prefix?
    ↪ credentials-file=${credentials-file-path}' --send-credentials-to-tikv
    ↪ =true
```

13.10.7.6.4 See also

To learn other external storages supported by BR, see External storages.

13.10.7.7 Back Up and Restore RawKV

Backup & Restore (BR) supports data backup and restore for products that use RawKV (TiKV and PD) without TiDB. This document describes how to back up and restore RawKV.

> **Warning:**
>
> This feature is in the experiment, without being thoroughly tested. It is **NOT** recommended that you use it in the production environment.

13.10.7.7.1 Back up RawKV

In some scenarios, TiKV might run independently of TiDB. Given that, BR supports bypassing the TiDB layer and backing up data in TiKV.

```
br backup raw --pd $PD_ADDR \
    -s "local://$BACKUP_DIR" \
    --start 31 \
    --ratelimit 128 \
    --end 3130303030303030 \
    --format hex \
    --cf default
```

The preceding command backs up all keys between `[0x31, 0x3130303030303030)` in the default CF to `$BACKUP_DIR`.

In this command, the values of `--start` and `--end` are decoded using the format specified by `--format` before being sent to TiKV. Currently, the following formats are available:

- "raw": The input string is directly encoded as a key in binary format.
- "hex": The default encoding format. The input string is treated as a hexadecimal number.
- "escaped": First escape (backslash) the input string, and then encode it into binary format, for example, `abc\xFF\x00\r\n`.

Note:

- If you use the local storage, you **should** copy all back up SST files to every TiKV node in the path specified by `--storage`. Even if each TiKV node eventually only needs to read a part of the SST files, they all need full access to the complete archive because:
  - Data is replicated into multiple peers. When ingesting SSTs, these files have to be present on all peers. This is unlike backup where reading from a single node is enough.
  - Where each peer is scattered to during restoration is random. You have no idea in advance which node will read which file.
- These can be avoided using shared storage, for example, mounting an NFS on the local path, or using S3. With network storage, every node can automatically read every SST file. In this case, the preceding caveats no longer apply.
- Also, note that you can only run one restoration operation for a single cluster at the same time. Otherwise, unexpected behaviors might occur. For details, see FAQs.

13.10.7.7.2 Restore RawKV

Similar to backing up RawKV, you can run the following command to restore RawKV:

```
br restore raw --pd $PD_ADDR \
    -s "local://$BACKUP_DIR" \
    --start 31 \
    --end 3130303030303030 \
    --ratelimit 128 \
    --format hex \
    --cf default
```

In this example, all the backed up keys in the range `[0x31, 0x3130303030303030)` are restored to the TiKV cluster. The coding formats of these keys are identical to that of keys during the backup process.

13.10.7.8 Back up and Restore Data Using Dumpling and TiDB Lightning

This document introduces how to use Dumpling and TiDB Lightning to back up and restore full data of TiDB.

If you need to back up a small amount of data (for example, less than 50 GB) and do not require high backup speed, you can use Dumpling to export data from the TiDB database

and then use TiDB Lightning to import the data into another TiDB database. For more information about backup and restore, see Use BR to Back Up Cluster Data and Use BR to Restore Cluster Data.

13.10.7.8.1 Requirements

- Install and start Dumpling:

```
tiup install dumpling && tiup dumpling
```

- Install and start TiDB Lightning:

```
tiup install tidb lightning && tiup tidb lightning
```

- Grant the source database privileges required for Dumpling
- Grant the target database privileges required for TiDB Lightning

13.10.7.8.2 Resource requirements

Operating system: The example in this document uses fresh CentOS 7 instances. You can deploy a virtual machine either on your local host or in the cloud. Because TiDB Lightning consumes as much CPU resources as needed by default, it is recommended that you deploy it on a dedicated server. If this is not possible, you can deploy it on a single server together with other TiDB components (for example, `tikv-server`) and then configure `region-concurrency` to limit the CPU usage from TiDB Lightning. Usually, you can configure the size to 75% of the logical CPU.

Memory and CPU: Because TiDB Lightning consumes high resources, it is recommended to allocate more than 64 GiB of memory and more than 32 CPU cores. To get the best performance, make sure that the CPU core to memory (GiB) ratio is greater than 1:2.

Disk space:

It is recommended to use Amazon S3, Google Cloud Storage (GCS), or Azure Blob Storage as the external storage. With such a cloud storage, you can store backup files quickly without being limited by the disk space.

If you need to save data of one backup task to the local disk, note the following limitations:

- Dumpling requires a disk space that can store the whole data source (or to store all upstream tables to be exported). To calculate the required space, see Downstream storage space requirements.
- During the import, TiDB Lightning needs temporary space to store the sorted key-value pairs. The disk space should be enough to hold the largest single table from the data source.

Note: It is difficult to calculate the exact data volume exported by Dumpling from MySQL, but you can estimate the data volume by using the following SQL statement to summarize the `data-length` field in the `information_schema.tables` table:

```
/* Calculate the size of all schemas, in MiB. Replace ${schema_name} with
    ↪ your schema name. */
SELECT table_schema,SUM(data_length)/1024/1024 AS data_length,SUM(
    ↪ index_length)/1024/1024 AS index_length,SUM(data_length+index_length)
    ↪ /1024/1024 AS SUM FROM information_schema.tables WHERE table_schema =
    ↪  "${schema_name}" GROUP BY table_schema;

/* Calculate the size of the largest table, in MiB. Replace ${schema_name}
    ↪  with your schema name. */
SELECT table_name,table_schema,SUM(data_length)/1024/1024 AS data_length,
    ↪ SUM(index_length)/1024/1024 AS index_length,SUM(data_length+
    ↪ index_length)/1024/1024 AS SUM from information_schema.tables WHERE
    ↪ table_schema = "${schema_name}" GROUP BY table_name,table_schema
    ↪ ORDER BY SUM DESC LIMIT 5;
```

Disk space for the target TiKV cluster

The target TiKV cluster must have enough disk space to store the imported data. In addition to the standard hardware requirements, the storage space of the target TiKV cluster must be larger than **the size of the data source x the number of replicas x 2**. For example, if the cluster uses 3 replicas by default, the target TiKV cluster must have a storage space larger than 6 times the size of the data source. The formula has x 2 because:

- Index might take extra space.
- RocksDB has a space amplification effect.

13.10.7.8.3 Use Dumpling to back up full data

1. Run the following command to export full data from TiDB to `s3://my-bucket/sql-backup` in Amazon S3:

    ```
    tiup dumpling -h ${ip} -P 3306 -u root -t 16 -r 200000 -F 256MiB -B
        ↪ my_db1 -f 'my_db1.table[12]' -o 's3://my-bucket/sql-backup'
    ```

    Dumpling exports data in SQL files by default. You can specify a different file format by adding the `--filetype` option.

    For more configurations of Dumpling, see Option list of Dumpling.

2. After the export is completed, you can view the backup files in the directory `s3://my-bucket/sql-backup`.

13.10.7.8.4 Use TiDB Lightning to restore full data

1. Edit the `tidb-lightning.toml` file to import full data backed up using Dumpling from `s3://my-bucket/sql-backup` to the target TiDB cluster:

```
[lightning]
# log
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
# "local": Default backend. The local backend is recommended to import
    ↪ large volumes of data (1 TiB or more). During the import, the
    ↪ target TiDB cluster cannot provide any service.
# "tidb": The "tidb" backend is recommended to import data less than 1
    ↪ TiB. During the import, the target TiDB cluster can provide
    ↪ service normally. For more information on the backends, refer to
    ↪ https://docs.pingcap.com/tidb/stable/tidb-lightning-backends.
backend = "local"
# Sets the temporary storage directory for the sorted Key-Value files.
    ↪ The directory must be empty, and the storage space must be
    ↪ greater than the size of the dataset to be imported. For better
    ↪ import performance, it is recommended to use a directory
    ↪ different from `data-source-dir` and use flash storage, which can
    ↪  use I/O exclusively.
sorted-kv-dir = "${sorted-kv-dir}"

[mydumper]
# The data source directory. The same directory where Dumpling exports
    ↪ data in "Use Dumpling to back up full data".
data-source-dir = "${data-path}" # A local path or S3 path. For example
    ↪ , 's3://my-bucket/sql-backup'

[tidb]
# The target TiDB cluster information.
host = ${host}             # e.g.: 172.16.32.1
port = ${port}             # e.g.: 4000
user = "${user_name}"      # e.g.: "root"
password = "${password}"   # e.g.: "rootroot"
status-port = ${status-port} # During the import, TiDB Lightning needs
    ↪ to obtain the table schema information from the TiDB status port.
    ↪  e.g.: 10080
pd-addr = "${ip}:${port}" # The address of the PD cluster, e.g.:
    ↪ 172.16.31.3:2379. TiDB Lightning obtains some information from PD
    ↪ . When backend = "local", you must specify status-port and pd-
    ↪ addr correctly. Otherwise, the import will be abnormal.
```

For more information on TiDB Lightning configuration, refer to TiDB Lightning Configuration.

2. Start the import by running `tidb-lightning`. If you launch the program directly in the command line, the process might exit unexpectedly after receiving a `SIGHUP` signal. In this case, it is recommended to run the program using a `nohup` or `screen` tool. For example:

   If you import data from S3, pass the SecretKey and AccessKey that have access to the S3 storage path as environment variables to the TiDB Lightning node. You can also read the credentials from `~/.aws/credentials`.

```
export AWS_ACCESS_KEY_ID=${access_key}
export AWS_SECRET_ACCESS_KEY=${secret_key}
nohup tiup tidb-lightning -config tidb-lightning.toml > nohup.out 2>&1
    ↪ &
```

3. After the import starts, you can `grep` the keyword `progress` in the log to check the progress of the import. The progress is updated every 5 minutes by default.

4. After TiDB Lightning completes the import, it exits automatically. Check whether `tidb-lightning.log` contains `the whole procedure completed` in the last lines. If yes, the import is successful. If no, the import encounters an error. Address the error as instructed in the error message.

> **Note:**
>
> Whether the import is successful or not, the last line of the log shows `tidb ↪ lightning exit`. It means that TiDB Lightning exits normally, but does not necessarily mean that the import is successful.

If the import fails, refer to TiDB Lightning FAQ for troubleshooting.

13.10.7.9 Backup & Restore FAQs

This document lists the frequently asked questions (FAQs) and the solutions about Backup & Restore (BR).

13.10.7.9.1 In TiDB v5.4.0 and later versions, when backup tasks are performed on the cluster under high workload, why does the speed of backup tasks become slow?

Starting from TiDB v5.4.0, BR introduces the auto-tune feature for backup tasks. For clusters in v5.4.0 or later versions, this feature is enabled by default. When the cluster workload is heavy, the feature limits the resources used by backup tasks to reduce the impact on the online cluster. For more information, refer to BR Auto-Tune.

TiKV supports dynamically configuring the auto-tune feature. You can enable or disable the feature by the following methods without restarting your cluster:

- Disable auto-tune: Set the TiKV configuration item `backup.enable-auto-tune` to `false`.
- Enable auto-tune: Set `backup.enable-auto-tune` to `true`. For clusters upgraded from v5.3.x to v5.4.0 or later versions, the auto-tune feature is disabled by default. You need to manually enable it.

To use `tikv-ctl` to enable or disable auto-tune, refer to Use auto-tune.

In addition, auto-tune reduces the default number of threads used by backup tasks. For details, see `backup.num-threads`](/tikv-configuration-file.md#num-threads-1). Therefore, on the Grafana Dashboard, the speed, CPU usage, and I/O resource utilization used by backup tasks are lower than those of versions earlier than v5.4.0. Before v5.4.0, the default value of `backup.num-threads` was `CPU * 0.75`, that is, the number of threads used by backup tasks makes up 75% of the logical CPU cores. The maximum value of it was `32` ↪ . Starting from v5.4.0, the default value of this configuration item is `CPU * 0.5`, and its maximum value is `8`.

When you perform backup tasks on an offline cluster, to speed up the backup, you can modify the value of `backup.num-threads` to a larger number using `tikv-ctl`.

13.10.7.9.2 What should I do if the error message `could not read local://...:download sst failed` is returned during data restoration?

When you restore data, each node must have access to **all** backup files (SST files). By default, if `local` storage is used, you cannot restore data because the backup files are scattered among different nodes. Therefore, you have to copy the backup file of each TiKV node to the other TiKV nodes.

It is recommended that you mount an NFS disk as a backup disk during backup. For details, see Back up a single table to a network disk.

13.10.7.9.3 How much impact does a backup operation have on the cluster?

For TiDB v5.4.0 or later versions, BR not only reduces the default CPU utilization used by backup tasks, but also introduces the BR Auto-tune feature to limit the resources used by backup tasks in the cluster with heavy workloads. Therefore, when you use the default configuration for backup tasks in a v5.4.0 cluster with heavy workloads, the impact of the

tasks on the cluster performance is significantly less than that on the clusters earlier than v5.4.0.

The following is an internal test on a single node. The test results show that when you use the default configuration of v5.4.0 and v5.3.0 in the **full-speed backup** scenario, the impact of backup using BR on cluster performance is quite different. The detailed test results are as follows:

- When BR uses the default configuration of v5.3.0, the QPS of write-only workload is reduced by 75%.
- When BR uses the default configuration of v5.4.0, the QPS for the same workload is reduced by 25%. However, when this configuration is used, the duration of backup tasks using BR becomes correspondingly longer. The time required is 1.7 times that of the v5.3.0 configuration.

You can use either of the following solutions to manually control the impact of backup tasks on cluster performance. Note that these methods reduce the impact of backup tasks on the cluster, but they also reduce the speed of backup tasks.

- Use the `--ratelimit` parameter to limit the speed of backup tasks. Note that this parameter limits the speed of **saving backup files to external storage**. When calculating the total size of backup files, use the `backup data size(after compressed)` in the backup log as a benchmark.
- Adjust the TiKV configuration item `backup.num-threads` to limit the number of threads used by backup tasks. When BR uses no more than `8` threads for backup tasks, and the total CPU utilization of the cluster does not exceed 60%, the backup tasks have little impact on the cluster, regardless of the read and write workload.

13.10.7.9.4 Does BR back up system tables? During data restoration, do they raise conflicts?

Before v5.1.0, BR filters out data from the system schemas `mysql.*` during the backup. Since v5.1.0, BR **backs up** all data by default, including the system schemas `mysql.*`.

The technical implementation of restoring the system tables in `mysql.*` is not complete yet, so the tables in the system schema `mysql` are **not restored** by default, which means no conflicts will be raised. For more details, refer to Restore tables created in the `mysql` schema (experimental).

13.10.7.9.5 What should I do to handle the `Permission denied` or `No such file or directory` error, even if I have tried to run BR using root in vain?

You need to confirm whether TiKV has access to the backup directory. To back up data, confirm whether TiKV has the write permission. To restore data, confirm whether it has the read permission.

During the backup operation, if the storage medium is the local disk or a network file system (NFS), make sure that the user to start BR and the user to start TiKV are consistent (if BR and TiKV are on different machines, the users' UIDs must be consistent). Otherwise, the `Permission denied` issue might occur.

Running BR with the root access might fail due to the disk permission, because the backup files (SST files) are saved by TiKV.

> **Note:**
>
> You might encounter the same problem during data restoration. When the SST files are read for the first time, the read permission is verified. The execution duration of DDL suggests that there might be a long interval between checking the permission and running BR. You might receive the error message `Permission denied` after waiting for a long time.
>
> Therefore, it is recommended to check the permission before data restore according to the following steps:

1. Run the Linux-native command for process query:

```
ps aux | grep tikv-server
```

The output of the above command:

```
tidb_ouo 9235 10.9 3.8 2019248 622776 ?  Ssl 08:28  1:12 bin/tikv-
    ↪ server --addr 0.0.0.0:20162 --advertise-addr 172.16.6.118:20162
    ↪ --status-addr 0.0.0.0:20188 --advertise-status-addr
    ↪ 172.16.6.118:20188 --pd 172.16.6.118:2379 --data-dir /home/user1/
    ↪ tidb-data/tikv-20162 --config conf/tikv.toml --log-file /home/
    ↪ user1/tidb-deploy/tikv-20162/log/tikv.log
tidb_ouo 9236 9.8  3.8 2048940 631136 ?   Ssl 08:28  1:05 bin/tikv-
    ↪ server --addr 0.0.0.0:20161 --advertise-addr 172.16.6.118:20161
    ↪ --status-addr 0.0.0.0:20189 --advertise-status-addr
    ↪ 172.16.6.118:20189 --pd 172.16.6.118:2379 --data-dir /home/user1/
    ↪ tidb-data/tikv-20161 --config conf/tikv.toml --log-file /home/
    ↪ user1/tidb-deploy/tikv-20161/log/tikv.log
```

Or you can run the following command:

```
ps aux | grep tikv-server | awk '{print $1}'
```

The output of the above command:

```
tidb_ouo
tidb_ouo
```

2. Query the startup information of the cluster using the TiUP command:

```
tiup cluster list
```

The output of the above command:

```
[root@Copy-of-VM-EE-CentOS76-v1 br]# tiup cluster list
Starting component `cluster`: /root/.tiup/components/cluster/v1.5.2/
    ↪ tiup-cluster list
Name         User     Version Path
    ↪                                          PrivateKey
----         ----     ------- ----
    ↪                                          ----------
tidb_cluster tidb_ouo v5.0.2 /root/.tiup/storage/cluster/clusters/
    ↪ tidb_cluster /root/.tiup/storage/cluster/clusters/tidb_cluster/
    ↪ ssh/id_rsa
```

3. Check the permission for the backup directory. For example, `backup` is for backup data storage:

```
ls -al backup
```

The output of the above command:

```
[root@Copy-of-VM-EE-CentOS76-v1 user1]# ls -al backup
total 0
drwxr-xr-x 2 root root 6 Jun 28 17:48 .
drwxr-xr-x 11 root root 310 Jul 4 10:35 ..
```

From the above output, you can find that the `tikv-server` instance is started by the user `tidb_ouo`. But the user `tidb_ouo` does not have the write permission for `backup`. Therefore, the backup fails.

13.10.7.9.6 What should I do to handle the `Io(Os...)` error?

Almost all of these problems are system call errors that occur when TiKV writes data to the disk, for example, `Io(Os {code: 13, kind: PermissionDenied...})` or `Io(Os {` ↪ `code: 2, kind: NotFound...})`.

To address such problems, first check the mounting method and the file system of the backup directory, and try to back up data to another folder or another hard disk.

For example, you might encounter the `Code: 22(invalid argument)` error when backing up data to the network disk built by `samba`.

13.10.7.9.7 What should I do to handle the `rpc error: code = Unavailable desc =...` error occurred in BR?

This error might occur when the capacity of the cluster to restore (using BR) is insufficient. You can further confirm the cause by checking the monitoring metrics of this cluster or the TiKV log.

To handle this issue, you can try to scale out the cluster resources, reduce the concurrency during restoration, and enable the `RATE_LIMIT` option.

13.10.7.9.8 Where are the backed up files stored when I use `local` storage?

When you use `local` storage, `backupmeta` is generated on the node where BR is running, and backup files are generated on the Leader nodes of each Region.

13.10.7.9.9 How about the size of the backup data? Are there replicas of the backup?

During data backup, backup files are generated on the Leader nodes of each Region. The size of the backup is equal to the data size, with no redundant replicas. Therefore, the total data size is approximately the total number of TiKV data divided by the number of replicas.

However, if you want to restore data from local storage, the number of replicas is equal to that of the TiKV nodes, because each TiKV must have access to all backup files.

13.10.7.9.10 What should I do when BR restores data to the upstream cluster of TiCDC/Drainer?

- **The data restored using BR cannot be replicated to the downstream**. This is because BR directly imports SST files but the downstream cluster currently cannot obtain these files from the upstream.
- Before v4.0.3, DDL jobs generated during the BR restore might cause unexpected DDL executions in TiCDC/Drainer. Therefore, if you need to perform restore on the upstream cluster of TiCDC/Drainer, add all tables restored using BR to the TiCDC/Drainer block list.

You can use `filter.rules` to configure the block list for TiCDC and use `syncer.ignore-table` to configure the block list for Drainer.

13.10.7.9.11 Does BR back up the `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` information of a table? Does the restored table have multiple Regions?

Yes. BR backs up the `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` information of a table. The data of the restored table is also split into multiple Regions.

13.10.7.9.12 What should I do if the restore fails with the error message `the entry too large, the max entry size is 6291456, the size of data is 7690800`?

You can try to reduce the number of tables to be created in a batch by setting `--ddl-batch-size` to `128` or a smaller value.

When using BR to restore the backup data with the value of `--ddl-batch-size` greater than `1`, TiDB writes a DDL job of table creation to the DDL jobs queue that is maintained by TiKV. At this time, the total size of all tables schema sent by TiDB at one time should not exceed 6 MB, because the maximum value of job messages is `6 MB` by default (it is **not recommended** to modify this value. For details, see `txn-entry-size-limit` and `raft-entry-max-size`). Therefore, if you set `--ddl-batch-size` to an excessively large value, the schema size of the tables sent by TiDB in a batch at one time exceeds the specified value, which causes BR to report the `entry too large, the max entry size is 6291456, the size of data is 7690800` error.

13.10.7.9.13 Why is the `region is unavailable` error reported for a SQL query after I use BR to restore the backup data?

If the cluster backed up using BR has TiFlash, `TableInfo` stores the TiFlash information when BR restores the backup data. If the cluster to be restored does not have TiFlash, the `region is unavailable` error is reported.

13.10.7.9.14 Does BR support in-place full restoration of some historical backup?

No. BR does not support in-place full restoration of some historical backup.

13.10.7.9.15 How can I use BR for incremental backup on Kubernetes?

To get the `commitTs` field of the last BR backup, run the `kubectl -n ${namespace} get bk ${name}` command using kubectl. You can use the content of this field as `--lastbackupts`.

13.10.7.9.16 How can I convert BR backupTS to Unix time?

BR `backupTS` defaults to the latest timestamp obtained from PD before the backup starts. You can use `pd-ctl tso timestamp` to parse the timestamp to obtain an accurate value, or use `backupTS >> 18` to quickly obtain an estimated value.

13.10.7.9.17 After BR restores the backup data, do I need to execute the `ANALYZE` statement on the table to update the statistics of TiDB on the tables and indexes?

BR does not back up statistics (except in v4.0.9). Therefore, after restoring the backup data, you need to manually execute `ANALYZE TABLE` or wait for TiDB to automatically execute `ANALYZE`.

In v4.0.9, BR backs up statistics by default, which consumes too much memory. To ensure that the backup process goes well, the backup for statistics is disabled by default starting from v4.0.10.

If you do not execute `ANALYZE` on the table, TiDB will fail to select the optimized execution plan due to inaccurate statistics. If query performance is not a key concern, you can ignore `ANALYZE`.

13.10.7.9.18 Can I use multiple BR processes at the same time to restore the data of a single cluster?

It is strongly not recommended to use multiple BR processes at the same time to restore the data of a single cluster for the following reasons:

- When BR restores data, it modifies some global configurations of PD. Therefore, if you use multiple BR processes for data restore at the same time, these configurations might be mistakenly overwritten and cause abnormal cluster status.
- BR consumes a lot of cluster resources to restore data, so in fact, running BR processes in parallel improves the restore speed only to a limited extent.
- There has been no test for running multiple BR processes in parallel for data restore, so it is not guaranteed to succeed.

13.10.7.9.19 What should I do if the backup log reports `key locked Error`?

Error message in the log: `log - ["backup occur kv error"][error="{\"KvError` ↪ `\":{\"locked\":`

If a key is locked during the backup process, BR tries to resolve the lock. If this error occurs only occasionally, the correctness of the backup is not affected.

13.10.7.9.20 What should I do if a backup operation fails?

Error message in the log: `log - Error: msg:"Io(Custom { kind: AlreadyExists,` ↪ `error: \"[5_5359_42_123_default.sst] is already exists in /dir/backup_local` ↪ `/\" })"`

If a backup operation fails and the preceding message occurs, perform one of the following operations and then start the backup again:

- Change the directory for the backup. For example, change `/dir/backup_local/` to `/dir/backup-2020-01-01/`.
- Delete the backup directories of all TiKV nodes and BR nodes.

13.10.7.9.21 What should I do if the disk usage shown on the monitoring node is inconsistent after BR backup or restoration?

This inconsistency is caused by the fact that the data compression rate used in backup is different from the default rate used in restoration. If the checksum succeeds, you can ignore this issue.

13.10.7.9.22 Why does an error occur when I restore placement rules to a cluster?

Before v6.0.0, BR does not support placement rules. Starting from v6.0.0, BR supports placement rules and introduces a command-line option `--with-tidb-placement-mode =strict/ignore` to control the backup and restore mode of placement rules. With the default value `strict`, BR imports and validates placement rules, but ignores all placement rules when the value is `ignore`.

13.10.7.9.23 Why does BR report `new_collations_enabled_on_first_bootstrap` mismatch?

Since TiDB v6.0.0, the default value of `new_collations_enabled_on_first_bootstrap` has changed from `false` to `true`. BR backs up the `new_collations_enabled_on_first_bootstrap` configuration of the upstream cluster and then checks whether the value of this configuration is consistent between the upstream and downstream clusters. If the value is consistent, BR safely restores the data backed up in the upstream cluster to the downstream cluster. If the value is inconsistent, BR does not perform the data restore and reports an error.

Suppose that you have backed up the data in a TiDB cluster of an earlier version of v6.0.0, and you want to restore this data to a TiDB cluster of v6.0.0 or later versions. In this situation, you need to manually check whether the value of `new_collations_enabled_on_first_bootstrap` is consistent between the upstream and downstream clusters:

- If the value is consistent, you can add `--check-requirements=false` to the restoration command to skip this configuration check.
- If the value is inconsistent, and you forcibly perform the restoration, BR reports a data validation error.

13.11 TiDB Binlog

13.11.1 TiDB Binlog Cluster Overview

This document introduces the architecture and the deployment of the cluster version of TiDB Binlog.

TiDB Binlog is a tool used to collect binlog data from TiDB and provide near real-time backup and replication to downstream platforms.

TiDB Binlog has the following features:

- **Data replication:** replicate the data in the TiDB cluster to other databases
- **Real-time backup and restoration:** back up the data in the TiDB cluster and restore the TiDB cluster when the cluster fails

> **Note:**
>
> TiDB Binlog is not compatible with some features introduced in TiDB v5.0 and they cannot be used together. For details, see Notes. It is recommended to use TiCDC instead of TiDB Binlog.

13.11.1.1 TiDB Binlog architecture

The TiDB Binlog architecture is as follows:

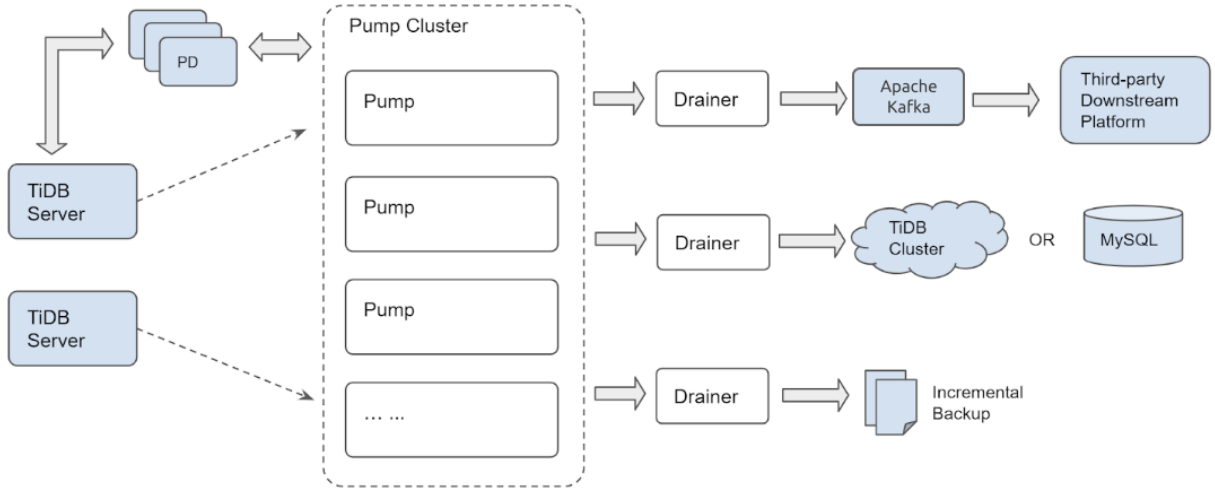


Figure 300: TiDB Binlog architecture

The TiDB Binlog cluster is composed of Pump and Drainer.

13.11.1.1.1 Pump

Pump is used to record the binlogs generated in TiDB, sort the binlogs based on the commit time of the transaction, and send binlogs to Drainer for consumption.

13.11.1.1.2 Drainer

Drainer collects and merges binlogs from each Pump, converts the binlog to SQL or data of a specific format, and replicates the data to a specific downstream platform.

13.11.1.1.3 `binlogctl` guide

`binlogctl` is an operations tool for TiDB Binlog with the following features:

- Obtaining the current `tso` of TiDB cluster
- Checking the Pump/Drainer state
- Modifying the Pump/Drainer state
- Pausing or closing Pump/Drainer

13.11.1.2 Main features

- Multiple Pumps form a cluster which can scale out horizontally
- TiDB uses the built-in Pump Client to send the binlog to each Pump
- Pump stores binlogs and sends the binlogs to Drainer in order
- Drainer reads binlogs of each Pump, merges and sorts the binlogs, and sends the binlogs downstream
- Drainer supports relay log. By the relay log, Drainer ensures that the downstream clusters are in a consistent state.

13.11.1.3 Notes

- In v5.1, the incompatibility between the clustered index feature introduced in v5.0 and TiDB Binlog has been resolved. After you upgrade TiDB Binlog and TiDB Server to v5.1 and enable TiDB Binlog, TiDB will support creating tables with clustered indexes; data insertion, deletion, and update on the created tables with clustered indexes will be replicated to the downstream via TiDB Binlog. When you use TiDB Binlog to replicate the tables with clustered indexes, pay attention to the following:
  - If you have upgraded the cluster to v5.1 from v5.0 by manually controlling the upgrade sequence, make sure that TiDB binlog is upgraded to v5.1 before upgrading the TiDB server to v5.1.
  - It is recommended to configure the system variable `tidb_enable_clustered_index` ↪ to a same value to ensure that the structure of TiDB clustered index tables between the upstream and downstream is consistent.
- TiDB Binlog is incompatible with the following features introduced in TiDB v5.0 and they cannot be used together.

    - TiDB Clustered Index: After TiDB Binlog is enabled, TiDB does not allow creating clustered indexes with non-single integer columns as primary keys; data insertion, deletion, and update of the created clustered index tables will not be replicated downstream via TiDB Binlog. If you need to replicate tables with clustered indexes, upgrade your cluster to v5.1 or use TiCDC instead.
    - TiDB system variable tidb_enable_async_commit: After TiDB Binlog is enabled, performance cannot be improved by enabling this option. It is recommended to use TiCDC instead of TiDB Binlog.
    - TiDB system variable tidb_enable_1pc: After TiDB Binlog is enabled, performance cannot be improved by enabling this option. It is recommended to use TiCDC instead of TiDB Binlog.

- TiDB Binlog is incompatible with the following feature introduced in TiDB v4.0.7 and they cannot be used together:

    - TiDB system variable tidb_enable_amend_pessimistic_txn: The two features have compatibility issues. Using them together might cause the issue that TiDB Binlog replicates data inconsistently.

- Drainer supports replicating binlogs to MySQL, TiDB, Kafka or local files. If you need to replicate binlogs to other Drainer unsuppored destinations, you can set Drainer to replicate the binlog to Kafka and read the data in Kafka for customized processing according to binlog consumer protocol. See Binlog Consumer Client User Guide.

- To use TiDB Binlog for recovering incremental data, set the config `db-type` to `file` ↪ (local files in the proto buffer format). Drainer converts the binlog to data in the specified proto buffer format and writes the data to local files. In this way, you can use Reparo to recover data incrementally.

  Pay attention to the value of `db-type`:

    - If your TiDB version is earlier than 2.1.9, set `db-type="pb"`.
    - If your TiDB version is 2.1.9 or later, set `db-type="file"` or `db-type="pb"`.

- If the downstream is MySQL, MariaDB, or another TiDB cluster, you can use sync-diff-inspector to verify the data after data replication.

13.11.2 TiDB Binlog Tutorial

This tutorial starts with a simple TiDB Binlog deployment with a single node of each component (Placement Driver, TiKV Server, TiDB Server, Pump, and Drainer), set up to push data into a MariaDB Server instance.

This tutorial is targeted toward users who have some familiarity with the TiDB Architecture, who may have already set up a TiDB cluster (not mandatory), and who wants to gain hands-on experience with TiDB Binlog. This tutorial is a good way to "kick the tires" of TiDB Binlog and to familiarize yourself with the concepts of its architecture.

Warning:

The instructions to deploy TiDB in this tutorial should **not** be used to deploy TiDB in a production or development setting.

This tutorial assumes you're using a modern Linux distribution on x86-64. A minimal CentOS 7 installation running in VMware is used in this tutorial for the examples. It's recommended that you start from a clean install, so that you aren't impacted by quirks of your existing environment. If you don't want to use local virtualization, you can easily start a CentOS 7 VM using your cloud service.

13.11.2.1 TiDB Binlog Overview

TiDB Binlog is a solution to collect binary log data from TiDB and provide real-time data backup and replication. It pushes incremental data updates from a TiDB Server cluster into downstream platforms.

You can use TiDB Binlog for incremental backups, to replicate data from one TiDB cluster to another, or to send TiDB updates through Kafka to a downstream platform of your choice.

TiDB Binlog is particularly useful when you migrate data from MySQL or MariaDB to TiDB, in which case you may use the TiDB DM (Data Migration) platform to get data from a MySQL/MariaDB cluster into TiDB, and then use TiDB Binlog to keep a separate, downstream MySQL/MariaDB instance/cluster in sync with your TiDB cluster. TiDB Binlog enables application traffic to TiDB to be pushed to a downstream MySQL or MariaDB instance/cluster, which reduces the risk of a migration to TiDB because you can easily revert the application to MySQL or MariaDB without downtime or data loss.

See TiDB Binlog Cluster User Guide for more information.

13.11.2.2 Architecture

TiDB Binlog comprises two components: the **Pump** and the **Drainer**. Several Pump nodes make up a pump cluster. Each Pump node connects to TiDB Server instances and receives updates made to each of the TiDB Server instances in a cluster. A Drainer connects to the Pump cluster and transforms the received updates into the correct format for a particular downstream destination, for example, Kafka, another TiDB Cluster or a MySQL/MariaDB server.

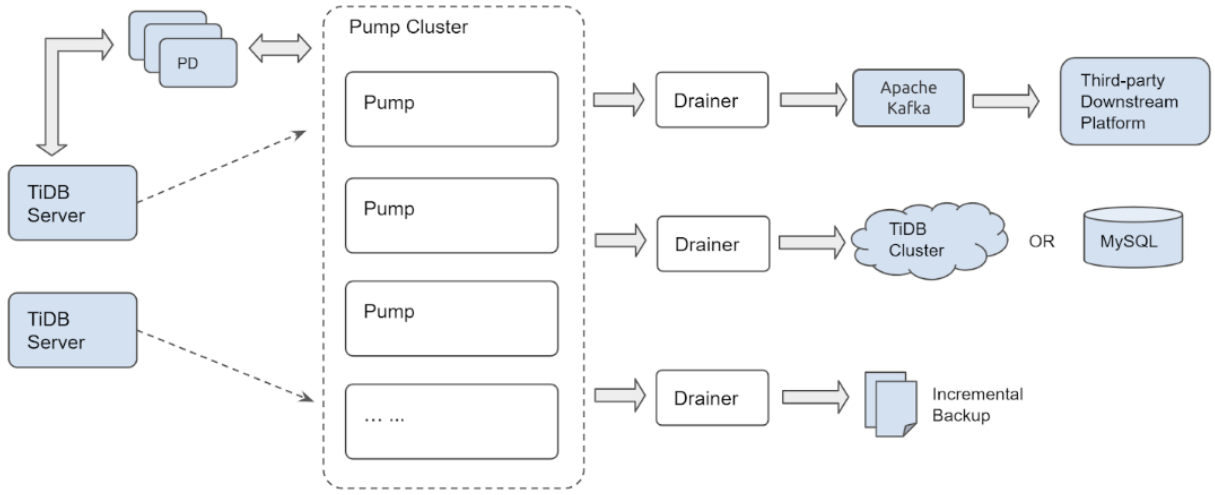


Figure 301: TiDB-Binlog architecture

The clustered architecture of Pump ensures that updates won't be lost as new TiDB Server instances join or leave the TiDB Cluster or Pump nodes join or leave the Pump cluster.

13.11.2.3 Installation

We're using MariaDB Server in this case instead of MySQL Server because RHEL/CentOS 7 includes MariaDB Server in their default package repositories. We'll need the client as well as the server for later use. Let's install them now:

```
sudo yum install -y mariadb-server
```

```
curl -L https://download.pingcap.org/tidb-community-server-v6.1.7-linux-
    ↪ amd64.tar.gz | tar xzf -
cd tidb-latest-linux-amd64
```

Expected output:

```
[kolbe@localhost ~]$ curl -LO https://download.pingcap.org/tidb-latest-linux
    ↪ -amd64.tar.gz | tar xzf -
  % Total    % Received % Xferd Average Speed Time  Time    Time Current
                                Dload Upload  Total  Spent   Left Speed
100  368M 100  368M   0     0  8394k     0  0:00:44 0:00:44 --:--:-- 11.1M
[kolbe@localhost ~]$ cd tidb-latest-linux-amd64
[kolbe@localhost tidb-latest-linux-amd64]$
```

13.11.2.4 Configuration

Now we'll start a simple TiDB cluster, with a single instance for each of `pd-server`, `tikv-server`, and `tidb-server`.

Populate the config files using:

```
printf > pd.toml %s\\n 'log-file="pd.log"' 'data-dir="pd.data"'
printf > tikv.toml %s\\n 'log-file="tikv.log"' '[storage]' 'data-dir="tikv.
    ↪ data"' '[pd]' 'endpoints=["127.0.0.1:2379"]' '[rocksdb]' max-open-
    ↪ files=1024 '[raftdb]' max-open-files=1024
printf > pump.toml %s\\n 'log-file="pump.log"' 'data-dir="pump.data"' 'addr
    ↪ ="127.0.0.1:8250"' 'advertise-addr="127.0.0.1:8250"' 'pd-urls="http
    ↪ ://127.0.0.1:2379"'
printf > tidb.toml %s\\n 'store="tikv"' 'path="127.0.0.1:2379"' '[log.file
    ↪ ]' 'filename="tidb.log"' '[binlog]' 'enable=true'
printf > drainer.toml %s\\n 'log-file="drainer.log"' '[syncer]' 'db-type="
    ↪ mysql"' '[syncer.to]' 'host="127.0.0.1"' 'user="root"' 'password=""'
    ↪ 'port=3306'
```

Use the following commands to see the configuration details:

```
for f in *.toml; do echo "$f:"; cat "$f"; echo; done
```

Expected output:

```
drainer.toml:
log-file="drainer.log"
[syncer]
db-type="mysql"
[syncer.to]
host="127.0.0.1"
user="root"
password=""
port=3306

pd.toml:
log-file="pd.log"
data-dir="pd.data"

pump.toml:
log-file="pump.log"
data-dir="pump.data"
addr="127.0.0.1:8250"
advertise-addr="127.0.0.1:8250"
pd-urls="http://127.0.0.1:2379"

tidb.toml:
```

```
store="tikv"
path="127.0.0.1:2379"
[log.file]
filename="tidb.log"
[binlog]
enable=true

tikv.toml:
log-file="tikv.log"
[storage]
data-dir="tikv.data"
[pd]
endpoints=["127.0.0.1:2379"]
[rocksdb]
max-open-files=1024
[raftdb]
max-open-files=1024
```

13.11.2.5 Bootstrapping

Now we can start each component. This is best done in a specific order - firstly the Placement Driver (PD), then TiKV Server, then Pump (because TiDB must connect to the Pump service to send the binary log), and finally the TiDB Server.

Start all the services using:

```
./bin/pd-server --config=pd.toml &>pd.out &
./bin/tikv-server --config=tikv.toml &>tikv.out &
./pump --config=pump.toml &>pump.out &
sleep 3
./bin/tidb-server --config=tidb.toml &>tidb.out &
```

Expected output:

```
[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/pd-server --config=pd.toml
    ↪ &>pd.out &
[1] 20935
[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/tikv-server --config=tikv.
    ↪ toml &>tikv.out &
[2] 20944
[kolbe@localhost tidb-latest-linux-amd64]$ ./pump --config=pump.toml &>pump.
    ↪ out &
[3] 21050
[kolbe@localhost tidb-latest-linux-amd64]$ sleep 3
[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/tidb-server --config=tidb.
    ↪ toml &>tidb.out &
```

```
[4] 21058
```

If you execute `jobs`, you should see a list of running daemons:

```
[kolbe@localhost tidb-latest-linux-amd64]$ jobs
[1]   Running                 ./bin/pd-server --config=pd.toml &>pd.out &
[2]   Running                 ./bin/tikv-server --config=tikv.toml &>tikv.out &
[3]-  Running                 ./pump --config=pump.toml &>pump.out &
[4]+  Running                 ./bin/tidb-server --config=tidb.toml &>tidb.out &
```

If one of the services has failed to start (if you see "`Exit 1`" instead of "`Running`", for example), try to restart that individual service.

13.11.2.6 Connecting

You should have all 4 components of our TiDB Cluster running now, and you can now connect to the TiDB Server on port 4000 using the MariaDB/MySQL command-line client:

```
mysql -h 127.0.0.1 -P 4000 -u root -e 'select tidb_version()\G'
```

Expected output:

```
[kolbe@localhost tidb-latest-linux-amd64]$ mysql -h 127.0.0.1 -P 4000 -u
   ↪ root -e 'select tidb_version()\G'
*************************** 1. row ***************************
tidb_version(): Release Version: v3.0.0-beta.1-154-gd5afff70c
Git Commit Hash: d5afff70cdd825d5fab125c8e52e686cc5fb9a6e
Git Branch: master
UTC Build Time: 2019-04-24 03:10:00
GoVersion: go version go1.12 linux/amd64
Race Enabled: false
TiKV Min Version: 2.1.0-alpha.1-ff3dd160846b7d1aed9079c389fc188f7f5ea13e
Check Table Before Drop: false
```

At this point we have a TiDB Cluster running, and we have `pump` reading binary logs from the cluster and storing them as relay logs in its data directory. The next step is to start a MariaDB server that `drainer` can write to.

Start `drainer` using:

```
sudo systemctl start mariadb
./drainer --config=drainer.toml &>drainer.out &
```

If you are using an operating system that makes it easier to install MySQL server, that's also OK. Just make sure it's listening on port 3306 and that you can either connect to it as user "root" with an empty password, or adjust drainer.toml as necessary.

```
mysql -h 127.0.0.1 -P 3306 -u root
```

```
show databases;
```

Expected output:

```
[kolbe@localhost ~]$ mysql -h 127.0.0.1 -P 3306 -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 20
Server version: 5.5.60-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
    ↪ statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
| tidb_binlog        |
+--------------------+
5 rows in set (0.01 sec)
```

Here we can already see the `tidb_binlog` database, which contains the `checkpoint` table used by `drainer` to record up to what point binary logs from the TiDB cluster have been applied.

```
MariaDB [tidb_binlog]> use tidb_binlog;
Database changed
MariaDB [tidb_binlog]> select * from checkpoint;
+--------------------+-------------------------------------------------+
| clusterID          | checkPoint                                      |
+--------------------+-------------------------------------------------+
| 6678715361817107733 | {"commitTS":407637466476445697,"ts-map":{}} |
+--------------------+-------------------------------------------------+
1 row in set (0.00 sec)
```

Now, let's open another client connection to the TiDB server, so that we can create a table and insert some rows into it. (It's recommended that you do this under a GNU screen so you can keep multiple clients open at the same time.)

```
mysql -h 127.0.0.1 -P 4000 --prompt='TiDB [\d]> ' -u root
```

```
create database tidbtest;
use tidbtest;
create table t1 (id int unsigned not null AUTO_INCREMENT primary key);
insert into t1 () values (),(),(),(),();
select * from t1;
```

Expected output:

```
TiDB [(none)]> create database tidbtest;
Query OK, 0 rows affected (0.12 sec)

TiDB [(none)]> use tidbtest;
Database changed
TiDB [tidbtest]> create table t1 (id int unsigned not null AUTO_INCREMENT
    ↪ primary key);
Query OK, 0 rows affected (0.11 sec)

TiDB [tidbtest]> insert into t1 () values (),(),(),(),();
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0

TiDB [tidbtest]> select * from t1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
+----+
5 rows in set (0.00 sec)
```

Switching back to the MariaDB client, we should find the new database, new table, and the newly inserted rows:

```
use tidbtest;
show tables;
select * from t1;
```

Expected output:

```
MariaDB [(none)]> use tidbtest;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```
Database changed
MariaDB [tidbtest]> show tables;
+--------------------+
| Tables_in_tidbtest |
+--------------------+
| t1                 |
+--------------------+
1 row in set (0.00 sec)

MariaDB [tidbtest]> select * from t1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
+----+
5 rows in set (0.00 sec)
```

You should see the same rows that you inserted into TiDB when querying the MariaDB server. Congratulations! You've just set up TiDB Binlog!

13.11.2.7 binlogctl

Information about Pumps and Drainers that have joined the cluster is stored in PD. You can use the binlogctl tool query and manipulate information about their states. See binlogctl guide for more information.

Use `binlogctl` to get a view of the current status of Pumps and Drainers in the cluster:

```
./binlogctl -cmd drainers
./binlogctl -cmd pumps
```

Expected output:

```
[kolbe@localhost tidb-latest-linux-amd64]$ ./binlogctl -cmd drainers
[2019/04/11 17:44:10.861 -04:00] [INFO] [nodes.go:47] ["query node"] [type=
    ↪ drainer] [node="{NodeID: localhost.localdomain:8249, Addr:
    ↪ 192.168.236.128:8249, State: online, MaxCommitTS: 407638907719778305,
    ↪  UpdateTime: 2019-04-11 17:44:10 -0400 EDT}"]

[kolbe@localhost tidb-latest-linux-amd64]$ ./binlogctl -cmd pumps
[2019/04/11 17:44:13.904 -04:00] [INFO] [nodes.go:47] ["query node"] [type=
    ↪ pump] [node="{NodeID: localhost.localdomain:8250, Addr:
```

```
    ↪ 192.168.236.128:8250, State: online, MaxCommitTS: 407638914024079361,
    ↪  UpdateTime: 2019-04-11 17:44:13 -0400 EDT}"]
```

If you kill a Drainer, the cluster puts it in the "paused" state, which means that the cluster expects it to rejoin:

```
pkill drainer
./binlogctl -cmd drainers
```

Expected output:

```
[kolbe@localhost tidb-latest-linux-amd64]$ pkill drainer
[kolbe@localhost tidb-latest-linux-amd64]$ ./binlogctl -cmd drainers
[2019/04/11 17:44:22.640 -04:00] [INFO] [nodes.go:47] ["query node"] [type=
    ↪ drainer] [node="{NodeID: localhost.localdomain:8249, Addr:
    ↪ 192.168.236.128:8249, State: paused, MaxCommitTS: 407638915597467649,
    ↪  UpdateTime: 2019-04-11 17:44:18 -0400 EDT}"]
```

You can use "NodeIDs" with `binlogctl` to control individual nodes. In this case, the NodeID of the drainer is "localhost.localdomain:8249" and the NodeID of the Pump is "localhost.localdomain:8250".

The main use of `binlogctl` in this tutorial is likely to be in the event of a cluster restart. If you end all processes in the TiDB cluster and try to restart them (not including the downstream MySQL/MariaDB server or Drainer), Pump will refuse to start because it cannot contact Drainer and believe that Drainer is still "online".

There are 3 solutions to this issue:

- Stop Drainer using `binlogctl` instead of killing the process:

  ```
  ./binlogctl --pd-urls=http://127.0.0.1:2379 --cmd=drainers
  ./binlogctl --pd-urls=http://127.0.0.1:2379 --cmd=offline-drainer --
      ↪ node-id=localhost.localdomain:8249
  ```

- Start Drainer *before* starting Pump.

- Use `binlogctl` after starting PD (but before starting Drainer and Pump) to update the state of the paused Drainer:

  ```
  ./binlogctl --pd-urls=http://127.0.0.1:2379 --cmd=update-drainer --node
      ↪ -id=localhost.localdomain:8249 --state=offline
  ```

13.11.2.8 Cleanup

To stop the TiDB cluster and TiDB Binlog processes, you can execute `pkill -P $$` in the shell where you started all the processes that form the cluster (pd-server, tikv-server, pump, tidb-server, drainer). To give each component enough time to shut down cleanly, it's helpful to stop them in a particular order:

```
for p in tidb-server drainer pump tikv-server pd-server; do pkill "$p";
   ↪ sleep 1; done
```

Expected output:

```
kolbe@localhost tidb-latest-linux-amd64]$ for p in tidb-server drainer pump
   ↪ tikv-server pd-server; do pkill "$p"; sleep 1; done
[4]- Done                    ./bin/tidb-server --config=tidb.toml &>tidb.out
[5]+ Done                    ./drainer --config=drainer.toml &>drainer.out
[3]+ Done                    ./pump --config=pump.toml &>pump.out
[2]+ Done                    ./bin/tikv-server --config=tikv.toml &>tikv.out
[1]+ Done                    ./bin/pd-server --config=pd.toml &>pd.out
```

If you wish to restart the cluster after all services exit, use the same commands you ran originally to start the services. As discussed in the binlogctl section above, you'll need to start `drainer` before `pump`, and `pump` before `tidb-server`.

```
./bin/pd-server --config=pd.toml &>pd.out &
./bin/tikv-server --config=tikv.toml &>tikv.out &
./drainer --config=drainer.toml &>drainer.out &
sleep 3
./pump --config=pump.toml &>pump.out &
sleep 3
./bin/tidb-server --config=tidb.toml &>tidb.out &
```

If any of the components fail to start, try to restart the failed individual component(s).

13.11.2.9 Conclusion

In this tutorial, we've set up TiDB Binlog to replicate from a TiDB cluster to a downstream MariaDB server, using a cluster with a single Pump and a single Drainer. As we've seen, TiDB Binlog is a comprehensive platform for capturing and processing changes to a TiDB cluster.

In a more robust development, testing, or production deployment, you'd have multiple TiDB servers for high availability and scaling purposes, and you'd use multiple Pump instances to ensure that application traffic to TiDB server instances is unaffected by problems in the Pump cluster. You may also use additional Drainer instances to push updates to different downstream platforms or to implement incremental backups.

13.11.3 TiDB Binlog Cluster Deployment

This document describes how to deploy TiDB Binlog using a Binary package.

13.11.3.1 Hardware requirements

Pump and Drainer are deployed and operate on 64-bit universal hardware server platforms with Intel x86-64 architecture.

In environments of development, testing and production, the requirements on server hardware are as follows:

Service	The Number of Servers	CPU	Disk	Memory
Pump	3	8 core+	SSD, 200 GB+	16G
Drainer	1	8 core+	SAS, 100 GB+ (If binlogs are output as local files, the disk size depends on how long these files are retained.)	16G

13.11.3.2 Deploy TiDB Binlog using TiUP

It is recommended to deploy TiDB Binlog using TiUP. To do that, when deploying TiDB using TiUP, you need to add the node information of `drainer` and `pump` of TiDB Binlog in TiDB Binlog Deployment Topology. For detailed deployment information, refer to Deploy a TiDB Cluster Using TiUP.

13.11.3.3 Deploy TiDB Binlog using a binary package

13.11.3.3.1 Download the official binary package

The binary package of TiDB Binlog is included in the TiDB Toolkit. To download the TiDB Toolkit, see Download TiDB Tools.

13.11.3.3.2 The usage example

Assuming that you have three PD nodes, one TiDB node, two Pump nodes, and one Drainer node, the information of each node is as follows:

Node	IP
TiDB	192.168.0.10
PD1	192.168.0.16
PD2	192.168.0.15
PD3	192.168.0.14

Node	IP
Pump	192.168.0.11
Pump	192.168.0.12
Drainer	192.168.0.13

The following part shows how to use Pump and Drainer based on the nodes above.

1. Deploy Pump using the binary.

    - To view the command line parameters of Pump, execute `./pump -help`:

```
Usage of Pump:
-L string
    the output information level of logs: debug, info, warn, error,
        ↪  fatal ("info" by default)
-V
    the print version information
-addr string
    the RPC address through which Pump provides the service (-addr=
        ↪ "192.168.0.11:8250")
-advertise-addr string
    the RPC address through which Pump provides the external
        ↪ service (-advertise-addr="192.168.0.11:8250")
-config string
    the path of the configuration file. If you specify the
        ↪ configuration file, Pump reads the configuration in the
        ↪ configuration file first. If the corresponding
        ↪ configuration also exits in the command line parameters,
        ↪ Pump uses the configuration of the command line
        ↪ parameters to cover that of the configuration file.
-data-dir string
    the path where the Pump data is stored
-gc int
    the number of days to retain the data in Pump ("7" by default)
-heartbeat-interval int
    the interval of the heartbeats Pump sends to PD (in seconds)
-log-file string
    the file path of logs
-log-rotate string
    the switch frequency of logs (hour/day)
-metrics-addr string
    the Prometheus Pushgateway address. If not set, it is forbidden
        ↪  to report the monitoring metrics.
-metrics-interval int
```

```
   the report frequency of the monitoring metrics ("15" by default
      ↪ , in seconds)
-node-id string
   the unique ID of a Pump node. If you do not specify this ID,
      ↪ the system automatically generates an ID based on the
      ↪ host name and listening port.
-pd-urls string
   the address of the PD cluster nodes (-pd-urls="http
      ↪ ://192.168.0.16:2379,http://192.168.0.15:2379,http
      ↪ ://192.168.0.14:2379")
-fake-binlog-interval int
   the frequency at which a Pump node generates fake binlog ("3"
      ↪ by default, in seconds)
```

- Taking deploying Pump on "192.168.0.11" as an example, the Pump configuration file is as follows:

```
# Pump Configuration

# the bound address of Pump
addr = "192.168.0.11:8250"

# the address through which Pump provides the service
advertise-addr = "192.168.0.11:8250"

# the number of days to retain the data in Pump ("7" by default)
gc = 7

# the directory where the Pump data is stored
data-dir = "data.pump"

# the interval of the heartbeats Pump sends to PD (in seconds)
heartbeat-interval = 2

# the address of the PD cluster nodes (each separated by a comma
   ↪ with no whitespace)
pd-urls = "http://192.168.0.16:2379,http://192.168.0.15:2379,http
   ↪ ://192.168.0.14:2379"

# [security]
# This section is generally commented out if no special security
   ↪ settings are required.
# The file path containing a list of trusted SSL CAs connected to
   ↪ the cluster.
# ssl-ca = "/path/to/ca.pem"
```

```
# The path to the X509 certificate in PEM format that is connected
    ↪ to the cluster.
# ssl-cert = "/path/to/drainer.pem"
# The path to the X509 key in PEM format that is connected to the
    ↪ cluster.
# ssl-key = "/path/to/drainer-key.pem"

# [storage]
# Set to true (by default) to guarantee reliability by ensuring
    ↪ binlog data is flushed to the disk
# sync-log = true

# When the available disk capacity is less than the set value, Pump
    ↪  stops writing data.
# 42 MB -> 42000000, 42 mib -> 44040192
# default: 10 gib
# stop-write-at-available-space = "10 gib"
# The LSM DB settings embedded in Pump. Unless you know this part
    ↪ well, it is usually commented out.
# [storage.kv]
# block-cache-capacity = 8388608
# block-restart-interval = 16
# block-size = 4096
# compaction-L0-trigger = 8
# compaction-table-size = 67108864
# compaction-total-size = 536870912
# compaction-total-size-multiplier = 8.0
# write-buffer = 67108864
# write-L0-pause-trigger = 24
# write-L0-slowdown-trigger = 17
```

- The example of starting Pump:

```
./pump -config pump.toml
```

If the command line parameters is the same with the configuration file parameters, the values of command line parameters are used.

2. Deploy Drainer using binary.

- To view the command line parameters of Drainer, execute `./drainer -help`:

```
Usage of Drainer:
-L string
    the output information level of logs: debug, info, warn, error,
        ↪  fatal ("info" by default)
```

```
-V
    the print version information
-addr string
    the address through which Drainer provides the service (-addr="
        ↪ 192.168.0.13:8249")
-c int
    the number of the concurrency of the downstream for replication
        ↪ . The bigger the value, the better throughput performance
        ↪  of the concurrency ("1" by default).
-cache-binlog-count int
    the limit on the number of binlog items in the cache ("8" by
        ↪ default)
    If a large single binlog item in the upstream causes OOM in
        ↪ Drainer, try to lower the value of this parameter to
        ↪ reduce memory usage.
-config string
    the directory of the configuration file. Drainer reads the
        ↪ configuration file first.
    If the corresponding configuration exists in the command line
        ↪ parameters, Drainer uses the configuration of the command
        ↪  line parameters to cover that of the configuration file.
-data-dir string
    the directory where the Drainer data is stored ("data.drainer"
        ↪ by default)
-dest-db-type string
    the downstream service type of Drainer
    The value can be "mysql", "tidb", "kafka", and "file". ("mysql"
        ↪  by default)
-detect-interval int
    the interval of checking the online Pump in PD ("10" by default
        ↪ , in seconds)
-disable-detect
    whether to disable the conflict monitoring
-disable-dispatch
    whether to disable the SQL feature of splitting a single binlog
        ↪  file. If it is set to "true", each binlog file is
        ↪ restored to a single transaction for replication based on
        ↪  the order of binlogs.
    It is set to "False", when the downstream is MySQL.
-ignore-schemas string
    the db filter list ("INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,
        ↪ mysql,test" by default)
    It does not support the Rename DDL operation on tables of `
        ↪ ignore schemas`.
-initial-commit-ts
```

```
    If Drainer does not have the related breakpoint information,
       ↪ you can configure the related breakpoint information
       ↪ using this parameter. ("-1" by default)
    If the value of this parameter is `-1`, Drainer automatically
       ↪ obtains the latest timestamp from PD.
-log-file string
    the path of the log file
-log-rotate string
    the switch frequency of log files, hour/day
-metrics-addr string
    the Prometheus Pushgateway address
    It it is not set, the monitoring metrics are not reported.
-metrics-interval int
    the report frequency of the monitoring metrics ("15" by default
       ↪ , in seconds)
-node-id string
    the unique ID of a Drainer node. If you do not specify this ID,
       ↪  the system automatically generates an ID based on the
       ↪ host name and listening port.
-pd-urls string
    the address of the PD cluster nodes (-pd-urls="http
       ↪ ://192.168.0.16:2379,http://192.168.0.15:2379,http
       ↪ ://192.168.0.14:2379")
-safe-mode
    Whether to enable safe mode so that data can be written into
       ↪ the downstream MySQL/TiDB repeatedly.
    This mode replaces the `INSERT` statement with the `REPLACE`
       ↪ statement and splits the `UPDATE` statement into `DELETE`
       ↪  plus `REPLACE`.
-txn-batch int
    the number of SQL statements of a transaction which are output
       ↪ to the downstream database ("1" by default)
```

- Taking deploying Drainer on "192.168.0.13" as an example, the Drainer configuration file is as follows:

```
# Drainer Configuration.

# the address through which Drainer provides the service
   ↪ ("192.168.0.13:8249")
addr = "192.168.0.13:8249"

# the address through which Drainer provides the external service
advertise-addr = "192.168.0.13:8249"
```

```
# the interval of checking the online Pump in PD ("10" by default,
    ↪ in seconds)
detect-interval = 10

# the directory where the Drainer data is stored "data.drainer" by
    ↪ default)
data-dir = "data.drainer"

# the address of the PD cluster nodes (each separated by a comma
    ↪ with no whitespace)
pd-urls = "http://192.168.0.16:2379,http://192.168.0.15:2379,http
    ↪ ://192.168.0.14:2379"

# the directory of the log file
log-file = "drainer.log"

# Drainer compresses the data when it gets the binlog from Pump.
    ↪ The value can be "gzip". If it is not configured, it will
    ↪ not be compressed
# compressor = "gzip"

# [security]
# This section is generally commented out if no special security
    ↪ settings are required.
# The file path containing a list of trusted SSL CAs connected to
    ↪ the cluster.
# ssl-ca = "/path/to/ca.pem"
# The path to the X509 certificate in PEM format that is connected
    ↪ to the cluster.
# ssl-cert = "/path/to/pump.pem"
# The path to the X509 key in PEM format that is connected to the
    ↪ cluster.
# ssl-key = "/path/to/pump-key.pem"

# Syncer Configuration
[syncer]
# If the item is set, the sql-mode will be used to parse the DDL
    ↪ statement.
# If the downstream database is MySQL or TiDB, then the downstream
    ↪ sql-mode
# is also set to this value.
# sql-mode = "STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION"

# the number of SQL statements of a transaction that are output to
    ↪ the downstream database ("20" by default)
```

```
txn-batch = 20

# the number of the concurrency of the downstream for replication.
   ↪ The bigger the value,
# the better throughput performance of the concurrency ("16" by
   ↪ default)
worker-count = 16

# whether to disable the SQL feature of splitting a single binlog
   ↪ file. If it is set to "true",
# each binlog file is restored to a single transaction for
   ↪ replication based on the order of binlogs.
# If the downstream service is MySQL, set it to "False".
disable-dispatch = false

# In safe mode, data can be written into the downstream MySQL/TiDB
   ↪ repeatedly.
# This mode replaces the `INSERT` statement with the `REPLACE`
   ↪ statement and replaces the `UPDATE` statement with `DELETE`
   ↪ plus `REPLACE` statements.
safe-mode = false

# the downstream service type of Drainer ("mysql" by default)
# Valid value: "mysql", "tidb", "file", and "kafka".
db-type = "mysql"

# If `commit ts` of the transaction is in the list, the transaction
   ↪  is filtered and not replicated to the downstream.
ignore-txn-commit-ts = []

# the db filter list ("INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql,
   ↪ test" by default)
# Does not support the Rename DDL operation on tables of `ignore
   ↪ schemas`.
ignore-schemas = "INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql"

# `replicate-do-db` has priority over `replicate-do-table`. When
   ↪ they have the same `db` name,
# regular expressions are supported for configuration.
# The regular expression should start with "~".

# replicate-do-db = ["~^b.*","s1"]

# [syncer.relay]
# It saves the directory of the relay log. The relay log is not
```

```
    ↪ enabled if the value is empty.
# The configuration only comes to effect if the downstream is TiDB
    ↪ or MySQL.
# log-dir = ""
# the maximum size of each file
# max-file-size = 10485760

# [[syncer.replicate-do-table]]
# db-name ="test"
# tbl-name = "log"

# [[syncer.replicate-do-table]]
# db-name ="test"
# tbl-name = "~^a.*"

# Ignore the replication of some tables
# [[syncer.ignore-table]]
# db-name = "test"
# tbl-name = "log"

# the server parameters of the downstream database when `db-type`
    ↪ is set to "mysql"
[syncer.to]
host = "192.168.0.13"
user = "root"
# If you do not want to set a cleartext `password` in the
    ↪ configuration file, you can create `encrypted_password`
    ↪ using `./binlogctl -cmd encrypt -text string`.
# When you have created an `encrypted_password` that is not empty,
    ↪ the `password` above will be ignored, because `
    ↪ encrypted_password` and `password` cannot take effect at the
    ↪  same time.
password = ""
encrypted_password = ""
port = 3306

[syncer.to.checkpoint]
# When the checkpoint type is "mysql" or "tidb", this option can be
# enabled to change the database that saves the checkpoint
# schema = "tidb_binlog"
# Currently only the "mysql" and "tidb" checkpoint types are
    ↪ supported
# You can remove the comment tag to control where to save the
    ↪ checkpoint
# The default method of saving the checkpoint for the downstream db
```

```
   ↪ -type:
# mysql/tidb -> in the downstream MySQL or TiDB database
# file/kafka -> file in `data-dir`
# type = "mysql"
# host = "127.0.0.1"
# user = "root"
# password = ""
# `encrypted_password` is encrypted using `./binlogctl -cmd encrypt
   ↪  -text string`.
# When `encrypted_password` is not empty, the `password` above will
   ↪  be ignored.
# encrypted_password = ""
# port = 3306

# the directory where the binlog file is stored when `db-type` is
   ↪ set to `file`
# [syncer.to]
# dir = "data.drainer"

# the Kafka configuration when `db-type` is set to "kafka"
# [syncer.to]
# only one of kafka-addrs and zookeeper-addrs is needed. If both
   ↪ are present, the program gives priority
# to the kafka address in zookeeper
# zookeeper-addrs = "127.0.0.1:2181"
# kafka-addrs = "127.0.0.1:9092"
# kafka-version = "0.8.2.0"
# The maximum number of messages (number of binlogs) in a broker
   ↪ request. If it is left blank or a value smaller than 0 is
   ↪ configured, the default value 1024 is used.
# kafka-max-messages = 1024
# The maximum size of a broker request (unit: byte). The default
   ↪ value is 1 GiB and the maximum value is 2 GiB.
# kafka-max-message-size = 1073741824

# the topic name of the Kafka cluster that saves the binlog data.
   ↪ The default value is <cluster-id>_obinlog.
# To run multiple Drainers to replicate data to the same Kafka
   ↪ cluster, you need to set different `topic-name`s for each
   ↪ Drainer.
# topic-name = ""
```

- Starting Drainer:

Note:
If the downstream is MySQL/TiDB, to guarantee the data integrity, you need to obtain the `initial-commit-ts` value and make a full backup of the data and restore the data before the initial start of Drainer.

When Drainer is started for the first time, use the `initial-commit-ts` parameter.

```
./drainer -config drainer.toml -initial-commit-ts {initial-commit-
    ↪ ts}
```

If the command line parameter and the configuration file parameter are the same, the parameter value in the command line is used.

3. Starting TiDB server:

- After starting Pump and Drainer, start TiDB server with binlog enabled by adding this section to your config file for TiDB server:

```
[binlog]
enable=true
```

- TiDB server will obtain the addresses of registered Pumps from PD and will stream data to all of them. If there are no registered Pump instances, TiDB server will refuse to start or will block starting until a Pump instance comes online.

Note:

- When TiDB is running, you need to guarantee that at least one Pump is running normally.
- To enable the TiDB Binlog service in TiDB server, use the `-enable-binlog` startup parameter in TiDB, or add enable=true to the binlog section of the TiDB server configuration file.
- Make sure that the TiDB Binlog service is enabled in all TiDB instances in a same cluster, otherwise upstream and downstream data inconsistency might occur during data replication. If you want to temporarily run a TiDB instance where the TiDB Binlog service is not enabled, set `run_ddl=false` in the TiDB configuration file.
- Drainer does not support the `rename` DDL operation on the table of `ignore schemas` (the schemas in the filter list).

- If you want to start Drainer in an existing TiDB cluster, generally you need to make a full backup of the cluster data, obtain **snapshot timestamp**, import the data to the target database, and then start Drainer to replicate the incremental data from the corresponding **snapshot timestamp**.
- When the downstream database is TiDB or MySQL, ensure that the `sql_mode` in the upstream and downstream databases are consistent. In other words, the `sql_mode` should be the same when each SQL statement is executed in the upstream and replicated to the downstream. You can execute the `select @@sql_mode;` statement in the upstream and downstream respectively to compare `sql_mode`.
- When a DDL statement is supported in the upstream but incompatible with the downstream, Drainer fails to replicate data. An example is to replicate the `CREATE TABLE t1(a INT)ROW_FORMAT=FIXED;` statement when the downstream database MySQL uses the InnoDB engine. In this case, you can configure skipping transactions in Drainer, and manually execute compatible statements in the downstream database.

13.11.4 TiDB Binlog Cluster Operations

This document introduces the following TiDB Binlog cluster operations:

- The state of a Pump and Drainer nodes
- Starting or exiting a Pump or Drainer process
- Managing the TiDB Binlog cluster by using the binlogctl tool or by directly performing SQL operations in TiDB

13.11.4.1 Pump or Drainer state

Pump or Drainer state description:

- `online`: running normally
- `pausing`: in the pausing process
- `paused`: has been stopped
- `closing`: in the offline process
- `offline`: has been offline

Note:

The state information of a Pump or Drainer node is maintained by the service itself and is regularly updated to the Placement Driver (PD).

13.11.4.2 Starting and exiting a Pump or Drainer process

13.11.4.2.1 Pump

- Starting: When started, the Pump node notifies all Drainer nodes in the `online` state. If the notification is successful, the Pump node sets its state to `online`. Otherwise, the Pump node reports an error, sets its state to `paused` and exits the process.
- Exiting: The Pump node enters the `paused` or `offline` state before the process is exited normally; if the process is exited abnormally (caused by the `kill -9` command, process panic, crash), the node is still in the `online` state.
    - Pause: You can pause a Pump process by using the `kill` command (not `kill -9` ↪), pressing Ctrl+C or using the `pause-pump` command in the binlogctl tool. After receiving the pause instruction, the Pump node sets its state to `pausing`, stops receiving binlog write requests and stops providing binlog data to Drainer nodes. After all threads are safely exited, the Pump node updates its state to `paused` and exits the process.
    - Offline: You can close a Pump process only by using the `offline-pump` command in the binlogctl tool. After receiving the offline instruction, the Pump node sets its state to `closing` and stops receiving the binlog write requests. The Pump node continues providing binlog to Drainer nodes until all binlog data is consumed by Drainer nodes. Then, the Pump node sets its state to `offline` and exits the process.

13.11.4.2.2 Drainer

- Starting: When started, the Drainer node sets its state to `online` and tries to pull binlogs from all Pump nodes which are not in the `offline` state. If it fails to get the binlogs, it keeps trying.
- Exiting: The Drainer node enters the `paused` or `offline` state before the process is exited normally; if the process is exited abnormally (caused by `kill -9`, process panic, crash), the Drainer node is still in the `online` state.
    - Pause: You can pause a Drainer process by using the `kill` command (not `kill` ↪ `-9`), pressing Ctrl+C or using the `pause-drainer` command in the binlogctl tool. After receiving the pause instruction, the Drainer node sets its state to `pausing` and stops pulling binlogs from Pump nodes. After all threads are safely exited, the Drainer node sets its state to `paused` and exits the process.
    - Offline: You can close a Drainer process only by using the `offline-drainer` command in the binlogctl tool. After receiving the offline instruction, the Drainer node sets its state to `closing` and stops pulling binlogs from Pump nodes. After all threads are safely exited, the Drainer node updates its state to `offline` and exits the process.

For how to pause, close, check, and modify the state of Drainer, see the binlogctl guide.

13.11.4.3 Use `binlogctl` to manage Pump/Drainer

`binlogctl` is an operations tool for TiDB Binlog with the following features:

- Checking the state of Pump or Drainer
- Pausing or closing Pump or Drainer
- Handling the abnormal state of Pump or Drainer

For detailed usage of `binlogctl`, refer to binlogctl overview.

13.11.4.4 Use SQL statements to manage Pump or Drainer

To view or modify binlog related states, execute corresponding SQL statements in TiDB.

- Check whether binlog is enabled:

```
show variables like "log_bin";
```

```
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | 0     |
+---------------+-------+
```

When the Value is `0`, binlog is enabled. When the Value is `1`, binlog is disabled.

- Check the status of all the Pump or Drainer nodes:

```
show pump status;
```

```
+--------|----------------|--------|--------------------|--------------------|
   ↪
| NodeID |   Address    | State |  Max_Commit_Ts  |    Update_Time     |
+--------|----------------|--------|--------------------|--------------------|
   ↪
| pump1 | 127.0.0.1:8250 | Online | 408553768673342237 | 2019-05-01
   ↪ 00:00:01 |
+--------|----------------|--------|--------------------|--------------------|
   ↪
| pump2 | 127.0.0.2:8250 | Online | 408553768673342335 | 2019-05-01
   ↪ 00:00:02 |
+--------|----------------|--------|--------------------|--------------------|
   ↪
```

```
show drainer status;
```

```
+----------|----------------|--------|--------------------|---------------------|
   ↪
|  NodeID |     Address    | State |   Max_Commit_Ts   |    Update_Time     |
+----------|----------------|--------|--------------------|---------------------|
   ↪
| drainer1 | 127.0.0.3:8249 | Online | 408553768673342532 | 2019-05-01
   ↪ 00:00:03 |
+----------|----------------|--------|--------------------|---------------------|
   ↪
| drainer2 | 127.0.0.4:8249 | Online | 408553768673345531 | 2019-05-01
   ↪ 00:00:04 |
+----------|----------------|--------|--------------------|---------------------|
   ↪
```

- Modify the state of a Pump or Drainer node in abnormal situations

```
change pump to node_state ='paused' for node_id 'pump1';
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
change drainer to node_state ='paused' for node_id 'drainer1';
```

```
Query OK, 0 rows affected (0.01 sec)
```

Executing the above SQL statements works the same as the `update-pump` or `update ↪ -drainer` commands in binlogctl. Use the above SQL statements **only** when the Pump or Drainer node is in abnormal situations.

Note:

- Checking whether binlog is enabled and the running status of Pump or Drainer is supported in TiDB v2.1.7 and later versions.
- Modifying the status of Pump or Drainer is supported in TiDB v3.0.0-rc.1 and later versions. This feature only supports modifying the status of Pump or Drainer nodes stored in PD. To pause or close the node, use the `binlogctl` tool.

13.11.5 TiDB Binlog Configuration File

This document introduces the configuration items of TiDB Binlog.

13.11.5.1 Pump

This section introduces the configuration items of Pump. For the example of a complete Pump configuration file, see Pump Configuration.

13.11.5.1.1 addr

- Specifies the listening address of HTTP API in the format of `host:port`.
- Default value: `127.0.0.1:8250`

13.11.5.1.2 advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of `host:port`.
- Default value: `127.0.0.1:8250`

13.11.5.1.3 socket

- The Unix socket address that HTTP API listens to.
- Default value: “”

13.11.5.1.4 pd-urls

- Specifies the comma-separated list of PD URLs. If multiple addresses are specified, when the PD client fails to connect to one address, it automatically tries to connect to another address.
- Default value: `http://127.0.0.1:2379`

13.11.5.1.5 data-dir

- Specifies the directory where binlogs and their indexes are stored locally.
- Default value: `data.pump`

13.11.5.1.6 heartbeat-interval

- Specifies the heartbeat interval (in seconds) at which the latest status is reported to PD.
- Default value: `2`

13.11.5.1.7 gen-binlog-interval

- Specifies the interval (in seconds) at which data is written into fake binlog.
- Default value: `3`

13.11.5.1.8 gc

- Specifies the number of days (integer) that binlogs can be stored locally. Binlogs stored longer than the specified number of days are automatically deleted.
- Default value: `7`

13.11.5.1.9 log-file

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: ""

13.11.5.1.10 log-level

- Specifies the log level.
- Default value: `info`

13.11.5.1.11 node-id

- Specifies the Pump node ID. With this ID, this Pump process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8250`.

13.11.5.1.12 security

This section introduces configuration items related to security.

ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, `/path/to/ca.pem`.
- Default value: ""

ssl-cert

- Specifies the path of the X509 certificate file encoded in the Privacy Enhanced Mail (PEM) format. For example, `/path/to/pump.pem`.
- Default value: ""

ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, `/path/to/pump-key.pem`.
- Default value: ""

13.11.5.1.13 storage

This section introduces configuration items related to storage.

sync-log

- Specifies whether to use `fsync` after each **batch** write to binlog to ensure data safety.
- Default value: `true`

kv_chan_cap

- Specifies the number of write requests that the buffer can store before Pump receives these requests.
- Default value: `1048576` (that is, 2 to the power of 20)

slow_write_threshold

- The threshold (in seconds). If it takes longer to write a single binlog file than this specified threshold, the write is considered slow write and `"take a long time to write binlog"` is output in the log.
- Default value: `1`

stop-write-at-available-space

- Binlog write requests is no longer accepted when the available storage space is below this specified value. You can use the format such as `900 MB`, `5 GB`, and `12 GiB` to specify the storage space. If there is more than one Pump node in the cluster, when a Pump node refuses a write request because of the insufficient space, TiDB will automatically write binlogs to other Pump nodes.
- Default value: `10 GiB`

kv

Currently the storage of Pump is implemented based on GoLevelDB. Under `storage` there is also a `kv` subgroup that is used to adjust the GoLevel configuration. The supported configuration items are shown as below:

- block-cache-capacity
- block-restart-interval
- block-size
- compaction-L0-trigger
- compaction-table-size
- compaction-total-size
- compaction-total-size-multiplier
- write-buffer
- write-L0-pause-trigger
- write-L0-slowdown-trigger

For the detailed description of the above items, see GoLevelDB Document.

13.11.5.2 Drainer

This section introduces the configuration items of Drainer. For the example of a complete Drainer configuration file, see Drainer Configuration

13.11.5.2.1 addr

- Specifies the listening address of HTTP API in the format of `host:port`.
- Default value: `127.0.0.1:8249`

13.11.5.2.2 advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of `host:port`.
- Default value: `127.0.0.1:8249`

13.11.5.2.3 log-file

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: “”

13.11.5.2.4 log-level

- Specifies the log level.
- Default value: `info`

13.11.5.2.5 node-id

- Specifies the Drainer node ID. With this ID, this Drainer process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8249`.

13.11.5.2.6 data-dir

- Specifies the directory used to store files that need to be saved during Drainer operation.
- Default value: `data.drainer`

13.11.5.2.7 detect-interval

- Specifies the interval (in seconds) at which PD updates the Pump information.
- Default value: `5`

13.11.5.2.8 pd-urls

- The comma-separated list of PD URLs. If multiple addresses are specified, the PD client will automatically attempt to connect to another address if an error occurs when connecting to one address.
- Default value: `http://127.0.0.1:2379`

13.11.5.2.9 initial-commit-ts

- Specifies from which commit timestamp of the transaction the replication process starts. This configuration is applicable only to the Drainer node that is in the replication process for the first time. If a checkpoint already exists in the downstream, the replication will be performed according to the time recorded in the checkpoint.
- commit ts (commit timestamp) is a specific point in time for transaction commits in TiDB. It is a globally unique and increasing timestamp from PD as the unique ID of the current transaction. You can get the `initial-commit-ts` configuration in the following typical ways:
    - If BR is used, you can get `initial-commit-ts` from the backup TS recorded in the metadata backed up by BR (backupmeta).
    - If Dumpling is used, you can get `initial-commit-ts` from the Pos recorded in the metadata backed up by Dumpling (metadata),
    - If PD Control is used, `initial-commit-ts` is in the output of the `tso` command.
- Default value: `-1`. Drainer will get a new timestamp from PD as the starting time, which means that the replication process starts from the current time.

13.11.5.2.10 synced-check-time

- You can access the `/status` path through the HTTP API to query the status of Drainer replication. `synced-check-time` specifies how many minutes from the last successful replication is considered as `synced`, that is, the replication is complete.
- Default value: `5`

13.11.5.2.11 compressor

- Specifies the compression algorithm used for data transfer between Pump and Drainer. Currently only the `gzip` algorithm is supported.
- Default value: "", which means no compression.

13.11.5.2.12 security

This section introduces configuration items related to security.

ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, `/path/to/ca.pem`.
- Default value: “”

ssl-cert

- Specifies the path of the X509 certificate file encoded in the PEM format. For example, `/path/to/drainer.pem`.
- Default value: “”

ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, `/path/to/pump-key.pem`.
- Default value: “”

13.11.5.2.13 syncer

The `syncer` section includes configuration items related to the downstream.

db-type

Currently, the following downstream types are supported:

- `mysql`
- `tidb`
- `kafka`
- `file`

Default value: `mysql`

sql-mode

- Specifies the SQL mode when the downstream is the `mysql` or `tidb` type. If there is more than one mode, use commas to separate them.
- Default value: “”

ignore-txn-commit-ts

- Specifies the commit timestamp at which the binlog is ignored, such as `[416815754209656834, 421349811963822081]`.
- Default value: `[]`

ignore-schemas

- Specifies the database to be ignored during replication. If there is more than one database to be ignored, use commas to separate them. If all changes in a binlog file are filtered, the whole binlog file is ignored.
- Default value: `INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql`

ignore-table

Ignores the specified table changes during replication. You can specify multiple tables to be ignored in the `toml` file. For example:

```
[[syncer.ignore-table]]
db-name = "test"
tbl-name = "log"

[[syncer.ignore-table]]
db-name = "test"
tbl-name = "audit"
```

If all changes in a binlog file are filtered, the whole binlog file is ignored.

Default value: `[]`

replicate-do-db

- Specifies the database to be replicated. For example, `[db1, db2]`.
- Default value: `[]`

replicate-do-table

Specifies the table to be replicated. For example:

```
[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "log"

[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "~^a.*"
```

Default value: `[]`

txn-batch

- When the downstream is the `mysql` or `tidb` type, DML operations are executed in different batches. This parameter specifies how many DML operations can be included in each transaction.
- Default value: `20`

worker-count

- When the downstream is the `mysql` or `tidb` type, DML operations are executed concurrently. This parameter specifies the concurrency numbers of DML operations.
- Default value: `16`

disable-dispatch

- Disables the concurrency and forcibly set `worker-count` to `1`.
- Default value: `false`

safe-mode

If the safe mode is enabled, Drainer modifies the replication updates in the following way:

- `Insert` is modified to `Replace Into`
- `Update` is modified to `Delete` plus `Replace Into`

Default value: `false`

13.11.5.2.14 syncer.to

The `syncer.to` section introduces different types of downstream configuration items according to configuration types.

mysql/tidb

The following configuration items are related to connection to downstream databases:

- `host`: If this item is not set, TiDB Binlog tries to check the `MYSQL_HOST` environment variable which is `localhost` by default.
- `port`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PORT` environment variable which is `3306` by default.
- `user`: If this item is not set, TiDB Binlog tries to check the `MYSQL_USER` environment variable which is `root` by default.
- `password`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PSWD` environment variable which is `""` by default.
- `read-timeout`: Specifies the I/O read timeout of the downstream database connection. The default value is `1m`. If Drainer keeps failing on some DDLs that take a long time, you can set this configuration to a larger value.

file

- `dir`: Specifies the directory where binlog files are stored. If this item is not set, `data-dir` is used.

kafka

When the downstream is Kafka, the valid configuration items are as follows:

- `zookeeper-addrs`
- `kafka-addrs`
- `kafka-version`
- `kafka-max-messages`
- `kafka-max-message-size`
- `topic-name`

13.11.5.2.15 syncer.to.checkpoint

- `type`: Specifies in what way the replication progress is saved. Currently, the available options are `mysql`, `tidb`, and `file`.

  This configuration item is the same as the downstream type by default. For example, when the downstream is `file`, the checkpoint progress is saved in the local file `<data-dir>/savepoint`; when the downstream is `mysql`, the progress is saved in the downstream database. If you need to explicitly specify using `mysql` or `tidb` to store the progress, make the following configuration:

- `schema`: `"tidb_binlog"` by default.

> **Note:**
> When deploying multiple Drainer nodes in the same TiDB cluster, you need to specify a different checkpoint schema for each node. Otherwise, the replication progress of two instances will overwrite each other.

- `host`
- `user`
- `password`
- `port`

13.11.5.3 TiDB Binlog Configuration File

This document introduces the configuration items of TiDB Binlog.

13.11.5.3.1 Pump

This section introduces the configuration items of Pump. For the example of a complete Pump configuration file, see Pump Configuration.

addr

- Specifies the listening address of HTTP API in the format of `host:port`.
- Default value: `127.0.0.1:8250`

advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of `host:port`.
- Default value: `127.0.0.1:8250`

socket

- The Unix socket address that HTTP API listens to.
- Default value: “”

pd-urls

- Specifies the comma-separated list of PD URLs. If multiple addresses are specified, when the PD client fails to connect to one address, it automatically tries to connect to another address.
- Default value: `http://127.0.0.1:2379`

data-dir

- Specifies the directory where binlogs and their indexes are stored locally.
- Default value: `data.pump`

heartbeat-interval

- Specifies the heartbeat interval (in seconds) at which the latest status is reported to PD.
- Default value: `2`

gen-binlog-interval

- Specifies the interval (in seconds) at which data is written into fake binlog.
- Default value: `3`

gc

- Specifies the number of days (integer) that binlogs can be stored locally. Binlogs stored longer than the specified number of days are automatically deleted.
- Default value: `7`

log-file

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: “”

log-level

- Specifies the log level.
- Default value: `info`

node-id

- Specifies the Pump node ID. With this ID, this Pump process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8250`.

security

This section introduces configuration items related to security.

ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, `/path/to/ca.pem`.
- Default value: “”

ssl-cert

- Specifies the path of the X509 certificate file encoded in the Privacy Enhanced Mail (PEM) format. For example, `/path/to/pump.pem`.
- Default value: “”

ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, `/path/to/pump-key.pem`.

- Default value: “”

storage

This section introduces configuration items related to storage.

sync-log

- Specifies whether to use `fsync` after each **batch** write to binlog to ensure data safety.
- Default value: `true`

kv_chan_cap

- Specifies the number of write requests that the buffer can store before Pump receives these requests.
- Default value: `1048576` (that is, 2 to the power of 20)

slow_write_threshold

- The threshold (in seconds). If it takes longer to write a single binlog file than this specified threshold, the write is considered slow write and `"take a long time to write binlog"` is output in the log.
- Default value: `1`

stop-write-at-available-space

- Binlog write requests is no longer accepted when the available storage space is below this specified value. You can use the format such as `900 MB`, `5 GB`, and `12 GiB` to specify the storage space. If there is more than one Pump node in the cluster, when a Pump node refuses a write request because of the insufficient space, TiDB will automatically write binlogs to other Pump nodes.
- Default value: `10 GiB`

kv

Currently the storage of Pump is implemented based on GoLevelDB. Under `storage` there is also a `kv` subgroup that is used to adjust the GoLevel configuration. The supported configuration items are shown as below:

- block-cache-capacity
- block-restart-interval
- block-size
- compaction-L0-trigger
- compaction-table-size

- compaction-total-size
- compaction-total-size-multiplier
- write-buffer
- write-L0-pause-trigger
- write-L0-slowdown-trigger

For the detailed description of the above items, see GoLevelDB Document.

13.11.5.3.2 Drainer

This section introduces the configuration items of Drainer. For the example of a complete Drainer configuration file, see Drainer Configuration

addr

- Specifies the listening address of HTTP API in the format of `host:port`.
- Default value: `127.0.0.1:8249`

advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of `host:port`.
- Default value: `127.0.0.1:8249`

log-file

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: “”

log-level

- Specifies the log level.
- Default value: `info`

node-id

- Specifies the Drainer node ID. With this ID, this Drainer process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8249`.

data-dir

- Specifies the directory used to store files that need to be saved during Drainer operation.
- Default value: `data.drainer`

detect-interval

- Specifies the interval (in seconds) at which PD updates the Pump information.
- Default value: `5`

pd-urls

- The comma-separated list of PD URLs. If multiple addresses are specified, the PD client will automatically attempt to connect to another address if an error occurs when connecting to one address.
- Default value: `http://127.0.0.1:2379`

initial-commit-ts

- Specifies from which commit timestamp of the transaction the replication process starts. This configuration is applicable only to the Drainer node that is in the replication process for the first time. If a checkpoint already exists in the downstream, the replication will be performed according to the time recorded in the checkpoint.
- commit ts (commit timestamp) is a specific point in time for transaction commits in TiDB. It is a globally unique and increasing timestamp from PD as the unique ID of the current transaction. You can get the `initial-commit-ts` configuration in the following typical ways:
    - If BR is used, you can get `initial-commit-ts` from the backup TS recorded in the metadata backed up by BR (backupmeta).
    - If Dumpling is used, you can get `initial-commit-ts` from the Pos recorded in the metadata backed up by Dumpling (metadata),
    - If PD Control is used, `initial-commit-ts` is in the output of the `tso` command.
- Default value: `-1`. Drainer will get a new timestamp from PD as the starting time, which means that the replication process starts from the current time.

synced-check-time

- You can access the `/status` path through the HTTP API to query the status of Drainer replication. `synced-check-time` specifies how many minutes from the last successful replication is considered as `synced`, that is, the replication is complete.
- Default value: `5`

compressor

- Specifies the compression algorithm used for data transfer between Pump and Drainer. Currently only the `gzip` algorithm is supported.
- Default value: "", which means no compression.

security

This section introduces configuration items related to security.

ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, `/path/to/ca.pem`.
- Default value: ""

ssl-cert

- Specifies the path of the X509 certificate file encoded in the PEM format. For example, `/path/to/drainer.pem`.
- Default value: ""

ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, `/path/to/pump-key.pem`.
- Default value: ""

syncer

The `syncer` section includes configuration items related to the downstream.

db-type

Currently, the following downstream types are supported:

- `mysql`
- `tidb`
- `kafka`
- `file`

Default value: `mysql`

sql-mode

- Specifies the SQL mode when the downstream is the `mysql` or `tidb` type. If there is more than one mode, use commas to separate them.
- Default value: ""

ignore-txn-commit-ts

- Specifies the commit timestamp at which the binlog is ignored, such as `[416815754209656834, 421349811963822081]`.
- Default value: `[]`

ignore-schemas

- Specifies the database to be ignored during replication. If there is more than one database to be ignored, use commas to separate them. If all changes in a binlog file are filtered, the whole binlog file is ignored.
- Default value: `INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql`

ignore-table

Ignores the specified table changes during replication. You can specify multiple tables to be ignored in the `toml` file. For example:

```
[[syncer.ignore-table]]
db-name = "test"
tbl-name = "log"

[[syncer.ignore-table]]
db-name = "test"
tbl-name = "audit"
```

If all changes in a binlog file are filtered, the whole binlog file is ignored.

Default value: `[]`

replicate-do-db

- Specifies the database to be replicated. For example, `[db1, db2]`.
- Default value: `[]`

replicate-do-table

Specifies the table to be replicated. For example:

```
[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "log"

[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "~^a.*"
```

Default value: `[]`

txn-batch

- When the downstream is the `mysql` or `tidb` type, DML operations are executed in different batches. This parameter specifies how many DML operations can be included in each transaction.
- Default value: `20`

worker-count

- When the downstream is the `mysql` or `tidb` type, DML operations are executed concurrently. This parameter specifies the concurrency numbers of DML operations.
- Default value: `16`

disable-dispatch

- Disables the concurrency and forcibly set `worker-count` to `1`.
- Default value: `false`

safe-mode

If the safe mode is enabled, Drainer modifies the replication updates in the following way:

- `Insert` is modified to `Replace Into`
- `Update` is modified to `Delete` plus `Replace Into`

Default value: `false`

syncer.to

The `syncer.to` section introduces different types of downstream configuration items according to configuration types.

mysql/tidb

The following configuration items are related to connection to downstream databases:

- `host`: If this item is not set, TiDB Binlog tries to check the `MYSQL_HOST` environment variable which is `localhost` by default.
- `port`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PORT` environment variable which is `3306` by default.
- `user`: If this item is not set, TiDB Binlog tries to check the `MYSQL_USER` environment variable which is `root` by default.

- `password`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PSWD` environment variable which is `""` by default.
- `read-timeout`: Specifies the I/O read timeout of the downstream database connection. The default value is `1m`. If Drainer keeps failing on some DDLs that take a long time, you can set this configuration to a larger value.

file

- `dir`: Specifies the directory where binlog files are stored. If this item is not set, `data-dir` is used.

kafka

When the downstream is Kafka, the valid configuration items are as follows:

- `zookeeper-addrs`
- `kafka-addrs`
- `kafka-version`
- `kafka-max-messages`
- `kafka-max-message-size`
- `topic-name`

syncer.to.checkpoint

- `type`: Specifies in what way the replication progress is saved. Currently, the available options are `mysql`, `tidb`, and `file`.

  This configuration item is the same as the downstream type by default. For example, when the downstream is `file`, the checkpoint progress is saved in the local file `<data-dir>/savepoint`; when the downstream is `mysql`, the progress is saved in the downstream database. If you need to explicitly specify using `mysql` or `tidb` to store the progress, make the following configuration:

- `schema`: `"tidb_binlog"` by default.

> **Note:**
> When deploying multiple Drainer nodes in the same TiDB cluster, you need to specify a different checkpoint schema for each node. Otherwise, the replication progress of two instances will overwrite each other.

- `host`
- `user`
- `password`
- `port`

13.11.6 Upgrade TiDB Binlog

This document introduces how to upgrade TiDB Binlog that is deployed manually to the latest cluster version. There is also a section on how to upgrade TiDB Binlog from an earlier incompatible version (Kafka/Local version) to the latest version.

13.11.6.1 Upgrade TiDB Binlog deployed manually

Follow the steps in this section if you deploy TiDB Binlog manually.

13.11.6.1.1 Upgrade Pump

First, upgrade each Pump instance in the cluster one by one. This ensures that there are always Pump instances in the cluster that can receive binlogs from TiDB. The steps are as below:

1. Replace the original file with the new version of `pump`.
2. Restart the Pump process.

13.11.6.1.2 Upgrade Drainer

Second, upgrade the Drainer component:

1. Replace the original file with the new version of `drainer`.
2. Restart the Drainer process.

13.11.6.2 Upgrade TiDB Binlog from Kafka/Local version to the cluster version

The new TiDB versions (v2.0.8-binlog, v2.1.0-rc.5 or later) are not compatible with the Kafka version or Local version of TiDB Binlog. If TiDB is upgraded to one of the new versions, it is required to use the cluster version of TiDB Binlog. If the Kafka or local version of TiDB Binlog is used before upgrading, you need to upgrade your TiDB Binlog to the cluster version.

The corresponding relationship between TiDB Binlog versions and TiDB versions is shown in the following table:

TiDB Binlog version	TiDB version	Note
Local	TiDB 1.0 or earlier	
Kafka	TiDB 1.0 ~ TiDB 2.1 RC5	TiDB 1.0 supports both the local and Kafka versions of TiDB Binlog.
Cluster	TiDB v2.0.8-binlog, TiDB 2.1 RC5 or later	TiDB v2.0.8-binlog is a special 2.0 version supporting the cluster version of TiDB Binlog.

13.11.6.2.1 Upgrade process

> **Note:**
>
> If importing the full data is acceptable, you can abandon the old version and deploy TiDB Binlog following TiDB Binlog Cluster Deployment.

If you want to resume replication from the original checkpoint, perform the following steps to upgrade TiDB Binlog:

1. Deploy the new version of Pump.
2. Stop the TiDB cluster service.
3. Upgrade TiDB and the configuration, and write the binlog data to the new Pump cluster.
4. Reconnect the TiDB cluster to the service.
5. Make sure that the old version of Drainer has replicated the data in the old version of Pump to the downstream completely;

   Query the `status` interface of Drainer, command as below:

   ```
   curl 'http://172.16.10.49:8249/status'
   ```

   ```
   {"PumpPos":{"172.16.10.49:8250":{"offset":32686}},"Synced": true ,"
       ↪ DepositWindow":{"Upper":398907800202772481,"Lower
       ↪ ":398907799455662081}}
   ```

   If the return value of `Synced` is True, it means Drainer has replicated the data in the old version of Pump to the downstream completely.
6. Start the new version of Drainer.
7. Close the Pump and Drainer of the old versions and the dependent Kafka and ZooKeeper.

13.11.7 TiDB Binlog Monitoring

After you have deployed TiDB Binlog successfully, you can go to the Grafana Web (default address: http://grafana_ip:3000, default account: admin, password: admin) to check the state of Pump and Drainer.

13.11.7.1 Monitoring metrics

TiDB Binlog consists of two components: Pump and Drainer. This section shows the monitoring metrics of Pump and Drainer.

13.11.7.1.1 Pump monitoring metrics

To understand the Pump monitoring metrics, check the following table:

Pump monitoring metrics	Description
Storage Size	Records the total disk space (capacity) and the available disk space (available)

Pump monitoring metrics	Description
Metadata	Records the biggest TSO (`gc_tso` ↪) of the binlog that each Pump node can delete, and the biggest commit TSO (`max_commit_tso` ↪) of the saved binlog

Pump monitoring metrics	Description
Write Binlog QPS by Instance	Shows QPS of writing binlog requests received by each Pump node
Write Binlog Latency	Records the latency time of each Pump node writing binlog

Pump monitoring metrics	Description
Storage Write Binlog Size	Shows the size of the binlog data written by Pump
Storage Write Binlog Latency	Records the latency time of the Pump storage module writing binlog

Pump monitoring metrics	Description
Pump Storage Error By Type	Records the number of errors encountered by Pump, counted based on the type of error
Query TiKV	The number of times that Pump queries the transaction status through TiKV

13.11.7.1.2 Drainer monitoring metrics

To understand the Drainer monitoring metrics, check the following table:

Drainer monitoring metrics	Description
Drainer monitoring metrics	Description
Checkpoint TSO	Shows the biggest TSO time of the binlog that Drainer has already replicated into the downstream. You can get the lag by using the current time to subtract the bin-

Drainer monitoring metrics	Description
Pump Handle TSO	Records the biggest TSO time among the binlog files that Drainer obtains from each Pump node
Pull Binlog QPS by Pump NodeID	Shows the QPS when Drainer obtains binlog from each Pump node

Drainer monitoring metrics	Description
95% Binlog Reach Duration By Pump	Records the delay from the time when binlog is written into Pump to the time when the binlog is obtained by Drainer

Drainer monitoring metrics	Description
Error By Type	Shows the number of errors encountered by Drainer, counted based on the type of error
SQL Query Time	Records the time it takes Drainer to execute the SQL statement in the downstream

Drainer monitoring metrics	Description
Drainer Event	Shows the number of various types of events, including “ddl”, “insert”, “delete”, “update”, “flush”, and “savepoint”

Drainer monitoring metrics	Description
Execute Time	Records the time it takes to write binlog into the downstream syncing module
95% Binlog Size	Shows the size of the binlog data that Drainer obtains from each Pump node

Drainer monitoring metrics	Description
DDL Job Count	Records the number of DDL statements handled by Drainer
Queue Size	Records the work queue size in Drainer

13.11.7.2 Alert rules

This section gives the alert rules for TiDB Binlog. According to the severity level, TiDB Binlog alert rules are divided into three categories (from high to low): emergency-level, critical-level and warning-level.

13.11.7.2.1 Emergency-level alerts

Emergency-level alerts are often caused by a service or node failure. Manual intervention is required immediately.

`binlog_pump_storage_error_count`

- Alert rule:

  `changes(binlog_pump_storage_error_count[1m])> 0`

- Description:

  Pump fails to write the binlog data to the local storage.

- Solution:

  Check whether an error exists in the `pump_storage_error` monitoring and check the Pump log to find the causes.

13.11.7.2.2 Critical-level alerts

For the critical-level alerts, a close watch on the abnormal metrics is required.

`binlog_drainer_checkpoint_high_delay`

- Alert rule:

  `(time()- binlog_drainer_checkpoint_tso / 1000)> 3600`

- Description:

  The delay of Drainer replication exceeds one hour.

- Solution:

  - Check whether it is too slow to obtain the data from Pump:

    You can check `handle tso` of Pump to get the time for the latest message of each Pump. Check whether a high latency exists for Pump and make sure the corresponding Pump is running normally.
  - Check whether it is too slow to replicate data in the downstream based on Drainer `event` and Drainer `execute latency`:
    - If Drainer `execute time` is too large, check the network bandwidth and latency between the machine with Drainer deployed and the machine with the target database deployed, and the state of the target database.
    - If Drainer `execute time` is not too large and Drainer `event` is too small, add `work count` and `batch` and retry.
  - If the two solutions above cannot work, get support from PingCAP or the community.

13.11.7.2.3 Warning-level alerts

Warning-level alerts are a reminder for an issue or error.

`binlog_pump_write_binlog_rpc_duration_seconds_bucket`

- Alert rule:

  ```
  histogram_quantile(0.9, rate(binlog_pump_rpc_duration_seconds_bucket{
  ↪ method="WriteBinlog"}[5m]))> 1
  ```

- Description:

  It takes too much time for Pump to handle the TiDB request of writing binlog.

- Solution:
    - Verify the disk performance pressure and check the disk performance monitoring via `node exported`.
    - If both `disk latency` and `util` are low, get support from PingCAP or the community.

`binlog_pump_storage_write_binlog_duration_time_bucket`

- Alert rule:

```
histogram_quantile(0.9, rate(binlog_pump_storage_write_binlog_duration_time_bucket
↪ {type="batch"}[5m]))> 1
```

- Description:

  The time it takes for Pump to write the local binlog to the local disk.

- Solution:

  Check the state of the local disk of Pump and fix the problem.

`binlog_pump_storage_available_size_less_than_20G`

- Alert rule:

```
binlog_pump_storage_storage_size_bytes{type="available"} < 20 * 1024 *
↪ 1024 * 1024
```

- Description:

  The available disk space of Pump is less than 20 GB.

- Solution:

  Check whether Pump `gc_tso` is normal. If not, adjust the GC time configuration of Pump or get the corresponding Pump offline.

`binlog_drainer_checkpoint_tso_no_change_for_1m`

- Alert rule:

```
changes(binlog_drainer_checkpoint_tso[1m])< 1
```

- Description:

  Drainer `checkpoint` has not been updated for one minute.

- Solution:

  Check whether all the Pumps that are not offline are running normally.

`binlog_drainer_execute_duration_time_more_than_10s`

- Alert rule:

  ```
  histogram_quantile(0.9, rate(binlog_drainer_execute_duration_time_bucket
  ↪ [1m]))> 10
  ```

- Description:

  The transaction time it takes Drainer to replicate data to TiDB. If it is too large, the Drainer replication of data is affected.

- Solution:

  - Check the TiDB cluster state.
  - Check the Drainer log or monitor. If a DDL operation causes this problem, you can ignore it.

13.11.8 Reparo User Guide

Reparo is a TiDB Binlog tool, used to recover the incremental data. To back up the incremental data, you can use Drainer of TiDB Binlog to output the binlog data in the protobuf format to files. To restore the incremental data, you can use Reparo to parse the binlog data in the files and apply the binlog in TiDB/MySQL.

The Reparo installation package (`reparo`) is included in the TiDB Toolkit. To download the TiDB Toolkit, see Download TiDB Tools.

13.11.8.1 Reparo usage

13.11.8.1.1 Description of command line parameters

```
Usage of Reparo:
-L string
    The level of the output information of logs
    Value: "debug"/"info"/"warn"/"error"/"fatal" ("info" by default)
-V Prints the version.
-c int
    The number of concurrencies in the downstream for the replication
       ↪ process (`16` by default). A higher value indicates a better
       ↪ throughput for the replication.
-config string
    The path of the configuration file
    If the configuration file is specified, Reparo reads the configuration
       ↪ data in this file.
```

```
   If the configuration data also exists in the command line parameters,
      ↪ Reparo uses the configuration data in the command line parameters
      ↪ to cover that in the configuration file.
-data-dir string
   The storage directory for the binlog file in the protobuf format that
      ↪ Drainer outputs ("data.drainer" by default)
-dest-type string
   The downstream service type
   Value: "print"/"mysql" ("print" by default)
   If it is set to "print", the data is parsed and printed to standard
      ↪ output while the SQL statement is not executed.
   If it is set to "mysql", you need to configure the "host", "port", "user
      ↪ " and "password" information in the configuration file.
-log-file string
   The path of the log file
-log-rotate string
   The switch frequency of log files
   Value: "hour"/"day"
-start-datetime string
   Specifies the time point for starting recovery.
   Format: "2006-01-02 15:04:05"
   If it is not set, the recovery process starts from the earliest binlog
      ↪ file.
-stop-datetime string
   Specifies the time point of finishing the recovery process.
   Format: "2006-01-02 15:04:05"
   If it is not set, the recovery process ends up with the last binlog file
      ↪ .
-safe-mode bool
   Specifies whether to enable safe mode. When enabled, it supports
      ↪ repeated replication.
-txn-batch int
   The number of SQL statements in a transaction that is output to the
      ↪ downstream database (`20` by default).
```

13.11.8.1.2 Description of the configuration file

```
### The storage directory for the binlog file in the protobuf format that
   ↪ Drainer outputs
data-dir = "./data.drainer"

### The level of the output information of logs
### Value: "debug"/"info"/"warn"/"error"/"fatal" ("info" by default)
log-level = "info"
```

```
### Uses `start-datetime` and `stop-datetime` to specify the time range in
    ↪ which
### the binlog files are to be recovered.
### Format: "2006-01-02 15:04:05"
### start-datetime = ""
### stop-datetime = ""

### Correspond to `start-datetime` and `stop-datetime` respectively.
### They are used to specify the time range in which the binlog files are to
    ↪  be recovered.
### If `start-datetime` and `stop-datetime` are set, there is no need to set
    ↪  `start-tso` and `stop-tso`.
### When you perform a full recovery or resume an incremental recovery, set
    ↪ start-tso to tso + 1 or stop-tso + 1, respectively.
### start-tso = 0
### stop-tso = 0

### The downstream service type
### Value: "print"/"mysql" ("print" by default)
### If it is set to "print", the data is parsed and printed to standard
    ↪ output
### while the SQL statement is not executed.
### If it is set to "mysql", you need to configure `host`, `port`, `user`
    ↪ and `password` in [dest-db].
dest-type = "mysql"

### The number of SQL statements in a transaction that is output to the
    ↪ downstream database (`20` by default).
txn-batch = 20

### The number of concurrencies in the downstream for the replication
    ↪ process (`16` by default). A higher value indicates a better
    ↪ throughput for the replication.
worker-count = 16

### Safe-mode configuration
### Value: "true"/"false" ("false" by default)
### If it is set to "true", Reparo splits the `UPDATE` statement into a `
    ↪ DELETE` statement plus a `REPLACE` statement.
safe-mode = false

### `replicate-do-db` and `replicate-do-table` specify the database and
    ↪ table to be recovered.
### `replicate-do-db` has priority over `replicate-do-table`.
```

```
### You can use a regular expression for configuration. The regular
    ↪ expression should start with "~".
### The configuration method for `replicate-do-db` and `replicate-do-table`
    ↪ is
### the same with that for `replicate-do-db` and `replicate-do-table` of
    ↪ Drainer.
### replicate-do-db = ["~^b.*","s1"]
### [[replicate-do-table]]
### db-name ="test"
### tbl-name = "log"
### [[replicate-do-table]]
### db-name ="test"
### tbl-name = "~^a.*"

### If `dest-type` is set to `mysql`, `dest-db` needs to be configured.
[dest-db]
host = "127.0.0.1"
port = 3309
user = "root"
password = ""
```

13.11.8.1.3 Start example

```
./reparo -config reparo.toml
```

Note:

- `data-dir` specifies the directory for the binlog file that Drainer outputs.
- Both `start-datatime` and `start-tso` are used to specify the time point for starting recovery, but they are different in the time format. If they are not set, the recovery process starts from the earliest binlog file by default.
- Both `stop-datetime` and `stop-tso` are used to specify the time point for finishing recovery, but they are different in the time format. If they are not set, the recovery process ends up with the last binlog file by default.
- `dest-type` specifies the destination type. Its value can be "mysql" and "print."
    - When it is set to `mysql`, the data can be recovered to MySQL or TiDB that uses or is compatible with the MySQL protocol. In this

case, you need to specify the database information in `[dest-db]` of the configuration information.
    - When it is set to `print`, only the binlog information is printed. It is generally used for debugging and checking the binlog information. In this case, there is no need to specify `[dest-db]`.
- `replicate-do-db` specifies the database for recovery. If it is not set, all the databases are to be recovered.
- `replicate-do-table` specifies the table for recovery. If it is not set, all the tables are to be recovered.

13.11.9 binlogctl

Binlog Control (`binlogctl` for short) is a command line tool for TiDB Binlog. You can use `binlogctl` to manage TiDB Binlog clusters.

You can use `binlogctl` to:

- Check the state of Pump or Drainer
- Pause or close Pump or Drainer
- Handle the abnormal state of Pump or Drainer

The following are its usage scenarios:

- An error occurs during data replication or you need to check the running state of Pump or Drainer.
- You need to pause or close Pump or Drainer when maintaining the cluster.
- A Pump or Drainer process exits abnormally, while the node state is not updated or is unexpected. This affects the data replication task.

13.11.9.1 Download `binlogctl`

`binlogctl` is included in the TiDB Toolkit. To download the TiDB Toolkit, see Download TiDB Tools.

13.11.9.2 Descriptions

Command line parameters:

```
Usage of binlogctl:
  -V   prints version and exit
  -cmd string
```

```
      operator: "generate_meta", "pumps", "drainers", "update-pump", "
         ↪ update-drainer", "pause-pump", "pause-drainer", "offline-pump
         ↪ ", "offline-drainer", "encrypt" (default "pumps")
-data-dir string
      meta directory path (default "binlog_position")
-node-id string
      id of node, used to update some nodes with operations update-pump,
         ↪ update-drainer, pause-pump, pause-drainer, offline-pump and
         ↪ offline-drainer
-pd-urls string
      a comma separated list of PD endpoints (default "http
         ↪ ://127.0.0.1:2379")
-show-offline-nodes
      include offline nodes when querying pumps/drainers
-ssl-ca string
      Path of file that contains list of trusted SSL CAs for connection
         ↪ with cluster components.
-ssl-cert string
      Path of file that contains X509 certificate in PEM format for
         ↪ connection with cluster components.
-ssl-key string
      Path of file that contains X509 key in PEM format for connection with
         ↪  cluster components.
-state string
      set node's state, can be set to online, pausing, paused, closing or
         ↪ offline.
-text string
      text to be encrypted when using encrypt command
-time-zone Asia/Shanghai
      set time zone if you want to save time info in savepoint file; for
         ↪ example, Asia/Shanghai for CST time, `Local` for local time
```

Command examples:

- Check the state of all the Pump or Drainer nodes.

  Set `cmd` to `pumps` or `drainers`. For example:

```
bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd pumps
```

```
[2019/04/28 09:29:59.016 +00:00] [INFO] [nodes.go:48] ["query node"] [
   ↪ type=pump] [node="{NodeID: 1.1.1.1:8250, Addr: pump:8250, State:
   ↪ online, MaxCommitTS: 408012403141509121, UpdateTime: 2019-04-28
   ↪ 09:29:57 +0000 UTC}"]
```

```
bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd drainers
```

```
[2019/04/28 09:29:59.016 +00:00] [INFO] [nodes.go:48] ["query node"] [
    ↪ type=drainer] [node="{NodeID: 1.1.1.1:8249, Addr: 1.1.1.1:8249,
    ↪ State: online, MaxCommitTS: 408012403141509121, UpdateTime:
    ↪ 2019-04-28 09:29:57 +0000 UTC}"]
```

- Pause or close Pump or Drainer.

  You can use the following commands to pause or close services:

Command	Description	Example
pause-pump	Pause Pump	`bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd pause-pump -node-id ip-127-0-0-1:8250`
pause-drainer	Pause Drainer	`bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd pause-drainer -node-id ip-127-0-0-1:8249`
offline-pump	Close Pump	`bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd offline-pump -node-id ip-127-0-0-1:8250`
offline-drainer	Close Drainer	`bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd offline-drainer -node-id ip-127-0-0-1:8249`

  `binlogctl` sends the HTTP request to the Pump or Drainer node. After receiving the request, the node executes the exiting procedures accordingly.

- Modify the state of a Pump or Drainer node in abnormal states.

  When a Pump or Drainer node runs normally or when it is paused or closed in the normal process, it is in the normal state. In abnormal states, the Pump or Drainer node cannot correctly maintain its state. This affects data replication tasks. In this case, use `binlogctl` to repair the state information.

  To update the state of a Pump or Drainer node, set `cmd` to `update-pump` or `update-drainer`. The state can be `paused` or `offline`. For example:

```
bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd update-pump -node-id
    ↪ ip-127-0-0-1:8250 -state paused
```

> **Note:**
> When a Pump or Drainer node runs normally, it regularly updates its state to PD. The above command directly modifies the Pump or Drainer state saved in PD; therefore, do not use the command when the Pump or Drainer node runs normally. For more information, refer to TiDB Binlog FAQ.

13.11.10 Binlog Consumer Client User Guide

Binlog Consumer Client is used to consume TiDB secondary binlog data from Kafka and output the data in a specific format. Currently, Drainer supports multiple kinds of down streaming, including MySQL, TiDB, file and Kafka. But sometimes users have customized requirements for outputting data to other formats, for example, Elasticsearch and Hive, so this feature is introduced.

13.11.10.1 Configure Drainer

Modify the configuration file of Drainer and set it to output the data to Kafka:

```
[syncer]
db-type = "kafka"

[syncer.to]
### the Kafka address
kafka-addrs = "127.0.0.1:9092"
### the Kafka version
kafka-version = "2.4.0"
```

13.11.10.2 Customized development

13.11.10.2.1 Data format

Firstly, you need to obtain the format information of the data which is output to Kafka by Drainer:

```
// `Column` stores the column data in the corresponding variable based on
   ↪ the data type.
message Column {
  // Indicates whether the data is null
  optional bool is_null = 1 [ default = false ];
  // Stores `int` data
  optional int64 int64_value = 2;
  // Stores `uint`, `enum`, and `set` data
  optional uint64 uint64_value = 3;
  // Stores `float` and `double` data
  optional double double_value = 4;
  // Stores `bit`, `blob`, `binary` and `json` data
  optional bytes bytes_value = 5;
  // Stores `date`, `time`, `decimal`, `text`, `char` data
  optional string string_value = 6;
}
```

```
// `ColumnInfo` stores the column information, including the column name,
    ↪ type, and whether it is the primary key.
message ColumnInfo {
  optional string name = 1 [ (gogoproto.nullable) = false ];
  // the lower case column field type in MySQL
  // https://dev.mysql.com/doc/refman/8.0/en/data-types.html
  // for the `numeric` type: int bigint smallint tinyint float double
      ↪ decimal bit
  // for the `string` type: text longtext mediumtext char tinytext varchar
  // blob longblob mediumblob binary tinyblob varbinary
  // enum set
  // for the `json` type: json
  optional string mysql_type = 2 [ (gogoproto.nullable) = false ];
  optional bool is_primary_key = 3 [ (gogoproto.nullable) = false ];
}

// `Row` stores the actual data of a row.
message Row { repeated Column columns = 1; }

// `MutationType` indicates the DML type.
enum MutationType {
  Insert = 0;
  Update = 1;
  Delete = 2;
}

// `Table` contains mutations in a table.
message Table {
  optional string schema_name = 1;
  optional string table_name = 2;
  repeated ColumnInfo column_info = 3;
  repeated TableMutation mutations = 4;
}

// `TableMutation` stores mutations of a row.
message TableMutation {
  required MutationType type = 1;
  // data after modification
  required Row row = 2;
  // data before modification. It only takes effect for `Update MutationType
      ↪ `.
  optional Row change_row = 3;
}

// `DMLData` stores all the mutations caused by DML in a transaction.
```

```
message DMLData {
  // `tables` contains all the table changes in the transaction.
  repeated Table tables = 1;
}

// `DDLData` stores the DDL information.
message DDLData {
  // the database used currently
  optional string schema_name = 1;
  // the relates table
  optional string table_name = 2;
  // `ddl_query` is the original DDL statement query.
  optional bytes ddl_query = 3;
}

// `BinlogType` indicates the binlog type, including DML and DDL.
enum BinlogType {
  DML = 0; // Has `dml_data`
  DDL = 1; // Has `ddl_query`
}

// `Binlog` stores all the changes in a transaction. Kafka stores the
   ↪ serialized result of the structure data.
message Binlog {
  optional BinlogType type = 1 [ (gogoproto.nullable) = false ];
  optional int64 commit_ts = 2 [ (gogoproto.nullable) = false ];
  optional DMLData dml_data = 3;
  optional DDLData ddl_data = 4;
}
```

For the definition of the data format, see `secondary_binlog.proto`

13.11.10.2.2 Driver

The TiDB-Tools project provides Driver, which is used to read the binlog data in Kafka. It has the following features:

- Read the Kafka data.
- Locate the binlog stored in Kafka based on `commit ts`.

You need to configure the following information when using Driver:

- `KafkaAddr`: the address of the Kafka cluster
- `CommitTS`: from which `commit ts` to start reading the binlog

- `Offset`: from which Kafka `offset` to start reading data. If `CommitTS` is set, you needn't configure this parameter.
- `ClusterID`: the cluster ID of the TiDB cluster
- `Topic`: the topic name of Kafka. If Topic is empty, use the default name in Drainer `<ClusterID>_obinlog`.

You can use Driver by quoting the Driver code in package and refer to the example code provided by Driver to learn how to use Driver and parse the binlog data.

Currently, two examples are provided:

- Using Driver to replicate data to MySQL. This example shows how to convert a binlog to SQL
- Using Driver to print data

> **Note:**
>
> - The example code only shows how to use Driver. If you want to use Driver in the production environment, you need to optimize the code.
> - Currently, only the Golang version of Driver and example code are available. If you want to use other languages, you need to generate the code file in the corresponding language based on the binlog proto file and develop an application to read the binlog data in Kafka, parse the data, and output the data to the downstream. You are also welcome to optimize the example code and submit the example code of other languages to TiDB-Tools.

13.11.11 TiDB Binlog Relay Log

When replicating binlogs, Drainer splits transactions from the upstream and replicates the split transactions concurrently to the downstream.

In extreme cases where the upstream clusters are not available and Drainer exits abnormally, the downstream clusters (MySQL or TiDB) might be in the intermediate states with inconsistent data. In such cases, Drainer can use the relay log to ensure that the downstream clusters are in a consistent state.

13.11.11.1 Consistent state during Drainer replication

The downstream clusters reaching a consistent state means the data of the downstream clusters are the same as the snapshot of the upstream which sets `tidb_snapshot = ts`.

The checkpoint consistency means Drainer checkpoint saves the consistent state of replication in `consistent`. When Drainer runs, `consistent` is `false`. After Drainer exits normally, `consistent` is set to `true`.

You can query the downstream checkpoint table as follows:

```
select * from tidb_binlog.checkpoint;
```

```
+--------------------+-------------------------------------------------------------------+
   ↪
| clusterID        | checkPoint
   ↪                                                  |
+--------------------+-------------------------------------------------------------------+
   ↪
| 6791641053252586769 | {"consistent":false,"commitTS":414529105591271429,"
   ↪ ts-map":{}} |
+--------------------+-------------------------------------------------------------------+
   ↪
```

13.11.11.2 Implementation principles

After Drainer enables the relay log, it first writes the binlog events to the disks and then replicates the events to the downstream clusters.

If the upstream clusters are not available, Drainer can restore the downstream clusters to a consistent state by reading the relay log.

> **Note:**
>
> If the relay log data is lost at the same time, this method does not work, but its incidence is very low. In addition, you can use the Network File System to ensure data safety of the relay log.

13.11.11.2.1 Trigger scenarios where Drainer consumes binlogs from the relay log

When Drainer is started, if it fails to connect to the Placement Driver (PD) of the upstream clusters, and it detects that `consistent = false` in the checkpoint, Drainer will try to read the relay log, and restore the downstream clusters to a consistent state. After that, the Drainer process sets the checkpoint `consistent` to `true` and then exits.

13.11.11.2.2 GC mechanism of relay log

Before data is replicated to the downstream, Drainer writes data to the relay log file. If the size of a relay log file reaches 10 MB (by default) and the binlog data of the current transaction is completely written, Drainer starts to write data to the next relay log file. After Drainer successfully replicates data to the downstream, it automatically cleans up the relay log files whose data has been replicated. The relay log into which data is currently being written will not be cleaned up.

13.11.11.3 Configuration

To enable the relay log, add the following configuration in Drainer:

```
[syncer.relay]
### It saves the directory of the relay log. The relay log is not enabled if
    ↪ the value is empty.
### The configuration only comes to effect if the downstream is TiDB or
    ↪ MySQL.
log-dir = "/dir/to/save/log"
### The size limit of a single relay log file (unit: byte).
### When the size of a relay log file reaches this limit, data is written to
    ↪ the next relay log file.
max-file-size = 10485760
```

13.11.12 Bidirectional Replication between TiDB Clusters

> **Warning:**
>
> Currently, bidirectional replication is still an experimental feature. It is **NOT** recommended to use it in the production environment.

This document describes the bidirectional replication between two TiDB clusters, how the replication works, how to enable it, and how to replicate DDL operations.

13.11.12.1 User scenario

If you want two TiDB clusters to exchange data changes with each other, TiDB Binlog allows you to do that. For example, you want cluster A and cluster B to replicate data with each other.

Note:

The data written to these two clusters must be conflict-free, that is, in the two clusters, the same primary key or the rows with the unique index of the tables must not be modified.

The user scenario is shown as below:

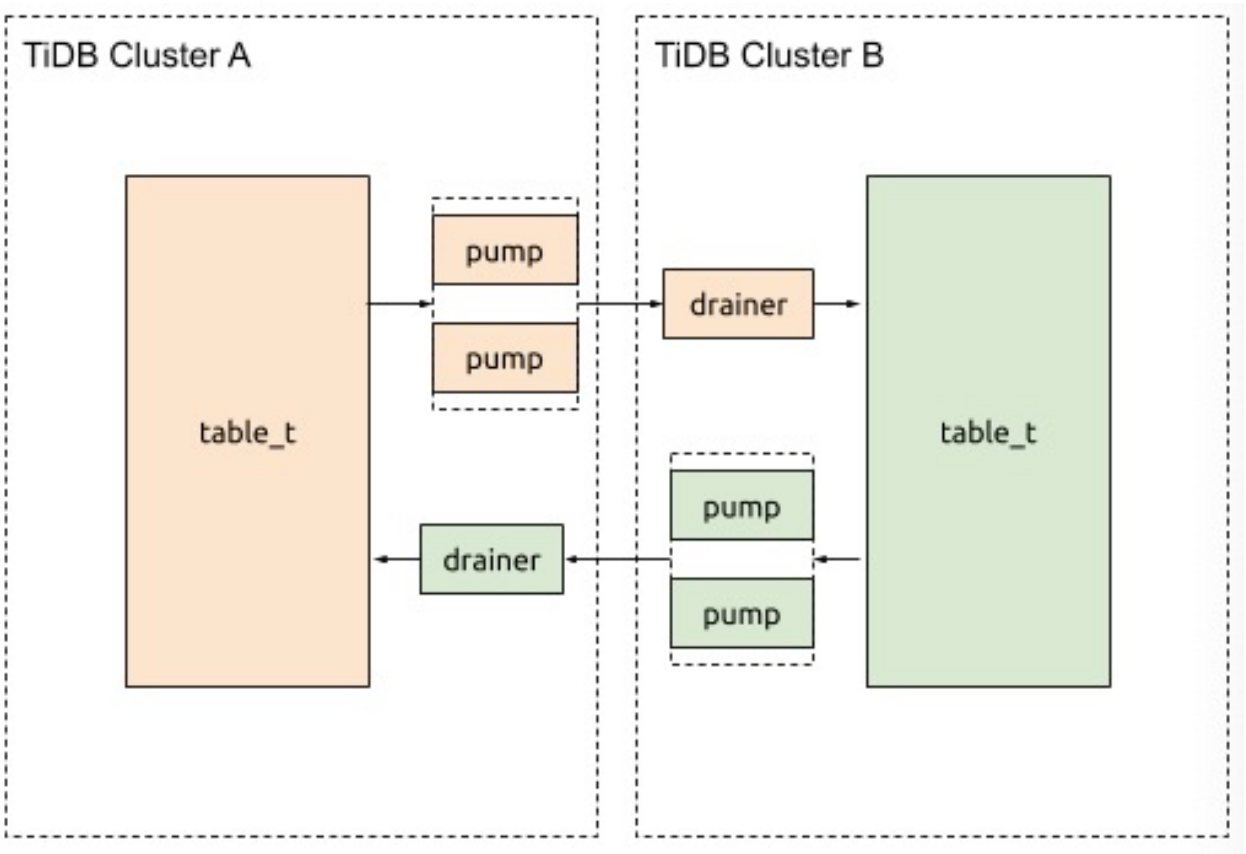


Figure 302: Architect

13.11.12.2 Implementation details

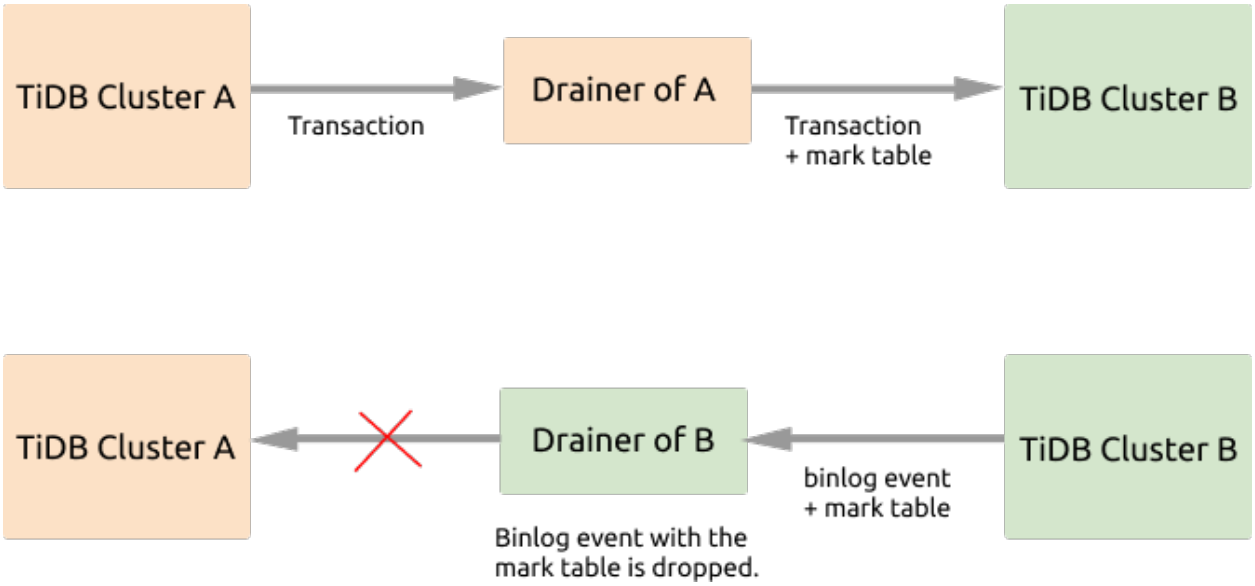


Figure 303: Mark Table

If the bidirectional replication is enabled between cluster A and cluster B, the data written to cluster A will be replicated to cluster B, and then these data changes will be replicated back to cluster A, which causes an infinite loop of replication. From the figure above, you can see that during the data replication, Drainer marks the binlog events, and filters out the marked events to avoid such a replication loop.

The detailed implementation is described as follows:

1. Start the TiDB Binlog replication program for each of the two clusters.
2. When the transaction to be replicated passes through the Drainer of cluster A, this Drainer adds the `_drainer_repl_mark` table to the transaction, writes this DML event update to the mark table, and replicate this transaction to cluster B.
3. Cluster B returns binlog events with the `_drainer_repl_mark` mark table to cluster A. The Drainer of cluster B identifies the mark table with the DML event when parsing the binlog event, and gives up replicating this binlog event to cluster A.

The replication process from cluster B to cluster A is the same as above. The two clusters can be upstream and downstream of each other.

Note:

- When updating the `_drainer_repl_mark` mark table, data changes are required to generate binlogs.

- DDL operations are not transactional, so you need to use the one-way replication method to replicate DDL operations. See Replicate DDL operations for details.

Drainer can use a unique ID for each connection to downstream to avoid conflicts. `channel_id` is used to indicate a channel for bidirectional replication. The two clusters should have the same `channel_id` configuration (with the same value).

If you add or delete columns in the upstream, there might be extra or missing columns of the data to be replicated to the downstream. Drainer allows this situation by ignoring the extra columns or by inserting default values to the missing columns.

13.11.12.3 Mark table

The `_drainer_repl_mark` mark table has the following structure:

```
CREATE TABLE `_drainer_repl_mark` (
  `id` bigint(20) NOT NULL,
  `channel_id` bigint(20) NOT NULL DEFAULT '0',
  `val` bigint(20) DEFAULT '0',
  `channel_info` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`id`,`channel_id`)
);
```

Drainer uses the following SQL statement to update `_drainer_repl_mark`, which ensures data change and the generation of binlog:

```
update drainer_repl_mark set val = val + 1 where id = ? && channel_id = ?;
```

13.11.12.4 Replicate DDL operations

Because Drainer cannot add the mark table to DDL operations, you can only use the one-way replication method to replicate DDL operations.

For example, if DDL replication is enabled from cluster A to cluster B, then the replication is disabled from cluster B to cluster A. This means that all DDL operations are performed on cluster A.

Note:

DDL operations cannot be executed on two clusters at the same time. When a DDL operation is executed, if any DML operation is being executed at the same time or any DML binlog is being replicated, the upstream and downstream table structures of the DML replication might be inconsistent.

13.11.12.5 Configure and enable bidirectional replication

For bidirectional replication between cluster A and cluster B, assume that all DDL operations are executed on cluster A. On the replication path from cluster A to cluster B, add the following configuration to Drainer:

```
[syncer]
loopback-control = true
channel-id = 1 # Configures the same ID for both clusters to be replicated.
sync-ddl = true # Enables it if you need to perform DDL replication.

[syncer.to]
### 1 means SyncFullColumn and 2 means SyncPartialColumn.
### If set to SyncPartialColumn, Drainer allows the downstream table
### structure to have more or fewer columns than the data to be replicated
### And remove the STRICT_TRANS_TABLES of the SQL mode to allow fewer
   ↪ columns, and insert zero values to the downstream.
sync-mode = 2

### Ignores the checkpoint table.
[[syncer.ignore-table]]
db-name = "tidb_binlog"
tbl-name = "checkpoint"
```

On the replication path from cluster B to cluster A, add the following configuration to Drainer:

```
[syncer]
loopback-control = true
channel-id = 1 # Configures the same ID for both clusters to be replicated.
sync-ddl = false # Disables it if you do not need to perform DDL replication
   ↪ .

[syncer.to]
### 1 means SyncFullColumn and 2 means SyncPartialColumn.
### If set to SyncPartialColumn, Drainer allows the downstream table
### structure to have more or fewer columns than the data to be replicated
### And remove the STRICT_TRANS_TABLES of the SQL mode to allow fewer
   ↪ columns, and insert zero values to the downstream.
sync-mode = 2

### Ignores the checkpoint table.
[[syncer.ignore-table]]
db-name = "tidb_binlog"
tbl-name = "checkpoint"
```

13.11.13 TiDB Binlog Glossary

This document lists the terms used in the logs, monitoring, configurations, and documentation of TiDB Binlog.

13.11.13.1 Binlog

In TiDB Binlog, binlogs refer to the binary log data from TiDB. They also refer to the binary log data that Drainer writes to Kafka or files. The former and the latter are in different formats. In addition, binlogs in TiDB and binlogs in MySQL are also in different formats.

13.11.13.2 Binlog event

The DML binlogs from TiDB have three types of event: `INSERT`, `UPDATE`, and `DELETE`. In the monitoring dashboard of Drainer, you can see the number of different events that correspond to the replication data.

13.11.13.3 Checkpoint

A checkpoint indicates the position from which a replication task is paused and resumed, or is stopped and restarted. It records the commit-ts that Drainer replicates to the downstream. When restarted, Drainer reads the checkpoint and starts replicating data from the corresponding commit-ts.

13.11.13.4 Safe mode

Safe mode refers to the mode that supports the idempotent import of DML when a primary key or unique index exists in the table schema in the incremental replication task.

In this mode, the `INSERT` statement is re-written as `REPLACE`, and the `UPDATE` statement is re-written as `DELETE` and `REPLACE`. Then the re-written statement is executed to the downstream. Safe mode is automatically enabled within 5 minutes after Drainer is started. You can manually enable the mode by modifying the `safe-mode` parameter in the configuration file, but this configuration is valid only when the downstream is MySQL or TiDB.

13.11.14 Troubleshoot

13.11.14.1 TiDB Binlog Troubleshooting

This document describes how to troubleshoot TiDB Binlog to find the problem.

If you encounter errors while running TiDB Binlog, take the following steps to troubleshoot:

1. Check whether each monitoring metric is normal or not. Refer to TiDB Binlog Monitoring for details.

2. Use the binlogctl tool to check whether the state of each Pump or Drainer node is normal or not.

3. Check whether `ERROR` or `WARN` exists in the Pump log or Drainer log.

After finding out the problem by the above steps, refer to FAQ and TiDB Binlog Error Handling for the solution. If you fail to find the solution or the solution provided does not help, submit an issue for help.

13.11.14.2 TiDB Binlog Error Handling

This document introduces common errors that you might encounter and solutions to these errors when you use TiDB Binlog.

13.11.14.2.1 `kafka server: Message was too large, server rejected it to avoid allocation error` is returned when Drainer replicates data to Kafka

Cause: Executing a large transaction in TiDB generates binlog data of a large size, which might exceed Kafka's limit on the message size.

Solution: Adjust the configuration parameters of Kafka as shown below:

```
message.max.bytes=1073741824
replica.fetch.max.bytes=1073741824
fetch.message.max.bytes=1073741824
```

13.11.14.2.2 Pump returns `no space left on device` error

Cause: The local disk space is insufficient for Pump to write binlog data normally.

Solution: Clean up the disk space and then restart Pump.

13.11.14.2.3 `fail to notify all living drainer` is returned when Pump is started

Cause: When Pump is started, it notifies all Drainer nodes that are in the `online` state. If it fails to notify Drainer, this error log is printed.

Solution: Use the binlogctl tool to check whether each Drainer node is normal or not. This is to ensure that all Drainer nodes that are in the `online` state are working normally. If the state of a Drainer node is not consistent with its actual working status, use the binlogctl tool to change its state and then restart Pump.

13.11.14.2.4 Data loss occurs during the TiDB Binlog replication

You need to confirm that TiDB Binlog is enabled on all TiDB instances and runs normally. If the cluster version is later than v3.0, use the `curl {TiDB_IP}:{STATUS_PORT}/info/all` command to confirm the TiDB Binlog status on all TiDB instances.

13.11.14.2.5 When the upstream transaction is large, Pump reports an error `rpc error: code = ResourceExhausted desc = trying to send message larger than max (2191430008 vs. 2147483647)`

This error occurs because the gRPC message sent by TiDB to Pump exceeds the size limit. You can adjust the maximum size of a gRPC message that Pump allows by specifying `max-message-size` when starting Pump.

13.11.14.2.6 Is there any cleaning mechanism for the incremental data of the file format output by Drainer? Will the data be deleted?

- In Drainer v3.0.x, there is no cleaning mechanism for incremental data of the file format.
- In the v4.0.x version, there is a time-based data cleaning mechanism. For details, refer to Drainer's `retention-time` configuration item.

13.11.15 TiDB Binlog FAQs

This document collects the frequently asked questions (FAQs) about TiDB Binlog.

13.11.15.1 What is the impact of enabling TiDB Binlog on the performance of TiDB?

- There is no impact on the query.
- There is a slight performance impact on `INSERT`, `DELETE` and `UPDATE` transactions. In latency, a p-binlog is written concurrently in the TiKV prewrite stage before the transactions are committed. Generally, writing binlog is faster than TiKV prewrite, so it does not increase latency. You can check the response time of writing binlog in Pump's monitoring panel.

13.11.15.2 How high is the replication latency of TiDB Binlog?

The latency of TiDB Binlog replication is measured in seconds, which is generally about 3 seconds during off-peak hours.

13.11.15.3 What privileges does Drainer need to replicate data to the downstream MySQL or TiDB cluster?

To replicate data to the downstream MySQL or TiDB cluster, Drainer must have the following privileges:

- Insert
- Update

- Delete
- Create
- Drop
- Alter
- Execute
- Index
- Select
- Create View

13.11.15.4 What can I do if the Pump disk is almost full?

1. Check whether Pump's GC works well:

    - Check whether the **gc_tso** time in Pump's monitoring panel is identical with that of the configuration file.

2. If GC works well, perform the following steps to reduce the amount of space required for a single Pump:

    - Modify the **GC** parameter of Pump to reduce the number of days to retain data.
    - Add pump instances.

13.11.15.5 What can I do if Drainer replication is interrupted?

Execute the following command to check whether the status of Pump is normal and whether all the Pump instances that are not in the `offline` state are running.

```
binlogctl -cmd pumps
```

Then, check whether the Drainer monitor or log outputs corresponding errors. If so, resolve them accordingly.

13.11.15.6 What can I do if Drainer is slow to replicate data to the downstream MySQL or TiDB cluster?

Check the following monitoring items:

- For the **Drainer Event** monitoring metric, check the speed of Drainer replicating `INSERT`, `UPDATE` and `DELETE` transactions to the downstream per second.
- For the **SQL Query Time** monitoring metric, check the time Drainer takes to execute SQL statements in the downstream.

Possible causes and solutions for slow replication:

- If the replicated database contains a table without a primary key or unique index, add a primary key to the table.
- If the latency between Drainer and the downstream is high, increase the value of the `worker-count` parameter of Drainer. For cross-datacenter replication, it is recommended to deploy Drainer in the downstream.
- If the load in the downstream is not high, increase the value of the `worker-count` parameter of Drainer.

13.11.15.7 What can I do if a Pump instance crashes?

If a Pump instance crashes, Drainer cannot replicate data to the downstream because it cannot obtain the data of this instance. If this Pump instance can recover to the normal state, Drainer resumes replication; if not, perform the following steps:

1. Use binlogctl to change the state of this Pump instance to `offline` to discard the data of this Pump instance.
2. Because Drainer cannot obtain the data of this pump instance, the data in the downstream and upstream is inconsistent. In this situation, perform full and incremental backups again. The steps are as follows:
    1. Stop the Drainer.
    2. Perform a full backup in the upstream.
    3. Clear the data in the downstream including the `tidb_binlog.checkpoint` table.
    4. Restore the full backup to the downstream.
    5. Deploy Drainer and use `initialCommitTs` (set `initialCommitTs` as the snapshot timestamp of the full backup) as the start point of initial replication.

13.11.15.8 What is checkpoint?

Checkpoint records the `commit-ts` that Drainer replicates to the downstream. When Drainer restarts, it reads the checkpoint and then replicates data to the downstream starting from the corresponding `commit-ts`. The `["write save point"] [ts =411222863322546177]` Drainer log means saving the checkpoint with the corresponding timestamp.

Checkpoint is saved in different ways for different types of downstream platforms:

- For MySQL/TiDB, it is saved in the `tidb_binlog.checkpoint` table.
- For Kafka/file, it is saved in the file of the corresponding configuration directory.

The data of kafka/file contains `commit-ts`, so if the checkpoint is lost, you can check the latest `commit-ts` of the downstream data by consuming the latest data in the downstream .

Drainer reads the checkpoint when it starts. If Drainer cannot read the checkpoint, it uses the configured `initialCommitTs` as the start point of the initial replication.

13.11.15.9 How to redeploy Drainer on the new machine when Drainer fails and the data in the downstream remains?

If the data in the downstream is not affected, you can redeploy Drainer on the new machine as long as the data can be replicated from the corresponding checkpoint.

- If the checkpoint is not lost, perform the following steps:
    1. Deploy and start a new Drainer (Drainer can read checkpoint and resumes replication).
    2. Use binlogctl to change the state of the old Drainer to `offline`.
- If the checkpoint is lost, perform the following steps:
    1. To deploy a new Drainer, obtain the `commit-ts` of the old Drainer as the `initialCommitTs` of the new Drainer.
    2. Use binlogctl to change the state of the old Drainer to `offline`.

13.11.15.10 How to restore the data of a cluster using a full backup and a binlog backup file?

1. Clean up the cluster and restore a full backup.
2. To restore the latest data of the backup file, use Reparo to set `start-tso` = {snapshot timestamp of the full backup + 1} and `end-ts` = 0 (or you can specify a point in time).

13.11.15.11 How to redeploy Drainer when enabling `ignore-error` in Primary-Secondary replication triggers a critical error?

If a critical error is triggered when TiDB fails to write binlog after enabling `ignore-error`, TiDB stops writing binlog and binlog data loss occurs. To resume replication, perform the following steps:

1. Stop the Drainer instance.
2. Restart the `tidb-server` instance that triggers critical error and resume writing binlog (TiDB does not write binlog to Pump after critical error is triggered).
3. Perform a full backup in the upstream.
4. Clear the data in the downstream including the `tidb_binlog.checkpoint` table.
5. Restore the full backup to the downstream.
6. Deploy Drainer and use `initialCommitTs` (set `initialCommitTs` as the snapshot timestamp of the full backup) as the start point of initial replication.

13.11.15.12 When can I pause or close a Pump or Drainer node?

Refer to TiDB Binlog Cluster Operations to learn the description of the Pump or Drainer state and how to start and exit the process.

Pause a Pump or Drainer node when you need to temporarily stop the service. For example:

- Version upgrade

Use the new binary to restart the service after the process is stopped.

- Server maintenance

When the server needs a downtime maintenance, exit the process and restart the service after the maintenance is finished.

Close a Pump or Drainer node when you no longer need the service. For example:

- Pump scale-in

If you do not need too many Pump services, close some of them.

- Cancelling replication tasks

If you no longer need to replicate data to a downstream database, close the corresponding Drainer node.

- Service migration

If you need to migrate the service to another server, close the service and re-deploy it on the new server.

13.11.15.13 How can I pause a Pump or Drainer process?

- Directly kill the process.

> **Note:**
> Do not use the `kill -9` command. Otherwise, the Pump or Drainer node cannot process signals.

- If the Pump or Drainer node runs in the foreground, pause it by pressing Ctrl+C.
- Use the `pause-pump` or `pause-drainer` command in binlogctl.

13.11.15.14 Can I use the `update-pump` or `update-drainer` command in binlogctl to pause the Pump or Drainer service?

No. The `update-pump` or `update-drainer` command directly modifies the state information saved in PD without notifying Pump or Drainer to perform the corresponding operation. Misusing the two commands can interrupt data replication and might even cause data loss.

13.11.15.15 Can I use the `update-pump` or `update-drainer` command in binlogctl to close the Pump or Drainer service?

No. The `update-pump` or `update-drainer` command directly modifies the state information saved in PD without notifying Pump or Drainer to perform the corresponding operation. Misusing the two commands interrupts data replication and might even cause data inconsistency. For example:

- When a Pump node runs normally or is in the `paused` state, if you use the `update-pump` command to set the Pump state to `offline`, the Drainer node stops pulling the binlog data from the `offline` Pump. In this situation, the newest binlog cannot be replicated to the Drainer node, causing data inconsistency between upstream and downstream.
- When a Drainer node runs normally, if you use the `update-drainer` command to set the Drainer state to `offline`, the newly started Pump node only notifies Drainer nodes in the `online` state. In this situation, the `offline` Drainer fails to pull the binlog data from the Pump node in time, causing data inconsistency between upstream and downstream.

13.11.15.16 When can I use the `update-pump` command in binlogctl to set the Pump state to `paused`?

In some abnormal situations, Pump fails to correctly maintain its state. Then, use the `update-pump` command to modify the state.

For example, when a Pump process is exited abnormally (caused by directly exiting the process when a panic occurs or mistakenly using the `kill -9` command to kill the process), the Pump state information saved in PD is still `online`. In this situation, if you do not need to restart Pump to recover the service at the moment, use the `update-pump` command to update the Pump state to `paused`. Then, interruptions can be avoided when TiDB writes binlogs and Drainer pulls binlogs.

13.11.15.17 When can I use the `update-drainer` command in binlogctl to set the Drainer state to `paused`?

In some abnormal situations, the Drainer node fails to correctly maintain its state, which has influenced the replication task. Then, use the `update-drainer` command to modify the state.

For example, when a Drainer process is exited abnormally (caused by directly exiting the process when a panic occurs or mistakenly using the `kill -9` command to kill the process), the Drainer state information saved in PD is still `online`. When a Pump node is started, it fails to notify the exited Drainer node (the `notify drainer ...` error), which cause the Pump node failure. In this situation, use the `update-drainer` command to update the Drainer state to `paused` and restart the Pump node.

13.11.15.18 How can I close a Pump or Drainer node?

Currently, you can only use the `offline-pump` or `offline-drainer` command in binlogctl to close a Pump or Drainer node.

13.11.15.19 When can I use the `update-pump` command in binlogctl to set the Pump state to `offline`?

You can use the `update-pump` command to set the Pump state to `offline` in the following situations:

- When a Pump process is exited abnormally and the service cannot be recovered, the replication task is interrupted. If you want to recover the replication and accept some losses of binlog data, use the `update-pump` command to set the Pump state to `offline` ↪ . Then, the Drainer node stops pulling binlog from the Pump node and continues replicating data.
- Some stale Pump nodes are left over from historical tasks. Their processes have been exited and their services are no longer needed. Then, use the `update-pump` command to set their state to `offline`.

For other situations, use the `offline-pump` command to close the Pump service, which is the regular process.

> **Warning:**
>
> Do not use the `update-pump` command unless you can tolerate binlog data loss and data inconsistency between upstream and downstream, or you no longer need the binlog data stored in the Pump node.

13.11.15.20 Can I use the `update-pump` command in binlogctl to set the Pump state to `offline` if I want to close a Pump node that is exited and set to `paused`?

When a Pump process is exited and the node is in the `paused` state, not all the binlog data in the node is consumed in its downstream Drainer node. Therefore, doing so might risk data inconsistency between upstream and downstream. In this situation, restart the Pump and use the `offline-pump` command to close the Pump node.

13.11.15.21 When can I use the `update-drainer` command in binlogctl to set the Drainer state to `offline`?

Some stale Drainer nodes are left over from historical tasks. Their processes have been exited and their services are no longer needed. Then, use the `update-drainer` command to set their state to `offline`.

13.11.15.22 Can I use SQL operations such as `change pump` and `change drainer` to pause or close the Pump or Drainer service?

No. For more details on these SQL operations, refer to Use SQL statements to manage Pump or Drainer.

These SQL operations directly modifies the state information saved in PD and are functionally equivalent to the `update-pump` and `update-drainer` commands in binlogctl. To pause or close the Pump or Drainer service, use the binlogctl tool.

13.11.15.23 What can I do when some DDL statements supported by the upstream database cause error when executed in the downstream database?

To solve the problem, follow these steps:

1. Check `drainer.log`. Search `exec failed` for the last failed DDL operation before the Drainer process is exited.
2. Change the DDL version to the one compatible to the downstream. Perform this step manually in the downstream database.
3. Check `drainer.log`. Search for the failed DDL operation and find the `commit-ts` of this operation. For example:

```
[2020/05/21 09:51:58.019 +08:00] [INFO] [syncer.go:398] ["add ddl item
    ↪ to syncer, you can add this commit ts to `ignore-txn-commit-ts`
    ↪ to skip this ddl if needed"] [sql="ALTER TABLE `test` ADD INDEX
    ↪ (`index1`)"] ["commit ts"=416815754209656834].
```

4. Modify the `drainer.toml` configuration file. Add the `commit-ts` in the `ignore-txn-commit-ts` item and restart the Drainer node.

13.11.15.24 TiDB fails to write to binlog and gets stuck, and `listener stopped, waiting for manual stop` appears in the log

In TiDB v3.0.12 and earlier versions, the binlog write failure causes TiDB to report the fatal error. TiDB does not automatically exit but only stops the service, which seems like getting stuck. You can see the `listener stopped, waiting for manual stop` error in the log.

You need to determine the specific causes of the binlog write failure. If the failure occurs because binlog is slowly written into the downstream, you can consider scaling out Pump or increasing the timeout time for writing binlog.

Since v3.0.13, the error-reporting logic is optimized. The binlog write failure causes transaction execution to fail and TiDB Binlog will return an error but will not get TiDB stuck.

13.11.15.25 TiDB writes duplicate binlogs to Pump

This issue does not affect the downstream and replication logic.

When the binlog write fails or becomes timeout, TiDB retries writing binlog to the next available Pump node until the write succeeds. Therefore, if the binlog write to a Pump node is slow and causes TiDB timeout (default 15s), then TiDB determines that the write fails and tries to write to the next Pump node. If binlog is actually successfully written to the timeout-causing Pump node, the same binlog is written to multiple Pump nodes. When Drainer processes the binlog, it automatically de-duplicates binlogs with the same TSO, so this duplicate write does not affect the downstream and replication logic.

13.11.15.26 Reparo is interrupted during the full and incremental restore process. Can I use the last TSO in the log to resume replication?

Yes. Reparo does not automatically enable the safe-mode when you start it. You need to perform the following steps manually:

1. After Reparo is interrupted, record the last TSO in the log as `checkpoint-tso`.
2. Modify the Reparo configuration file, set the configuration item `start-tso` to `checkpoint-tso + 1`, set `stop-tso` to `checkpoint-tso + 80,000,000,000` (approximately five minutes after the `checkpoint-tso`), and set `safe-mode` to `true`. Start Reparo, and Reparo replicates data to `stop-tso` and then stops automatically.
3. After Reparo stops automatically, set `start-tso` to `checkpoint tso + 80,000,000,001` ↪ , set `stop-tso` to `0`, and set `safe-mode` to `false`. Start Reparo to resume replication.

13.12 PingCAP Clinic Diagnostic Service

13.12.1 PingCAP Clinic Overview

PingCAP Clinic Diagnostic Service (PingCAP Clinic) is a diagnostic service provided by PingCAP for TiDB clusters that are deployed using either TiUP or TiDB Operator. This service helps to troubleshoot cluster problems remotely and provides a quick check of cluster status locally. With PingCAP Clinic, you can ensure the stable operation of your TiDB cluster for its full life-cycle, predict potential problems, reduce the probability of problems, troubleshoot cluster problems quickly, and fix cluster problems.

PingCAP Clinic provides the following two components to diagnose cluster problems:

- Diag client:

  Diag client (Diag) is a diagnostic tool deployed on the cluster side. Diag is used to collect cluster diagnostic data, upload diagnostic data to the Clinic Server, and perform a quick health check locally on your cluster. For a full list of diagnostic data that can be collected by Diag, see PingCAP Clinic Diagnostic Data.

> **Note:**
> Diag supports TiDB v4.0 and later versions, but **does not support** collecting data from clusters deployed using TiDB Ansible.

- Clinic Server:

  Clinic Server is a cloud service deployed in the cloud. By providing diagnostic services in the SaaS model, the Clinic Server can not only receive uploaded diagnostic data but also work as an online diagnostic environment to store data, view data, and provide cluster diagnostic reports. Clinic Server provides two independent services depending on the storage location:

  - Clinic Server for international users: Data is stored in AWS in US.
  - Clinic Server for users in the Chinese mainland: Data is stored in AWS in China (Beijing) regions.

13.12.1.1 User scenarios

- Troubleshoot cluster problems remotely

  When your cluster has some problems that cannot be fixed quickly, you can get support from PingCAP or the community. When contacting technical support for remote assistance, you need to save various diagnostic data from the cluster and forward the data to the support staff. In this case, you can use Diag to collect diagnostic data with one click. Diag helps you to collect complete diagnostic data quickly, which can avoid

complex manual data collection operations. After collecting data, you can upload the data to the Clinic Server for PingCAP technical support staff to troubleshoot cluster problems. The Clinic Server provides secure storage for uploaded diagnostic data and supports the online diagnosis, which greatly improves the troubleshooting efficiency.

- Quickly check cluster status

  Even if your cluster is running stably for now, it is necessary to periodically check the cluster to detect potential stability risks. You can identify potential health risks of a cluster using the local and server-side quick check feature provided by PingCAP Clinic.

13.12.1.2 Implementation principles

This section introduces the implementation principles about how Diag collects diagnostic data from a cluster.

First, Diag gets cluster topology information from the deployment tool TiUP (tiup-cluster) or TiDB Operator (tidb-operator). Then, Diag collects different types of diagnostic data through various data collection methods as follows:

- Transfer server files through SCP

  For clusters deployed using TiUP, Diag can collect log files and configuration files directly from the nodes of the target component through the Secure copy protocol (SCP).

- Collect data by running commands remotely through SSH

  For clusters deployed using TiUP, Diag can connect to the target component system through SSH (Secure Shell) and run commands (such as Insight) to obtain system information, including kernel logs, kernel parameters, and basic information of the system and hardware.

- Collect data through HTTP call

  - By calling the HTTP interface of TiDB components, Diag can get the real-time configuration sampling information and the real-time performance sampling information of TiDB, TiKV, PD, and other components.
  - By calling the HTTP interface of Prometheus, Diag can get alert information and monitoring metrics data.

- Query database parameters through SQL statements

  Using SQL statements, Diag can query system variables and other information of TiDB. To use this method, you need to **additionally provide** the username and password to access TiDB when collecting data.

13.12.1.3 The limitations of Clinic Server

Note:

- Clinic Server is free from July 15, 2022 to April 15, 2025. You will be notified through email before April 15, 2025 if the service starts charging fee afterwards.
- If you want to adjust the usage limitations, get support from PingCAP.

Service Type	Limitation
Number of clusters	10/organization
Storage capacity	50 GB/cluster
Storage duration	180 days
Data size	3 GB/package
Saving duration of the data rebuild environment	3 days

13.12.1.4 Next step

- Use PingCAP Clinic in a self-hosted environment
  - Quick Start with PingCAP Clinic
  - Troubleshoot Clusters using PingCAP Clinic
  - PingCAP Clinic Diagnostic Data
- Use PingCAP Clinic on Kubernetes
  - Troubleshoot TiDB Cluster using PingCAP Clinic
  - PingCAP Clinic Diagnostic Data

13.12.2 Quick Start Guide for PingCAP Clinic

This document describes how to use PingCAP Clinic diagnosis service (PingCAP Clinic) to collect, upload, and view cluster diagnosis data quickly.

PingCAP Clinic consists of two components: Diag client (shorten as Diag) and Clinic Server cloud service (shorten as Clinic Server). For details of these two components, refer to PingCAP Clinic Overview.

13.12.2.1 User scenarios

- To accurately identify and quickly resolve problems in your cluster when seeking help remotely from PingCAP technical support, you can collect diagnostic data with Diag, upload the collected data to the Clinic Server, and provide the data access link to the technical support.
- When the cluster is running properly and you need to check the status of the cluster, you can use Diag to collect diagnostic data, upload the data to Clinic Server, and view the results of Health Report.

Note:

- The following methods to collect and upload data are **only** applicable to clusters deployed using TiUP. For clusters deployed using TiDB Operator on Kubernetes, see PingCAP Clinic for TiDB Operator environments.
- The diagnostic data collected by PingCAP Clinic is **only** used for troubleshooting cluster problems.

13.12.2.2 Prerequisites

Before using PingCAP Clinic, you need to install Diag and prepare an environment for uploading data.

1. On your control machine with TiUP installed, run the following command to install Diag:

```
tiup install diag
```

2. Log in to Clinic Server.

   Go to the Clinic Server for international users and select **Sign in with TiDB Account** to enter the TiDB Cloud login page. If you do not have a TiDB Cloud account, create one on that page.

Note:

A TiDB Cloud account is only used for logging in to Clinic Server in SSO mode and is not mandatory for accessing the TiDB Cloud service.

Go to the Clinic Server for users in the Chinese mainland and select **Sign in with AskTUG** to enter the AskTUG community login page. If you do not have an AskTUG account, create one on that page

3. Create an organization on the Clinic Server. Organization is a collection of TiDB clusters. You can upload diagnostic data on the created organization.

4. Get an access token to upload data. When uploading collected data through Diag, you need a token for user authentication to ensure the data is isolated securely. If you already get a token from the Clinic Server, you can reuse the token.

   To get a token, click the icon in the lower-right corner of the Cluster page, select **Get Access Token For Diag Tool**, and click + in the pop-up window. Make sure that you have copied and saved the token that is displayed.

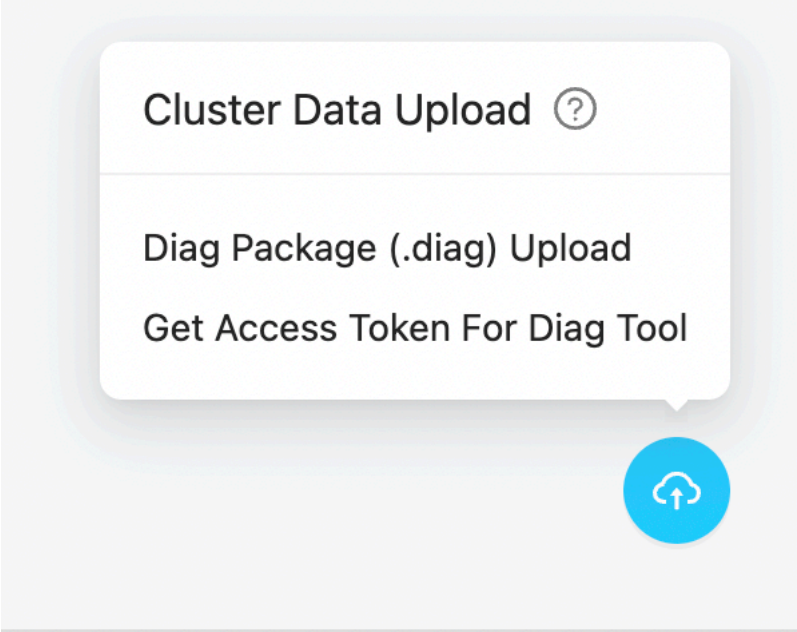


Figure 304: An example of a token

Note:

- For data security, TiDB only displays the token information when it is created. If you lost the information, you can delete the old token and create a new one.
- A token is only used for uploading data.

5. Set the token and `region` in Diag.

   - Run the following command to set the `clinic.token`:

     ```
     tiup diag config clinic.token ${token-value}
     ```

   - Run the following command to set the `clinic.region`:

`region` determines the encryption certificate used for packing data and the target service when uploading the data. For example:

> **Note:**
>
> - Diag v0.9.0 and later versions support setting `region`.
> - For versions earlier than Diag v0.9.0, data is uploaded to Clinic Server in the Chinese region by default. To set `region` in these versions, run the `tiup update diag` command to upgrade Diag to the latest version and then set `region` in Diag.

When using Clinic Server for international users, set `region` to `US` using the following command:

```
tiup diag config clinic.region US
```

When using Clinic Server for users in the Chinese mainland, set `region` to `CN` using the following command:

```
tiup diag config clinic.region CN
```

6. (Optional) Enable log redaction.

   When TiDB provides detailed log information, it might print sensitive information (for example, user data) in the log. If you want to avoid leaking sensitive information in the local log and Clinic Server, you can enable log redaction in the TiDB side. For more information, see log redaction.

13.12.2.3 Steps

1. Run Diag to collect diagnostic data.

   For example, to collect the diagnostic data from 4 hours ago to 2 hours ago based on the current time, run the following command:

   ```
   tiup diag collect ${cluster-name} -f="-4h" -t="-2h"
   ```

   After you run the command, Diag does not start collecting data immediately. Instead, Diag provides the estimated data size and the target data storage path in the output for you to confirm whether to continue. To confirm that you want to start collecting data, enter `Y`.

   After the collection is complete, Diag provides the folder path where the collected data is located.

2. Upload the collected data to Clinic Server.

> **Note:**
> The size of data (the compressed file with collected data) to be uploaded should be **no larger than** 3 GB. Otherwise, the data upload fails.

- If the network where your cluster is located can access the internet, you can directly upload the folder with collected data using the following command:

```
tiup diag upload ${filepath}
```

  After the upload is completed, the `Download URL` is displayed in the output.

> **Note:**
> When uploading data using this method, you need to use Diag v0.9.0 or a later version. You can get the Diag version when you run it. If the Diag version is earlier than 0.9.0, you can use the `tiup update diag` command to upgrade Diag to the latest version.

- If the network where your cluster is located cannot access the internet, you need to pack the collected data and upload the package. For details, see Method 2. Pack and upload data.

3. After the upload is complete, get the data access link from `Download URL` in the command output.

   By default, the diagnostic data includes the cluster name, cluster topology information, log content in the collected diagnostic data, and Grafana Dashboard information reorganized based on the metrics in the collected data.

   You can use the data to troubleshoot cluster problems by yourself, or you can provide the data access link to PingCAP technical support staff to facilitate the remote troubleshooting.

4. View the results of Health Report

   After data is uploaded, Clinic Server processes the data automatically in the background. The Health Report is generated in approximately 5 to 15 minutes. You can view the report by opening the diagnostic data link and click the “Health Report”.

13.12.2.4 What’s next

- PingCAP Clinic Overview
- Troubleshoot Clusters Using PingCAP Clinic
- PingCAP Clinic Diagnostic Data

13.12.3 Troubleshoot Clusters Using PingCAP Clinic

For TiDB clusters and DM clusters deployed using TiUP, you can use PingCAP Clinic Diagnostic Service (PingCAP Clinic) to troubleshoot cluster problems remotely and perform a quick check on cluster status locally using Diag client (Diag) and Clinic Server.

> **Note:**
>
> - This document **only** applies to clusters deployed using TiUP in a self-hosted environment. For clusters deployed using TiDB Operator on Kubernetes, see PingCAP Clinic for TiDB Operator environments.
> - PingCAP Clinic **does not support** collecting data from clusters deployed using TiDB Ansible.

13.12.3.1 User scenarios

- Troubleshoot cluster problems remotely
  - When your cluster has some problems, if you need to get support from PingCAP, you can perform the following operations to facilitate the remote troubleshooting: collect diagnostic data with Diag, upload the collected data to the Clinic Server, and provide the data access link to the technical support staff.
  - When your cluster has some problems, if you cannot analyze the problems immediately, you can use Diag to collect and save the data for later analysis.
- Perform a quick check on the cluster status locally

  Even if your cluster is running stably for now, it is necessary to periodically check the cluster to detect potential stability risks. You can identify potential health risks of a cluster using the local quick check feature provided by PingCAP Clinic. The local check only checks configuration. To check more items, such as metrics and logs, it is recommended to upload the diagnostic data to the Clinic Server and use the Health Report feature.

13.12.3.2 Prerequisites

Before using PingCAP Clinic, you need to install Diag (a component to collect data provided by PingCAP Clinic) and prepare the environment to upload data.

1. Install Diag.

- If you have installed TiUP on your control machine, run the following command to install Diag:

```
```bash
tiup install diag
```
```

    - If you have installed Diag, you can use the following command to upgrade Diag to the latest version:

```
tiup update diag
```

> **Note:**
>
> - For clusters without an internet connection, you need to deploy Diag offline. For details, refer to Deploy TiUP offline: Method 2.
> - Diag is **only** provided in the TiDB Server offline mirror package of v5.4.0 or later.

2. Get and set an access token (token) to upload data.

   When uploading collected data through Diag, you need a token for user authentication. If you already set a token Diag, you can reuse the token and skip this step.

   To get a token, perform the following steps:

   - Log in to the Clinic Server.

     Clinic Server for international users: Data is stored in AWS in US.

     Clinic Server for users in the Chinese mainland: Data is stored in AWS in China (Beijing) regions.
   - Click the icon in the lower-right corner of the Cluster page, select **Get Access Token For Diag Tool**, and click + in the pop-up window. Make sure that you have copied and saved the token that is displayed.

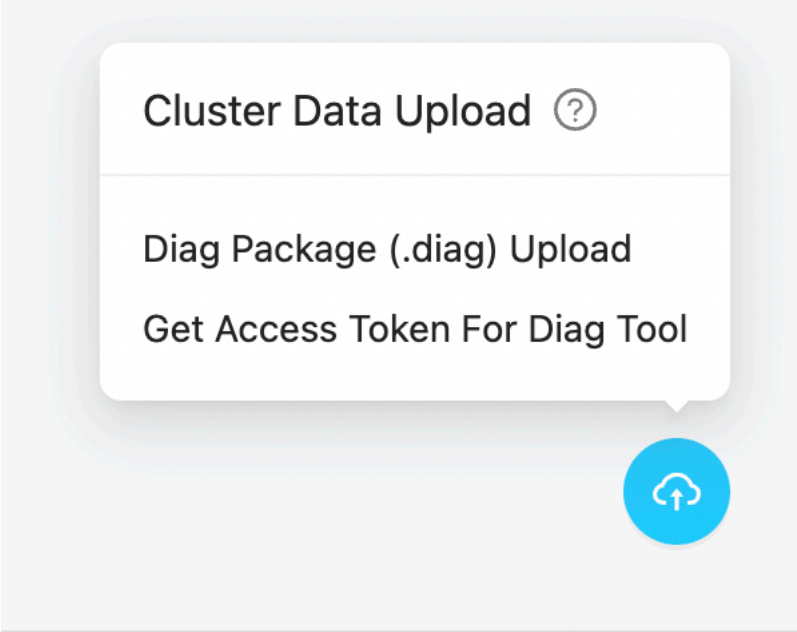


Figure 305: Get the Token

Note:

- When accessing Clinic Server for the first time, before getting a token, you need to prepare the environment by referring to Quick Start with PingCAP Clinic.
- For data security, TiDB only displays the token upon the token creation. If you have lost the token, delete the old token and create a new one.
- A token is only used for uploading data.

- Then, set the token in Diag. For example:

```
tiup diag config clinic.token ${token-value}
```

3. Set the `region` in Diag.

   `region` determines the encryption certificate used for packing data and the target service when uploading the data. For example:

Note:

- Diag v0.9.0 and later versions support setting `region`.
- For versions earlier than Diag v0.9.0, data is uploaded to Clinic Server in the Chinese region by default. To set `region` in these

versions, run the `tiup update diag` command to upgrade Diag to the latest version and then set `region` in Diag.

When using Clinic Server for international users, set `region` to `US` using the following command:

```
tiup diag config clinic.region US
```

When using Clinic Server for users in the Chinese mainland, set `region` to `CN` using the following command:

```
tiup diag config clinic.region CN
```

4. (Optional) Enable log redaction.

   When TiDB provides detailed log information, it might print sensitive information (for example, user data) in the log. If you want to avoid leaking sensitive information in the local log and Clinic Server, you can enable log redaction in the TiDB side. For more information, see log redaction.

13.12.3.3 Troubleshoot cluster problems remotely

You can use Diag to quickly collect diagnostic data from TiDB clusters and DM clusters, including monitoring data and configuration information.

13.12.3.3.1 Step 1. Check the data to be collected

For a full list of data that can be collected by Diag, see PingCAP Clinic Diagnostic Data.

To improve the efficiency of the later diagnosis, you are recommended to collect full diagnostic data including monitoring data and configuration information. For details, see Collect data from clusters.

13.12.3.3.2 Step 2. Collect data

With Diag, you can collect data from the TiDB clusters and the DM clusters deployed using TiUP.

1. Run the data collection command of Diag.

   For example, to collect the diagnostic data from 4 hours ago to 2 hours ago based on the current time, run the following command:

```
tiup diag collect ${cluster-name} -f="-4h" -t="-2h"
```

```
tiup diag collectdm ${dm-cluster-name} -f="-4h" -t="-2h"
```

Descriptions of the parameters for data collection:

- `-f/--from`: specifies the start time of the data collection. If you do not specify this parameter, the default start time is 2 hours before the current time. To modify the time zone, use the `-f="12:30 +0800"` syntax. If you do not specify the time zone information in this parameter, such as `+0800`, the time zone is UTC by default.
- `-t/--to`: specifies the end time of the data collection. If you do not specify this parameter, the default end time is the current moment. To modify the time zone, use the `-f="12:30 +0800"` syntax. If you do not specify the time zone information in this parameter, such as `+0800`, the time zone is UTC by default.

Parameter usage tips:

In addition to specifying the data collection time, you can use Diag to specify more parameters. To get all parameters, run the `tiup diag collect -h` or `tiup diag collectdm -h` command.

Note:

- Diag does not collect system variables data (db_vars) by default. To collect this data, you need to additionally provide a username and password that can access the database. Note that the reading access to system variables should be enabled in this database.
- Diag does not collect performance data (`perf`) and debug data (`debug`) by default.
- To collect full diagnostic data including system variables, use the command `tiup diag collect <cluster-name> --include="system,monitor,log,config,db_vars,perf,debug"`.

- `-l`: the bandwidth limit for transferring files, the unit is Kbit/s, and the default value is `100000` (the `-l` parameter of scp).
- `-N/--node`: only collects data from a specified node. The format is `ip:port`.
- `--include`: only collects specific types of data. The optional values are `system`, `monitor`, `log`, `config`, and `db_vars`. To include two or more types, you can use `,` as a separator between the types.
- `--exclude`: does not collect specific types of data. The optional values are `system`, `monitor`, `log`, `config`, and `db_vars`. To exclude two or more types, you can use `,` as a separator between the types.

After you run the command, Diag does not start collecting data immediately. Instead, Diag provides the estimated data size and the target data storage path in the output for you to confirm whether to continue. For example:

```
Estimated size of data to collect:
Host              Size      Target
----              ----      ------
172.16.7.129:9090 43.57 MB 1775 metrics, compressed
172.16.7.87       0 B       /tidb-deploy/tidb-4000/log/tidb_stderr.log
... ...
172.16.7.179      325 B     /tidb-deploy/tikv-20160/conf/tikv.toml
Total             2.01 GB   (inaccurate)
These data will be stored in /home/user/diag-fNTnz5MGhr6
Do you want to continue? [y/N]: (default=N)
```

2. Enter `Y` to confirm that you want to start collecting data.

   Collecting data takes a certain amount of time. The time varies according to the volume of data to be collected. For example, in a test environment, collecting 1 GB of data takes about 10 minutes.

   After the collection is complete, Diag provides the folder path where the collected data is located. For example:

```
Collected data are stored in /home/user/diag-fNTnz5MGhr6
```

13.12.3.3.3 Step 3. View data locally (optional)

The collected data is stored in separate subdirectories based on its data source. These subdirectories are named after machine names and port numbers. The storage locations of the configuration, logs, and other files of each node are the same as their relative storage paths in the real server of your TiDB cluster:

- Basic information of the system and hardware: in `insight.json`
- Contents in the system `/etc/security/limits.conf`: in `limits.conf`
- List of kernel parameters: in `sysctl.conf`
- Kernel logs: in `dmesg.log`
- Network connection during data collection: in `ss.txt`
- Configuration data: in the `config.json` directory of each node
- Meta-information for the cluster itself: in `meta.yaml` (this file is located at the top level of the directory that stores collected data)
- Monitoring data: in the `/monitor` file directory. The monitoring data is compressed by default and cannot be viewed directly. To directly view the JSON files with monitoring data, disable compression with the `--compress-metrics=false` parameter when collecting data.

13.12.3.3.4 Step 4. Upload data

To provide cluster diagnostic data to PingCAP technical support staff, you need to upload the data to the Clinic Server first, and then send the obtained data access link to the staff. The Clinic Server is a cloud service that stores and shares diagnostic data securely.

Depending on the network connection of the cluster, you can choose one of the following methods to upload data:

- Methods 1: if the network where the cluster is located can access the internet, you can directly upload data using the upload command.
- Methods 2: if the network where the cluster is located cannot access the internet, you need to pack the data and then upload it.

> **Note:**
>
> If you did not set a token or `region` in Diag before uploading data, Diag reports the upload failure and reminds you to set a token or `region`. To set a token, see the second step in Prerequisites.

Method 1. Upload directly

If the network where the cluster is located can access the internet, you can directly upload the folder with collected data obtained in Step 2: Collect data using the following command:

```
tiup diag upload
```

After the upload is completed, the `Download URL` is displayed in the output. You can open the link of `Download URL` to see the uploaded data or send the link to the PingCAP technical support staff you contacted before.

Method 2. Pack and upload data

If the network where your cluster is located cannot access the internet, you need to pack the data on your intranet and upload the data package to the Clinic Server using a device with internet access. The detailed operations are as follows:

1. Pack the collected data obtained in Step 2. Collect data by running the following command:

   ```
   tiup diag package ${filepath}
   ```

   During packaging, Diag encrypts and compresses the data at the same time. In the test environment, 800 MB of data was compressed to 57 MB. The following is an example output:

```
Starting component `diag`: /root/.tiup/components/diag/v0.7.0/diag
    ↪ package diag-fNTnz5MGhr6
packaged data set saved to /home/user/diag-fNTnz5MGhr6.diag
```

After the packaging is complete, the data is packaged to the `.diag` format. The `.diag` file can only be decrypted and viewed after being uploaded to the Clinic Server. If you want to directly forward the collected data instead of uploading it to the Clinic Server, you can compress the data by your own method and forward it.

2. From a machine with internet access, upload the compressed data package:

```
tiup diag upload ${filepath}
```

The following is an example output:

```
[root@Copy-of-VM-EE-CentOS76-v1 user]# tiup diag upload /home/user/
    ↪ diag-fNTnz5MGhr6
Starting component `diag`: /root/.tiup/components/diag/v0.7.0/diag
    ↪ upload /home/user/diag-fNTnz5MGhr6
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<
Completed!
Download URL: "https://clinic.pingcap.com.cn/portal/#/orgs/4/clusters/
    ↪ XXXX"
```

3. After the upload is complete, you can open the link of `Download URL` to see the uploaded data or send the link to the PingCAP technical support staff you contacted before.

13.12.3.4 Perform a quick check on the cluster status locally

You can have a quick check on the cluster status locally using Diag. Even if your cluster is running stably for now, it is necessary to periodically check the cluster to detect potential stability risks. You can identify potential health risks of a cluster using the local quick check feature provided by PingCAP Clinic. The local check only checks configuration. To check more items, such as metrics and logs, it is recommended to upload the diagnostic data to the Clinic Server and use the Health Report feature.

1. Collect configuration data:

```
tiup diag collect ${cluster-name} --include="config"
```

The data of configuration files are relatively small. After the collection, the collected data is stored in the current path by default. In the test environment, for a cluster with 18 nodes, the data size of configuration files is less than 10 KB.

2. Diagnose configuration data:

```
tiup diag check ${subdir-in-output-data}
```

`${subdir-in-output-data}` in the above command is the path that stores the collected data, and this path has the `meta.yaml` file.

3. View the diagnostic result:

   The diagnostic result is returned on the command line. For example:

```
Starting component `diag`: /root/.tiup/components/diag/v0.7.0/diag
    ↪ check diag-fNTnz5MGhr6

# Diagnostic result
lili 2022-01-24T09:33:57+08:00

## 1. Cluster basic information
- Cluster ID: 7047403704292855808
- Cluster Name: lili
- Cluster Version: v5.3.0

## 2. Sampling information
- Sample ID: fNTnz5MGhr6
- Sampling Date: 2022-01-24T09:33:57+08:00
- Sample Content:: [system monitor log config]

## 3. Diagnostic result, including potential configuration problems
In this inspection, 22 rules were executed.

The results of **1** rules were abnormal and needed to be further
    ↪ discussed with support team.

The following is the details of the abnormalities.

### Diagnostic result summary
The configuration rules are all derived from PingCAP's OnCall Service.

If the results of the configuration rules are found to be abnormal,
    ↪ they may cause the cluster to fail.

There were **1** abnormal results.

#### Path to save the diagnostic result file

Rule Name: tidb-max-days
- RuleID: 100
- Variation: TidbConfig.log.file.max-days
```

```
- For more information, please visit: https://s.tidb.io/msmo6awg
- Check Result:
  TidbConfig_172.16.7.87:4000 TidbConfig.log.file.max-days:0 warning
  TidbConfig_172.16.7.86:4000 TidbConfig.log.file.max-days:0 warning
  TidbConfig_172.16.7.179:4000 TidbConfig.log.file.max-days:0 warning

Result report and record are saved at diag-fNTnz5MGhr6/report
    ↪ -220125153215
```

In the last section of the diagnostic result (under `#### Path to save the ↪ diagnostic result file` in the above example output), for each configuration potential risk found, Diag provides a corresponding knowledge base link with detailed configuration suggestions. In the example above, the relevant link is `https://s.tidb ↪ .io/msmo6awg`.

13.12.3.5 FAQ

1. If the data upload fails, can I re-upload it?

   Yes. Data upload supports breakpoint upload. If the upload fails, you can upload it again directly.

2. After uploading data, I cannot open the returned data access link. What should I do?

   Log in to Clinic Server first. If you still cannot open the link after login success, check whether you have access to data. If not, contact the data owner for permission. After getting the permission, log in to Clinic Server and open the link again.

3. How long will the uploaded data be kept on the Clinic Server?

   The longest time is 180 days. You can delete the data you uploaded on the Clinic Server page at any time.

13.12.4 PingCAP Clinic Diagnostic Data

This document provides the types of diagnostic data that can be collected by PingCAP Clinic Diagnostic Service (PingCAP Clinic) from the TiDB and DM clusters deployed using TiUP. Also, the document lists the parameters for data collection corresponding to each data type. When running a command to collect data using Diag client (Diag), you can add the required parameters to the command according to the types of the data to be collected.

The diagnostic data collected by PingCAP Clinic is **only** used for troubleshooting cluster problems.

A diagnostic service deployed in the cloud, Clinic Server provides two independent services depending on the data storage location:

- Clinic Server for international users: If you upload the collected data to Clinic Server for international users, the data will be stored in the Amazon S3 service deployed by PingCAP in AWS US regions. PingCAP uses strict data access policies and only authorized technical support can access the data.
- Clinic Server for users in the Chinese mainland: If you upload the collected data to Clinic Server for users in the Chinese mainland, the data will be stored in the Amazon S3 service deployed by PingCAP in China (Beijing) regions. PingCAP uses strict data access policies and only authorized technical support can access the data.

13.12.4.1 TiDB clusters

This section lists the types of diagnostic data that can be collected by Diag from the TiDB clusters deployed using TiUP.

13.12.4.1.1 TiDB cluster information

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Basic information of the cluster, including the cluster ID	`cluster.json`	The data is collected per run by default.
Detailed information of the cluster	`meta.yaml`	The data is collected per run by default.

13.12.4.1.2 TiDB diagnostic data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Log	`tidb.log`	`--include=log`
Error log	`tidb_stderr.log`	`--include=log`

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Slow log	`tidb_slow_query.log`	`--include=log`
Configuration file	`tidb.toml`	`--include=config`
Real-time configuration	`config.json`	`--include=config`

13.12.4.1.3 TiKV diagnostic data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Log	`tikv.log`	`--include=log`
Error log	`tikv_stderr.log`	`--include=log`
Configuration file	`tikv.toml`	`--include=config`
Real-time configuration	`config.json`	`--include=config`

13.12.4.1.4 PD diagnostic data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Log	`pd.log`	`--include=log`
Error log	`pd_stderr.log`	`--include=log`
Configuration file	`pd.toml`	`--include=config`
Real-time configuration	`config.json`	`--include=config`

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Outputs of the command `tiup ↪ ctl ↪ :< ↪ cluster ↪ - ↪ version ↪ > ↪ pd ↪ -u ↪ http ↪ :// ↪ ${ ↪ pd ↪ IP ↪ }:$ ↪ { ↪ PORT ↪ } ↪ store ↪`	`store. ↪ json ↪`	`-- ↪ include ↪ = ↪ config ↪`

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Outputs of the command `tiup ↪ ctl ↪ : ↪ cluster ↪ - ↪ version ↪ > ↪ pd ↪ -u ↪ http ↪ :// ↪ ${ ↪ pd ↪ IP ↪ }:$ ↪ { ↪ PORT ↪ } ↪ config ↪ ↪ placement ↪ - ↪ rules ↪ ↪ show ↪`	`placement ↪ - ↪ rule ↪ . ↪ json ↪`	`-- ↪ include ↪ = ↪ config ↪`

13.12.4.1.5 TiFlash diagnostic data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Log	`tiflash` ↪ `.` ↪ `log`	`--` ↪ `include` ↪ `=log`
Error log	`tiflash_stderr` ↪ `.` ↪ `log`	`--` ↪ `include` ↪ `=log`
Configuration file	`tiflash` ↪ `-` ↪ `learner` ↪ `.` ↪ `toml` ↪ `,` `tiflash` ↪ `-` ↪ `preprocessed` ↪ `.` ↪ `toml` ↪ `,` `tiflash` ↪ `.` ↪ `toml` ↪	`--` ↪ `include` ↪ `=` ↪ `config` ↪
Real-time configuration	`config` ↪ `.` ↪ `json` ↪	`--` ↪ `include` ↪ `=` ↪ `config` ↪

13.12.4.1.6 TiCDC diagnostic data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Log	`ticdc.log`	`--include=log`
Error log	`ticdc_stderr.log`	`--include=log`
Configuration file	`ticdc.toml`	`--include=config`
Debug data	`info.txt`, `status.txt`, `changefeeds.txt`, `captures.txt`, `processors.txt`	`--include=debug` (Diagnostics does not collect this data type by default)

13.12.4.1.7 Prometheus monitoring data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
All metrics data	`{metric_name}.json`	`--include=monitor`
All alerts data	`alerts.json`	`--include=monitor`

13.12.4.1.8 TiDB system variables

Data type	Exported file	Parameter for data collection by PingCAP Clinic
TiDB system variables	`mysql.tidb` ↪ `.` ↪ `csv`	`--include=db_vars` (Diag does not collect this data type by default; if you need to collect this data type, database credential is required)
	`global_variables` ↪ `.` ↪ `csv`	`--include=db_vars` (Diag does not collect this data type by default)

13.12.4.1.9 System information of the cluster node

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Kernel log	`dmesg.log`	`--include=system`
Basic information of the system and hardware	`insight.json`	`--include=system`
Contents in the `/etc/security/limits.conf`	`limits.conf`	`--include=system`
List of kernel parameters	`sysctl.conf`	`--include=system`
Socket system information, which is the output of the `ss` command	`ss.txt`	`--include=system`

13.12.4.2 DM clusters

This section lists the types of diagnostic data that can be collected by Diag from the DM clusters deployed using TiUP.

13.12.4.2.1 DM cluster information

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Basic information of the cluster, including the cluster ID	`cluster.json`	The data is collected per run by default.
Detailed information of the cluster	`meta.yaml`	The data is collected per run by default.

13.12.4.2.2 dm-master diagnostic data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Log	`m-master.log`	`--include=log`
Error log	`dm-master_stderr.log`	`--include=log`
Configuration file	`dm-master.toml`	`--include=config`

13.12.4.2.3 dm-worker diagnostic data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Log	`dm-worker.log`	`--include=log`
Error log	`dm-worker_stderr.log`	`--include=log`
Configuration file	`dm-work.toml`	`--include=config`

13.12.4.2.4 Prometheus monitoring data

Data type	Exported file	Parameter for data collection by PingCAP Clinic
All metrics data	`{metric_name}.json`	`--include=monitor`
All alerts data	`alerts.json`	`--include=monitor`

13.12.4.2.5 System information of the cluster node

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Kernel log	`dmesg.log`	`--include=system`
Basic information of the system and hardware	`insight.json`	`--include=system`
Contents in the `/etc/security/limits.conf` system	`limits.conf`	`--include=system`
List of kernel parameters	`sysctl.conf`	`--include=system`

Data type	Exported file	Parameter for data collection by PingCAP Clinic
Socket system information, which is the output of the `ss` command	`ss.txt`	`--include=system`

13.13 TiSpark

13.13.1 TiSpark User Guide

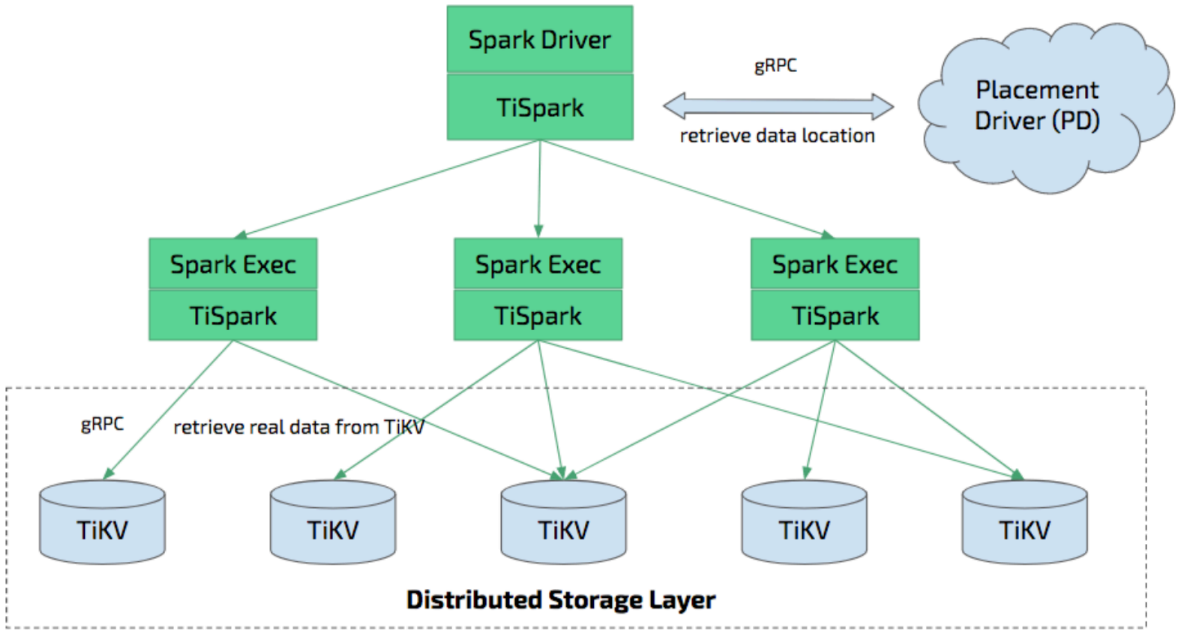


Figure 306: TiSpark architecture

TiSpark is a thin layer built for running Apache Spark on top of TiDB/TiKV to answer the complex OLAP queries. It takes advantages of both the Spark platform and the distributed TiKV cluster and seamlessly glues to TiDB, the distributed OLTP database, to provide a Hybrid Transactional/Analytical Processing (HTAP) solution to serve as a one-stop solution for both online transactions and analysis.

TiFlash is another tool that enables HTAP. Both TiFlash and TiSpark allow the use of multiple hosts to execute OLAP queries on OLTP data. TiFlash stores data in a columnar format, which allows more efficient analytical queries. TiFlash and TiSpark can be used together.

TiSpark depends on the TiKV cluster and the PD cluster. You also need to set up a Spark cluster. This document provides a brief introduction to how to setup and use TiSpark. It requires some basic knowledge of Apache Spark. For more information, see Apache Spark website.

Deeply integrating with Spark Catalyst Engine, TiSpark provides precise control on computing. This allows Spark to read data from TiKV efficiently. TiSpark also supports index seek, which enables high-speed point query.

TiSpark accelerates data queries by pushing computing to TiKV so as to reduce the volume of data to be processed by Spark SQL. Meanwhile, TiSpark can use TiDB built-in statistics to select the best query plan.

With TiSpark and TiDB, you can run both transaction and analysis tasks on the same platform without building and maintaining ETLs. This simplifies the system architecture and reduces the cost of maintenance.

You can use tools of the Spark ecosystem for data processing on TiDB:

- TiSpark: Data analysis and ETLs
- TiKV: Data retrieval
- Scheduling system: Report generation

Also, TiSpark supports distributed writes to TiKV. Compared with writes to TiDB by using Spark and JDBC, distributed writes to TiKV can implement transactions (either all data are written successfully or all writes fail), and the writes are faster.

> **Warning:**
>
> Because TiSpark accesses TiKV directly, the access control mechanisms used by TiDB Server are not applicable to TiSpark. Since TiSpark v2.5.0, TiSpark supports user authentication and authorization, for more information, see Security.

13.13.1.1 Environment setup

The following table lists the compatibility information of the supported TiSpark versions. You can choose a TiSpark version according to your need.

TiSpark version	TiDB, TiKV, and PD versions	Spark version	Scala version
2.4.x-scala_2.11	5.x, 4.x	2.3.x, 2.4.x	2.11
2.4.x-scala_2.12	5.x, 4.x	2.4.x	2.12
2.5.x	5.x, 4.x	3.0.x, 3.1.x	2.12
3.0.x	5.x, 4.x	3.0.x, 3.1.x, 3.2.x	2.12

TiSpark runs in any Spark mode such as YARN, Mesos, and Standalone.

13.13.1.2 Recommended configuration

This section describes the recommended configuration of independent deployment of TiKV and TiSpark, independent deployment of Spark and TiSpark, and co-deployed TiKV and TiSpark.

See also TiSpark Deployment Topology for more details about how to deploy TiSpark using TiUP.

13.13.1.2.1 Configuration of independent deployment of TiKV and TiSpark

For independent deployment of TiKV and TiSpark, it is recommended to refer to the following recommendations:

- Hardware configuration
    - For general purposes, refer to the TiDB and TiKV hardware configuration recommendations.
    - If the usage is more focused on the analysis scenarios, you can increase the memory of the TiKV nodes to at least 64G.

13.13.1.2.2 Configuration of independent deployment of Spark and TiSpark

See the Spark official website for the detail hardware recommendations. The following is a short overview of TiSpark configuration:

- It is recommended to allocate 32G memory for Spark, and reserve at least 25% of the memory for the operating system and buffer cache.
- It is recommended to provision at least 8 to 16 cores on per machine for Spark. Initially, you can assign all the CPU cores to Spark.

13.13.1.2.3 Configuration of co-deployed TiKV and TiSpark

To co-deploy TiKV and TiSpark, add TiSpark required resources to the TiKV reserved resources, and allocate 25% of the memory for the system.

13.13.1.3 Deploy the TiSpark cluster

Download TiSpark's jar package here and place it in the `$SPARKPATH/jars` folder.

> **Note:**
>
> TiSpark v2.1.x and older versions have file names that look like `tispark-core-2.1.9-spark_2.4-jar-with-dependencies.jar`. Please check the releases page on GitHub for the exact file name for the version you want.

The following is a short example of how to install TiSpark v2.4.1:

```
wget https://github.com/pingcap/tispark/releases/download/v2.4.1/tispark-assembly-2.4.1.jar
mv tispark-assembly-2.4.1.jar $SPARKPATH/jars/
```

Copy the `spark-defaults.conf` from the `spark-defaults.conf.template` file:

```
cp conf/spark-defaults.conf.template conf/spark-defaults.conf
```

In the `spark-defaults.conf` file, add the following lines:

```
spark.tispark.pd.addresses $pd_host:$pd_port
spark.sql.extensions org.apache.spark.sql.TiExtensions
```

The `spark.tispark.pd.addresses` configuration allows you to put in multiple PD servers. Specify the port number for each of them. For example, when you have multiple PD servers on `10.16.20.1,10.16.20.2,10.16.20.3` with the port 2379, put it as `10.16.20.1:2379,10.16.20.2:2379,10.16.20.3:2379`.

> **Note:**
>
> If TiSpark could not communicate properly, please check your firewall configuration. You can adjust the firewall rules or disable it on your need.

13.13.1.3.1 Deploy TiSpark on an existing Spark cluster

Running TiSpark on an existing Spark cluster does not require a reboot of the cluster. You can use Spark's `--jars` parameter to introduce TiSpark as a dependency:

```
spark-shell --jars $TISPARK_FOLDER/tispark-${name_with_version}.jar
```

13.13.1.3.2 Deploy TiSpark without a Spark cluster

If you do not have a Spark cluster, we recommend using the standalone mode. For more information, see Spark Standalone. If you encounter any problem, see Spark official website. And you are welcome to file an issue on our GitHub.

13.13.1.4 Use Spark Shell and Spark SQL

Assume that you have successfully started the TiSpark cluster as described above. The following describes how to use Spark SQL for OLAP analysis on a table named `lineitem` in the `tpch` database.

To generate the test data via a TiDB server available on `192.168.1.101`:

```
tiup bench tpch prepare --host 192.168.1.101 --user root
```

Assuming that your PD node is located at `192.168.1.100`, port `2379`, add the following command to `$SPARK_HOME/conf/spark-defaults.conf`:

```
spark.tispark.pd.addresses 192.168.1.100:2379
spark.sql.extensions org.apache.spark.sql.TiExtensions
```

Start the Spark Shell:

```
./bin/spark-shell
```

And then enter the following command in the Spark Shell as in native Apache Spark:

```
spark.sql("use tpch")
spark.sql("select count(*) from lineitem").show
```

The result is:

```
+-------------+
| Count (1) |
+-------------+
| 2000     |
+-------------+
```

Besides Spark Shell, there is also Spark SQL available. To use Spark SQL, run:

```
./bin/spark-sql
```

You can run the same query:

```
use tpch;
select count(*) from lineitem;
```

The result is:

```
2000
Time taken: 0.673 seconds, Fetched 1 row(s)
```

13.13.1.5 Use JDBC support with ThriftServer

You can use Spark Shell or Spark SQL without JDBC support. However, JDBC support is required for tools like beeline. JDBC support is provided by Thrift server. To use Spark's Thrift server, run:

```
./sbin/start-thriftserver.sh
```

To connect JDBC with Thrift server, you can use JDBC supported tools including beeline.

For example, to use it with beeline:

```
./bin/beeline jdbc:hive2://localhost:10000
```

If the following message is displayed, you have enabled beeline successfully.

```
Beeline version 1.2.2 by Apache Hive
```

Then, you can run the query command:

```
1: jdbc:hive2://localhost:10000> use testdb;
+---------+--+
| Result |
+---------+--+
+---------+--+
No rows selected (0.013 seconds)

select count(*) from account;
+-----------+--+
| count(1) |
+-----------+--+
| 1000000 |
+-----------+--+
1 row selected (1.97 seconds)
```

13.13.1.6 Use TiSpark together with Hive

You can use TiSpark together with Hive. Before starting Spark, you need to set the `HADOOP_CONF_DIR` environment variable to your Hadoop configuration folder and copy `hive-site.xml` to the `spark/conf` folder.

```
val tisparkDF = spark.sql("select * from tispark_table").toDF
tisparkDF.write.saveAsTable("hive_table") // save table to hive
spark.sql("select * from hive_table a, tispark_table b where a.col1 = b.col1").show // join table across Hive and Tispark
```

13.13.1.7 Batch write DataFrames into TiDB using TiSpark

Starting from v2.3, TiSpark natively supports batch writing DataFrames into TiDB clusters. This writing mode is implemented through the two-phase commit protocol of TiKV.

Compared with the writing through Spark + JDBC, the TiSpark batch writing has the following advantages:

Aspects to compare	TiSpark batch writes	Spark + JDBC writes
Atomicity	The DataFrames either are all written successfully or all fail to write.	If the Spark task fails and exits during the writing process, a part of the data might be written successfully.
Isolation	During the writing process, the data being written is invisible to other transactions.	During the writing process, some successfully written data is visible to other transactions.

Aspects to compare	TiSpark batch writes	Spark + JDBC writes
Error recovery	If the batch write fails, you only need to re-run Spark.	An application is required to achieve idempotence. For example, if the batch write fails, you need to clean up the part of the successfully written data and re-run Spark. You need to set `spark.task.maxFailures=1` to prevent data duplication caused by task retry.
Speed	Data is directly written into TiKV, which is faster.	Data is written to TiKV through TiDB, which affects the speed.

The following example shows how to batch write data using TiSpark via the scala API:

```
// select data to write
val df = spark.sql("select * from tpch.ORDERS")

// write data to tidb
df.write.
  format("tidb").
  option("tidb.addr", "127.0.0.1").
  option("tidb.port", "4000").
  option("tidb.user", "root").
  option("tidb.password", "").
  option("database", "tpch").
  option("table", "target_orders").
```

```
  mode("append").
  save()
```

If the amount of data to write is large and the writing time exceeds ten minutes, you need to ensure that the GC time is longer than the writing time.

```
UPDATE mysql.tidb SET VARIABLE_VALUE="6h" WHERE VARIABLE_NAME="
    ↪ tikv_gc_life_time";
```

Refer to this document for details.

13.13.1.8 Load Spark Dataframe into TiDB using JDBC

In addition to using TiSpark to batch write DataFrames into the TiDB cluster, you can also use Spark's native JDBC support for the data writing:

```
import org.apache.spark.sql.execution.datasources.jdbc.JDBCOptions

val customer = spark.sql("select * from customer limit 100000")
// You might repartition the source to make it balance across nodes
// and increase the concurrency.
val df = customer.repartition(32)
df.write
.mode(saveMode = "append")
.format("jdbc")
.option("driver", "com.mysql.jdbc.Driver")
 // Replace the host and port with that of your own and be sure to use the
     ↪  rewrite batch
.option("url", "jdbc:mysql://127.0.0.1:4000/test?rewriteBatchedStatements=
    ↪ true")
.option("useSSL", "false")
// As tested, 150 is good practice
.option(JDBCOptions.JDBC_BATCH_INSERT_SIZE, 150)
.option("dbtable", s"cust_test_select") // database name and table name
    ↪ here
.option("isolationLevel", "NONE") // recommended to set isolationLevel to
    ↪ NONE if you have a large DF to load.
.option("user", "root") // TiDB user here
.save()
```

It is recommended to set `isolationLevel` to `NONE` to avoid large single transactions which might potentially lead to TiDB OOM.

Note:

When you use JDBC, the default value of `isolationLevel` is `READ_UNCOMMITTED`, which causes the error of unsupported isolation level transactions. It is recommended to set the value of `isolationLevel` to `NONE`.

13.13.1.9 Statistics information

TiSpark uses TiDB statistic information for the following items:

1. Determining which index to ues in your query plan with the estimated lowest cost.
2. Small table broadcasting, which enables efficient broadcast join.

If you would like TiSpark to use statistic information, first you need to make sure that concerning tables have already been analyzed. Read more about how to analyze tables.

Starting from TiSpark 2.0, statistics information is default to auto load.

13.13.1.10 Security

If you are using TiSpark v2.5.0 or a later version, you can authenticate and authorize TiSpark users by using TiDB.

The authentication and authorization feature is disabled by default. To enable it, add the following configurations to the Spark configuration file `spark-defaults.conf`.

```
// Enable authentication and authorization
spark.sql.auth.enable true

// Configure TiDB information
spark.sql.tidb.addr $your_tidb_server_address
spark.sql.tidb.port $your_tidb_server_port
spark.sql.tidb.user $your_tidb_server_user
spark.sql.tidb.password $your_tidb_server_password
```

For more information, see Authorization and authentication through TiDB server.

Note:

After enabling the authentication and authorization feature, TiSpark Spark SQL can only use TiDB as the data source, so switching to other data sources (such as Hive) makes tables invisible.

13.13.1.11 TiSpark FAQ

Q: What are the pros/cons of independent deployment as opposed to a shared resource with an existing Spark / Hadoop cluster?

A: You can use the existing Spark cluster without a separate deployment, but if the existing cluster is busy, TiSpark will not be able to achieve the desired speed.

Q: Can I mix Spark with TiKV?

A: If TiDB and TiKV are overloaded and run critical online tasks, consider deploying TiSpark separately. You also need to consider using different NICs to ensure that OLTP's network resources are not compromised and affect online business. If the online business requirements are not high or the loading is not large enough, you can consider mixing TiSpark with TiKV deployment.

Q: What can I do if `warning: WARN ObjectStore:568 - Failed to get database` is returned when executing SQL statements using TiSpark?

A: You can ignore this warning. It occurs because Spark tries to load two nonexistent databases (`default` and `global_temp`) in its catalog. If you want to mute this warning, modify log4j by adding `log4j.logger.org.apache.hadoop.hive.metastore.` ↪ `ObjectStore=ERROR` to the `log4j` file in `tispark/conf`. You can add the parameter to the `log4j` file of the `config` under Spark. If the suffix is `template`, you can use the `mv` command to change it to `properties`.

Q: What can I do if `java.sql.BatchUpdateException: Data Truncated` is returned when executing SQL statements using TiSpark?

A: This error occurs because the length of the data written exceeds the length of the data type defined by the database. You can check the field length and adjust it accordingly.

Q: Does TiSpark read Hive metadata by default?

A: By default, TiSpark searches for the Hive database by reading the Hive metadata in hive-site. If the search task fails, it searches for the TiDB database instead, by reading the TiDB metadata.

If you do not need this default behavior, do not configure the Hive metadata in hive-site.

Q: What can I do if `Error: java.io.InvalidClassException: com.pingcap.tikv.` ↪ `region.TiRegion; local class incompatible: stream classdesc serialVersionUID` ↪ `...` is returned when TiSpark is executing a Spark task?

A: The error message shows a `serialVersionUID` conflict, which occurs because you have used `class` and `TiRegion` of different versions. Because `TiRegion` only exists in TiSpark, multiple versions of TiSpark packages might be used. To fix this error, you need to make sure the version of TiSpark dependency is consistent among all nodes in the cluster.

13.14 sync-diff-inspector

13.14.1 sync-diff-inspector User Guide

sync-diff-inspector is a tool used to compare data stored in the databases with the MySQL protocol. For example, it can compare the data in MySQL with that in TiDB, the data in MySQL with that in MySQL, or the data in TiDB with that in TiDB. In addition, you can also use this tool to repair data in the scenario where a small amount of data is inconsistent.

This guide introduces the key features of sync-diff-inspector and describes how to configure and use this tool. To download sync-diff-inspector, use one of the following methods:

- Binary package. The sync-diff-inspector binary package is included in the TiDB Toolkit. To download the TiDB Toolkit, see Download TiDB Tools.
- Docker image. Execute the following command to download:

```
docker pull pingcap/tidb-enterprise-tools:nightly
```

13.14.1.1 Key features

- Compare the table schema and data
- Generate the SQL statements used to repair data if the data inconsistency exists
- Support data check for tables with different schema or table names
- Support data check in the sharding scenario
- Support data check for TiDB upstream-downstream clusters
- Support data check in the DM replication scenario

13.14.1.2 Restrictions of sync-diff-inspector

- Online check is not supported for data migration between MySQL and TiDB. Ensure that no data is written into the upstream-downstream checklist, and that data in a certain range is not changed. You can check data in this range by setting `range`.
- Data of the `JSON` type is not supported. When you perform a data check, you need to set `ignore-columns` to skip checking this type of data.
- In TiDB and MySQL, `FLOAT`, `DOUBLE` and other floating-point types are implemented differently. `FLOAT` and `DOUBLE` respectively take 6 and 15 significant digits for calculating checksum. If you do not want to use this feature, set `ignore-columns` to skip checking these columns.
- Support checking tables that do not contain the primary key or the unique index. However, if data is inconsistent, the generated SQL statements might not be able to repair the data correctly.

13.14.1.3 Database privileges for sync-diff-inspector

To access table schemas and query data, sync-diff-inspector requires specific database privileges. Grant the following privileges on both the upstream and downstream databases:

- `SELECT`: required to compare data.
- `RELOAD`: required to view table schemas.

> **Note**:
>
> - **DO NOT** grant the `SHOW DATABASES` privilege on all databases (`*.*` ↪). Otherwise, sync-diff-inspector will attempt to access inaccessible databases, which causes errors.
> - For MySQL data sources, ensure that the `skip_show_database` system variable is set to `OFF`. If this variable is set to `ON`, the check might fail.

13.14.1.4 Configuration file description

The configuration of sync-diff-inspector consists of the following parts:

- `Global config`: General configurations, such as number of threads to check, whether to export SQL statement to fix inconsistent tables, and whether to campare the data.
- `Databases config`: Configures the instances of the upstream and downstream databases.
- `Routes`: Rules for upstream multiple schema names to match downstream single schema names **(optional)**.
- `Task config`: Configures the tables for checking. If some tables have a certain mapping relationship between the upstream and downstream databases or have some special requirements, you must configure these tables.
- `Table config`: Special configurations for specific tables, such as specified ranges and columns to be ignored **(optional)**.

Below is the description of a complete configuration file:

- Note: configurations with `s` after their name can have multiple values, so you need to use square brackets `[]` to contain the configuration values.

```
### Diff Configuration.

########################### Global config ###########################
```

```
### The number of goroutines created to check data. The number of
    ↪ connections between sync-diff-inspector and upstream/downstream
    ↪ databases is slightly greater than this value.
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables.
export-fix-sql = true

### Only compares the table structure instead of the data.
check-struct-only = false

############################ Datasource config ############################
[data-sources]
[data-sources.mysql1] # mysql1 is the only custom ID for the database
    ↪ instance. It is used for the following `task.source-instances/task.
    ↪ target-instance` configuration.
    host = "127.0.0.1"
    port = 3306
    user = "root"
    password = ""

    # (optional) Use mapping rules to match multiple upstream sharded tables
        ↪ . Rule1 and rule2 are configured in the following Routes section.
    route-rules = ["rule1", "rule2"]

[data-sources.tidb0]
    host = "127.0.0.1"
    port = 4000
    user = "root"
    password = ""
    # (optional) Uses the snapshot feature. If enabled, historical data is
        ↪ used for comparison
    # snapshot = "386902609362944000"

############################## Routes ##############################
### To compare the data of a large number of tables with different schema
    ↪ names or table names, or check the data of multiple upstream sharded
    ↪ tables and downstream table family, use the table-rule to configure
    ↪ the mapping relationship. You can configure the mapping rule only for
    ↪  the schema or table. Also, you can configure the mapping rules for
    ↪ both the schema and the table.
[routes]
[routes.rule1] # rule1 is the only custom ID for the configuration. It is
    ↪ used for the above `data-sources.route-rules` configuration.
schema-pattern = "test_*"  # Matches the schema name of the data source.
```

```
    ↪ Supports the wildcards "*" and "?"
table-pattern = "t_*"      # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test"     # The name of the schema in the target database
target-table = "t"         # The name of the target table
[routes.rule2]
schema-pattern = "test2_*"  # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "t2_*"      # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test2"     # The name of the schema in the target database
target-table = "t2"         # The name of the target table

########################### task config ##########################
### Configures the tables of the target database that need to be compared.
[task]
    # output-dir saves the following information:
    # 1 sql: The SQL file to fix tables that is generated after error is
        ↪ detected. One chunk corresponds to one SQL file.
    # 2 log: sync-diff.log
    # 3 summary: summary.txt
    # 4 checkpoint: a dir
    output-dir = "./output"
    # The upstream database. The value is the unique ID declared by data-
        ↪ sources.
    source-instances = ["mysql1"]
    # The downstream database. The value is the unique ID declared by data-
        ↪ sources.
    target-instance = "tidb0"
    # The tables of downstream databases to be compared. Each table needs to
        ↪  contain the schema name and the table name, separated by '.'
    # Use "?" to match any character and "*" to match characters of any
        ↪ length.
    # For detailed match rules, refer to golang regexp pkg: https://github.
        ↪ com/google/re2/wiki/Syntax.
    target-check-tables = ["schema*.table*", "!c.*", "test2.t2"]
    # (optional) Extra configurations for some tables, Config1 is defined in
        ↪  the following table config example.
    target-configs = ["config1"]

########################### Table config ##########################
### Special configurations for specific tables. The tables to be configured
    ↪ must be in `task.target-check-tables`.
[table-configs.config1] # config1 is the only custom ID for this
    ↪ configuration. It is used for the above `task.target-configs`
```

```
    ↪ configuration.
### The name of the target table, you can use regular expressions to match
    ↪ multiple tables, but one table is not allowed to be matched by
    ↪ multiple special configurations at the same time.
target-tables = ["schema*.test*", "test2.t2"]
### (optional) Specifies the range of the data to be checked
### It needs to comply with the syntax of the WHERE clause in SQL.
range = "age > 10 AND age < 20"
### (optional) Specifies the column used to divide data into chunks. If you
    ↪ do not configure it,
### sync-diff-inspector chooses an appropriate column (primary key, unique
    ↪ key, or a field with index).
index-fields = ["col1","col2"]
### (optional) Ignores checking some columns such as some types (json, bit,
    ↪ blob, etc.)
### that sync-diff-inspector does not currently support.
### The floating-point data type behaves differently in TiDB and MySQL. You
    ↪ can use
### `ignore-columns` to skip checking these columns.
ignore-columns = ["",""]
### (optional) Specifies the size of the chunk for dividing the table. If
    ↪ not specified, this configuration can be deleted or be set as 0.
chunk-size = 0
### (optional) Specifies the "collation" for the table. If not specified,
    ↪ this configuration can be deleted or be set as an empty string.
collation = ""
```

13.14.1.5 Run sync-diff-inspector

Run the following command:

```
./sync_diff_inspector --config=./config.toml
```

This command outputs a check report `summary.txt` in the `output-dir` of `config.toml` and the log `sync_diff.log`. In the `output-dir`, a folder named by the hash value of the `config. toml` file is also generated. This folder includes the checkpoint node information of breakpoints and the SQL file generated when the data is inconsistent.

13.14.1.5.1 Progress information

sync-diff-inspector sends progress information to `stdout` when running. Progress information includes the comparison results of table structures, comparison results of table data and the progress bar.

Note:

To ensure the display effect, keep the display window width above 80 characters.

```
A total of 2 tables need to be compared

Comparing the table structure of ``sbtest`.`sbtest96`` ... equivalent
Comparing the table structure of ``sbtest`.`sbtest99`` ... equivalent
Comparing the table data of ``sbtest`.`sbtest96`` ... failure
Comparing the table data of ``sbtest`.`sbtest99`` ...
--------------------------------------------------------------------------------
   ↪
Progress [============================================================>--] 98%
   ↪  193/200
```

```
A total of 2 tables need to be compared

Comparing the table structure of ``sbtest`.`sbtest96`` ... equivalent
Comparing the table structure of ``sbtest`.`sbtest99`` ... equivalent
Comparing the table data of ``sbtest`.`sbtest96`` ... failure
Comparing the table data of ``sbtest`.`sbtest99`` ... failure
--------------------------------------------------------------------------------
   ↪
Progress [============================================================>]
   ↪ 100% 0/0
The data of `sbtest`.`sbtest99` is not equal
The data of `sbtest`.`sbtest96` is not equal

The rest of tables are all equal.
The patch file has been generated in
       'output/fix-on-tidb2/'
You can view the comparision details through 'output/sync_diff.log'
```

13.14.1.5.2 Output file

The directory structure of the output file is as follows:

```
output/
|-- checkpoint # Saves the breakpoint information
| |-- bbfec8cc8d1f58a5800e63aa73e5 # Config hash. The placeholder file which
   ↪  identifies the configuration file corresponding to the output
   ↪ directory (output/)
```

```
  |-- DO_NOT_EDIT_THIS_DIR
   -- sync_diff_checkpoints.pb # The breakpoint information
|
|-- fix-on-target # Saves SQL files to fix data inconsistency
| |-- xxx.sql
| |-- xxx.sql
|  -- xxx.sql
|
|-- summary.txt # Saves the summary of the check results
 -- sync_diff.log # Saves the output log informatiion when sync-diff-
    ↪ inspector is running
```

13.14.1.5.3 Log

The log of sync-diff-inspector is saved in `${output}/sync_diff.log`, among which `${output}` is the value of `output-dir` in the `config.toml` file.

13.14.1.5.4 Progress

The running sync-diff-inspector periodically (every 10 seconds) prints the progress in checkpoint, which is located at `${output}/checkpoint/sync_diff_checkpoints.pb`, among which `${output}` is the value of `output-dir` in the `config.toml` file.

13.14.1.5.5 Result

After the check is finished, sync-diff-inspector outputs a report. It is located at `${output}/summary.txt`, and `${output}` is the value of `output-dir` in the `config.toml` file.

```
+--------------------+--------------------+---------------+---------+-----------+
    ↪
|       TABLE        | STRUCTURE EQUALITY | DATA DIFF ROWS | UPCOUNT |
    ↪ DOWNCOUNT |
+--------------------+--------------------+---------------+---------+-----------+
    ↪
| `sbtest`.`sbtest99` | true           | +97/-97       |  999999 |   999999 |
| `sbtest`.`sbtest96` | true           | +0/-101       |  999999 |  1000100 |
+--------------------+--------------------+---------------+---------+-----------+
    ↪
Time Cost: 16.75370462s
Average Speed: 113.277149MB/s
```

- TABLE: The corresponding database and table names
- STRUCTURE EQUALITY: Checks whether the table structure is the same

- DATA DIFF ROWS: `rowAdd` / `rowDelete`. Indicates the number of rows that need to be added/deleted to fix the table
- `UPCOUNT`: The number of rows in this table in the upstream data source
- `DOWNCOUNT`: The number of rows in this table in the downstream data source

13.14.1.5.6 SQL statements to fix inconsistent data

If different rows exist during the data checking process, the SQL statements will be generated to fix them. If the data inconsistency exists in a chunk, a SQL file named by `chunk.Index` will be generated. The SQL file is located at `${output}/fix-on-${instance}`, and `${instance}` is the value of `task.target-instance` in the `config.toml` file.

A SQL file contains the tale to which the chunk belong and the range information. For the SQL files, you should consider the following three situations:

- If the rows in the downstream database are missing, REPLACE statements will be applied
- If the rows in the downstream database are redundant, DELETE statements will be applied
- If some data of the rows in the downstream database is inconsistent, REPLACE statements will be applied and inconsistent columns will be marked with annotation in the SQL file

```
-- table: sbtest.sbtest99
-- range in sequence: (3690708) < (id) <= (3720581)
/*
  DIFF COLUMNS   `K`                  `C`                                  `PAD`

    ↪
  source data   2501808   'hello'                          'world'

    ↪
  target data   5003616   '0709824117-9809973320-4456050422'
      ↪ '1714066100-7057807621-1425865505'

    ↪
*/
REPLACE INTO `sbtest`.`sbtest99`(`id`,`k`,`c`,`pad`) VALUES
    ↪ (3700000,2501808,'hello','world');
```

13.14.1.6 Note

- sync-diff-inspector consumes a certain amount of server resources when checking data. Avoid using sync-diff-inspector to check data during peak business hours.
- Before comparing the data in MySQL with that in TiDB, pay attention to the collation configuration of the tables. If the primary key or unique key is the `varchar` type and the collation configuration in MySQL differs from that in TiDB, the final check result might be incorrect because of the collation issue. You need to add collation to the sync-diff-inspector configuration file.
- sync-diff-inspector divides data into chunks first according to TiDB statistics and you need to guarantee the accuracy of the statistics. You can manually run the `analyze table {table_name}` command when the TiDB server's *workload is light*.
- Pay special attention to `table-rules`. If you configure `schema-pattern="test1"`, `table-pattern = "t_1"`, `target-schema="test2"` and `target-table = "t_2"`, the `test1.t_1` schema in the source database and the `test2.t_2` schema in the target database are compared. Sharding is enabled by default in sync-diff-inspector, so if the source database has a `test2.t_2` table, the `test1.t_1` table and `test2.t_2` table in the source database serving as sharding are compared with the `test2.t_2` table in the target database.
- The generated SQL file is only used as a reference for repairing data, and you need to confirm it before executing these SQL statements to repair data.

13.14.2 Data Check for Tables with Different Schema or Table Names

When using replication tools such as TiDB Data Migration, you can set `route-rules` to replicate data to a specified table in the downstream. sync-diff-inspector enables you to verify tables with different schema names or table names by setting `rules`.

The following is a simple configuration example. To learn the complete configuration, refer to Sync-diff-inspector User Guide.

```
########################## Datasource config ##########################
[data-sources.mysql1]
    host = "127.0.0.1"
    port = 3306
    user = "root"
    password = ""
    route-rules = ["rule1"]

[data-sources.tidb0]
    host = "127.0.0.1"
    port = 4000
    user = "root"
    password = ""
############################## Routes ##############################
[routes.rule1]
schema-pattern = "test_1"  # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
```

```
table-pattern = "t_1"      # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test_2"   # The name of the schema in the target database
target-table = "t_2"       # The name of the target table
```

This configuration can be used to check `test_2.t_2` in the downstream and `test_1.t_1` in the `mysql1` instance.

To check a large number of tables with different schema names or table names, you can simplify the configuration by setting the mapping relationship by using `rules`. You can configure the mapping relationship of either schema or table, or of both. For example, all the tables in the upstream `test_1` database are replicated to the downstream `test_2` database, which can be checked through the following configuration:

```
########################## Datasource config ##########################
[data-sources.mysql1]
    host = "127.0.0.1"
    port = 3306
    user = "root"
    password = ""
    route-rules = ["rule1"]

[data-sources.tidb0]
    host = "127.0.0.1"
    port = 4000
    user = "root"
    password = ""
############################## Routes ##############################
[routes.rule1]
schema-pattern = "test_1"  # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "*"        # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test_2"   # The name of the schema in the target database
target-table = "t_2"       # The name of the target table
```

13.14.2.1 Note

If `test_2.t_2` exists in the upstream database, the downstream database also compares this table.

13.14.3 Data Check in the Sharding Scenario

sync-diff-inspector supports data check in the sharding scenario. Assume that you use the TiDB Data Migration tool to replicate data from multiple MySQL instances into TiDB, you can use sync-diff-inspector to check upstream and downstream data.

For scenarios where the number of upstream sharded tables is small and the naming rules of sharded tables do not have a pattern as shown below, you can use `Datasource config` to configure `table-0`, set corresponding `rules` and configure the tables that have the mapping relationship between the upstream and downstream databases. This configuration method requires setting all sharded tables.

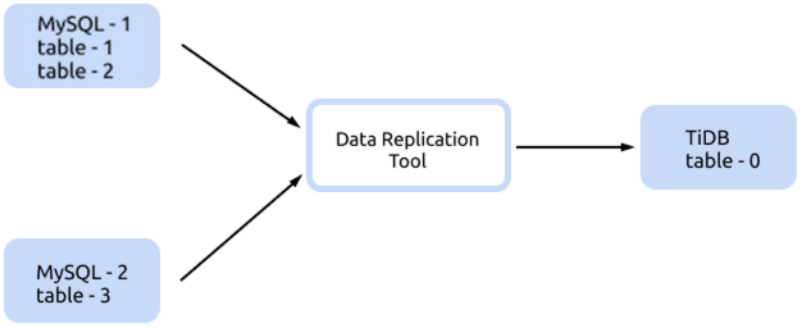


Figure 307: shard-table-replica-1

Below is a complete example of the sync-diff-inspector configuration.

```
### Diff Configuration.

############################# Global config ############################

### The number of goroutines created to check data. The number of
    ↪ connections between upstream and downstream databases are slightly
    ↪ greater than this value
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables
export-fix-sql = true

### Only compares the table structure instead of the data
check-struct-only = false

############################# Datasource config ############################
[data-sources.mysql1]
    host = "127.0.0.1"
    port = 3306
    user = "root"
    password = ""

    route-rules = ["rule1"]
```

```
[data-sources.mysql2]
    host = "127.0.0.1"
    port = 3306
    user = "root"
    password = ""

    route-rules = ["rule2"]

[data-sources.tidb0]
    host = "127.0.0.1"
    port = 4000
    user = "root"
    password = ""

############################## Routes ############################
[routes.rule1]
schema-pattern = "test"    # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "table-[1-2]" # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test"     # The name of the schema in the target database
target-table = "table-0"   # The name of the target table

[routes.rule2]
schema-pattern = "test"  # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "table-3" # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test"    # The name of the schema in the target database
target-table = "table-0"  # The name of the target table

########################### Task config ##########################
[task]
    output-dir = "./output"

    source-instances = ["mysql1", "mysql2"]

    target-instance = "tidb0"

    # The tables of downstream databases to be compared. Each table needs to
        ↪  contain the schema name and the table name, separated by '.'
    target-check-tables = ["test.table-0"]
```

You can use `table-rules` for configuration when there are a large number of upstream sharded tables and the naming rules of all sharded tables have a pattern, as shown below:

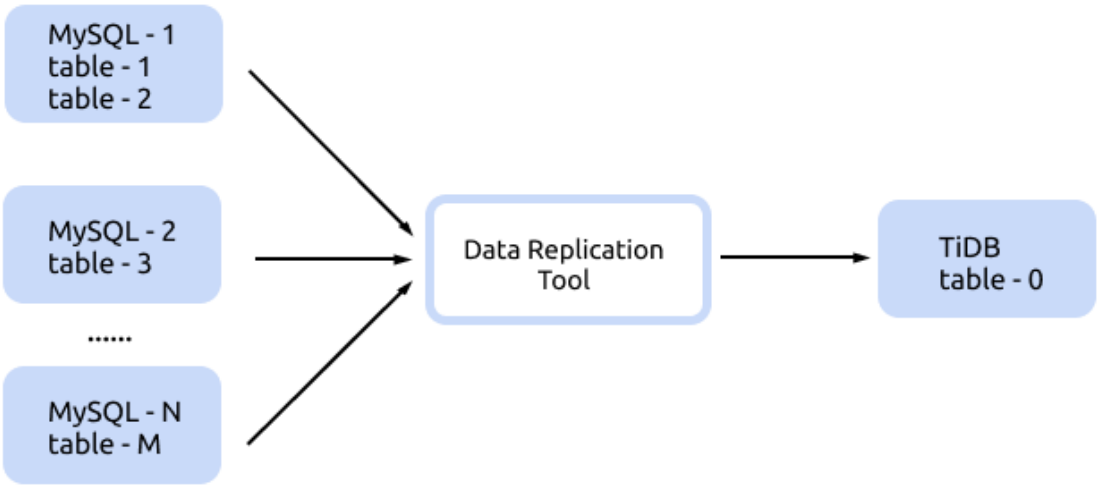


Figure 308: shard-table-replica-2

Below is a complete example of the sync-diff-inspector configuration.

```
### Diff Configuration.
############################ Global config ############################

### The number of goroutines created to check data. The number of
    ↪ connections between upstream and downstream databases are slightly
    ↪ greater than this value.
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables.
export-fix-sql = true

### Only compares the table structure instead of the data.
check-struct-only = false

########################## Datasource config ##########################
[data-sources.mysql1]
    host = "127.0.0.1"
    port = 3306
    user = "root"
    password = ""

[data-sources.mysql2]
    host = "127.0.0.1"
    port = 3306
    user = "root"
```

```
    password = ""

[data-sources.tidb0]
    host = "127.0.0.1"
    port = 4000
    user = "root"
    password = ""

############################## Routes ############################
[routes.rule1]
schema-pattern = "test"  # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "table-*" # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test"   # The name of the schema in the target database
target-table = "table-0" # The name of the target table

########################## Task config ########################
[task]
    output-dir = "./output"
    source-instances = ["mysql1", "mysql2"]

    target-instance = "tidb0"

    # The tables of downstream databases to be compared. Each table needs to
       ↪  contain the schema name and the table name, separated by '.'
    target-check-tables = ["test.table-0"]
```

13.14.3.1 Note

If `test.table-0` exists in the upstream database, the downstream database also compares this table.

13.14.4 Data Check for TiDB Upstream and Downstream Clusters

You can use TiDB Binlog to build upstream and downstream clusters of TiDB. When Drainer replicates data to TiDB, the checkpoint is saved and the TSO mapping relationship between the upstream and downstream is also saved as `ts-map`. To check data between the upstream and downstream, configure `snapshot` in sync-diff-inspector.

13.14.4.1 Step 1: obtain `ts-map`

To obtain `ts-map`, execute the following SQL statement in the downstream TiDB cluster:

```
mysql> select * from tidb_binlog.checkpoint;
+--
    ↪ ------------------+----------------------------------------------------------
    ↪
| clusterID         | checkPoint
    ↪
    ↪ |
+--
    ↪ ------------------+----------------------------------------------------------
    ↪
| 6711243465327639221 | {"commitTS":409622383615541249,"ts-map":{"primary-ts
    ↪ ":409621863377928194,"secondary-ts":409621863377928345}} |
+--
    ↪ ------------------+----------------------------------------------------------
    ↪
```

13.14.4.2 Step 2: configure snapshot

Then configure the snapshot information of the upstream and downstream databases by using the `ts-map` information obtained in Step 1.

Here is a configuration example of the `Datasource config` section:

```
############################ Datasource config ########################
[data-sources.uptidb]
    host = "172.16.0.1"
    port = 4000
    user = "root"
    password = ""
    snapshot = "409621863377928194"

[data-sources.downtidb]
    host = "172.16.0.2"
    port = 4000
    user = "root"
    snapshot = "409621863377928345"
```

Note:

- Set `db-type` of Drainer to `tidb` to ensure that `ts-map` is saved in the checkpoint.

- Modify the Garbage Collection (GC) time of TiKV to ensure that the historical data corresponding to snapshot is not collected by GC during the data check. It is recommended that you modify the GC time to 1 hour and recover the setting after the check.
- In some versions of TiDB Binlog, `master-ts` and `slave-ts` are stored in `ts-map`. `master-ts` is equivalent to `primary-ts` and `slave-ts` is equivalent to `secondary-ts`.
- The above example only shows the section of `Datasource config`. For complete configuration, refer to sync-diff-inspector User Guide.

13.14.5 Data Check in the DM Replication Scenario

When using replication tools such as TiDB Data Migration, you need to check the data consistency before and after the replication process. You can set a specific `task-name` configuration from `DM-master` to perform a data check.

The following is a simple configuration example. To learn the complete configuration, refer to Sync-diff-inspector User Guide.

```
### Diff Configuration.

############################# Global config ############################

### The number of goroutines created to check data. The number of
    ↪ connections between upstream and downstream databases are slightly
    ↪ greater than this value.
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables.
export-fix-sql = true

### Only compares the table structure instead of the data.
check-struct-only = false

### The IP address of dm-master and the format is "http://127.0.0.1:8261".
dm-addr = "http://127.0.0.1:8261"

### Specifies the `task-name` of DM.
dm-task = "test"

############################# Task config ############################
[task]
    output-dir = "./output"
```

```
# The tables of downstream databases to be compared. Each table needs to
    ↪ contain the schema name and the table name, separated by '.'
target-check-tables = ["hb_test.*"]
```

This example is configured in dm-task = "test", which checks all the tables of hb_test schema under the "test" task. It automatically gets the regular matching of the schemas between upstream and downstream databases to verify the data consistency after DM replication.

14 Reference

14.1 Cluster Architecture

14.1.1 TiDB Architecture

Compared with the traditional standalone databases, TiDB has the following advantages:

- Has a distributed architecture with flexible and elastic scalability.
- Fully compatible with the MySQL 5.7 protocol, common features and syntax of MySQL. To migrate your applications to TiDB, you do not need to change a single line of code in many cases.
- Supports high availability with automatic failover when a minority of replicas fail; transparent to applications.
- Supports ACID transactions, suitable for scenarios requiring strong consistency such as bank transfer.

- Provides a rich series of data migration tools for migrating, replicating, or backing up data.

As a distributed database, TiDB is designed to consist of multiple components. These components communicate with each other and form a complete TiDB system. The architecture is as follows:

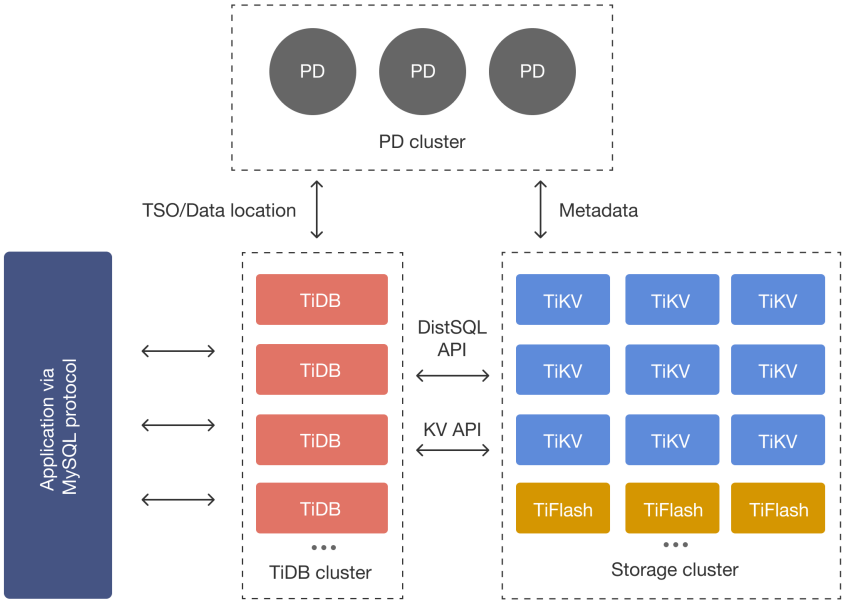


Figure 309: TiDB Architecture

14.1.1.1 TiDB server

The TiDB server is a stateless SQL layer that exposes the connection endpoint of the MySQL protocol to the outside. The TiDB server receives SQL requests, performs SQL parsing and optimization, and ultimately generates a distributed execution plan. It is horizontally scalable and provides the unified interface to the outside through the load balancing components such as Linux Virtual Server (LVS), HAProxy, or F5. It does not store data and is only for computing and SQL analyzing, transmitting actual data read request to TiKV nodes (or TiFlash nodes).

14.1.1.2 Placement Driver (PD) server

The PD server is the metadata managing component of the entire cluster. It stores metadata of real-time data distribution of every single TiKV node and the topology structure of the entire TiDB cluster, provides the TiDB Dashboard management UI, and allocates transaction IDs to distributed transactions. The PD server is "the brain" of the entire TiDB cluster because it not only stores metadata of the cluster, but also sends data scheduling command to specific TiKV nodes according to the data distribution state reported by TiKV nodes in real time. In addition, the PD server consists of three nodes at least and has high availability. It is recommended to deploy an odd number of PD nodes.

14.1.1.3 Storage servers

14.1.1.3.1 TiKV server

The TiKV server is responsible for storing data. TiKV is a distributed transactional key-value storage engine.

Region is the basic unit to store data. Each Region stores the data for a particular Key Range which is a left-closed and right-open interval from StartKey to EndKey.

Multiple Regions exist in each TiKV node. TiKV APIs provide native support to distributed transactions at the key-value pair level and supports the Snapshot Isolation level isolation by default. This is the core of how TiDB supports distributed transactions at the SQL level. After processing SQL statements, the TiDB server converts the SQL execution plan to an actual call to the TiKV API. Therefore, data is stored in TiKV. All the data in TiKV is automatically maintained in multiple replicas (three replicas by default), so TiKV has native high availability and supports automatic failover.

14.1.1.3.2 TiFlash server

The TiFlash server is a special type of storage server. Unlike ordinary TiKV nodes, TiFlash stores data by column, mainly designed to accelerate analytical processing.

14.1.2 TiDB Storage

This document introduces some design ideas and key concepts of TiKV.

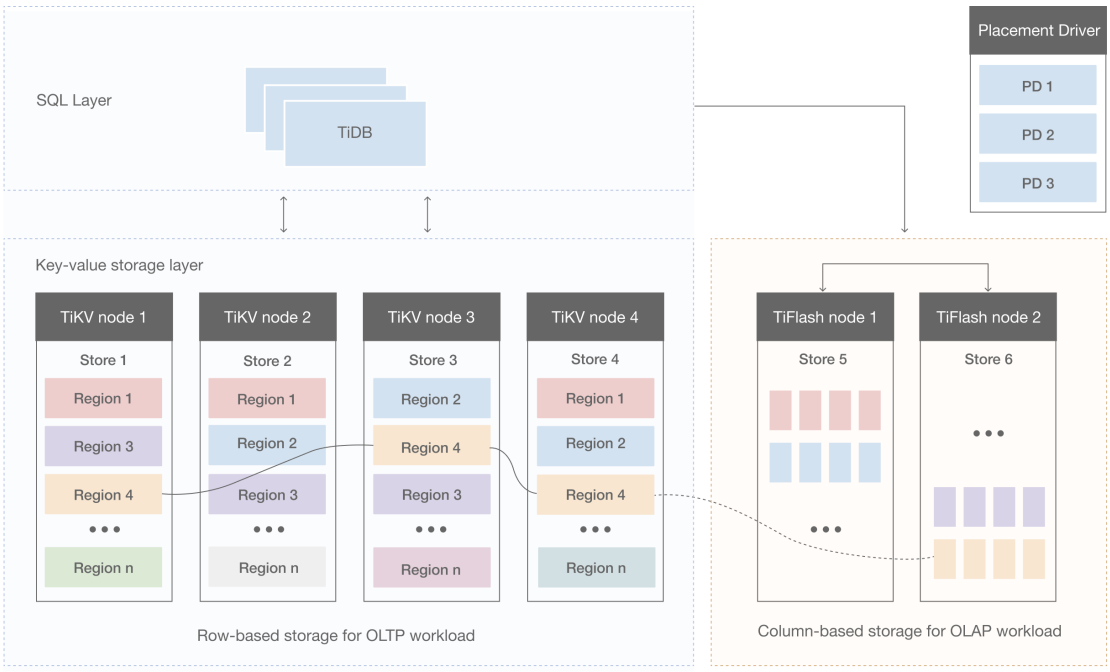


Figure 310: storage-architecture

14.1.2.1 Key-Value pairs

The first thing to decide for a data storage system is the data storage model, that is, in what form the data is saved. TiKV's choice is the Key-Value model and provides an ordered traversal method. There are two key points for TiKV data storage model:

- This is a huge Map (similar to `std::Map` in C++) , which stores Key-Value pairs.
- The Key-Value pair in the Map is ordered according to Keys' binary order, which means you can Seek the position of a particular Key and then call the Next method to get the Key-Value pairs larger than this Key in incremental order.

Note that the KV storage model for TiKV described in this document has nothing to do with tables of SQL. This document does not discuss any concepts related to SQL and only focuses on how to implement a high-performance, high-reliability, distributed Key-Value storage such as TiKV.

14.1.2.2 Local storage (RocksDB)

For any persistent storage engine, data is eventually saved on disk, and TiKV is no exception. TiKV does not write data directly on the disk, but stores data in RocksDB, which is responsible for the data storage. The reason is that it costs a lot to develop a standalone storage engine, especially a high-performance standalone engine that requires careful optimization.

RocksDB is an excellent standalone storage engine open-sourced by Facebook. This engine can meet various requirements of TiKV for a single engine. Here, you can simply consider RocksDB as a single persistent Key-Value Map.

14.1.2.3 Raft protocol

What's more, the implementation of TiKV faces a more difficult thing: to secure data safety in case a single machine fails.

A simple way is to replicate data to multiple machines, so that even if one machine fails, the replicas on other machines are still available. In other words, you need a data replication scheme that is reliable, efficient, and able to handle the situation of a failed replica. All of these are made possible by the Raft algorithm.

Raft is a consensus algorithm. This document only briefly introduces Raft. For more details, you can see In Search of an Understandable Consensus Algorithm. The Raft has several important features:

- Leader election
- Membership changes (such as adding replicas, deleting replicas, and transferring leaders)
- Log replication

TiKV use Raft to perform data replication. Each data change will be recorded as a Raft log. Through Raft log replication, data is safely and reliably replicated to multiple nodes of the Raft group. However, according to Raft protocol, successful writes only need that data is replicated to the majority of nodes.

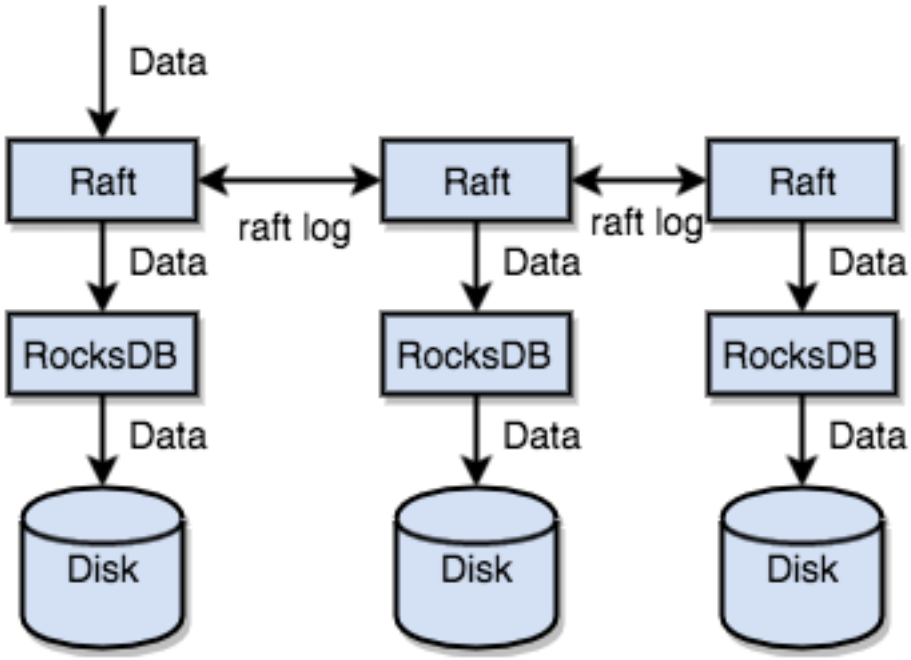


Figure 311: Raft in TiDB

In summary, TiKV can quickly store data on disk via the standalone machine RocksDB, and replicate data to multiple machines via Raft in case of machine failure. Data is written through the interface of Raft instead of to RocksDB. With the implementation of Raft, TiKV becomes a distributed Key-Value storage. Even with a few machine failures, TiKV can automatically complete replicas by virtue of the native Raft protocol, which does not impact the application.

14.1.2.4 Region

To make it easy to understand, let's assume that all data only has one replica. As mentioned earlier, TiKV can be regarded as a large, orderly KV Map, so data is distributed across multiple machines in order to achieve horizontal scalability. For a KV system, there are two typical solutions to distributing data across multiple machines:

- Hash: Create Hash by Key and select the corresponding storage node according to the Hash value.
- Range: Divide ranges by Key, where a segment of serial Key is stored on a node.

TiKV chooses the second solution that divides the whole Key-Value space into a series of consecutive Key segments. Each segment is called a Region. Each Region can be described by `[StartKey, EndKey)`, a left-closed and right-open interval. The default size limit for each Region is 96 MiB and the size can be configured.

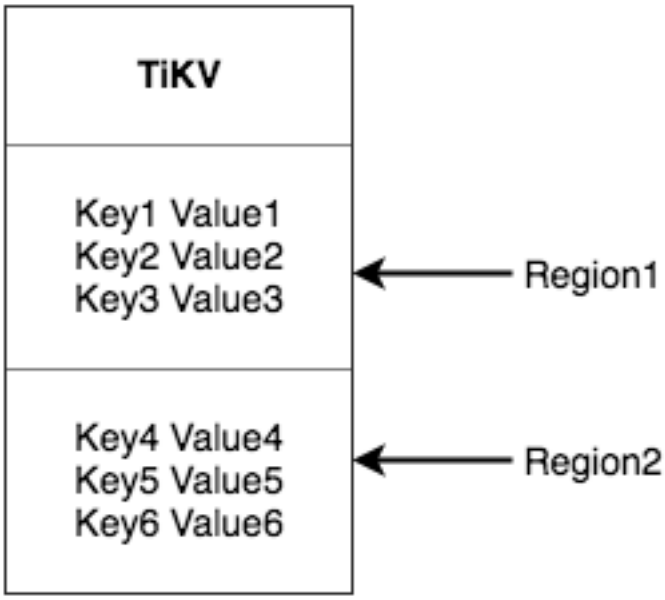


Figure 312: Region in TiDB

Note that the Region here has nothing to do with the table in SQL. In this document, forget about SQL and focus on KV for now. After dividing data into Regions, TiKV will perform two important tasks:

- Distributing data to all nodes in the cluster and use Region as the basic unit. Try its best to ensure that the number of Regions on each node is roughly similar.
- Performing Raft replication and membership management in Region.

These two tasks are very important and will be introduced one by one.

- First, data is divided into many Regions according to Key, and the data for each Region is stored on only one node (ignoring multiple replicas). The TiDB system has a PD component that is responsible for spreading Regions as evenly as possible across all nodes in the cluster. In this way, on one hand, the storage capacity is scaled horizontally (Regions on the other nodes are automatically scheduled to the newly added node); on the other hand, load balancing is achieved (the situation where one node has a lot of data while the others have little will not occur).

  At the same time, in order to ensure that the upper client can access the needed data, there is a component (PD) in the system to record the distribution of Regions on the node, that is, the exact Region of a Key and the node of that Region placed through any Key.

- For the second task, TiKV replicates data in Regions, which means that data in one Region will have multiple replicas with the name "Replica". Multiple Replicas of a Region are stored on different nodes to form a Raft Group, which is kept consistent through the Raft algorithm.

  One of the Replicas serves as the Leader of the Group and other as the Follower. By default, all reads and writes are processed through the Leader, where reads are done and write are replicated to followers. The following diagram shows the whole picture about Region and Raft group.

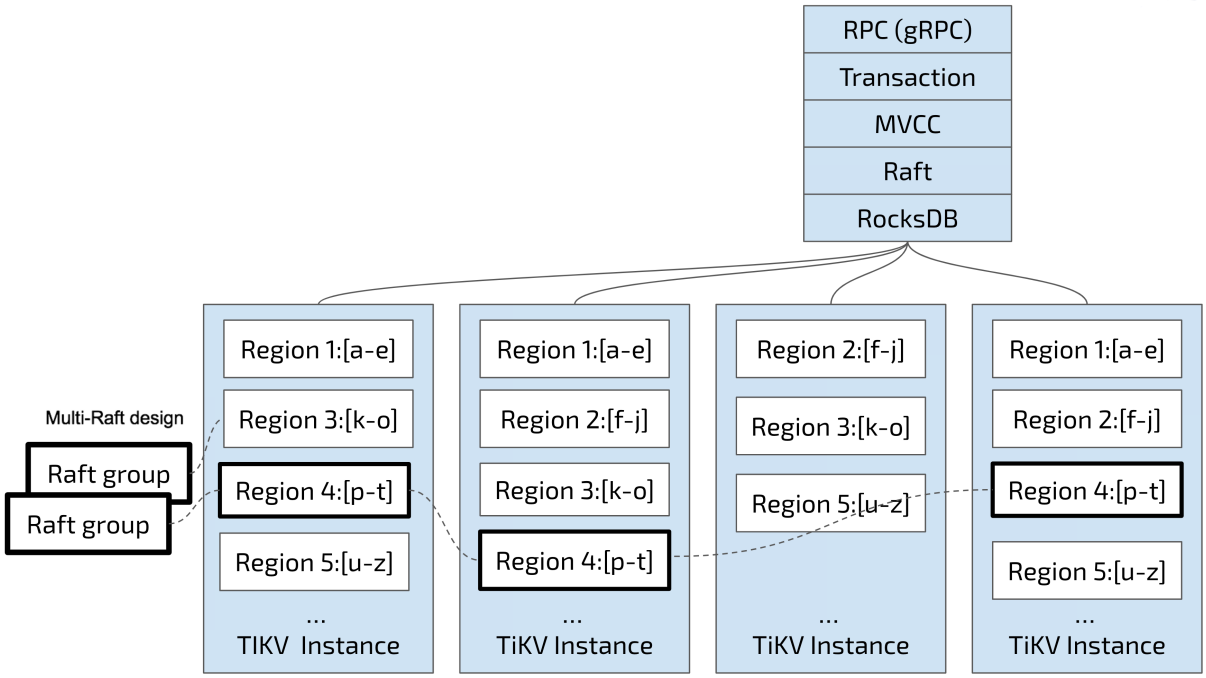


Figure 313: TiDB Storage

As we distribute and replicate data in Regions, we have a distributed Key-Value system that, to some extent, has the capability of disaster recovery. You no longer need to worry about the capacity, or disk failure and data loss.

14.1.2.5 MVCC

Many databases implement multi-version concurrency control (MVCC), and TiKV is no exception. Imagine the situation where two clients modify the value of a Key at the same time. Without MVCC, the data needs to be locked. In a distributed scenario, it might cause performance and deadlock problems. TiKV's MVCC implementation is achieved by appending a version number to Key. In short, without MVCC, TiKV's data layout can be seen as:

```
Key1 -> Value
Key2 -> Value
.....
KeyN -> Value
```

With MVCC, the key array of TiKV is like this:

```
Key1_Version3 -> Value
Key1_Version2 -> Value
Key1_Version1 -> Value
.....
Key2_Version4 -> Value
Key2_Version3 -> Value
Key2_Version2 -> Value
Key2_Version1 -> Value
.....
KeyN_Version2 -> Value
KeyN_Version1 -> Value
.....
```

Note that for multiple versions of the same Key, versions with larger numbers are placed first (see the Key-Value section where Keys are arranged in order), so that when you obtain Value through Key + Version, the Key of MVCC can be constructed with Key and Version, which is `Key_Version`. Then you can directly locate the first position greater than or equal to this `Key_Version` through RocksDB's `SeekPrefix(Key_Version)` API.

14.1.2.6 Distributed ACID transaction

Transaction of TiKV adopts the model used by Google in BigTable: Percolator. TiKV's implementation is inspired by this paper, with a lot of optimizations. See transaction overview for details.

14.1.3 TiDB Computing

Based on the distributed storage provided by TiKV, TiDB builds the computing engine that combines great capability of transactional processing with that of data analysis. This document starts by introducing a data mapping algorithm that maps data from TiDB database tables to (Key, Value) key-value pairs in TiKV, then introduces how TiDB manages metadata, and finally illustrates the architecture of the TiDB SQL layer.

For the storage solution on which the computing layer is dependent, this document only introduces the row-based storage structure of TiKV. For OLAP services, TiDB introduces a column-based storage solution TiFlash as a TiKV extension.

14.1.3.1 Mapping table data to Key-Value

This section describes the scheme for mapping data to (Key, Value) key-value pairs in TiDB. Data to be mapped here includes the following two types:

- Data of each row in the table, hereinafter referred to as table data.
- Data of all indexes in the table, hereinafter referred to as index data.

14.1.3.1.1 Mapping of table data to Key-Value

In a relational database, a table might have many columns. To map the data of each column in a row to a (Key, Value) key-value pair, you need to consider how to construct the Key. First of all, in OLTP scenarios, there are many operations such as adding, deleting, changing, and searching for data on a single or multiple rows, which needs the database to read a row of data quickly. Therefore, each key should have a unique ID (either explicit or implicit) to make it quick to locate. Then, many OLAP queries require a full table scan. If you can encode the keys of all rows in a table into a range, the whole table can be efficiently scanned by range queries.

Based on the considerations above, the mapping of table data to Key-Value in TiDB is designed as follows:

- To ensure that data from the same table is kept together for easy searching, TiDB assigns a table ID to each table represented by `TableID`. Table ID is an integer that is unique throughout the cluster.
- TiDB assigns a row ID, represented by `RowID`, to each row of data in the table. The row ID is also an integer, unique within the table. For row ID, TiDB has made a small optimization: if a table has an integer type primary key, TiDB uses the value of this primary key as the row ID.

Each row of data is encoded as a (Key, Value) key-value pair according to the following rule:

```
Key:   tablePrefix{TableID}_recordPrefixSep{RowID}
Value: [col1, col2, col3, col4]
```

`tablePrefix` and `recordPrefixSep` are both special string constants used to distinguish other data in Key space. The exact values of the string constants are introduced in Summary of mapping relationships.

14.1.3.1.2 Mapping of indexed data to Key-Value

TiDB supports both primary keys and secondary indexes (both unique and non-unique indexes). Similar to the table data mapping scheme, TiDB assigns an index ID to each index of the table represented by `IndexID`.

For primary keys and unique indexes, it is needed to quickly locate the corresponding `RowID` based on the key-value pair, so such a key-value pair is encoded as follows.

```
Key:  tablePrefix{tableID}_indexPrefixSep{indexID}_indexedColumnsValue
Value: RowID
```

For ordinary secondary indexes that do not need to satisfy the uniqueness constraint, a single key might correspond to multiple rows. It needs to query corresponding `RowID` according to the range of keys. Therefore, the key-value pair must be encoded according to the following rule:

```
Key:  tablePrefix{TableID}_indexPrefixSep{IndexID}_indexedColumnsValue_{
    ↪ RowID}
Value: null
```

14.1.3.1.3 Summary of mapping relationships

`tablePrefix`, `recordPrefixSep`, and `indexPrefixSep` in all of the above encoding rules are string constants that are used to distinguish a KV from other data in the Key space, which are defined as follows:

```
tablePrefix   = []byte{'t'}
recordPrefixSep = []byte{'r'}
indexPrefixSep = []byte{'i'}
```

Also note that in the above encoding schemes, no matter table data or index data key encoding scheme, all rows in a table have the same key prefix, and all data of an index also has the same prefix. Data with the same prefixes are thus arranged together in TiKV's key space. Therefore, by carefully designing the encoding scheme of the suffix part to ensure that the pre-encoding and post-encoding comparisons remain the same, the table data or index data can be stored in TiKV in an ordered manner. Using this encoding scheme, all row data in a table is arranged orderly by `RowID` in the TiKV's key space, and the data of a particular index is also arranged sequentially in the key space according to the specific value of the index data (`indexedColumnsValue`).

14.1.3.1.4 Example of Key-Value mapping relationship

This section shows a simple example for you to understand the Key-Value mapping relationship of TiDB. Suppose the following table exists in TiDB.

```
CREATE TABLE User (
    ID int,
    Name varchar(20),
    Role varchar(20),
    Age int,
    PRIMARY KEY (ID),
    KEY idxAge (Age)
);
```

Suppose there are 3 rows of data in the table.

```
1, "TiDB", "SQL Layer", 10
2, "TiKV", "KV Engine", 20
3, "PD", "Manager", 30
```

Each row of data is mapped to a (Key, Value) key-value pair, and the table has an `int` type primary key, so the value of `RowID` is the value of this primary key. Suppose the table's `TableID` is `10`, and then its table data stored on TiKV is:

```
t10_r1 --> ["TiDB", "SQL Layer", 10]
t10_r2 --> ["TiKV", "KV Engine", 20]
t10_r3 --> ["PD", " Manager", 30]
```

In addition to the primary key, the table has a non-unique ordinary secondary index, `idxAge`. Suppose the `IndexID` is `1`, and then its index data stored on TiKV is:

```
t10_i1_10_1 --> null
t10_i1_20_2 --> null
t10_i1_30_3 --> null
```

The above example shows the mapping rule from a relational model to a Key-Value model in TiDB, and the consideration behind this mapping scheme.

14.1.3.2 Metadata management

Each database and table in TiDB has metadata that indicates its definition and various attributes. This information also needs to be persisted, and TiDB stores this information in TiKV as well.

Each database or table is assigned a unique ID. As the unique identifier, when table data is encoded to Key-Value, this ID is encoded in the Key with the `m_` prefix. This constructs a key-value pair with the serialized metadata stored in it.

In addition, TiDB also uses a dedicated (Key, Value) key-value pair to store the latest version number of structure information of all tables. This key-value pair is global, and its version number is increased by `1` each time the state of the DDL operation changes. TiDB stores this key-value pair persistently in the PD server with the key of `/tidb/ddl` ↪ `/global_schema_version`, and Value is the version number value of the `int64` type. Meanwhile, because TiDB applies schema changes online, it keeps a background thread that constantly checks whether the version number of the table structure information stored in the PD server changes. This thread also ensures that the changes of version can be obtained within a certain period of time.

14.1.3.3 SQL layer overview

TiDB's SQL layer, TiDB Server, translates SQL statements into Key-Value operations, forwards the operations to TiKV, the distributed Key-Value storage layer, assembles the results returned by TiKV, and finally returns the query results to the client.

The nodes at this layer are stateless. These nodes themselves do not store data and are completely equivalent.

14.1.3.3.1 SQL computing

The simplest solution to SQL computing is the mapping of table data to Key-Value as described in the previous section, which maps SQL queries to KV queries, acquires the corresponding data through the KV interface, and performs various computations.

For example, to execute the `select count(*)from user where name = "TiDB"` SQL statement, TiDB needs to read all data in the table, then checks whether the `name` field is `TiDB`, and if so, returns this row. The process is as follows:

1. Construct the Key Range: all `RowID` in a table are in `[0, MaxInt64)` range. According to the row data `Key` encoding rule, using `0` and `MaxInt64` can construct a `[StartKey ↪ , EndKey)` range that is left-closed and right-open.
2. Scan Key Range: read the data in TiKV according to the key range constructed above.
3. Filter data: for each row of data read, calculate the `name = "TiDB"` expression. If the result is `true`, return to this row. If not, skip this row.
4. Calculate `Count(*)`: for each row that meets the requirements, add up to the result of `Count(*)`.

The entire process is illustrated as follows:

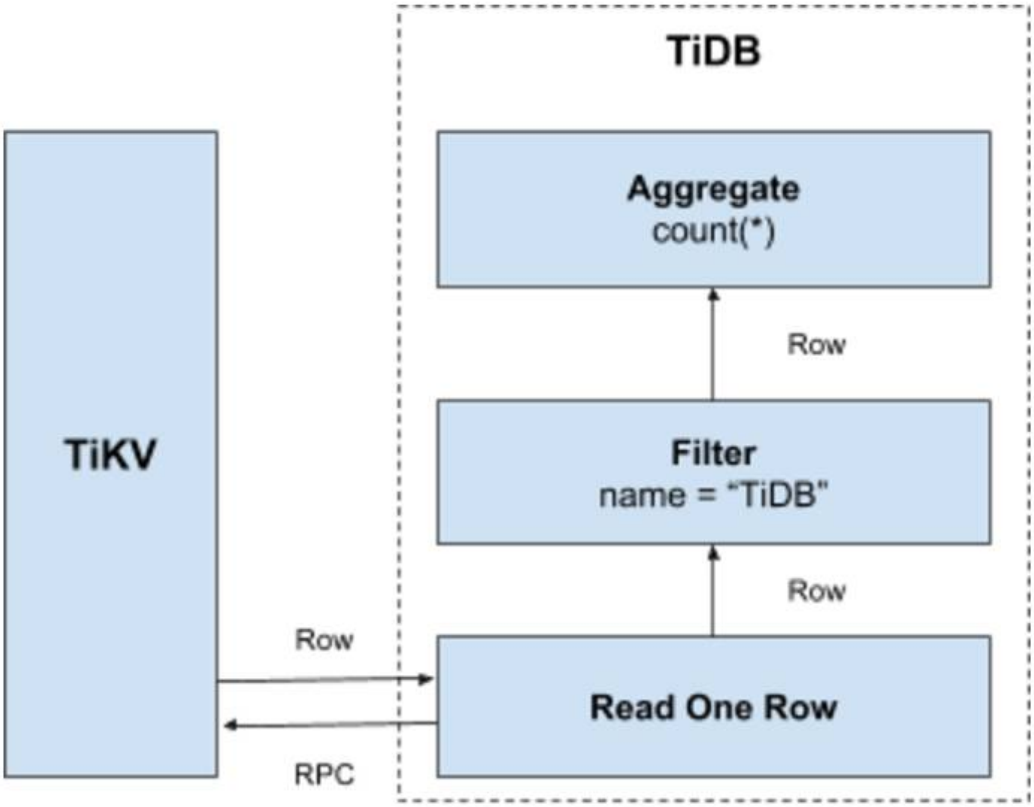


Figure 314: naive sql flow

This solution is intuitive and feasible, but has some obvious problems in a distributed database scenario:

- As the data is being scanned, each row is read from TiKV via a KV operation with at least one RPC overhead, which can be very high if there is a large amount of data to be scanned.
- It is not applicable to all rows. Data that does not meet the conditions does not need to be read.
- From the returned result of this query, only the number of rows that match the requirements is needed, not the value of those rows.

14.1.3.3.2 Distributed SQL operations

To solve the problems above, the computation should be as close to the storage node as possible to avoid a large number of RPC callings. First of all, the SQL predicate condition `name = "TiDB"` should be pushed down to the storage node for computation, so that only valid rows are returned, which avoids meaningless network transfers. Then, the aggregation

function `Count(*)` can also be pushed down to the storage nodes for pre-aggregation, and each node only has to return a result of `Count(*)`. The SQL layer will sum up the `Count(*)` results returned by each node.

The following image shows how data returns layer by layer:

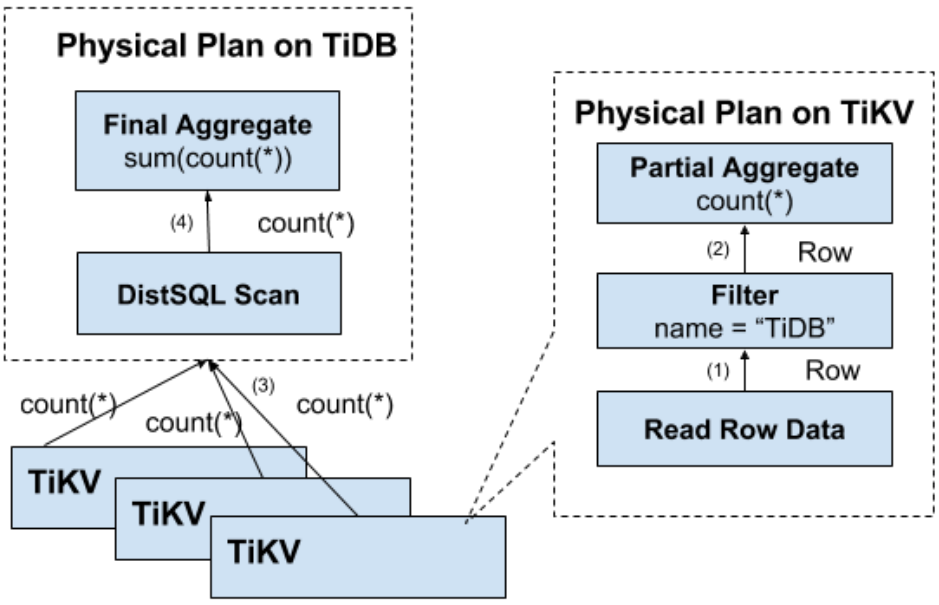


Figure 315: dist sql flow

14.1.3.3.3 Architecture of SQL layer

The previous sections introduce some functions of the SQL layer and I hope you have a basic understanding of how SQL statements are handled. In fact, TiDB's SQL layer is much more complicated, with many modules and layers. The following diagram lists the important modules and calling relationships:

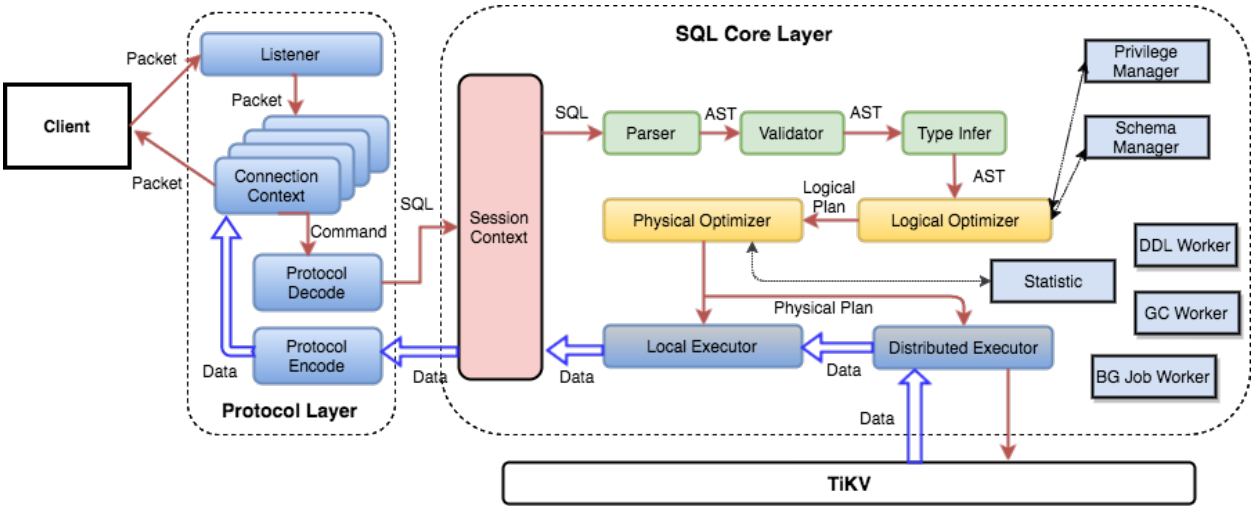


Figure 316: tidb sql layer

The user's SQL request is sent to TiDB Server either directly or via `Load Balancer`. TiDB Server will parse `MySQL Protocol Packet`, get the content of requests, parse the SQL request syntactically and semantically, develop and optimize query plans, execute a query plan, get and process the data. All data is stored in the TiKV cluster, so in this process, TiDB Server needs to interact with TiKV and get the data. Finally, TiDB Server needs to return the query results to the user.

14.1.4 TiDB Scheduling

The Placement Driver (PD) works as the manager in a TiDB cluster, and it also schedules Regions in the cluster. This article introduces the design and core concepts of the PD scheduling component.

14.1.4.1 Scheduling situations

TiKV is the distributed key-value storage engine used by TiDB. In TiKV, data is organized as Regions, which are replicated on several stores. In all replicas, a leader is responsible for reading and writing, and followers are responsible for replicating Raft logs from the leader.

Now consider about the following situations:

- To utilize storage space in a high-efficient way, multiple Replicas of the same Region need to be properly distributed on different nodes according to the Region size;
- For multiple data center topologies, one data center failure only causes one replica to fail for all Regions;
- When a new TiKV store is added, data can be rebalanced to it;
- When a TiKV store fails, PD needs to consider:

    - Recovery time of the failed store.
        - If it's short (for example, the service is restarted), whether scheduling is necessary or not.
        - If it's long (for example, disk fault and data is lost), how to do scheduling.
    - Replicas of all Regions.
        - If the number of replicas is not enough for some Regions, PD needs to complete them.
        - If the number of replicas is more than expected (for example, the failed store re-joins into the cluster after recovery), PD needs to delete them.
- Read/Write operations are performed on leaders, which cannot be distributed only on a few individual stores;
- Not all Regions are hot, so loads of all TiKV stores need to be balanced;
- When Regions are in balancing, data transferring utilizes much network/disk traffic and CPU time, which can influence online services.

These situations can occur at the same time, which makes it harder to resolve. Also, the whole system is changing dynamically, so a scheduler is needed to collect all information about the cluster, and then adjust the cluster. So, PD is introduced into the TiDB cluster.

14.1.4.2 Scheduling requirements

The above situations can be classified into two types:

1. A distributed and highly available storage system must meet the following requirements:

    - The right number of replicas.
    - Replicas need to be distributed on different machines according to different topologies.
    - The cluster can perform automatic disaster recovery from TiKV peers' failure.

2. A good distributed system needs to have the following optimizations:

    - All Region leaders are distributed evenly on stores;
    - Storage capacity of all TiKV peers are balanced;
    - Hot spots are balanced;
    - Speed of load balancing for the Regions needs to be limited to ensure that online services are stable;
    - Maintainers are able to take peers online/offline manually.

After the first type of requirements is satisfied, the system will be failure tolerable. After the second type of requirements is satisfied, resources will be utilized more efficiently and the system will have better scalability.

To achieve the goals, PD needs to collect information firstly, such as state of peers, information about Raft groups and the statistics of accessing the peers. Then we need to specify some strategies for PD, so that PD can make scheduling plans from these information and strategies. Finally, PD distributes some operators to TiKV peers to complete scheduling plans.

14.1.4.3 Basic scheduling operators

All scheduling plans contain three basic operators:

- Add a new replica
- Remove a replica
- Transfer a Region leader between replicas in a Raft group

They are implemented by the Raft commands `AddReplica`, `RemoveReplica`, and `TransferLeader`.

14.1.4.4 Information collection

Scheduling is based on information collection. In short, the PD scheduling component needs to know the states of all TiKV peers and all Regions. TiKV peers report the following information to PD:

- State information reported by each TiKV peer:

  Each TiKV peer sends heartbeats to PD periodically. PD not only checks whether the store is alive, but also collects `StoreState` in the heartbeat message. `StoreState` includes:

  - Total disk space
  - Available disk space
  - The number of Regions
  - Data read/write speed
  - The number of snapshots that are sent/received (The data might be replicated between replicas through snapshots)
  - Whether the store is overloaded
  - Labels (See Perception of Topology)

  You can use PD control to check the status of a TiKV store, which can be Up, Disconnect, Offline, Down, or Tombstone. The following is a description of all statuses and their relationship.

  - **Up**: The TiKV store is in service.
  - **Disconnect**: Heartbeat messages between the PD and the TiKV store are lost for more than 20 seconds. If the lost period exceeds the time specified by `max-store-down-time`, the status "Disconnect" changes to "Down".

- **Down**: Heartbeat messages between the PD and the TiKV store are lost for a time longer than `max-store-down-time` (30 minutes by default). In this status, the TiKV store starts replenishing replicas of each Region on the surviving store.
- **Offline**: A TiKV store is manually taken offline through PD Control. This is only an intermediate status for the store to go offline. The store in this status moves all its Regions to other "Up" stores that meet the relocation conditions. When `leader_count` and `region_count` (obtained through PD Control) both show `0`, the store status changes to "Tombstone" from "Offline". In the "Offline" status, **do not** disable the store service or the physical server where the store is located. During the process that the store goes offline, if the cluster does not have target stores to relocate the Regions (for example, inadequate stores to hold replicas in the cluster), the store is always in the "Offline" status.
- **Tombstone**: The TiKV store is completely offline. You can use the `remove-tombstone` interface to safely clean up TiKV in this status.

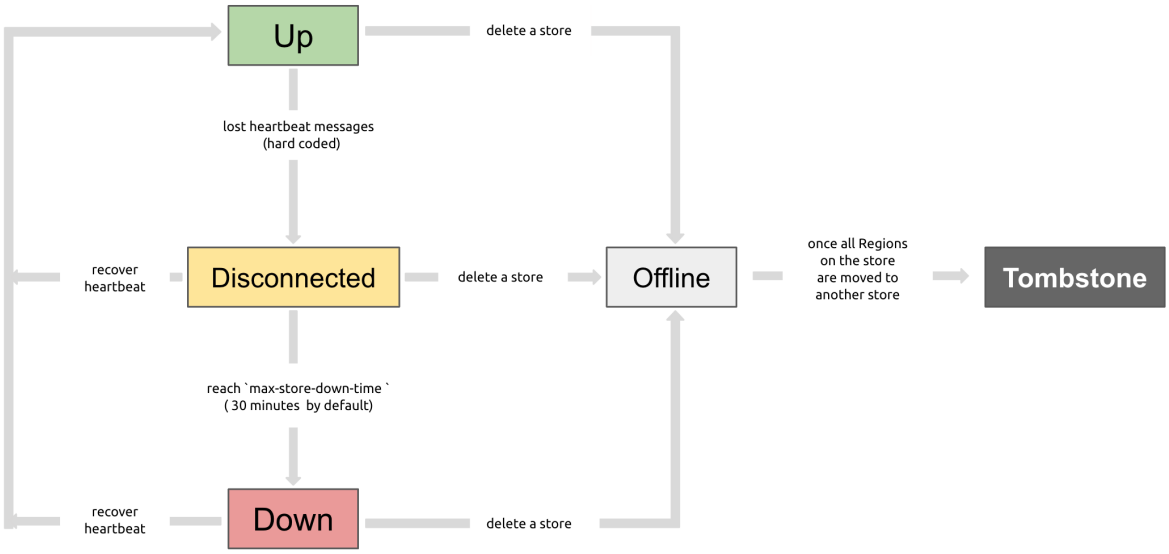


Figure 317: TiKV store status relationship

- Information reported by Region leaders:

  Each Region leader sends heartbeats to PD periodically to report `RegionState`, including:

  - Position of the leader itself
  - Positions of other replicas
  - The number of offline replicas

- data read/write speed

PD collects cluster information by the two types of heartbeats and then makes decision based on it.

Besides, PD can get more information from an expanded interface to make a more precise decision. For example, if a store's heartbeats are broken, PD can't know whether the peer steps down temporarily or forever. It just waits a while (by default 30min) and then treats the store as offline if there are still no heartbeats received. Then PD balances all regions on the store to other stores.

But sometimes stores are manually set offline by a maintainer, so the maintainer can tell PD this by the PD control interface. Then PD can balance all regions immediately.

14.1.4.5 Scheduling strategies

After collecting the information, PD needs some strategies to make scheduling plans.

Strategy 1: The number of replicas of a Region needs to be correct

PD can know that the replica count of a Region is incorrect from the Region leader's heartbeat. If it happens, PD can adjust the replica count by adding/removing replica(s). The reason for incorrect replica count can be:

- Store failure, so some Region's replica count is less than expected;
- Store recovery after failure, so some Region's replica count could be more than expected;
- `max-replicas` is changed.

Strategy 2: Replicas of a Region need to be at different positions

Note that here "position" is different from "machine". Generally PD can only ensure that replicas of a Region are not at a same peer to avoid that the peer's failure causes more than one replicas to become lost. However in production, you might have the following requirements:

- Multiple TiKV peers are on one machine;
- TiKV peers are on multiple racks, and the system is expected to be available even if a rack fails;
- TiKV peers are in multiple data centers, and the system is expected to be available even if a data center fails;

The key to these requirements is that peers can have the same "position", which is the smallest unit for failure toleration. Replicas of a Region must not be in one unit. So, we can configure labels for the TiKV peers, and set location-labels on PD to specify which labels are used for marking positions.

Strategy 3: Replicas need to be balanced between stores

The size limit of a Region replica is fixed, so keeping the replicas balanced between stores is helpful for data size balance.

Strategy 4: Leaders need to be balanced between stores

Read and write operations are performed on leaders according to the Raft protocol, so that PD needs to distribute leaders into the whole cluster instead of several peers.

Strategy 5: Hot spots need to be balanced between stores

PD can detect hot spots from store heartbeats and Region heartbeats, so that PD can distribute hot spots.

Strategy 6: Storage size needs to be balanced between stores

When started up, a TiKV store reports `capacity` of storage, which indicates the store's space limit. PD will consider this when scheduling.

Strategy 7: Adjust scheduling speed to stabilize online services

Scheduling utilizes CPU, memory, network and I/O traffic. Too much resource utilization will influence online services. Therefore, PD needs to limit the number of the concurrent scheduling tasks. By default this strategy is conservative, while it can be changed if quicker scheduling is required.

14.1.4.6 Scheduling implementation

PD collects cluster information from store heartbeats and Region heartbeats, and then makes scheduling plans from the information and strategies. Scheduling plans are a sequence of basic operators. Every time PD receives a Region heartbeat from a Region leader, it checks whether there is a pending operator on the Region or not. If PD needs to dispatch a new operator to a Region, it puts the operator into heartbeat responses, and monitors the operator by checking follow-up Region heartbeats.

Note that here "operators" are only suggestions to the Region leader, which can be skipped by Regions. Leader of Regions can decide whether to skip a scheduling operator or not based on its current status.

14.2 Storage Engine - TiKV

14.2.1 TiKV Overview

TiKV is a distributed and transactional key-value database, which provides transactional APIs with ACID compliance. With the implementation of the Raft consensus algorithm and consensus state stored in RocksDB, TiKV guarantees data consistency between multiple replicas and high availability. As the storage layer of the TiDB distributed database, TiKV provides the read and write service, and persist the written data from applications. It also stores the statistics data of the TiDB cluster.

14.2.1.1 Architecture Overview

TiKV implements the multi-raft-group replica mechanism based on the design of Google Spanner. A Region is a basic unit of the key-value data movement and refers to a data range in a Store. Each Region is replicated to multiple nodes. These multiple replicas form a Raft group. A replica of a Region is called a Peer. Typically there are 3 peers in a Region. One of them is the leader, which provides the read and write services. The PD component balances all the Regions automatically to guarantee that the read and write throughput is balanced among all the nodes in the TiKV cluster. With PD and carefully designed Raft groups, TiKV excels in horizontal scalability and can easily scale to store more than 100 TBs of data.

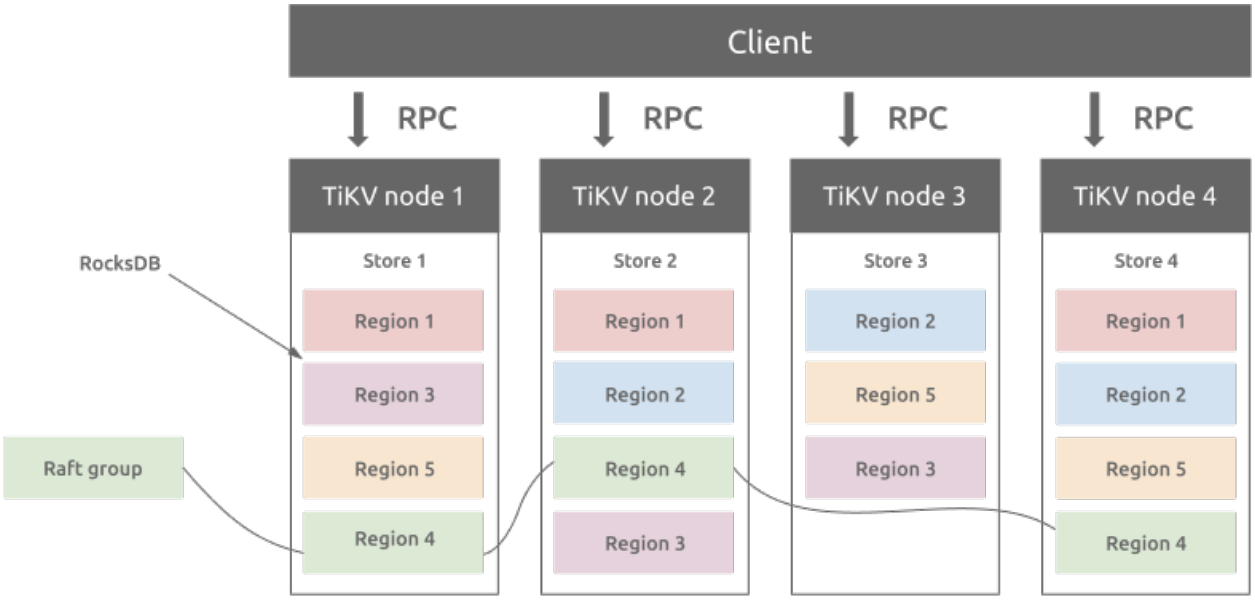


Figure 318: TiKV Architecture

14.2.1.1.1 Region and RocksDB

There is a RocksDB database within each Store and it stores data into the local disk. All the Region data are stored in the same RocksDB instance in each Store. All the logs used for the Raft consensus algorithm is stored in another RocksDB instance in each Store. This is because the performance of sequential I/O is better than random I/O. With different RocksDB instances storing raft logs and Region data, TiKV combines all the data write operations of raft logs and TiKV Regions into one I/O operation to improve the performance.

14.2.1.1.2 Region and Raft Consensus Algorithm

Data consistency between replicas of a Region is guaranteed by the Raft Consensus Algorithm. Only the leader of the Region can provide the writing service, and only when the data is written to the majority of replicas of a Region, the write operation succeeds.

TiKV tries to keep an appropriate size for each Region in the cluster. The Region size is currently 96 MiB by default. This mechanism helps the PD component to balance Regions among nodes in a TiKV cluster. When the size of a Region exceeds a threshold (144 MiB by default), TiKV splits it into two or more Regions. When the size of a Region is smaller than the threshold (20 MiB by default), TiKV merges the two smaller adjacent Regions into one Region.

When PD moves a replica from one TiKV node to another, it firstly adds a Learner replica on the target node, after the data in the Learner replica is nearly the same as that in the Leader replica, PD changes it to a Follower replica and removes the Follower replica on the source node.

Moving Leader replica from one node to another has a similar mechanism. The difference is that after the Learner replica becomes the Follower replica, there is a "Leader Transfer" operation in which the Follower replica actively proposes an election to elect itself as the Leader. Finally, the new Leader removes the old Leader replica in the source node.

14.2.1.2 Distributed Transaction

TiKV supports distributed transactions. Users (or TiDB) can write multiple key-value pairs without worrying about whether they belong to the same Region. TiKV uses two-phase commit to achieve ACID constraints. See TiDB Optimistic Transaction Model for details.

14.2.1.3 TiKV Coprocessor

TiDB pushes some data computation logic to TiKV Coprocessor. TiKV Coprocessor processes the computation for each Region. Each request sent to TiKV Coprocessor only involves the data of one Region.

14.2.2 RocksDB Overview

RocksDB is an LSM-tree storage engine that provides key-value store and read-write functions. It is developed by Facebook and based on LevelDB. Key-value pairs written by the user are firstly inserted into Write Ahead Log (WAL) and then written to the SkipList in memory (a data structure called MemTable). LSM-tree engines convert the random modification (insertion) to sequential writes to the WAL file, so they provide better write throughput than B-tree engines.

Once the data in memory reaches a certain size, RocksDB flushes the content into a Sorted String Table (SST) file in the disk. SST files are organized in multiple levels (the default is up to 6 levels). When the total size of a level reaches the threshold, RocksDB chooses part of the SST files and merges them into the next level. Each subsequent level is 10 times larger than the previous one, so 90% of the data is stored in the last layer.

RocksDB allows users to create multiple Column Families (CFs). CFs have their own SkipList and SST files, and they share the same WAL file. In this way, different CFs can have different settings according to the application characteristics. It does not increase the number of writes to WAL at the same time.

14.2.2.1 TiKV architecture

The architecture of TiKV is illustrated as follows:

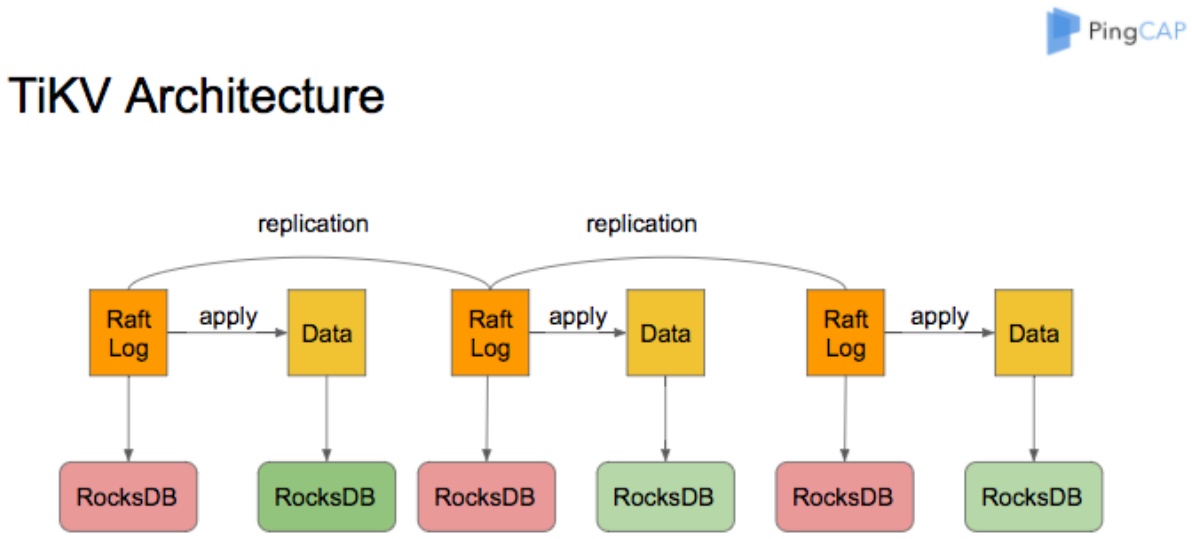


Figure 319: TiKV RocksDB

As the storage engine of TiKV, RocksDB is used to store Raft logs and user data. All data in a TiKV node shares two RocksDB instances. One is for Raft log (often called raftdb), and the other is for user data and MVCC metadata (often called kvdb). There are four CFs in kvdb: raft, lock, default, and write:

- raft CF: Store metadata of each Region. It occupies only a very small amount of space, and users do not need to care.
- lock CF: Store the pessimistic lock of pessimistic transactions and the Prewrite lock for distributed transactions. After the transaction is committed, the corresponding data in lock CF is deleted quickly. Therefore, the size of data in lock CF is usually very small (less than 1 GB). If the data in lock CF increases a lot, it means that a large number of transactions are waiting to be committed, and that the system meets a bug or failure.
- write CF: Store the user's real written data and MVCC metadata (the start timestamp and commit timestamp of the transaction to which the data belongs). When the user writes a row of data, it is stored in the write CF if the data length is less than 255 bytes. Otherwise, it is stored in the default CF. In TiDB, the secondary index only occupies the space of write CF, since the value stored in the non-unique index is empty and the value stored in the unique index is the primary key index.
- default CF: Store data longer than 255 bytes.

14.2.2.2 RocksDB memory usage

To improve the reading performance and reduce the reading operations to the disk, RocksDB divides the files stored on the disk into blocks based on a certain size (the default is 64 KB). When reading a block, it first checks if the data already exists in BlockCache in memory. If true, it can read the data directly from memory without accessing the disk.

BlockCache discards the least recently used data according to the LRU algorithm. By default, TiKV devotes 45% of the system memory to BlockCache. Users can also modify the `storage.block-cache.capacity` configuration to an appropriate value by themselves. However, it is not recommended to exceed 60% of the total system memory.

The data written to RocksDB is written to MemTable firstly. When the size of a MemTable exceeds 128 MB, it switches to a new MemTable. There are 2 RocksDB instances in TiKV, a total of 4 CFs. The size limit of a single MemTable for each CF is 128 MB. A maximum of 5 MemTables can exist at the same time; otherwise, the foreground writes is blocked. The memory occupied by this part is at most 2.5 GB (4 x 5 x 128 MB). It is not recommended to change this limit since it costs less memory.

14.2.2.3 RocksDB space usage

- Multi-version: As RocksDB is a key-value storage engine with LSM-tree structure, the data in MemTable is flushed to L0 first. Because the file is arranged in the order of which they are generated, there might be overlap between the ranges of SSTs at the L0. As a result, the same key might have multiple versions in L0. When a file is merged from L0 to L1, it is cut into multiple files in a certain size (the default is 8 MB). The key range of each file on the same level does not overlap with each other, so there is only one version for each key on L1 and subsequent levels.
- Space amplification: The total size of files on each level is x (the default is 10) times that of the previous level, so 90% of the data is stored in the last level. It also means that the space amplification of RocksDB does not exceed 1.11 (L0 has fewer data and can be ignored).
- Space amplification of TiKV: TiKV has it's own MVCC strategy. When a user writes a key, the real data written to RocksDB is key + commit_ts, that is to say, the update and deletion also write a new key to RocksDB. TiKV deletes the old version of the data (through the Delete interface of RocksDB) at intervals, so it can be considered that the actual space of the data stored by the user on TiKV is enlarged to 1.11 plus the data written in the last 10 minutes (assuming that TiKV cleans up the old data promptly).

14.2.2.4 RocksDB background threads and compaction

In RocksDB, operations such as converting the MemTable into SST files and merging SST files at various levels are performed in the background thread pool. The default size of the background thread pool is 8. When the number of CPUs in the machine is less than or equal to 8, the default size of the background thread pool is the number of CPUs minus one.

Generally speaking, users do not need to change this configuration. If the user deploys multiple TiKV instances on a machine, or the machine has a relatively high read load and a

low write load, you can adjust the `rocksdb/max-background-jobs` to 3 or 4 as appropriate.

14.2.2.5 WriteStall

The L0 of RocksDB is different from other levels. The SSTs of L0 are arranged in the order of generation. The key ranges between the SSTs can overlap. Therefore, each SST in L0 must be queried in turn when a query is performed. In order not to affect query performance, WriteStall is triggered to block writing when there are too many files in L0.

When encountering a sudden sharp increase in write delay, you can first check the **WriteStall Reason** metric on the Grafana RocksDB KV panel. If it is a WriteStall caused by too many L0 files, you can adjust the following configurations to 64.

```
rocksdb.defaultcf.level0-slowdown-writes-trigger
rocksdb.writecf.level0-slowdown-writes-trigger
rocksdb.lockcf.level0-slowdown-writes-trigger
rocksdb.defaultcf.level0-stop-writes-trigger
rocksdb.writecf.level0-stop-writes-trigger
rocksdb.lockcf.level0-stop-writes-trigger
```

14.2.3 Titan Overview

Titan is a high-performance RocksDB plugin for key-value separation. Titan can reduce write amplification in RocksDB when large values are used.

When the value size in Key-Value pairs is large (larger than 1 KB or 512 B), Titan performs better than RocksDB in write, update, and point read scenarios. However, Titan gets a higher write performance by sacrificing storage space and range query performance. As the price of SSDs continues to decrease, this trade-off will be more and more meaningful.

14.2.3.1 Key features

- Reduce write amplification by separating values from the log-structured merge-tree (LSM-tree) and storing them independently.
- Seamlessly upgrade RocksDB instances to Titan. The upgrade does not require human intervention and does not impact online services.
- Achieve 100% compatibility with all RocksDB features used by the current TiKV.

14.2.3.2 Usage scenarios

Titan is suitable for the scenarios where a huge volume of data is written to the TiKV foreground:

- RocksDB triggers a large amount of compactions, which consumes a lot of I/O bandwidth or CPU resources. This causes poor read and write performance of the foreground.

- The RocksDB compaction lags much behind (due to the I/O bandwidth limit or CPU bottleneck) and frequently causes write stalls.
- RocksDB triggers a large amount of compactions, which causes a lot of I/O writes and affects the life of the SSD disk.

14.2.3.3 Prerequisites

The prerequisites for enabling Titan are as follows:

- The average size of values is large, or the size of all large values accounts for much of the total value size. Currently, the size of a value greater than 1 KB is considered as a large value. In some situations, this number (1 KB) can be 512 B. Note that a single value written to TiKV cannot exceed 8 MB due to the limitation of the TiKV Raft layer. You can adjust the `raft-entry-max-size` configuration value to relax the limit.
- No range query will be performed or you do not need a high performance of range query. Because the data stored in Titan is not well-ordered, its performance of range query is poorer than that of RocksDB, especially for the query of a large range. According PingCAP's internal test, Titan's range query performance is 40% to a few times lower than that of RocksDB.
- Sufficient disk space (consider reserving a space twice of the RocksDB disk consumption with the same data volume). This is because Titan reduces write amplification at the cost of disk space. In addition, Titan compresses values one by one, and its compression rate is lower than that of RocksDB. RocksDB compresses blocks one by one. Therefore, Titan consumes more storage space than RocksDB, which is expected and normal. In some situations, Titan's storage consumption can be twice that of RocksDB.

If you want to improve the performance of Titan, see the blog post Titan: A RocksDB Plugin to Reduce Write Amplification.

14.2.3.4 Architecture and implementation

The following figure shows the architecture of Titan:

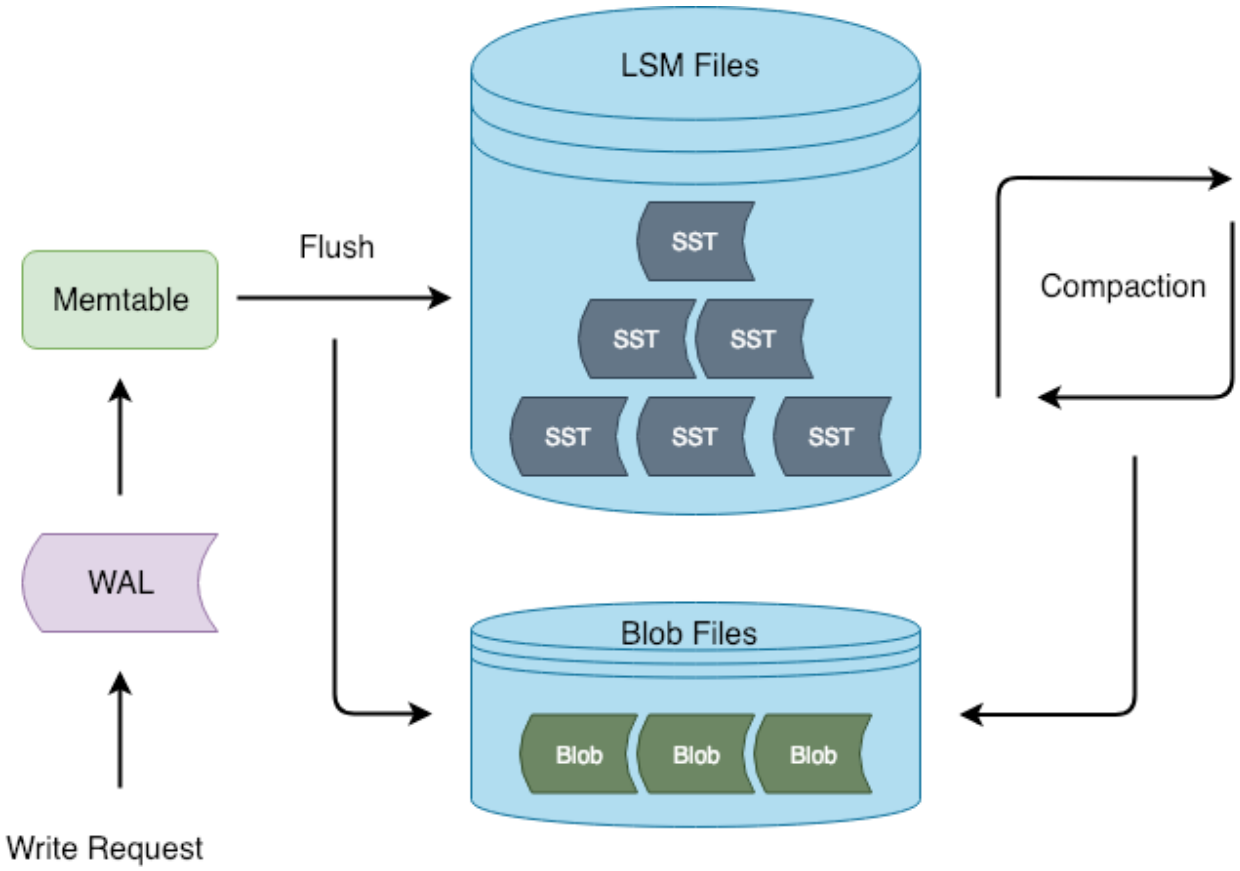


Figure 320: Titan Architecture

During flush and compaction operations, Titan separates values from the LSM-tree. The advantage of this approach is that the write process is consistent with RocksDB, which reduces the chance of invasive changes to RocksDB.

14.2.3.4.1 BlobFile

When Titan separates the value file from the LSM-tree, it stores the value file in the BlobFile. The following figure shows the BlobFile format:

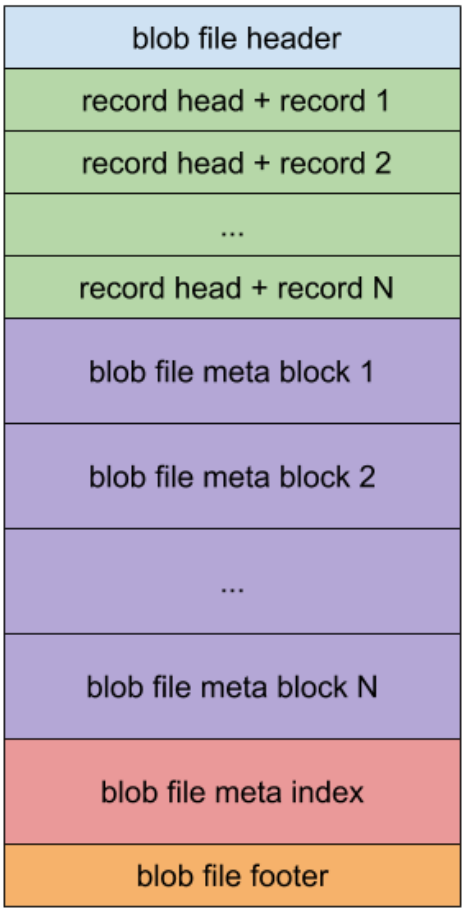


Figure 321: BlobFile Format

A blob file mainly consists of blob records, meta blocks, a meta index block, and a footer. Each block record stores a Key-Value pair. The meta blocks are used for scalability, and store properties related to the blob file. The meta index block is used for meta block searching.

Note:

- The Key-Value pairs in the blob file are stored in order, so that when the Iterator is implemented, the sequential reading performance can be improved via prefetching.
- Each blob record keeps a copy of the user key corresponding to the value. This way, when Titan performs Garbage Collection (GC), it can query the user key and identify whether the corresponding value is outdated. However, this process introduces some write amplification.

- BlobFile supports compression at the blob record level. Titan supports multiple compression algorithms, such as Snappy, LZ4, and Zstd. Currently, the default compression algorithm Titan uses is LZ4.

14.2.3.4.2 TitanTableBuilder

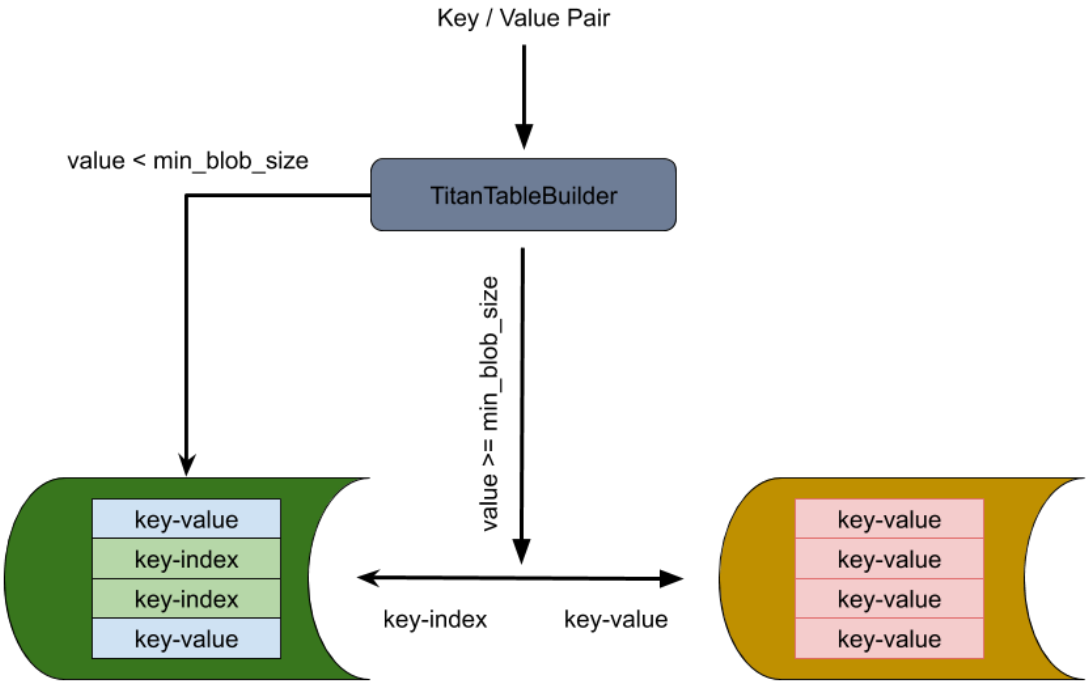


Figure 322: TitanTableBuilder

TitanTableBuilder is the key to achieving Key-Value separation. TitanTableBuilder determines the Key-Pair value size, and based on that, decides whether to separate the value from the Key-Value pair and store it in the blob file.

- If the value size is greater than or equal to `min_blob_size`, TitanTableBuilder separates the value and stores it in the blob file. TitanTableBuilder also generates an index and writes it into the SST.
- If the value size is smaller than `min_blob_size`, TitanTableBuilder writes the value directly into the SST.

Titan can also be downgraded to RocksDB in the process above. When RocksDB is performing compactions, the separated value can be written back to the newly generated SST files.

14.2.3.5 Garbage Collection

Titan uses Garbage Collection (GC) to reclaim space. As the keys are being reclaimed in the LSM-tree compaction, some values stored in blob files are not deleted at the same time. Therefore, Titan needs to perform GC periodically to delete outdated values. Titan provides the following two types of GC:

- Blob files are periodically integrated and rewritten to delete outdated values. This is the regular way of performing GC.
- Blob files are rewritten while the LSM-tree compaction is performed at the same time. This is the feature of Level Merge.

14.2.3.5.1 Regular GC

Titan uses the TablePropertiesCollector and EventListener components of RocksDB to collect the information for GC.

TablePropertiesCollector

RocksDB supports using BlobFileSizeCollector, a custom table property collector, to collect properties from the SST which are written into corresponding SST files. The collected properties are named BlobFileSizeProperties. The following figure shows the BlobFileSizeCollector workflow and data formats:

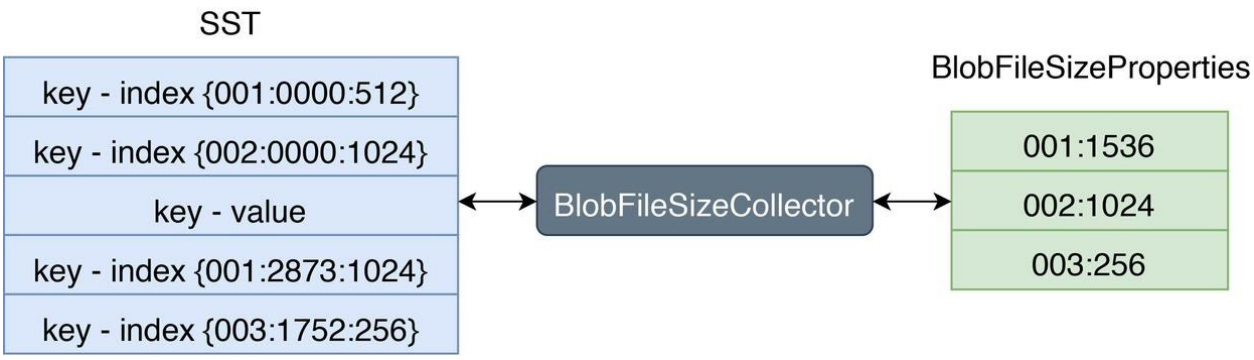


Figure 323: BlobFileSizeProperties

On the left is the SST index format. The first column is the blob file ID; the second column is the offset for the blob record in the blob file; the third column is the blob record size.

On the right is the BlobFileSizeProperties format. Each line represents a blob file and how much data is saved in this blob file. The first column is the blob file ID; the second column is the size of the data.

EventListener

RocksDB uses compaction to discard old data and reclaim space. After each compaction, some blob files in Titan might contain partly or entirely outdated data. Therefore, you can

trigger GC by listening to compaction events. During compaction, you can collect and compare the input/output blob file size properties of SST to determine which blob files require GC. The following figure shows the general process:

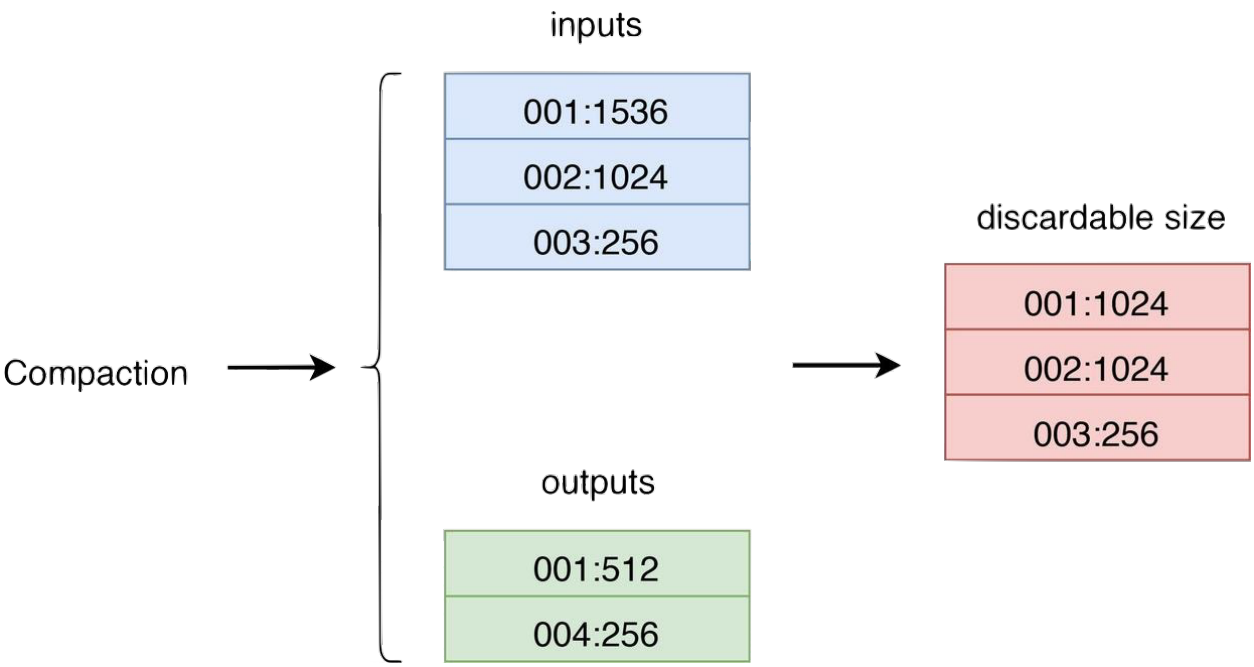


Figure 324: EventListener

- *inputs* stands for the blob file size properties for all SSTs that participate in the compaction.
- *outputs* stands for the blob file size properties for all SSTs generated in the compaction.
- *discardable size* is the size of the file to be discarded for each blob file, calculated based on inputs and outputs. The first column is the blob file ID. The second column is the size of the file to be discarded.

For each valid blob file, Titan maintains a discardable size variable in memory. After each compaction, this variable is accumulated for the corresponding blob file. Each time when GC starts, it picks the blob file with the greatest discardable size as the candidate file for GC. To reduce write amplification, a certain level of space amplification is allowed, which means GC can be started on a blob file only when the discardable file has reached a specific proportion in size.

For the selected blob file, Titan checks whether the blob index of the key corresponding to each value exists or has been updated to determine whether this value is outdated. If the value is not outdated, Titan merges and sorts the value into a new blob file, and writes the updated blob index into SST using WriteCallback or MergeOperator. Then, Titan records the latest sequence number of RocksDB and does not delete the old blob file until the sequence of the oldest snapshot exceeds the recorded sequence number. The reason is

that after the blob index is written back to SST, the old blob index is still accessible via the previous snapshot. Therefore, we need to ensure that no snapshot will access the old blob index before GC can safely deletes the corresponding blob file.

14.2.3.5.2 Level Merge

Level Merge is a newly introduced algorithm in Titan. According to the implementation principle of Level Merge, Titan merges and rewrites blob file that corresponds to the SST file, and generates new blob file while compactions are performed in LSM-tree. The following figure shows the general process:

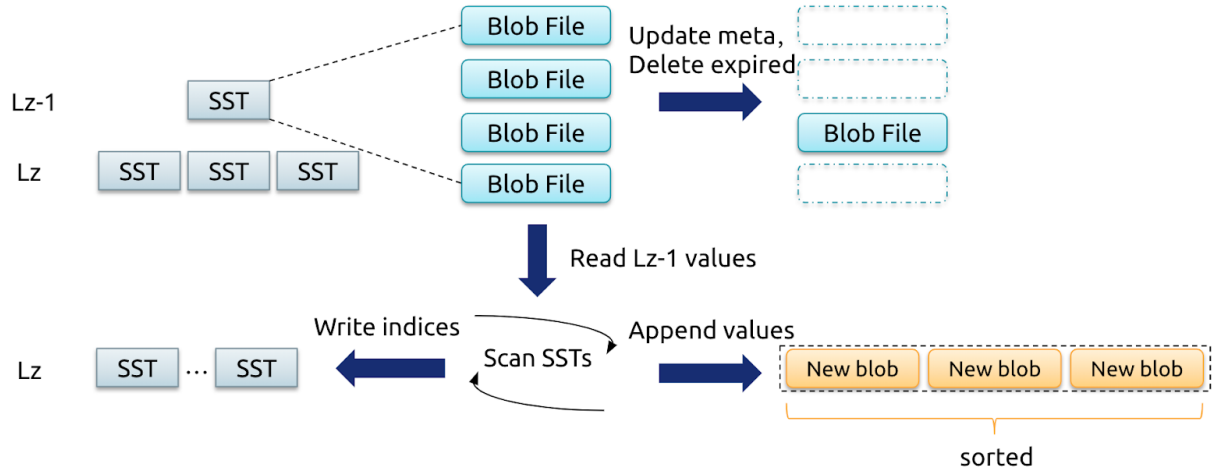


Figure 325: LevelMerge General Process

When compactions are performed on the SSTs of level z-1 and level z, Titan reads and writes Key-Value pairs in order. Then it writes the values of the selected blob files into new blob files in order, and updates the blob indexes of keys when new SSTs are generated. For the keys deleted in compactions, the corresponding values will not be written to the new blob file, which works similar to GC.

Compared with the regular way of GC, the Level Merge approach completes the blob GC while compactions are performed in LSM-tree. In this way, Titan no longer needs to check the status of blob index in LSM-tree or to write the new blob index into LSM-tree. This reduces the impact of GC on the foreground operations. As the blob file is repeatedly rewritten, fewer files overlap with each other, which makes the whole system in better order and improves the performance of scan.

However, layering blob files similar to tiering compaction brings write amplification. Because 99% of the data in LSM-tree is stored at the lowest two levels, Titan performs the Level Merge operation on the blob files corresponding to the data that is compacted only to the lowest two levels of LSM-tree.

Range Merge

Range Merge is an optimized approach of GC based on Level Merge. However, the bottom level of LSM-tree might be in poorer order in the following situations:

- When `level_compaction_dynamic_level_bytes` is enabled, data volume at each level of LSM-tree dynamically increases, and the sorted runs at the bottom level keep increasing.
- A specific range of data is frequently compacted, and this causes a lot of sorted runs in that range.

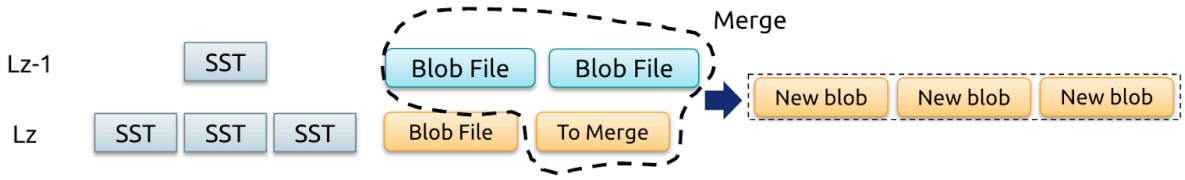


Figure 326: RangeMerge

Therefore, the Range Merge operation is needed to keep the number of sorted runs within a certain level. At the time of OnCompactionComplete, Titan counts the number of sorted runs in a range. If the number is large, Titan marks the corresponding blob file as ToMerge and rewrites it in the next compaction.

14.2.4 Titan Configuration

This document introduces how to enable and disable Titan using the corresponding configuration items, as well as the relevant parameters and the Level Merge feature.

14.2.4.1 Enable Titan

Titan is compatible with RocksDB, so you can directly enable Titan on the existing TiKV instances that use RocksDB. You can use one of the following two methods to enable Titan:

- Method 1: If you have deployed the cluster using TiUP, you can execute the `tiup cluster edit-config ${cluster-name}` command and edit the TiKV configuration file as the following example shows:

```
tikv:
  rocksdb.titan.enabled: true
```

Reload the configuration and TiKV will be rolling restarted dynamically:

```
tiup cluster reload ${cluster-name} -R tikv
```

For the detailed command, see Modify the configuration using TiUP.

- Method 2: Directly edit the TiKV configuration file to enable Titan (**NOT** recommended for the production environment).

```
[rocksdb.titan]
enabled = true
```

After Titan is enabled, the existing data stored in RocksDB is not immediately moved to the Titan engine. As new data is written to the TiKV foreground and RocksDB performs compaction, the values are progressively separated from keys and written to Titan. You can view the **TiKV Details** -> **Titan kv** -> **blob file size** panel to confirm the size of the data stored in Titan.

If you want to speed up the writing process, compact data of the whole TiKV cluster manually using tikv-ctl. For details, see manual compaction.

> **Note:**
>
> When Titan is disabled, RocksDB cannot read data that has been migrated to Titan. If Titan is incorrectly disabled on a TiKV instance with Titan already enabled (mistakenly set `rocksdb.titan.enabled` to `false`), TiKV will fail to start, and the `You have disabled titan when its data directory is not empty` error appears in the TiKV log. To correctly disabled Titan, see Disable Titan.

14.2.4.2 Parameters

To adjust Titan-related parameters using TiUP, refer to Modify the configuration.

- Titan GC thread count.

  From the **TiKV Details** -> **Thread CPU** -> **RocksDB CPU** panel, if you observe that the Titan GC threads are at full capacity for a long time, consider increasing the size of the Titan GC thread pool.

```
[rocksdb.titan]
max-background-gc = 1
```

- Value size threshold.

  When the size of the value written to the foreground is smaller than the threshold, this value is stored in RocksDB; otherwise, this value is stored in the blob file of Titan. Based on the distribution of value sizes, if you increase the threshold, more values are stored in RocksDB and TiKV performs better in reading small values. If you decrease the threshold, more values go to Titan, which further reduces RocksDB compactions.

```
[rocksdb.defaultcf.titan]
min-blob-size = "1KB"
```

- The algorithm used for compressing values in Titan, which takes value as the unit.

```
[rocksdb.defaultcf.titan]
blob-file-compression = "lz4"
```

- The size of value caches in Titan.

  Larger cache size means higher read performance of Titan. However, too large a cache size causes Out of Memory (OOM). It is recommended to set the value of `storage ↪ .block-cache.capacity` to the store size minus the blob file size and set `blob- ↪ cache-size` to `memory size * 50% - block cache size` according to the monitoring metrics when the database is running stably. This maximizes the blob cache size when the block cache is large enough for the whole RocksDB engine.

```
[rocksdb.defaultcf.titan]
blob-cache-size = 0
```

- When the ratio of discardable data (the corresponding key has been updated or deleted) in a blob file exceeds the following threshold, Titan GC is triggered.

```
discardable-ratio = 0.5
```

  When Titan writes the useful data of this blob file to another file, you can use the `discardable-ratio` value to estimate the upper limits of write amplification and space amplification (assuming the compression is disabled).

  Upper limit of write amplification = 1 / discardable_ratio

  Upper limit of space amplification = 1 / (1 - discardable_ratio)

  From the two equations above, you can see that decreasing the value of `discardable_ratio ↪` can reduce space amplification but causes GC to be more frequent in Titan. Increasing the value reduces Titan GC, the corresponding I/O bandwidth, and CPU consumption but increases disk usage.

- The following option limits the I/O rate of RocksDB compaction. During peak traffic, limiting RocksDB compaction, its I/O bandwidth, and its CPU consumption reduces its impact on the write and read performance of the foreground.

When Titan is enabled, this option limits the summed I/O rates of RocksDB compaction and Titan GC. If you find that the I/O and/or CPU consumption of RocksDB compaction and Titan GC is too large, set this option to a suitable value according the disk I/O bandwidth and the actual write traffic.

```
[rocksdb]
rate-bytes-per-sec = 0
```

14.2.4.3 Disable Titan (experimental)

> **Warning:**
>
> Disabling Titan is an experimental feature. It is **NOT** recommended to use it if not necessary.

To disable Titan, you can configure the `rocksdb.defaultcf.titan.blob-run-mode` option. The optional values for `blob-run-mode` are as follows:

- When the option is set to `normal`, Titan performs read and write operations normally.
- When the option is set to `read-only`, all newly written values are written into RocksDB, regardless of the value size.
- When the option is set to `fallback`, all newly written values are written into RocksDB, regardless of the value size. Also, all compacted values stored in the Titan blob file are automatically moved back to RocksDB.

To disable Titan for all existing and future data, you can take the following steps. Note that you can skip Step 2 because it greatly affects online traffic performance. In fact even without Step 2, the data compaction consumes extra I/O and CPU resources when it moves data from Titan to RocksDB, and performance will degrade (sometimes as much as 50%) when TiKV I/O or CPU resources are limited.

> **Warning:**
>
> When disabling Titan for TiDB versions earlier than v8.5.0, it is not recommended to modify the TiKV configuration item `rocksdb.titan.enabled` to `false`, as this might cause TiKV to crash. Following Step 1 is sufficient to disable Titan.

1. Update the configuration file of the TiKV nodes you wish to disable Titan for and restart TiKV:

```
[rocksdb.defaultcf.titan]
blob-run-mode = "fallback"
discardable-ratio = 1.0
merge-small-file-threshold = "0KB"
```

2. Perform a full compaction using tikv-ctl. This process will consume large amount of I/O and CPU resources.

```
tikv-ctl --pd <PD_ADDR> compact-cluster --bottommost force
```

3. After the compaction is finished, wait until the **Blob file count** metrics under **TiKV-Details/Titan - kv** decrease to `0`.

14.2.4.4 Level Merge (experimental)

In TiKV 4.0, Level Merge, a new algorithm, is introduced to improve the performance of range query and to reduce the impact of Titan GC on the foreground write operations. You can enable Level Merge using the following option:

```
[rocksdb.defaultcf.titan]
level-merge = true
```

Enabling Level Merge has the following benefits:

- Greatly improve the performance of Titan range query.
- Reduce the impact of Titan GC on the foreground write operations and improve write performance.
- Reduce space amplification of Titan and the disk usage (compared to the disk usage with the default configuration).

Accordingly, the write amplification with Level Merge enabled is slightly higher than that of Titan but is still lower than that of the native RocksDB.

14.3 Storage Engine - TiFlash

14.3.1 TiFlash Overview

TiFlash is the key component that makes TiDB essentially an Hybrid Transactional/-Analytical Processing (HTAP) database. As a columnar storage extension of TiKV, TiFlash provides both good isolation level and strong consistency guarantee.

In TiFlash, the columnar replicas are asynchronously replicated according to the Raft Learner consensus algorithm. When these replicas are read, the Snapshot Isolation level of consistency is achieved by validating Raft index and multi-version concurrency control (MVCC).

14.3.1.1 Architecture

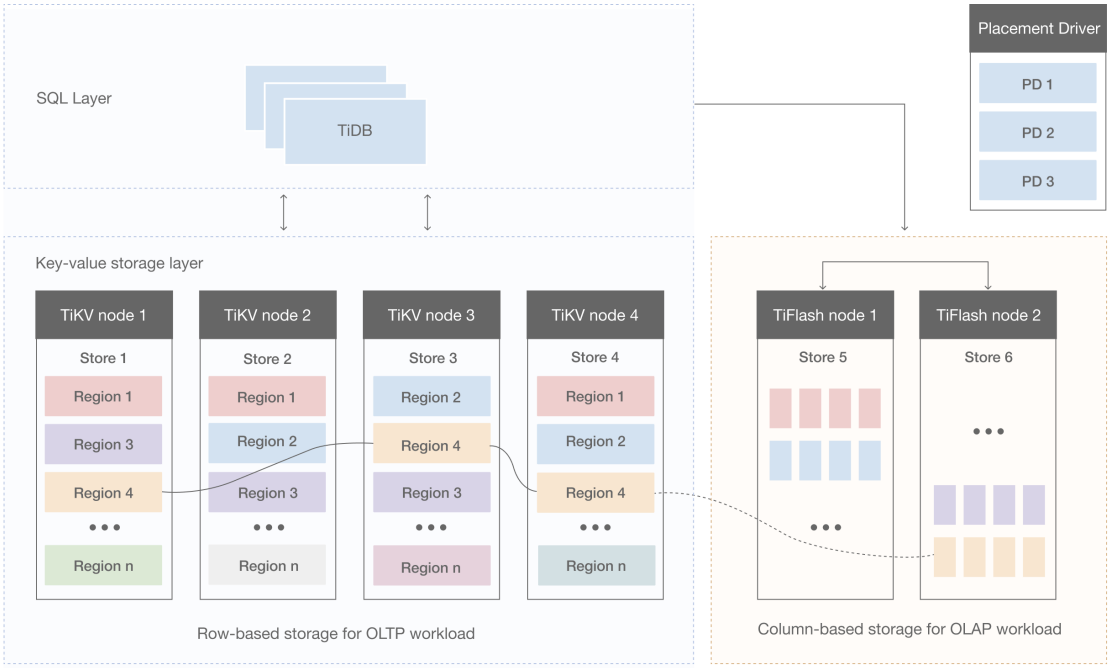


Figure 327: TiFlash Architecture

The above figure is the architecture of TiDB in its HTAP form, including TiFlash nodes.

TiFlash provides the columnar storage, with a layer of coprocessors efficiently implemented by ClickHouse. Similar to TiKV, TiFlash also has a Multi-Raft system, which supports replicating and distributing data in the unit of Region (see Data Storage for details).

TiFlash conducts real-time replication of data in the TiKV nodes at a low cost that does not block writes in TiKV. Meanwhile, it provides the same read consistency as in TiKV and ensures that the latest data is read. The Region replica in TiFlash is logically identical to those in TiKV, and is split and merged along with the Leader replica in TiKV at the same time.

TiFlash is compatible with both TiDB and TiSpark, which enables you to freely choose between these two computing engines.

It is recommended that you deploy TiFlash in different nodes from TiKV to ensure workload isolation. It is also acceptable to deploy TiFlash and TiKV in the same node if no business isolation is required.

Currently, data cannot be written directly into TiFlash. You need to write data in TiKV and then replicate it to TiFlash, because it connects to the TiDB cluster as a Learner role.

TiFlash supports data replication in the unit of table, but no data is replicated by default after deployment. To replicate data of a specified table, see Create TiFlash replicas for tables.

TiFlash has three components: the columnar storage module, `tiflash proxy`, and `pd buddy`. `tiflash proxy` is responsible for the communication using the Multi-Raft consensus algorithm. `pd buddy` works with PD to replicate data from TiKV to TiFlash in the unit of table.

When TiDB receives the DDL command to create replicas in TiFlash, the `pd buddy` component acquires the information of the table to be replicated via the status port of TiDB, and sends the information to PD. Then PD performs the corresponding data scheduling according to the information provided by `pd buddy`.

14.3.1.2 Key features

TiFlash has the following key features:

- Asynchronous replication
- Consistency
- Intelligent choice
- Computing acceleration

14.3.1.2.1 Asynchronous replication

The replica in TiFlash is asynchronously replicated as a special role, Raft Learner. This means when the TiFlash node is down or high network latency occurs, applications in TiKV can still proceed normally.

This replication mechanism inherits two advantages of TiKV: automatic load balancing and high availability.

- TiFlash does not rely on additional replication channels, but directly receives data from TiKV in a many-to-many manner.
- As long as the data is not lost in TiKV, you can restore the replica in TiFlash at any time.

14.3.1.2.2 Consistency

TiFlash provides the same Snapshot Isolation level of consistency as TiKV, and ensures that the latest data is read, which means that you can read the data previously written in TiKV. Such consistency is achieved by validating the data replication progress.

Every time TiFlash receives a read request, the Region replica sends a progress validation request (a lightweight RPC request) to the Leader replica. TiFlash performs the read operation only after the current replication progress includes the data covered by the timestamp of the read request.

14.3.1.2.3 Intelligent choice

TiDB can automatically choose to use TiFlash (column-wise) or TiKV (row-wise), or use both of them in one query to ensure the best performance.

This selection mechanism is similar to that of TiDB which chooses different indexes to execute query. TiDB optimizer makes the appropriate choice based on statistics of the read cost.

14.3.1.2.4 Computing acceleration

TiFlash accelerates the computing of TiDB in two ways:

- The columnar storage engine is more efficient in performing read operation.
- TiFlash shares part of the computing workload of TiDB.

TiFlash shares the computing workload in the same way as the TiKV Coprocessor does: TiDB pushes down the computing that can be completed in the storage layer. Whether the computing can be pushed down depends on the support of TiFlash. For details, see Supported pushdown calculations.

14.3.1.3 Use TiFlash

After TiFlash is deployed, data replication does not automatically begin. You need to manually specify the tables to be replicated.

You can either use TiDB to read TiFlash replicas for medium-scale analytical processing, or use TiSpark to read TiFlash replicas for large-scale analytical processing, which is based on your own needs. See the following sections for details:

- Create TiFlash Replicas
- Use TiDB to Read TiFlash Replicas
- Use TiSpark to Read TiFlash Replicas
- Use MPP Mode

To experience the whole process from importing data to querying in a TPC-H dataset, refer to Quick Start Guide for TiDB HTAP.

14.3.1.4 See also

- To deploy a new cluster with TiFlash nodes, see Deploy a TiDB cluster using TiUP.
- To add a TiFlash node in a deployed cluster, see Scale out a TiFlash cluster.
- Maintain a TiFlash cluster.
- Tune TiFlash performance.
- Configure TiFlash.
- Monitor the TiFlash cluster.
- Learn TiFlash alert rules.
- Troubleshoot a TiFlash cluster.
- Supported push-down calculations in TiFlash
- Data validation in TiFlash
- TiFlash compatibility

14.3.2 Create TiFlash Replicas

This document introduces how to create TiFlash replicas for tables and for databases, and set available zones for replica scheduling.

14.3.2.1 Create TiFlash replicas for tables

After TiFlash is connected to the TiKV cluster, data replication by default does not begin. You can send a DDL statement to TiDB through a MySQL client to create a TiFlash replica for a specific table:

```
ALTER TABLE table_name SET TIFLASH REPLICA count;
```

The parameter of the above command is described as follows:

- `count` indicates the number of replicas. When the value is `0`, the replica is deleted.

If you execute multiple DDL statements on the same table, only the last statement is ensured to take effect. In the following example, two DDL statements are executed on the table `tpch50`, but only the second statement (to delete the replica) takes effect.

Create two replicas for the table:

```
ALTER TABLE `tpch50`.`lineitem` SET TIFLASH REPLICA 2;
```

Delete the replica:

```
ALTER TABLE `tpch50`.`lineitem` SET TIFLASH REPLICA 0;
```

Notes:

- If the table `t` is replicated to TiFlash through the above DDL statements, the table created using the following statement will also be automatically replicated to TiFlash:

```
CREATE TABLE table_name like t;
```

- For versions earlier than v4.0.6, if you create the TiFlash replica before using TiDB Lightning to import the data, the data import will fail. You must import data to the table before creating the TiFlash replica for the table.
- If TiDB and TiDB Lightning are both v4.0.6 or later, no matter a table has TiFlash replica(s) or not, you can import data to that table using TiDB Lightning. Note that this might slow the TiDB Lightning procedure, which depends on the NIC bandwidth on the lightning host, the CPU and disk load of the TiFlash node, and the number of TiFlash replicas.
- It is recommended that you do not replicate more than 1,000 tables because this lowers the PD scheduling performance. This limit will be removed in later versions.
- In v5.1 and later versions, setting the replicas for the system tables is no longer supported. Before upgrading the cluster, you need to clear the replicas of the relevant system tables. Otherwise, you cannot modify the replica settings of the system tables after you upgrade the cluster to a later version.

14.3.2.1.1 Check replication progress

You can check the status of the TiFlash replicas of a specific table using the following statement. The table is specified using the `WHERE` clause. If you remove the `WHERE` clause, you will check the replica status of all tables.

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = '<
    ↪ db_name>' and TABLE_NAME = '<table_name>';
```

In the result of above statement:

- `AVAILABLE` indicates whether the TiFlash replicas of this table are available or not. `1` means available and `0` means unavailable. Once the replicas become available, this status does not change. If you use DDL statements to modify the number of replicas, the replication status will be recalculated.
- `PROGRESS` means the progress of the replication. The value is between `0.0` and `1.0`. `1` means at least one replica is replicated.

14.3.2.2 Create TiFlash replicas for databases

Similar to creating TiFlash replicas for tables, you can send a DDL statement to TiDB through a MySQL client to create a TiFlash replica for all tables in a specific database:

```
ALTER DATABASE db_name SET TIFLASH REPLICA count;
```

In this statement, `count` indicates the number of replicas. When you set it to `0`, replicas are deleted.

Examples:

- Create two replicas for all tables in the database `tpch50`:

```
ALTER DATABASE `tpch50` SET TIFLASH REPLICA 2;
```

- Delete TiFlash replicas created for the database `tpch50`:

```
ALTER DATABASE `tpch50` SET TIFLASH REPLICA 0;
```

Note:

- This statement actually performs a series of DDL operations, which are resource-intensive. If the statement is interrupted during the execution, executed operations are not rolled back and unexecuted operations do not continue.
- After executing the statement, do not set the number of TiFlash replicas or perform DDL operations on this database until **all tables in this database are replicated**. Otherwise, unexpected results might occur, which include:
  - If you set the number of TiFlash replicas to 2 and then change the number to 1 before all tables in the database are replicated, the final number of TiFlash replicas of all the tables is not necessarily 1 or 2.
  - After executing the statement, if you create tables in this database before the completion of the statement execution, TiFlash replicas **might or might not** be created for these new tables.
  - After executing the statement, if you add indexes for tables in the database before the completion of the statement execution, the statement might hang and resume only after the indexes are added.
- If you create tables in this database **after** the completion of the statement execution, TiFlash replicas are not created automatically for these new tables.
- This statement skips system tables, views, temporary tables, and tables with character sets not supported by TiFlash.

14.3.2.2.1 Check replication progress

Similar to creating TiFlash replicas for tables, successful execution of the DDL statement does not mean the completion of replication. You can execute the following SQL statement to check the progress of replication on target tables:

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = '<
    ↪ db_name>';
```

To check tables without TiFlash replicas in the database, you can execute the following SQL statement:

```
SELECT TABLE_NAME FROM information_schema.tables where TABLE_SCHEMA = "<
    ↪ db_name>" and TABLE_NAME not in (SELECT TABLE_NAME FROM
    ↪ information_schema.tiflash_replica where TABLE_SCHEMA = "<db_name>");
```

14.3.2.3 Speed up TiFlash replication

Before TiFlash replicas are added, each TiKV instance performs a full table scan and sends the scanned data to TiFlash as a "snapshot" to create replicas. By default, TiFlash replicas are added slowly with fewer resources usage in order to minimize the impact on the online service. If there are spare CPU and disk IO resources in your TiKV and TiFlash nodes, you can accelerate TiFlash replication by performing the following steps.

1. Temporarily increase the snapshot write speed limit for each TiKV and TiFlash instance by adjusting the TiFlash Proxy and TiKV configuration. For example, when using TiUP to manage configurations, the configuration is as below:

```
yaml tikv: server.snap-max-write-bytes-per-sec: 300MiB # Default to
↪  100MiB. tiflash-learner: raftstore.snap-handle-pool-size: 10  # Default
↪  to 2. Can be adjusted to >= node's CPU num * 0.6. raftstore.apply-low-
↪ priority-pool-size: 10 # Default to 1. Can be adjusted to >= node's CPU
↪ num * 0.6. server.snap-max-write-bytes-per-sec: 300MiB # Default to 100
↪ MiB.
```

The configuration change takes effect after restarting the TiFlash and TiKV instances. The TiKV configuration can be also changed online by using the Dynamic Config SQL statement, which takes effect immediately without restarting TiKV instances:

```
sql SET CONFIG tikv `server.snap-max-write-bytes-per-sec` = '300MiB';
```

After adjusting the preceding configurations, you cannot observe the acceleration for now, as the replication speed is still restricted by the PD limit globally.

2. Use PD Control to progressively ease the new replica speed limit.

The default new replica speed limit is 30, which means, approximately 30 Regions add TiFlash replicas every minute. Executing the following command will adjust the limit to 60 for all TiFlash instances, which doubles the original speed:

```
shell  tiup ctl:v<CLUSTER_VERSION> pd -u http://<PD_ADDRESS>:2379 store
↪  limit all engine tiflash 60 add-peer
```

> In the preceding command, you need to replace `<CLUSTER_VERSION>` with the actual cluster version and `<PD_ADDRESS>:2379` with the address of any PD node. For example:
>
> ```
> tiup ctl:v6.1.7 pd -u http://192.168.1.4:2379 store limit all
>    ↪ engine tiflash 60 add-peer
> ```

Within a few minutes, you will observe a significant increase in CPU and disk IO resource usage of the TiFlash nodes, and TiFlash should create replicas faster. At the same time, the TiKV nodes' CPU and disk IO resource usage increases as well.

If the TiKV and TiFlash nodes still have spare resources at this point and the latency of your online service does not increase significantly, you can further ease the limit, for example, triple the original speed:

```
shell  tiup ctl:v<CLUSTER_VERSION> pd -u http://<PD_ADDRESS>:2379 store
↪  limit all engine tiflash 90 add-peer
```

3. After the TiFlash replication is complete, revert to the default configuration to reduce the impact on online services.

Execute the following PD Control command to restore the default new replica speed limit:

```
shell  tiup ctl:v<CLUSTER_VERSION> pd -u http://<PD_ADDRESS>:2379 store
↪  limit all engine tiflash 30 add-peer
```

Comment out the changed configuration in TiUP to restore the default snapshot write speed limit:

```
yaml  # tikv:  #  server.snap-max-write-bytes-per-sec: 300MiB # tiflash
↪ -learner: # raftstore.snap-handle-pool-size: 10 # raftstore.apply-low-
↪ priority-pool-size: 10 # server.snap-max-write-bytes-per-sec: 300MiB
```

14.3.2.4 Set available zones

When configuring replicas, if you need to distribute TiFlash replicas to multiple data centers for disaster recovery, you can configure available zones by following the steps below:

1. Specify labels for TiFlash nodes in the cluster configuration file.

```
tiflash_servers:
  - host: 172.16.5.81
      logger.level: "info"
    learner_config:
      server.labels:
        zone: "z1"
  - host: 172.16.5.82
    config:
      logger.level: "info"
    learner_config:
      server.labels:
        zone: "z1"
  - host: 172.16.5.85
    config:
      logger.level: "info"
    learner_config:
      server.labels:
        zone: "z2"
```

Note that the `flash.proxy.labels` configuration in earlier versions cannot handle special characters in the available zone name correctly. It is recommended to use the `server.labels` in `learner_config` to configure the name of an available zone.

2. After starting a cluster, specify the labels when creating replicas.

```
ALTER TABLE table_name SET TIFLASH REPLICA count LOCATION LABELS
    ↪ location_labels;
```

For example:

```
ALTER TABLE t SET TIFLASH REPLICA 2 LOCATION LABELS "zone";
```

3. PD schedules the replicas based on the labels. In this example, PD respectively schedules two replicas of the table `t` to two available zones. You can use pd-ctl to view the scheduling.

```
> tiup ctl:v<CLUSTER_VERSION> pd -u http://<PD_ADDRESS>:2379 store


    ...
    "address": "172.16.5.82:23913",
    "labels": [
      { "key": "engine", "value": "tiflash"},
      { "key": "zone", "value": "z1" }
    ],
    "region_count": 4,
```

```
    ...
    "address": "172.16.5.81:23913",
    "labels": [
      { "key": "engine", "value": "tiflash"},
      { "key": "zone", "value": "z1" }
    ],
    "region_count": 5,
    ...

    "address": "172.16.5.85:23913",
    "labels": [
      { "key": "engine", "value": "tiflash"},
      { "key": "zone", "value": "z2" }
    ],
    "region_count": 9,
    ...
```

For more information about scheduling replicas by using labels, see Schedule Replicas by Topology Labels, Multiple Data Centers in One City Deployment, and Three Data Centers in Two Cities Deployment.

14.3.3 Use TiDB to Read TiFlash Replicas

This document introduces how to use TiDB to read TiFlash replicas.

TiDB provides three ways to read TiFlash replicas. If you have added a TiFlash replica without any engine configuration, the CBO (cost-based optimization) mode is used by default.

14.3.3.1 Smart selection

For tables with TiFlash replicas, the TiDB optimizer automatically determines whether to use TiFlash replicas based on the cost estimation. You can use the `desc` or `explain analyze` statement to check whether or not a TiFlash replica is selected. For example:

```
desc select count(*) from test.t;
```

```
+-------------------------+---------+-------------+--------------+-------------------
    ↪
| id                      | estRows | task        | access object | operator
    ↪ info            |
+-------------------------+---------+-------------+--------------+-------------------
    ↪
| StreamAgg_9             | 1.00    | root        |               | funcs:count(1)
    ↪ ->Column#4    |
```

```
|  └─TableReader_17       | 1.00   | root       |              | data:
   ↪ TableFullScan_16      |
|    └─TableFullScan_16   | 1.00   | cop[tiflash] | table:t    | keep order:
   ↪ false, stats:pseudo |
+-------------------------+--------+--------------+------------+-----------------
   ↪
3 rows in set (0.00 sec)
```

```
explain analyze select count(*) from test.t;
```

```
+-------------------------+---------+---------+--------------+--------------+---------
   ↪
| id                      | estRows | actRows | task     | access object |
   ↪ execution info                                          | operator
   ↪ info               | memory   | disk |
+-------------------------+---------+---------+--------------+--------------+---------
   ↪
| StreamAgg_9             | 1.00   | 1      | root       |              | time
   ↪ :83.8372ms, loops:2                                   | funcs:count
   ↪ (1)->Column#4   | 372 Bytes | N/A |
|  └─TableReader_17       | 1.00   | 1      | root       |              | time
   ↪ :83.7776ms, loops:2, rpc num: 1, rpc time:83.5701ms, proc keys:0 |
   ↪ data:TableFullScan_16 | 152 Bytes | N/A |
|    └─TableFullScan_16   | 1.00   | 1      | cop[tiflash] | table:t    | time
   ↪ :43ms, loops:1                                        | keep order:
   ↪ false, stats:pseudo | N/A | N/A |
+-------------------------+---------+---------+--------------+--------------+---------
   ↪
```

`cop[tiflash]` means that the task will be sent to TiFlash for processing. If you have not selected a TiFlash replica, you can try to update the statistics using the `analyze table` statement, and then check the result using the `explain analyze` statement.

Note that if a table has only a single TiFlash replica and the related node cannot provide service, queries in the CBO mode will repeatedly retry. In this situation, you need to specify the engine or use the manual hint to read data from the TiKV replica.

14.3.3.2 Engine isolation

Engine isolation is to specify that all queries use a replica of the specified engine by configuring the corresponding variable. The optional engines are “tikv”, “tidb” (indicates the internal memory table area of TiDB, which stores some TiDB system tables and cannot be actively used by users), and “tiflash”.

You can specify the engines at the following two configuration levels:

- TiDB instance-level, namely, INSTANCE level. Add the following configuration item in the TiDB configuration file:

```
[isolation-read]
engines = ["tikv", "tidb", "tiflash"]
```

The INSTANCE-level default configuration is `["tikv", "tidb", "tiflash"]` .

- SESSION level. Use the following statement to configure:

```
set @@session.tidb_isolation_read_engines = "engine list separated by
    ↪ commas";
```

or

```
set SESSION tidb_isolation_read_engines = "engine list separated by
    ↪ commas";
```

The default configuration of the SESSION level inherits from the configuration of the TiDB INSTANCE level.

The final engine configuration is the session-level configuration, that is, the session-level configuration overrides the instance-level configuration. For example, if you have configured "tikv" in the INSTANCE level and "tiflash" in the SESSION level, then the TiFlash replicas are read. If the final engine configuration is "tikv" and "tiflash", then the TiKV and TiFlash replicas are both read, and the optimizer automatically selects a better engine to execute.

> **Note:**
>
> Because TiDB Dashboard and other components need to read some system tables stored in the TiDB memory table area, it is recommended to always add the "tidb" engine to the instance-level engine configuration.

If the queried table does not have a replica of the specified engine (for example, the engine is configured as "tiflash" but the table does not have a TiFlash replica), the query returns an error.

14.3.3.3 Manual hint

Manual hint can force TiDB to use specified replicas for specific table(s) on the premise of satisfying engine isolation. Here is an example of using the manual hint:

```
select /*+ read_from_storage(tiflash[table_name]) */ ... from table_name;
```

If you set an alias to a table in a query statement, you must use the alias in the statement that includes a hint for the hint to take effect. For example:

```
select /*+ read_from_storage(tiflash[alias_a,alias_b]) */ ... from
    ↪ table_name_1 as alias_a, table_name_2 as alias_b where alias_a.
    ↪ column_1 = alias_b.column_2;
```

In the above statements, `tiflash[]` prompts the optimizer to read the TiFlash replicas. You can also use `tikv[]` to prompt the optimizer to read the TiKV replicas as needed. For hint syntax details, refer to READ_FROM_STORAGE.

If the table specified by a hint does not have a replica of the specified engine, the hint is ignored and a warning is reported. In addition, a hint only takes effect on the premise of engine isolation. If the engine specified in a hint is not in the engine isolation list, the hint is also ignored and a warning is reported.

> **Note:**
>
> The MySQL client of 5.7.7 or earlier versions clears optimizer hints by default. To use the hint syntax in these early versions, start the client with the `--comments` option, for example, `mysql -h 127.0.0.1 -P 4000 -uroot --comments`.

14.3.3.4 The relationship of smart selection, engine isolation, and manual hint

In the above three ways of reading TiFlash replicas, engine isolation specifies the overall range of available replicas of engines; within this range, manual hint provides statement-level and table-level engine selection that is more fine-grained; finally, CBO makes the decision and selects a replica of an engine based on cost estimation within the specified engine list.

> **Note:**
>
> Before v4.0.3, the behavior of reading from TiFlash replica in a non-read-only SQL statement (for example, `INSERT INTO ... SELECT`, `SELECT ... FOR UPDATE`, `UPDATE ...`, `DELETE ...`) is undefined. In v4.0.3 and later versions, internally TiDB ignores the TiFlash replica for a non-read-only SQL statement to guarantee the data correctness. That is, for smart selection, TiDB automatically selects the non-TiFlash replica; for engine isolation that specifies TiFlash replica **only**, TiDB reports an error; and for manual hint, TiDB ignores the hint.

14.3.4 Use TiSpark to Read TiFlash Replicas

This document introduces how to use TiSpark to read TiFlash replicas.

Currently, you can use TiSpark to read TiFlash replicas in a method similar to the engine isolation in TiDB. This method is to configure the `spark.tispark.isolation_read_engines` parameter. The parameter value defaults to `tikv,tiflash`, which means that TiDB reads data from TiFlash or from TiKV according to CBO's selection. If you set the parameter value to `tiflash`, it means that TiDB forcibly reads data from TiFlash.

> **Note:**
>
> When this parameter is set to `tiflash`, only the TiFlash replicas of all tables involved in the query are read and these tables must have TiFlash replicas; for tables that do not have TiFlash replicas, an error is reported. When this parameter is set to `tikv`, only the TiKV replica is read.

You can configure this parameter in one of the following ways:

- Add the following item in the `spark-defaults.conf` file:

```
spark.tispark.isolation_read_engines tiflash
```

- Add `--conf spark.tispark.isolation_read_engines=tiflash` in the initialization command when initializing Spark shell or Thrift server.
- Set `spark.conf.set("spark.tispark.isolation_read_engines", "tiflash")` in Spark shell in a real-time manner.
- Set `set spark.tispark.isolation_read_engines=tiflash` in Thrift server after the server is connected via beeline.

14.3.5 Use TiFlash MPP Mode

This document introduces the MPP mode of TiFlash and how to use it.

TiFlash supports using the MPP mode to execute queries, which introduces cross-node data exchange (data shuffle process) into the computation. TiDB automatically determines whether to select the MPP mode using the optimizer's cost estimation. You can change the selection strategy by modifying the values of `tidb_allow_mpp` and `tidb_enforce_mpp`.

14.3.5.1 Control whether to select the MPP mode

The `tidb_allow_mpp` variable controls whether TiDB can select the MPP mode to execute queries. The `tidb_enforce_mpp` variable controls whether the optimizer's cost estimation is ignored and the MPP mode of TiFlash is forcibly used to execute queries.

The results corresponding to all values of these two variables are as follows:

	tidb_allow_mpp=off	tidb_allow_mpp=on (by default)
tidb_enforce_mpp=off (by default)	The MPP mode is not used.	The optimizer selects the MPP mode based on cost estimation. (by default)
tidb_enforce_mpp=on	The MPP mode is not used.	TiDB ignores the cost estimation and selects the MPP mode.

For example, if you do not want to use the MPP mode, you can execute the following statements:

```
set @@session.tidb_allow_mpp=0;
```

If you want TiDB's cost-based optimizer to automatically decide whether to use the MPP mode (by default), you can execute the following statements:

```
set @@session.tidb_allow_mpp=1;
set @@session.tidb_enforce_mpp=0;
```

If you want TiDB to ignore the optimizer's cost estimation and to forcibly select the MPP mode, you can execute the following statements:

```
set @@session.tidb_allow_mpp=1;
set @@session.tidb_enforce_mpp=1;
```

The initial value of the `tidb_enforce_mpp` session variable is equal to the `enforce-mpp` configuration value of this tidb-server instance (which is `false` by default). If multiple tidb-server instances in a TiDB cluster only perform analytical queries and you want to make sure that the MPP mode is used on these instances, you can change their `enforce-mpp` configuration values to `true`.

> **Note:**
>
> When `tidb_enforce_mpp=1` takes effect, the TiDB optimizer will ignore the cost estimation to choose the MPP mode. However, if other factors block the MPP mode, TiDB will not select the MPP mode. These factors include the absence of TiFlash replica, unfinished replication of TiFlash replicas, and

statements containing operators or functions that are not supported by the MPP mode.

If TiDB optimizer cannot select the MPP mode due to reasons other than cost estimation, when you use the `EXPLAIN` statement to check out the execution plan, a warning is returned to explain the reason. For example:

```
set @@session.tidb_enforce_mpp=1;
create table t(a int);
explain select count(*) from t;
show warnings;
```

```
+---------+------+-----------------------------------------------------------------
    ↪
| Level   | Code | Message
    ↪
    ↪ |
+---------+------+-----------------------------------------------------------------
    ↪
| Warning | 1105 | MPP mode may be blocked because there aren't
    ↪  tiflash replicas of table `t`. |
+---------+------+-----------------------------------------------------------------
    ↪
```

14.3.5.2 Algorithm support for the MPP mode

The MPP mode supports these physical algorithms: Broadcast Hash Join, Shuffled Hash Join, Shuffled Hash Aggregation, Union All, TopN, and Limit. The optimizer automatically determines which algorithm to be used in a query. To check the specific query execution plan, you can execute the `EXPLAIN` statement. If the result of the `EXPLAIN` statement shows ExchangeSender and ExchangeReceiver operators, it indicates that the MPP mode has taken effect.

The following statement takes the table structure in the TPC-H test set as an example:

```
explain select count(*) from customer c join nation n on c.c_nationkey=n.
    ↪ n_nationkey;
+--
    ↪ ----------------------------------+-----------+--------------+-----------
    ↪
| id                                  | estRows   | task         | access
    ↪ object | operator info
    ↪                                                          |
+--
```

```
    ↪ ----------------------------------------+-----------+--------------+-----------
    ↪
| HashAgg_23                            | 1.00      | root         |
    ↪              | funcs:count(Column#16)->Column#15
    ↪                                    |
|  ─TableReader_25                      | 1.00      | root         |
    ↪              | data:ExchangeSender_24
    ↪                                                |
|    ─ExchangeSender_24                 | 1.00      | mpp[tiflash] |
    ↪              | ExchangeType: PassThrough
    ↪                                              |
|      ─HashAgg_12                      | 1.00      | mpp[tiflash] |
    ↪              | funcs:count(1)->Column#16
    ↪                                             |
|        ─HashJoin_17                   | 3000000.00 | mpp[tiflash] |
    ↪           | inner join, equal:[eq(tpch.nation.n_nationkey, tpch.
    ↪ customer.c_nationkey)] |
|          ─ExchangeReceiver_21(Build) | 25.00     | mpp[tiflash] |
    ↪              |
    ↪                                                                    |
|            ─ExchangeSender_20         | 25.00     | mpp[tiflash] |
    ↪              | ExchangeType: Broadcast
    ↪                                               |
|              ─TableFullScan_18        | 25.00     | mpp[tiflash] | table:n
    ↪    | keep order:false
    ↪ |
|          ─TableFullScan_22(Probe)     | 3000000.00 | mpp[tiflash] | table:c
    ↪   | keep order:false                                                |
+--
    ↪ ----------------------------------------+-----------+--------------+-----------
    ↪
9 rows in set (0.00 sec)
```

In the example execution plan, the `ExchangeReceiver` and `ExchangeSender` operators are included. The execution plan indicates that after the `nation` table is read, the `ExchangeSender` operator broadcasts the table to each node, the `HashJoin` and `HashAgg` operations are performed on the `nation` table and the `customer` table, and then the results are returned to TiDB.

TiFlash provides the following two global/session variables to control whether to use Broadcast Hash Join:

- `tidb_broadcast_join_threshold_size`: The unit of the value is bytes. If the table size (in the unit of bytes) is less than the value of the variable, the Broadcast Hash Join algorithm is used. Otherwise, the Shuffled Hash Join algorithm is used.

- `tidb_broadcast_join_threshold_count`: The unit of the value is rows. If the objects of the join operation belong to a subquery, the optimizer cannot estimate the size of the subquery result set, so the size is determined by the number of rows in the result set. If the estimated number of rows in the subquery is less than the value of this variable, the Broadcast Hash Join algorithm is used. Otherwise, the Shuffled Hash Join algorithm is used.

14.3.5.3 Access partitioned tables in the MPP mode

To access partitioned tables in the MPP mode, you need to enable dynamic pruning mode first.

Example:

```
mysql> DROP TABLE if exists test.employees;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE test.employees
(id int(11) NOT NULL,
 fname varchar(30) DEFAULT NULL,
 lname varchar(30) DEFAULT NULL,
 hired date NOT NULL DEFAULT '1970-01-01',
 separated date DEFAULT '9999-12-31',
 job_code int DEFAULT NULL,
 store_id int NOT NULL) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=
    ↪ utf8mb4_bin
PARTITION BY RANGE (store_id)
(PARTITION p0 VALUES LESS THAN (6),
 PARTITION p1 VALUES LESS THAN (11),
 PARTITION p2 VALUES LESS THAN (16),
 PARTITION p3 VALUES LESS THAN (MAXVALUE));
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER table test.employees SET tiflash replica 1;
Query OK, 0 rows affected (0.09 sec)

mysql> SET tidb_partition_prune_mode=static;
Query OK, 0 rows affected (0.00 sec)

mysql> explain SELECT count(*) FROM test.employees;
+--
   ↪ ---------------------------------+----------+-------------------+---------------
   ↪
| id                              | estRows | task           | access object
   ↪             | operator info                 |
+--
```

```
    ↪ ------------------------------+---------+-----------------+---------------
    ↪
| HashAgg_18                    | 1.00    | root            |
    ↪                             | funcs:count(Column#10)->Column#9 |
|  ─PartitionUnion_20           | 4.00    | root            |
    ↪                             |                                  |
|    ─StreamAgg_35              | 1.00    | root            |
    ↪                             | funcs:count(Column#12)->Column#10 |
|      ─TableReader_36          | 1.00    | root            |
    ↪                             | data:StreamAgg_26                |
|        ─StreamAgg_26          | 1.00    | batchCop[tiflash] |
    ↪                           | funcs:count(1)->Column#12        |
|          ─TableFullScan_34    | 10000.00 | batchCop[tiflash] | table:
    ↪ employees, partition:p0 | keep order:false, stats:pseudo |
|    ─StreamAgg_52              | 1.00    | root            |
    ↪                             | funcs:count(Column#14)->Column#10 |
|      ─TableReader_53          | 1.00    | root            |
    ↪                             | data:StreamAgg_43                |
|        ─StreamAgg_43          | 1.00    | batchCop[tiflash] |
    ↪                           | funcs:count(1)->Column#14        |
|          ─TableFullScan_51    | 10000.00 | batchCop[tiflash] | table:
    ↪ employees, partition:p1 | keep order:false, stats:pseudo |
|    ─StreamAgg_69              | 1.00    | root            |
    ↪                             | funcs:count(Column#16)->Column#10 |
|      ─TableReader_70          | 1.00    | root            |
    ↪                             | data:StreamAgg_60                |
|        ─StreamAgg_60          | 1.00    | batchCop[tiflash] |
    ↪                           | funcs:count(1)->Column#16        |
|          ─TableFullScan_68    | 10000.00 | batchCop[tiflash] | table:
    ↪ employees, partition:p2 | keep order:false, stats:pseudo |
|    ─StreamAgg_86              | 1.00    | root            |
    ↪                             | funcs:count(Column#18)->Column#10 |
|      ─TableReader_87          | 1.00    | root            |
    ↪                             | data:StreamAgg_77                |
|        ─StreamAgg_77          | 1.00    | batchCop[tiflash] |
    ↪                           | funcs:count(1)->Column#18        |
|          ─TableFullScan_85    | 10000.00 | batchCop[tiflash] | table:
    ↪ employees, partition:p3 | keep order:false, stats:pseudo |
+--
    ↪ ------------------------------+---------+-----------------+---------------
    ↪
18 rows in set (0,00 sec)

mysql> SET tidb_partition_prune_mode=dynamic;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> explain SELECT count(*) FROM test.employees;
+--
   ↪ ---------------------------+---------+-------------+---------------+-------
   ↪
| id                          | estRows | task        | access object | operator
   ↪  info                                    |
+--
   ↪ ---------------------------+---------+-------------+---------------+-------
   ↪
| HashAgg_17                  | 1.00    | root        |               | funcs:
   ↪ count(Column#11)->Column#9                |
|  ─TableReader_19            | 1.00    | root        | partition:all | data:
   ↪ ExchangeSender_18                        |
|    ─ExchangeSender_18       | 1.00    | mpp[tiflash] |              |
   ↪ ExchangeType: PassThrough                  |
|      ─HashAgg_8             | 1.00    | mpp[tiflash] |              | funcs:
   ↪ count(1)->Column#11                       |
|        ─TableFullScan_16    | 10000.00 | mpp[tiflash] | table:employees |
   ↪ keep order:false, stats:pseudo, PartitionTableScan:true |
+--
   ↪ ---------------------------+---------+-------------+---------------+-------
   ↪
5 rows in set (0,00 sec)
```

14.3.5.4 Known issues of MPP

In the current version, TiFlash uses the `start_ts` of a query as the unique key of the query. In most cases, the `start_ts` of each query can uniquely identify a query, but in the following cases, different queries have the same `start_ts`:

- All queries in the same transaction have the same `start_ts`.
- When you use `tidb_snapshot` to read data at a specific historical time point, the same time point is manually specified.
- When Stale Read is enabled, the same time point is manually specified.

When `start_ts` cannot uniquely represent the MPP query, if TiFlash detects that different queries have the same `start_ts` at a given time, TiFlash might report an error. Typical error cases are as follows:

- When multiple queries with the same `start_ts` are sent to TiFlash at the same time, you might encounter the `task has been registered` error.

- When multiple simple queries with `LIMIT` are executed continuously in the same transaction, once the `LIMIT` condition is met, TiDB sends a cancel request to TiFlash to cancel the query. This request also uses `start_ts` to identify the query to be canceled. If there are other queries with the same `start_ts` in TiFlash, these queries might be canceled by mistake. An example of this issue can be found in #43426.

This issue is fixed in TiDB v6.6.0. It is recommended to use the latest LTS version.

14.3.6 Push-down Calculations Supported by TiFlash

This document introduces the push-down calculations supported by TiFlash.

14.3.6.1 Push-down operators

TiFlash supports the push-down of the following operators:

- TableScan: Reads data from tables.
- Selection: Filters data.
- HashAgg: Performs data aggregation based on the Hash Aggregation algorithm.
- StreamAgg: Performs data aggregation based on the Stream Aggregation algorithm. SteamAgg only supports the aggregation without the `GROUP BY` condition.
- TopN: Performs the TopN calculation.
- Limit: Performs the limit calculation.
- Project: Performs the projection calculation.
- HashJoin: Performs the join calculation using the Hash Join algorithm, but with the following conditions:
    - The operator can be pushed down only in the MPP mode.
    - Supported joins are Inner Join, Left Join, Semi Join, Anti Semi Join, Left Semi Join, and Anti Left Semi Join.
    - The preceding joins support both Equi Join and Non-Equi Join (Cartesian Join). When calculating Cartesian Join, the Broadcast algorithm, instead of the Shuffle Hash Join algorithm, is used.
- Window functions: Currently, TiFlash supports row_number(), rank(), and dense_rank().

In TiDB, operators are organized in a tree structure. For an operator to be pushed down to TiFlash, all of the following prerequisites must be met:

- All of its child operators can be pushed down to TiFlash.
- If an operator contains expressions (most of the operators contain expressions), all expressions of the operator can be pushed down to TiFlash.

14.3.6.2 Push-down expressions

TiFlash supports the following push-down expressions:

- Mathematical functions: `+, -, /, *, %, >=, <=, =, !=, <, >, round, abs, floor(int), ceil(int), ceiling(int), sqrt, log, log2, log10, ln, exp, pow, sign, radians, degrees, conv, crc32, greatest(int/real), least(int/real)`
- Logical functions: `and, or, not, case when, if, ifnull, isnull, in, like, coalesce, is`
- Bitwise operations: `bitand, bitor, bigneg, bitxor`
- String functions: `substr, char_length, replace, concat, concat_ws, left, right, ascii, length, trim, ltrim, rtrim, position, format, lower, ucase, upper, substring_index, lpad, rpad, strcmp, regexp`
- Date functions: `date_format, timestampdiff, from_unixtime, unix_timestamp (int), unix_timestamp(decimal), str_to_date(date), str_to_date(datetime), datediff, year, month, day, extract(datetime), date, hour , microsecond, minute, second, sysdate, date_add/adddate(datetime , int), date_add/adddate(string, int), date_add/adddate(string, real), date_sub/subdate(datetime, int), date_sub/subdate(string, int), date_sub/subdate(string, real), quarter, dayname, dayofmonth , dayofweek, dayofyear, last_day, monthname, to_seconds, to_days, from_days, weekofyear`
- JSON function: `json_length`
- Conversion functions: `cast(int as double), cast(int as decimal), cast(int as string), cast(int as time), cast(double as int), cast(double as decimal), cast(double as string), cast(double as time), cast(string as int), cast(string as double), cast(string as decimal), cast(string as time), cast(decimal as int), cast(decimal as string), cast (decimal as time), cast(time as int), cast(time as decimal), cast(time as string), cast(time as real)`
- Aggregate functions: `min, max, sum, count, avg, approx_count_distinct, group_concat`
- Miscellaneous functions: `inetntoa, inetaton, inet6ntoa, inet6aton`

14.3.6.3 Restrictions

- Expressions that contain the Bit, Set, and Geometry types cannot be pushed down to TiFlash.
- The `date_add`, `date_sub`, `adddate`, and `subdate` functions support the following interval types only. If other interval types are used, TiFlash reports errors.
  - DAY
  - WEEK

- MONTH
- YEAR
- HOUR
- MINUTE
- SECOND

- Window functions with the `ROWS` or `RANGE` type of frame cannot be pushed down to TiFlash.

If a query encounters unsupported push-down calculations, TiDB needs to complete the remaining calculations, which might greatly affect the TiFlash acceleration effect. The currently unsupported operators and expressions might be supported in future versions.

14.3.7 TiFlash Data validation

This document introduces the data validation mechanism and tools for TiFlash.

Data corruptions are usually caused by serious hardware failures. In such cases, even if you attempt to manually recover data, your data become less reliable.

To ensure data integrity, by default, TiFlash performs basic data validation on data files, using the `City128` algorithm. In the event of any data validation failure, TiFlash immediately reports an error and exits, avoiding secondary disasters caused by inconsistent data. At this time, you need to manually intervene and replicate the data again before you can restore the TiFlash node.

Starting from v5.4.0, TiFlash introduces more advanced data validation features. TiFlash uses the `XXH3` algorithm by default and allows you to customize the validation frame and algorithm.

14.3.7.1 Validation mechanism

The validation mechanism builds upon the DeltaTree File (DTFile). DTFile is the storage file that persists TiFlash data. DTFile has three formats:

Version	State	Validation mechanism	Notes
V1	Deprecated	Hashes are embedded in data files.	
V2	Default for versions < v6.0.0	Hashes are embedded in data files.	Compared to V1, V2 adds statistics of column data.
V3	Default for versions >= v6.0.0	V3 contains metadata and token data checksum, and supports multiple hash algorithms.	New in v5.4.0.

DTFile is stored in the `stable` folder in the data file directory. All formats currently enabled are in folder format, which means the data is stored in multiple files under a folder with a name like `dmf_<file id>`.

14.3.7.1.1 Use data validation

TiFlash supports both automatic and manual data validation:

- Automatic data validation:
  - v6.0.0 and later versions use the V3 validation mechanism by default.
  - Versions earlier than v6.0.0 use the V2 validation mechanism by default.
  - To manually switch the validation mechanism, refer to TiFlash configuration file. However, the default configuration is verified by tests and therefore recommended.
- Manual data validation. Refer to `DTTool inspect`.

> **Warning:**
>
> After you enable the V3 validation mechanism, the newly generated DTFile cannot be directly read by TiFlash earlier than v5.4.0. Since v5.4.0, TiFlash supports both V2 and V3 and does not actively upgrade or downgrade versions. If you need to upgrade or downgrade versions for existing files, you need to manually switch versions.

14.3.7.1.2 Validation tool

In addition to automatic data validation performed when TiFlash reads data, a tool for manually checking data integrity is introduced in v5.4.0. For details, refer to DTTool.

14.3.8 TiFlash MinTSO Scheduler

The TiFlash MinTSO scheduler is a distributed scheduler for MPP (Massively Parallel Processing) tasks in TiFlash. This document describes the implementation principles of the TiFlash MinTSO scheduler.

14.3.8.1 Background

When processing an MPP query, TiDB splits the query into one or more MPP tasks and sends these MPP tasks to the corresponding TiFlash nodes for compilation and execution. TiFlash needs to use several threads to execute each MPP task, with the specific number of threads depending on the complexity of the MPP task and the concurrency parameters set in TiFlash.

In high concurrency scenarios, TiFlash nodes receive multiple MPP tasks simultaneously. If the execution of MPP tasks is not controlled, the number of threads that TiFlash needs to request from the system will increase linearly along with the increasing number of MPP tasks. Too many threads can affect the execution efficiency of TiFlash, and because the operating system itself supports a limited number of threads, TiFlash will encounter errors when it requests more threads than the operating system can provide.

To improve TiFlash's processing capability in high concurrency scenarios, an MPP task scheduler needs to be introduced into TiFlash.

14.3.8.2 Implementation principles

As mentioned in the background, the initial purpose of introducing the TiFlash task scheduler is to control the number of threads used during MPP query execution. A simple scheduling strategy is to specify the maximum number of threads TiFlash can request. For each MPP task, the scheduler decides whether the MPP task can be scheduled based on the current number of threads used by the system and the expected number of threads the MPP task will use:

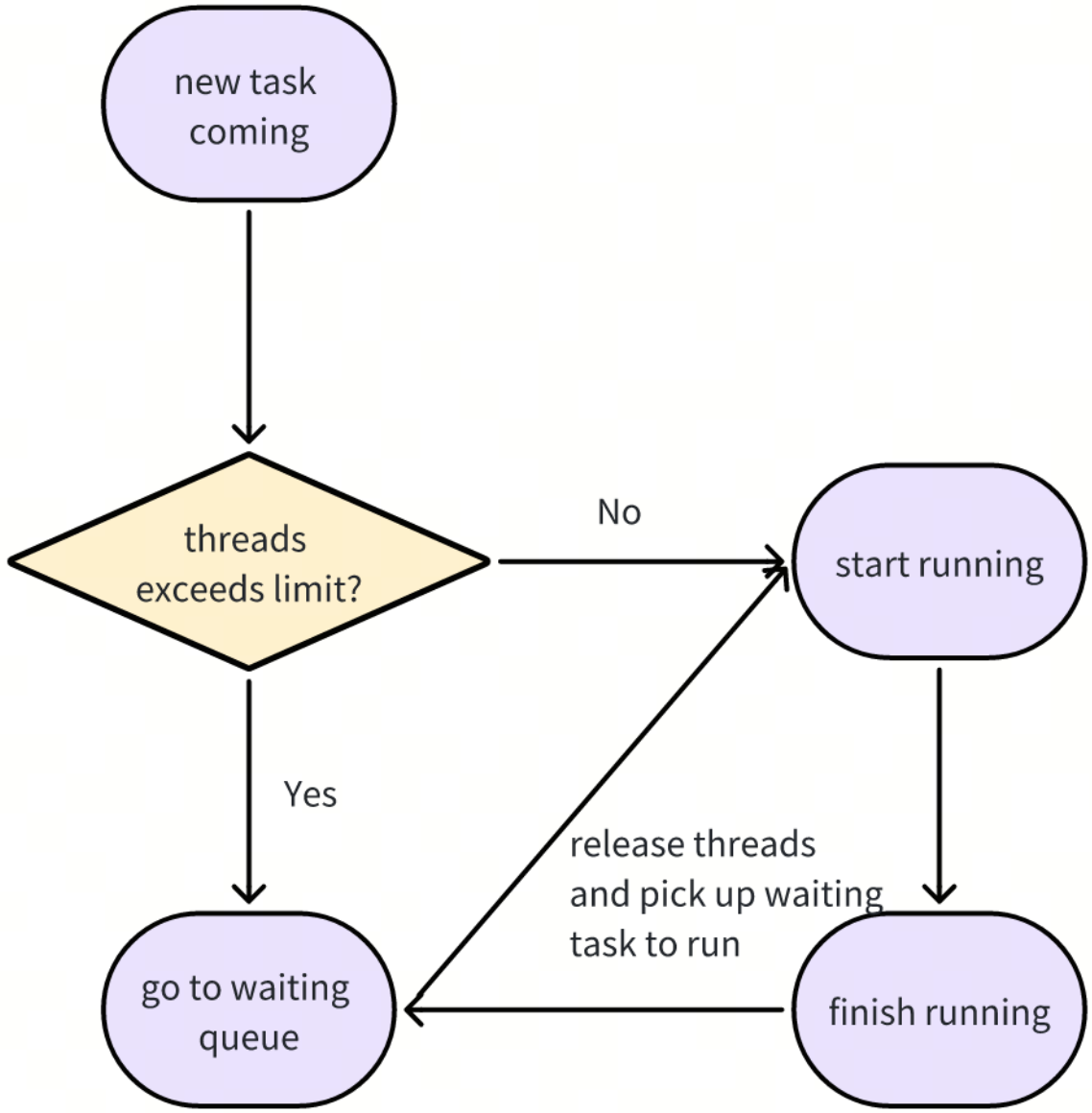


Figure 328: TiFlash MinTSO Scheduler v1

Although the preceding scheduling strategy can effectively control the number of system threads, an MPP task is not the smallest independent execution unit, and dependencies exist between different MPP tasks:

```
EXPLAIN SELECT count(*) FROM t0 a JOIN t0 b ON a.id = b.id;
```

```
+-------------------------------------------+----------+--------------+-----------------+
    ↪
```

```
| id                                    | estRows | task       | access
   ↪ object | operator info                                    |
+-------------------------------------------+----------+--------------+---------------+
   ↪
| HashAgg_44                            | 1.00    | root       |
   ↪             | funcs:count(Column#8)->Column#7                  |
|  ─TableReader_46                      | 1.00    | root       |
   ↪             | MppVersion: 2, data:ExchangeSender_45            |
|    ─ExchangeSender_45                 | 1.00    | mpp[tiflash] |
   ↪           | ExchangeType: PassThrough                        |
|      ─HashAgg_13                      | 1.00    | mpp[tiflash] |
   ↪           | funcs:count(1)->Column#8                         |
|        ─Projection_43                 | 12487.50 | mpp[tiflash] |
   ↪          | test.t0.id                                       |
|          ─HashJoin_42                 | 12487.50 | mpp[tiflash] |
   ↪          | inner join, equal:[eq(test.t0.id, test.t0.id)]   |
|            ─ExchangeReceiver_22(Build) | 9990.00 | mpp[tiflash] |
   ↪            |                                                |
|              ─ExchangeSender_21       | 9990.00 | mpp[tiflash] |
   ↪            | ExchangeType: Broadcast, Compression: FAST       |
|                ─Selection_20          | 9990.00 | mpp[tiflash] |
   ↪            | not(isnull(test.t0.id))                          |
|                  ─TableFullScan_19    | 10000.00 | mpp[tiflash] | table:a
   ↪    | pushed down filter:empty, keep order:false, stats:pseudo |
|            ─Selection_24(Probe)       | 9990.00 | mpp[tiflash] |
   ↪            | not(isnull(test.t0.id))                          |
|              ─TableFullScan_23        | 10000.00 | mpp[tiflash] | table:b
   ↪    | pushed down filter:empty, keep order:false, stats:pseudo |
+-------------------------------------------+----------+--------------+---------------+
   ↪
```

For example, the preceding query generates two MPP tasks on each TiFlash node, where the MPP task containing the `ExchangeSender_45` executor depends on the MPP task containing the `ExchangeSender_21` executor. In high concurrency scenarios, if the scheduler schedules the MPP task containing `ExchangeSender_45` for each query, the system will enter a deadlock state.

To avoid deadlock, TiFlash introduces the following two levels of thread limits:

- thread_soft_limit: used to limit the number of threads used by the system. For specific MPP tasks, this limit can be broken to avoid deadlock.
- thread_hard_limit: used to protect the system. Once the number of threads used by the system exceeds the hard limit, TiFlash will report an error to avoid deadlock.

The soft limit and hard limit work together to avoid deadlock as follows: the soft limit restricts the total number of threads used by all queries, enabling full use of resources while

avoiding thread resource exhaustion; the hard limit ensures that in any situation, at least one query in the system can break the soft limit and continue to acquire thread resources and run, thus avoiding deadlock. As long as the number of threads does not exceed the hard limit, there will always be one query in the system where all its MPP tasks can be executed normally, thus preventing deadlock.

The goal of the MinTSO scheduler is to control the number of system threads while ensuring that there is always one and only one special query in the system, where all its MPP tasks can be scheduled. The MinTSO scheduler is a fully distributed scheduler, with each TiFlash node scheduling MPP tasks based only on its own information. Therefore, all MinTSO schedulers on TiFlash nodes need to identify the same "special" query. In TiDB, each query carries a read timestamp (`start_ts`), and the MinTSO scheduler defines the "special" query as the query with the smallest `start_ts` on the current TiFlash node. Based on the principle that the global minimum is also the local minimum, the "special" query selected by all TiFlash nodes must be the same, called the MinTSO query.

The scheduling process of the MinTSO Scheduler is as follows:

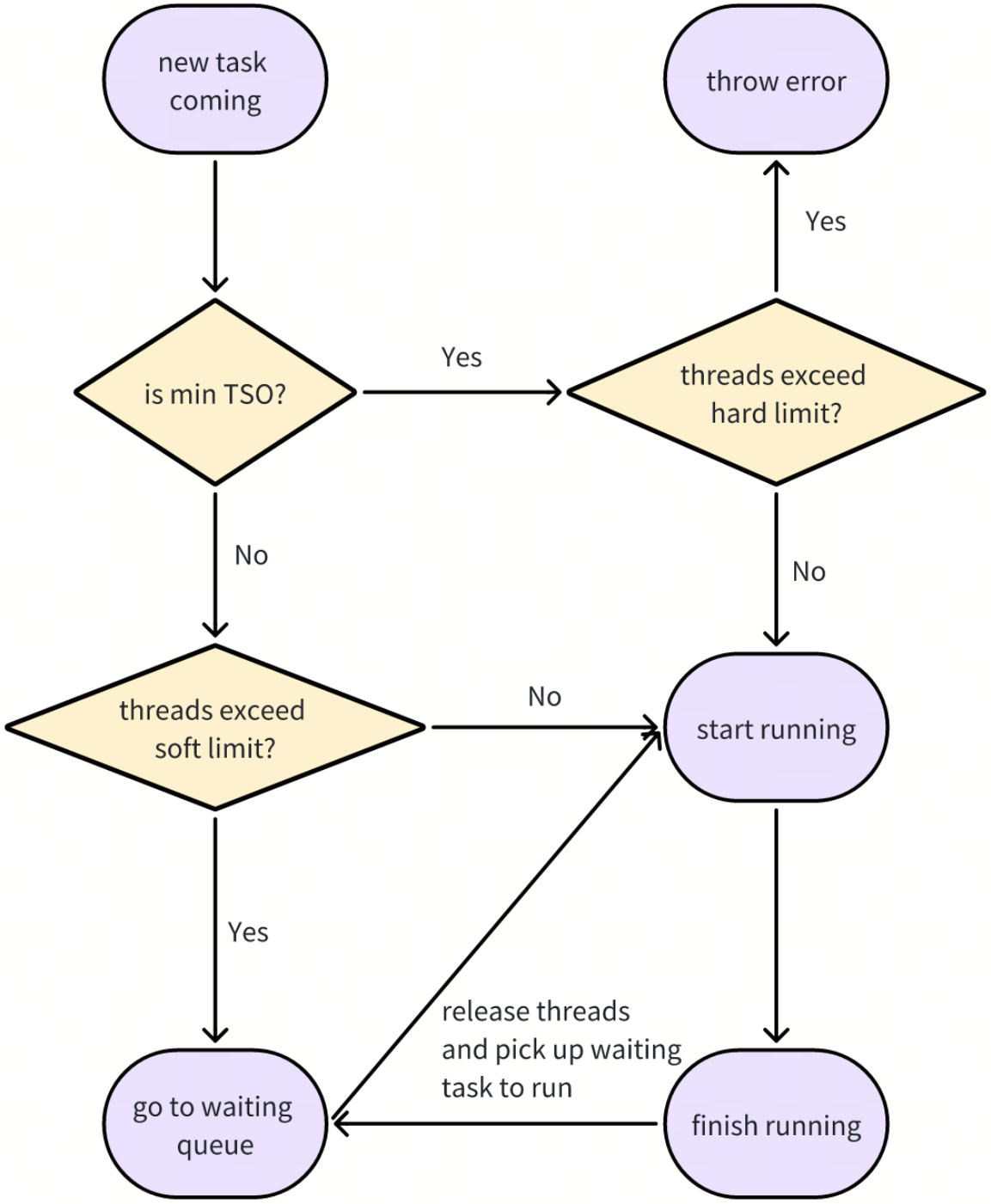


Figure 329: TiFlash MinTSO Scheduler v2

By introducing soft limit and hard limit, the MinTSO scheduler effectively avoids system deadlock while controlling the number of system threads. In high concurrency scenar-

ios, however, most queries might only have part of their MPP tasks scheduled. Queries with only part of MPP tasks scheduled cannot execute normally, leading to low system execution efficiency. To avoid this situation, TiFlash introduces a query-level limit for the MinTSO scheduler, called active_set_soft_limit. This limit allows only MPP tasks of up to active_set_soft_limit queries to participate in scheduling; MPP tasks of other queries do not participate in scheduling, and only after the current queries finish can new queries participate in scheduling. This limit is only a soft limit because for the MinTSO query, all its MPP tasks can be scheduled directly as long as the number of system threads does not exceed the hard limit.

14.3.8.3 See also

- Configure TiFlash: learn how to configure the MinTSO scheduler.

14.3.9 TiFlash Compatibility Notes

TiFlash is incompatible with TiDB in the following situations:

- In the TiFlash computation layer:
  - Checking overflowed numerical values is not supported. For example, adding two maximum values of the `BIGINT` type `9223372036854775807 + 9223372036854775807`. The expected behavior of this calculation in TiDB is to return the `ERROR 1690 (22003): BIGINT value is out of range` error. However, if this calculation is performed in TiFlash, an overflow value of `-2` is returned without any error.
  - The window function is not supported.
  - Reading data from TiKV is not supported.
  - Currently, the `sum` function in TiFlash does not support the string-type argument. But TiDB cannot identify whether any string-type argument has been passed into the `sum` function during the compiling. Therefore, when you execute statements similar to `select sum(string_col)from t`, TiFlash returns the `[FLASH:Coprocessor:Unimplemented] CastStringAsReal is not supported .` error. To avoid such an error in this case, you need to modify this SQL statement to `select sum(cast(string_col as double))from t`.
  - Currently, TiFlash's decimal division calculation is incompatible with that of TiDB. For example, when dividing decimal, TiFlash performs the calculation always using the type inferred from the compiling. However, TiDB performs this calculation using a type that is more precise than that inferred from the compiling. Therefore, some SQL statements involving the decimal division return different execution results when executed in TiDB + TiKV and in TiDB + TiFlash. For example:

```
mysql> create table t (a decimal(3,0), b decimal(10, 0));
Query OK, 0 rows affected (0.07 sec)
mysql> insert into t values (43, 1044774912);
Query OK, 1 row affected (0.03 sec)
mysql> alter table t set tiflash replica 1;
Query OK, 0 rows affected (0.07 sec)
mysql> set session tidb_isolation_read_engines='tikv';
Query OK, 0 rows affected (0.00 sec)
mysql> select a/b, a/b + 0.0000000000001 from t where a/b;
+--------+-----------------------+
| a/b    | a/b + 0.0000000000001 |
+--------+-----------------------+
| 0.0000 |     0.0000000410001 |
+--------+-----------------------+
1 row in set (0.00 sec)
mysql> set session tidb_isolation_read_engines='tiflash';
Query OK, 0 rows affected (0.00 sec)
mysql> select a/b, a/b + 0.0000000000001 from t where a/b;
Empty set (0.01 sec)
```

In the example above, `a/b`'s inferred type from the compiling is `Decimal(7,4)` both in TiDB and in TiFlash. Constrained by `Decimal(7,4)`, `a/b`'s returned type should be `0.0000`. In TiDB, `a/b`'s runtime precision is higher than `Decimal(7,4)`, so the original table data is not filtered by the `where a/b` condition. However, in TiFlash, the calculation of `a/b` uses `Decimal(7,4)` as the result type, so the original table data is filtered by the `where a/b` condition.

14.4 System Variables

TiDB system variables behave similar to MySQL, in that settings apply on a `SESSION` or `GLOBAL` scope:

- Changes on a `SESSION` scope will only affect the current session.
- Changes on a `GLOBAL` scope apply immediately. If this variable is also`SESSION` scoped, all sessions (including your session) will continue to use their current session value.
- Changes are made using the SET statement:

```
## These two identical statements change a session variable
SET tidb_distsql_scan_concurrency = 10;
SET SESSION tidb_distsql_scan_concurrency = 10;

## These two identical statements change a global variable
SET @@global.tidb_distsql_scan_concurrency = 10;
SET GLOBAL tidb_distsql_scan_concurrency = 10;
```

Note:

Several `GLOBAL` variables persist to the TiDB cluster. Some variables in this document have a `Persists to cluster` setting, which can be configured to `Yes` or `No`.

- For variables with the `Persists to cluster: Yes` setting, when a global variable is changed, a notification is sent to all TiDB servers to refresh their system variable cache. When you add additional TiDB servers or restart existing TiDB servers, the persisted configuration value is automatically used.
- For variables with the `Persists to cluster: No` setting, changes only apply to the local TiDB instance that you are connected to. To retain any values set, you need to specify the variables in your `tidb.toml` configuration file.

Additionally, TiDB presents several MySQL variables as both readable and settable. This is required for compatibility, because it is common for both applications and connectors to read MySQL variables. For example, JDBC connectors both read and set query cache settings, despite not relying on the behavior.

Note:

Larger values do not always yield better performance. It is also important to consider the number of concurrent connections that are executing statements, because most settings apply to each connection.

Consider the unit of a variable when you determine safe values:

- For threads, safe values are typically up to the number of CPU cores.
- For bytes, safe values are typically less than the amount of system memory.
- For time, pay attention that the unit might be seconds or milliseconds.

Variables using the same unit might compete for the same set of resources.

14.4.1 Variable Reference

14.4.1.1 allow_auto_random_explicit_insert New in v4.0.3

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- Determines whether to allow explicitly specifying the values of the column with the `AUTO_RANDOM` attribute in the `INSERT` statement.

14.4.1.2 auto_increment_increment

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1`
- Range: `[1, 65535]`
- Controls the step size of `AUTO_INCREMENT` values to be allocated to a column, and allocation rules for `AUTO_RANDOM` IDs. It is often used in combination with `auto_increment_offset`.

14.4.1.3 auto_increment_offset

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1`
- Range: `[1, 65535]`
- Controls the initial offset of `AUTO_INCREMENT` values to be allocated to a column, and allocation rules for `AUTO_RANDOM` IDs. This setting is often used in combination with `auto_increment_increment`. For example:

```
mysql> CREATE TABLE t1 (a int not null primary key auto_increment);
Query OK, 0 rows affected (0.10 sec)

mysql> set auto_increment_offset=1;
Query OK, 0 rows affected (0.00 sec)

mysql> set auto_increment_increment=3;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (),(),(),();
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
```

```
+----+
| a  |
+----+
|  1 |
|  4 |
|  7 |
| 10 |
+----+
4 rows in set (0.00 sec)
```

14.4.1.4 autocommit

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- Controls whether statements should automatically commit when not in an explicit transaction. See Transaction Overview for more information.

14.4.1.5 block_encryption_mode

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `aes-128-ecb`
- Value options: `aes-128-ecb`, `aes-192-ecb`, `aes-256-ecb`, `aes-128-cbc`, `aes-192-cbc`, `aes-256-cbc`, `aes-128-ofb`, `aes-192-ofb`, `aes-256-ofb`, `aes-128-cfb`, `aes-192-cfb`, `aes-256-cfb`
- This variable sets the encryption mode for the built-in functions `AES_ENCRYPT()` and `AES_DECRYPT()`.

14.4.1.6 character_set_client

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4`
- The character set for data sent from the client. See Character Set and Collation for details on the use of character sets and collations in TiDB. It is recommended to use `SET NAMES` to change the character set when needed.

14.4.1.7 character_set_connection

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4`
- The character set for string literals that do not have a specified character set.

14.4.1.8 character_set_database

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4`
- This variable indicates the character set of the default database in use. **It is NOT recommended to set this variable**. When a new default database is selected, the server changes the variable value.

14.4.1.9 character_set_results

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4`
- The character set that is used when data is sent to the client.

14.4.1.10 character_set_server

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4`
- The default character set for the server.

14.4.1.11 collation_connection

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4_bin`
- This variable indicates the collation used in the current connection. It is consistent with the MySQL variable `collation_connection`.

14.4.1.12 collation_database

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4_bin`
- This variable indicates the default collation of the database in use. **It is NOT recommended to set this variable**. When a new database is selected, TiDB changes this variable value.

14.4.1.13 collation_server

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `utf8mb4_bin`
- The default collation used when the database is created.

14.4.1.14 cte_max_recursion_depth

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1000`
- Range: `[0, 4294967295]`
- Controls the maximum recursion depth in Common Table Expressions.

14.4.1.15 datadir

- Scope: NONE
- Default value: /tmp/tidb
- This variable indicates the location where data is stored. This location can be a local path or point to a PD server if the data is stored on TiKV.
- A value in the format of `ip_address:port` indicates the PD server that TiDB connects to on startup.

14.4.1.16 ddl_slow_threshold

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Default value: `300`
- Unit: Milliseconds
- Log DDL operations whose execution time exceeds the threshold value.

14.4.1.17 default_authentication_plugin

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `mysql_native_password`
- Possible values: `mysql_native_password` and `caching_sha2_password`
- This variable sets the authentication method that the server advertises when the server-client connection is being established.

For more possible values of this variable, see Authentication plugin status.

14.4.1.18 default_week_format

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `0`
- Range: `[0, 7]`
- Sets the week format used by the `WEEK()` function.

14.4.1.19 error_count

- Scope: NONE
- Type: Integer
- A read-only variable that indicates the number of errors that resulted from the last statement that generated messages.

14.4.1.20 foreign_key_checks

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- For compatibility, TiDB returns foreign key checks as `OFF`.

14.4.1.21 group_concat_max_len

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1024`
- Range: `[4, 18446744073709551615]`
- The maximum buffer size for items in the `GROUP_CONCAT()` function.

14.4.1.22 have_openssl

- Scope: NONE
- Default value: `DISABLED`
- A read-only variable for MySQL compatibility. Set to `YES` by the server when the server has TLS enabled.

14.4.1.23 have_ssl

- Scope: NONE
- Default value: `DISABLED`
- A read-only variable for MySQL compatibility. Set to `YES` by the server when the server has TLS enabled.

14.4.1.24 hostname

- Scope: NONE
- Default value: (system hostname)
- The hostname of the TiDB server as a read-only variable.

14.4.1.25 identity New in v5.3.0

This variable is an alias for `last_insert_id`.

14.4.1.26 init_connect

- Scope: GLOBAL
- Persists to cluster: Yes
- Default value: “”
- The `init_connect` feature permits a SQL statement to be automatically executed when you first connect to a TiDB server. If you have the `CONNECTION_ADMIN` or `SUPER` privileges, this `init_connect` statement will not be executed. If the `init_connect` statement results in an error, your user connection will be terminated.

14.4.1.27 innodb_lock_wait_timeout

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `50`
- Range: `[1, 3600]`
- Unit: Seconds
- The lock wait timeout for pessimistic transactions (default).

14.4.1.28 interactive_timeout

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `28800`
- Range: `[1, 31536000]`
- Unit: Seconds
- This variable represents the idle timeout of the interactive user session. Interactive user session refers to the session established by calling `mysql_real_connect()` API using the `CLIENT_INTERACTIVE` option (for example, MySQL Shell and MySQL Client). This variable is fully compatible with MySQL.

14.4.1.29 last_insert_id

- Scope: SESSION
- Default value: `0`
- This variable returns the last `AUTO_INCREMENT` or `AUTO_RANDOM` value generated by an insert statement.
- The value of `last_insert_id` is the same as the value returned by the function `LAST_INSERT_ID()`.

14.4.1.30 last_plan_from_binding New in v4.0

- Scope: SESSION
- Type: Boolean
- Default value: `OFF`
- This variable is used to show whether the execution plan used in the previous statement was influenced by a plan binding

14.4.1.31 last_plan_from_cache New in v4.0

- Scope: SESSION
- Type: Boolean
- Default value: `OFF`
- This variable is used to show whether the execution plan used in the previous `execute` statement is taken directly from the plan cache.

14.4.1.32 license

- Scope: NONE
- Default value: `Apache License 2.0`
- This variable indicates the license of your TiDB server installation.

14.4.1.33 log_bin

- Scope: NONE
- Type: Boolean
- Default value: `OFF`
- This variable indicates whether TiDB Binlog is used.

14.4.1.34 max_allowed_packet New in v6.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `67108864`
- Range: `[1024, 1073741824]`. The value should be an integer multiple of 1024. If the value is not divisible by 1024, a warning will be prompted and the value will be rounded down. For example, when the value is set to 1025, the actual value in TiDB is 1024.
- The maximum packet size allowed by the server and the client in one transmission of packets, in bytes.
- In the `SESSION` scope, this variable is read-only.
- This variable is compatible with MySQL.

14.4.1.35 max_connections

- Scope: GLOBAL
- Persists to cluster: No
- Type: Integer
- Default value: `151`
- Range: `[0, 100000]`
- The maximum number of connections permitted for a single TiDB instance (not effective). This is a noop variable and does not take effect in the current TiDB version. To set the maximum number of connections permitted for a single TiDB instance in the current version, use the TiDB configuration item `max-server-connections`.
- The value of `0` means no limit.

14.4.1.36 max_execution_time

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `0`
- Range: `[0, 2147483647]`
- Unit: Milliseconds
- The maximum execution time of a statement. The default value is unlimited (zero).

> **Note:**
>
> Unlike in MySQL, the `max_execution_time` system variable currently works on all kinds of statements in TiDB, not only restricted to the `SELECT` statement. The precision of the timeout value is roughly 100ms. This means the statement might not be terminated in accurate milliseconds as you specify.

For a SQL statement with the `MAX_EXECUTION_TIME` hint, the maximum execution time of this statement is limited by the hint instead of this variable. The hint can also be used with SQL bindings as described in the SQL FAQ.

14.4.1.37 max_prepared_stmt_count

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[-1, 1048576]`
- Specifies the maximum number of `PREPARE` statements in the current TiDB instance.
- The value of `-1` means no limit on the maximum number of `PREPARE` statements in the current TiDB instance.
- If you set the variable to a value that exceeds the upper limit `1048576`, `1048576` is used instead:

```
mysql> SET GLOBAL max_prepared_stmt_count = 1048577;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+--
    ↪ -------+------+-------------------------------------------------------------+
    ↪
| Level   | Code | Message                                                     |
+--
    ↪ -------+------+-------------------------------------------------------------+
    ↪
| Warning | 1292 | Truncated incorrect max_prepared_stmt_count value: '
    ↪ 1048577' |
+--
    ↪ -------+------+-------------------------------------------------------------+
    ↪
1 row in set (0.00 sec)
```

```
mysql> SHOW GLOBAL VARIABLES LIKE 'max_prepared_stmt_count';
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| max_prepared_stmt_count | 1048576 |
+-------------------------+---------+
1 row in set (0.00 sec)
```

14.4.1.38 plugin_dir

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Default value: “”
- Indicates the directory to load plugins as specified by a command-line flag.

14.4.1.39 plugin_load

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Default value: “”
- Indicates the plugins to load when TiDB is started. These plugins are specified by a command-line flag and separated by commas.

14.4.1.40 port

- Scope: NONE
- Type: Integer
- Default value: `4000`
- Range: `[0, 65535]`
- The port that the `tidb-server` is listening on when speaking the MySQL protocol.

14.4.1.41 rand_seed1

- Scope: SESSION
- Type: Integer
- Default value: `0`
- Range: `[0, 2147483647]`
- This variable is used to seed the random value generator used in the `RAND()` SQL function.
- The behavior of this variable is MySQL compatible.

14.4.1.42 rand_seed2

- Scope: SESSION
- Type: Integer
- Default value: `0`
- Range: `[0, 2147483647]`
- This variable is used to seed the random value generator used in the `RAND()` SQL function.
- The behavior of this variable is MySQL compatible.

14.4.1.43 require_secure_transport New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`

- This variable ensures that all connections to TiDB are either on a local socket, or using TLS. See Enable TLS between TiDB Clients and Servers for additional details.

- Setting this variable to `ON` requires you to connect to TiDB from a session that has TLS enabled. This helps prevent lock-out scenarios when TLS is not configured correctly.
- This setting was previously a `tidb.toml` option (`security.require-secure-transport`), but changed to a system variable starting from TiDB v6.1.0.

14.4.1.44 skip_name_resolve New in v5.2.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls whether the `tidb-server` instance resolves hostnames as a part of the connection handshake.
- When the DNS is unreliable, you can enable this option to improve network performance.

> **Note:**
>
> When `skip_name_resolve=ON`, users with a hostname in their identity will no longer be able to log into the server. For example:

```
CREATE USER 'appuser'@'apphost' IDENTIFIED BY 'app-password';
```

In this example, it is recommended to replace `apphost` with an IP address or the wildcard (`%`).

14.4.1.45 socket

- Scope: NONE
- Default value: “”
- The local unix socket file that the `tidb-server` is listening on when speaking the MySQL protocol.

14.4.1.46 sql_log_bin

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- Indicates whether to write changes to TiDB Binlog or not.

Note:

It is not recommended to set `sql_log_bin` as a global variable because the future versions of TiDB might only allow setting this as a session variable.

14.4.1.47 sql_mode

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,`
  `↪ NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTI`
  `↪`
- This variable controls a number of MySQL compatibility behaviors. See SQL Mode for more information.

14.4.1.48 sql_select_limit New in v4.0.2

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `18446744073709551615`
- Range: `[0, 18446744073709551615]`
- Unit: Rows
- The maximum number of rows returned by the `SELECT` statements.

14.4.1.49 ssl_ca

- Scope: NONE
- Default value: “”
- The location of the certificate authority file (if there is one). The value of this variable is defined by the TiDB configuration item `ssl-ca`.

14.4.1.50 ssl_cert

- Scope: NONE
- Default value: “”
- The location of the certificate file (if there is a file) that is used for SSL/TLS connections. The value of this variable is defined by the TiDB configuration item `ssl-cert`.

14.4.1.51 ssl_key

- Scope: NONE
- Default value: “”
- The location of the private key file (if there is one) that is used for SSL/TLS connections. The value of this variable is defined by TiDB configuration item `ssl-key`.

14.4.1.52 system_time_zone

- Scope: NONE
- Default value: (system dependent)
- This variable shows the system time zone from when TiDB was first bootstrapped. See also `time_zone`.

14.4.1.53 tidb_allow_batch_cop New in v4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1`
- Range: `[0, 2]`
- This variable is used to control how TiDB sends a coprocessor request to TiFlash. It has the following values:
    - `0`: Never send requests in batches
    - `1`: Aggregation and join requests are sent in batches
    - `2`: All coprocessor requests are sent in batches

14.4.1.54 tidb_allow_fallback_to_tikv New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: ""
- This variable is used to specify a list of storage engines that might fall back to TiKV. If the execution of a SQL statement fails due to a failure of the specified storage engine in the list, TiDB retries executing this SQL statement with TiKV. This variable can be set to "" or "tiflash". When this variable is set to "tiflash", if TiFlash returns a timeout error (error code: ErrTiFlashServerTimeout), TiDB retries executing this SQL statement with TiKV.

14.4.1.55 tidb_allow_function_for_expression_index New in v5.2.0

- Scope: NONE
- Default value: `lower, md5, reverse, tidb_shard, upper, vitess_hash`
- This variable is used to show the functions that are allowed to be used for creating expression indexes.

14.4.1.56 tidb_allow_mpp New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`

- Controls whether to use the MPP mode of TiFlash to execute queries. The value options are as follows:
  - `0` or `OFF`, which means that the MPP mode will not be used.
  - `1` or `ON`, which means that the optimizer determines whether to use the MPP mode based on the cost estimation (by default).

MPP is a distributed computing framework provided by the TiFlash engine, which allows data exchange between nodes and provides high-performance, high-throughput SQL algorithms. For details about the selection of the MPP mode, refer to Control whether to select the MPP mode.

14.4.1.57 tidb_allow_remove_auto_inc New in v2.1.18 and v3.0.4

- Scope: SESSION
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether the `AUTO_INCREMENT` property of a column is allowed to be removed by executing `ALTER TABLE MODIFY` or `ALTER TABLE CHANGE` statements. It is not allowed by default.

14.4.1.58 tidb_analyze_version New in v5.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `2`
- Range: `[1, 2]`
- Controls how TiDB collects statistics.
- In v5.3.0 and later versions, the default value of this variable is `2`. If your cluster is upgraded from a version earlier than v5.3.0 to v5.3.0 or later, the default value of `tidb_analyze_version` does not change. For detailed introduction, see Introduction to Statistics.

14.4.1.59 tidb_auto_analyze_end_time

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Time
- Default value: `23:59 +0000`
- This variable is used to restrict the time window that the automatic update of statistics is permitted. For example, to only allow automatic statistics updates between 1 AM and 3 AM in UTC time, set `tidb_auto_analyze_start_time='01:00 +0000'` and `tidb_auto_analyze_end_time='03:00 +0000'`.

14.4.1.60 tidb_auto_analyze_ratio

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Float
- Default value: `0.5`
- Range: `[0, 18446744073709551615]`
- This variable is used to set the threshold when TiDB automatically executes `ANALYZE TABLE` in a background thread to update table statistics. For example, a value of 0.5 means that auto-analyze is triggered when greater than 50% of the rows in a table have been modified. Auto-analyze can be restricted to only execute during certain hours of the day by specifying `tidb_auto_analyze_start_time` and `tidb_auto_analyze_end_time`.

> **Note:**
>
> This feature requires the system variable `tidb_enable_auto_analyze` set to `ON`.

14.4.1.61 tidb_auto_analyze_start_time

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Time
- Default value: `00:00 +0000`
- This variable is used to restrict the time window that the automatic update of statistics is permitted. For example, to only allow automatic statistics updates between 1 AM and 3 AM in UTC time, set `tidb_auto_analyze_start_time='01:00 +0000'` and `tidb_auto_analyze_end_time='03:00 +0000'`.

14.4.1.62 tidb_max_auto_analyze_time New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Default value: `43200`
- Range: `[0, 2147483647]`
- Unit: seconds
- This variable is used to specify the maximum execution time of automatic `ANALYZE` tasks. When the execution time of an automatic `ANALYZE` task exceeds the specified time, the task will be terminated. When the value of this variable is `0`, there is no limit to the maximum execution time of automatic `ANALYZE` tasks.

14.4.1.63 tidb_backoff_lock_fast

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `10`
- Range: `[1, 2147483647]`
- This variable is used to set the `backoff` time when the read request meets a lock.

14.4.1.64 tidb_backoff_weight

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `2`
- Range: `[0, 2147483647]`
- This variable is used to increase the weight of the maximum time of TiDB `backoff`, that is, the maximum retry time for sending a retry request when an internal network or other component (TiKV, PD) failure is encountered. This variable can be used to adjust the maximum retry time and the minimum value is 1.

  For example, the base timeout for TiDB to take TSO from PD is 15 seconds. When `tidb_backoff_weight = 2`, the maximum timeout for taking TSO is: *base time * 2 = 30 seconds*.

  In the case of a poor network environment, appropriately increasing the value of this variable can effectively alleviate error reporting to the application end caused by timeout. If the application end wants to receive the error information more quickly, minimize the value of this variable.

14.4.1.65 tidb_batch_commit

> **Warning:**
>
> It is **NOT** recommended to enable this variable.

- Scope: SESSION
- Persists to cluster: No
- Type: Boolean

- Default value: `OFF`
- The variable is used to control whether to enable the deprecated batch-commit feature. When this variable is enabled, a transaction might be split into multiple transactions by grouping a few statements and committed non-atomically, which is not recommended.

14.4.1.66 tidb_batch_delete

> **Warning:**
>
> This variable is associated with the deprecated batch-dml feature, which might cause data corruption. Therefore, it is not recommended to enable this variable for batch-dml. Instead, use non-transactional DML.

- Scope: SESSION
- Persists to cluster: No
- Type: Boolean
- Default value: `OFF`
- This variable is used to control whether to enable the batch-delete feature, which is a part of the deprecated batch-dml feature. When this variable is enabled, `DELETE` statements might be split into multiple transactions and committed non-atomically. To make it work, you also need to enable `tidb_enable_batch_dml` and set a positive value for `tidb_dml_batch_size`, which is not recommended.

14.4.1.67 tidb_batch_insert

> **Warning:**
>
> This variable is associated with the deprecated batch-dml feature, which might cause data corruption. Therefore, it is not recommended to enable this variable for batch-dml. Instead, use non-transactional DML.

- Scope: SESSION
- Persists to cluster: No
- Type: Boolean
- Default value: `OFF`
- This variable is used to control whether to enable the batch-insert feature, which is a part of the deprecated batch-dml feature. When this variable is enabled, `INSERT` statements might be split into multiple transactions and committed non-atomically. To make it work, you also need to enable `tidb_enable_batch_dml` and set a positive value for `tidb_dml_batch_size`, which is not recommended.

14.4.1.68 tidb_batch_pending_tiflash_count New in v6.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `4000`
- Scope: `[0, 2147483647]`
- Specifies the maximum number of permitted unavailable tables when you use `ALTER DATABASE SET TIFLASH REPLICA` to add TiFlash replicas. If the number of unavailable tables exceeds this limit, the operation will be stopped or setting TiFlash replicas for the remaining tables will be very slow.

14.4.1.69 tidb_broadcast_join_threshold_count New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `10240`
- Range: `[0, 9223372036854775807]`
- Unit: Rows
- If the objects of the join operation belong to a subquery, the optimizer cannot estimate the size of the subquery result set. In this situation, the size is determined by the number of rows in the result set. If the estimated number of rows in the subquery is less than the value of this variable, the Broadcast Hash Join algorithm is used. Otherwise, the Shuffled Hash Join algorithm is used.

14.4.1.70 tidb_broadcast_join_threshold_size New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `104857600` (100 MiB)
- Range: `[0, 9223372036854775807]`
- Unit: Bytes
- If the table size is less than the value of the variable, the Broadcast Hash Join algorithm is used. Otherwise, the Shuffled Hash Join algorithm is used.

14.4.1.71 tidb_build_stats_concurrency

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `4`
- Unit: Threads
- This variable is used to set the concurrency of executing the `ANALYZE` statement.
- When the variable is set to a larger value, the execution performance of other queries is affected.

14.4.1.72 tidb_capture_plan_baselines New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to control whether to enable the baseline capturing feature. This feature depends on the statement summary, so you need to enable the statement summary before you use baseline capturing.
- After this feature is enabled, the historical SQL statements in the statement summary are traversed periodically, and bindings are automatically created for SQL statements that appear at least twice.

14.4.1.73 tidb_check_mb4_value_in_utf8

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Boolean
- Default value: `ON`
- This variable is used to enforce that the `utf8` character set only stores values from the Basic Multilingual Plane (BMP). To store characters outside the BMP, it is recommended to use the `utf8mb4` character set.
- You might need to disable this option when upgrading your cluster from an earlier version of TiDB where the `utf8` checking was more relaxed. For details, see FAQs After Upgrade.

14.4.1.74 tidb_checksum_table_concurrency

- Scope: SESSION
- Default value: `4`
- Unit: Threads
- This variable is used to set the scan index concurrency of executing the `ADMIN CHECKSUM TABLE` statement.
- When the variable is set to a larger value, the execution performance of other queries is affected.

14.4.1.75 tidb_committer_concurrency New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `128`

- Range: `[1, 10000]`
- The number of goroutines for requests related to executing commit in the commit phase of the single transaction.
- If the transaction to commit is too large, the waiting time for the flow control queue when the transaction is committed might be too long. In this situation, you can increase the configuration value to speed up the commit.
- This setting was previously a `tidb.toml` option (`performance.committer-concurrency`), but changed to a system variable starting from TiDB v6.1.0.

14.4.1.76 tidb_config

- Scope: SESSION
- Default value: ""
- This variable is read-only. It is used to obtain the configuration information of the current TiDB server.

14.4.1.77 tidb_constraint_check_in_place

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This setting only applies to optimistic transactions. When this variable is set to `OFF`, checking for duplicate values in UNIQUE indexes is deferred until the transaction commits. This helps improve performance, but might be an unexpected behavior for some applications. See Constraints for details.
    - When set to zero and using optimistic transactions:

```
tidb> create table t (i int key);
tidb> insert into t values (1);
tidb> begin optimistic;
tidb> insert into t values (1);
Query OK, 1 row affected
tidb> commit; -- Check only when a transaction is committed.
ERROR 1062 : Duplicate entry '1' for key 'PRIMARY'
```

    - When set to 1 and using optimistic transactions:

```
tidb> set @@tidb_constraint_check_in_place=1;
tidb> begin optimistic;
tidb> insert into t values (1);
ERROR 1062 : Duplicate entry '1' for key 'PRIMARY'
```

Constraint checking is always performed in place for pessimistic transactions (default).

14.4.1.78 tidb_current_ts

- Scope: SESSION
- Default value: `0`
- This variable is read-only. It is used to obtain the timestamp of the current transaction.

14.4.1.79 tidb_ddl_error_count_limit

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `512`
- Range: `[0, 9223372036854775807]`
- This variable is used to set the number of retries when the DDL operation fails. When the number of retries exceeds the parameter value, the wrong DDL operation is canceled.

14.4.1.80 tidb_ddl_reorg_batch_size

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `256`
- Range: `[32, 10240]`
- Unit: Rows
- This variable is used to set the batch size during the `re-organize` phase of the DDL operation. For example, when TiDB executes the `ADD INDEX` operation, the index data needs to backfilled by `tidb_ddl_reorg_worker_cnt` (the number) concurrent workers. Each worker backfills the index data in batches.
    - If many updating operations such as `UPDATE` and `REPLACE` exist during the `ADD INDEX` operation, a larger batch size indicates a larger probability of transaction conflicts. In this case, you need to adjust the batch size to a smaller value. The minimum value is 32.
    - If the transaction conflict does not exist, you can set the batch size to a large value (consider the worker count. See Interaction Test on Online Workloads and `ADD INDEX` Operations for reference). This can increase the speed of the backfilling data, but the write pressure on TiKV also becomes higher.

14.4.1.81 tidb_ddl_reorg_priority

- Scope: SESSION
- Default value: `PRIORITY_LOW`
- This variable is used to set the priority of executing the `ADD INDEX` operation in the `re-organize` phase.
- You can set the value of this variable to `PRIORITY_LOW`, `PRIORITY_NORMAL` or `PRIORITY_HIGH`.

14.4.1.82 tidb_ddl_reorg_worker_cnt

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `4`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of the DDL operation in the `re-organize` phase.

14.4.1.83 tidb_disable_txn_auto_retry

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to set whether to disable the automatic retry of explicit optimistic transactions. The default value of `ON` means that transactions will not automatically retry in TiDB and `COMMIT` statements might return errors that need to be handled in the application layer.

  Setting the value to `OFF` means that TiDB will automatically retry transactions, resulting in fewer errors from `COMMIT` statements. Be careful when making this change, because it might result in lost updates.

  This variable does not affect automatically committed implicit transactions and internally executed transactions in TiDB. The maximum retry count of these transactions is determined by the value of `tidb_retry_limit`.

  For more details, see limits of retry.

  This variable only applies to optimistic transactions, not to pessimistic transactions. The number of retries for pessimistic transactions is controlled by `max_retry_count`.

14.4.1.84 tidb_distsql_scan_concurrency

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `15`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of the `scan` operation.
- Use a bigger value in OLAP scenarios, and a smaller value in OLTP scenarios.
- For OLAP scenarios, the maximum value should not exceed the number of CPU cores of all the TiKV nodes.
- If a table has a lot of partitions, you can reduce the variable value appropriately (determined by the size of the data to be scanned and the frequency of the scan) to avoid TiKV becoming out of memory (OOM).
- For a simple query with only a `LIMIT` clause, if the `LIMIT` value is less than 100000, the scan operation pushed down to TiKV treats the value of this variable as `1` to enhance execution efficiency.
- For the `SELECT MAX/MIN(col)FROM ...` query, if the `col` column has an index sorted in the same order required by the `MAX(col)` or `MIN(col)` function, TiDB will rewrite the query to `SELECT col FROM ... LIMIT 1` for processing, and the value of this variable will also be processed as `1`. For example, for `SELECT MIN(col)FROM ...`, if the `col` column has an ascending index, TiDB can quickly obtain the `MIN(col)` value by rewriting the query to `SELECT col FROM ... LIMIT 1` and directly reading the first row of the index.

14.4.1.85 tidb_dml_batch_size

> **Warning:**
>
> This variable is associated with the deprecated batch-dml feature, which might cause data corruption. Therefore, it is not recommended to enable this variable for batch-dml. Instead, use non-transactional DML.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `0`
- Range: `[0, 2147483647]`
- Unit: Rows

- When this value is greater than `0`, TiDB will batch commit statements such as `INSERT` or `LOAD DATA` into smaller transactions. This reduces memory usage and helps ensure that the `txn-total-size-limit` is not reached by bulk modifications.
- Only the value `0` provides ACID compliance. Setting this to any other value will break the atomicity and isolation guarantees of TiDB.
- To make this variable work, you also need to enable `tidb_enable_batch_dml` and at least one of `tidb_batch_insert` and `tidb_batch_delete`.

14.4.1.86 tidb_enable_1pc New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to specify whether to enable the one-phase commit feature for transactions that only affect one Region. Compared with the often-used two-phase commit, one-phase commit can greatly reduce the latency of transaction commit and increase the throughput.

> **Note:**
>
> - The default value of `ON` only applies to new clusters. if your cluster was upgraded from an earlier version of TiDB, the value `OFF` will be used instead.
> - If you have enabled TiDB Binlog, enabling this variable cannot improve the performance. To improve the performance, it is recommended to use TiCDC instead.
> - Enabling this parameter only means that one-phase commit becomes an optional mode of transaction commit. In fact, the most suitable mode of transaction commit is determined by TiDB.

14.4.1.87 tidb_enable_amend_pessimistic_txn New in v4.0.7

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`

- This variable is used to control whether to enable the `AMEND TRANSACTION` feature. If you enable the `AMEND TRANSACTION` feature in a pessimistic transaction, when concurrent DDL operations and SCHEMA VERSION changes exist on tables associated with this transaction, TiDB attempts to amend the transaction. TiDB corrects the transaction commit to make the commit consistent with the latest valid SCHEMA VERSION so that the transaction can be successfully committed without getting the `Information schema is changed` error. This feature is effective on the following concurrent DDL operations:
    - `ADD COLUMN` or `DROP COLUMN` operations.
    - `MODIFY COLUMN` or `CHANGE COLUMN` operations which increase the length of a field.
    - `ADD INDEX` or `DROP INDEX` operations in which the index column is created before the transaction is opened.

> **Note:**
>
> Currently, this feature is incompatible with TiDB Binlog in some scenarios and might cause semantic changes on a transaction. For more usage precautions of this feature, refer to Incompatibility issues about transaction semantic and Incompatibility issues about TiDB Binlog.

14.4.1.88 tidb_enable_async_commit New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls whether to enable the async commit feature for the second phase of the two-phase transaction commit to perform asynchronously in the background. Enabling this feature can reduce the latency of transaction commit.

> **Note:**
>
> - The default value of `ON` only applies to new clusters. if your cluster was upgraded from an earlier version of TiDB, the value `OFF` will be used instead.
> - If you have enabled TiDB Binlog, enabling this variable cannot improve the performance. To improve the performance, it is recommended to use TiCDC instead.

- Enabling this parameter only means that Async Commit becomes an optional mode of transaction commit. In fact, the most suitable mode of transaction commit is determined by TiDB.

14.4.1.89 tidb_enable_auto_analyze New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- Determines whether TiDB automatically updates table statistics as a background operation.
- This setting was previously a `tidb.toml` option (`performance.run-auto-analyze`), but changed to a system variable starting from TiDB v6.1.0.

14.4.1.90 tidb_enable_auto_increment_in_generated

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to determine whether to include the `AUTO_INCREMENT` columns when creating a generated column or an expression index.

14.4.1.91 tidb_enable_batch_dml

Warning:

This variable is associated with the deprecated batch-dml feature, which might cause data corruption. Therefore, it is not recommended to enable this variable for batch-dml. Instead, use non-transactional DML.

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`

- This variable controls whether to enable the deprecated batch-dml feature. When it is enabled, certain statements might be split into multiple transactions, which is non-atomic and should be used with care. When using batch-dml, you must ensure that there are no concurrent operations on the data you are operating on. To make it work, you must also specify a positive value for `tidb_batch_dml_size` and enable at least one of `tidb_batch_insert` and `tidb_batch_delete`.

14.4.1.92 tidb_enable_change_multi_schema

> **Warning:**
>
> TiDB will support more types of multi-schema changes in the future. This system variable will be deleted in a future release of TiDB.

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to control whether multiple columns or indexes can be altered in one `ALTER TABLE` statement. When the value of this variable is `ON`, only the following multi-schema changes are supported:
  - Add multiple columns. For example, `ALTER TABLE t ADD COLUMN c1 INT, ADD COLUMN c2 INT;`.
  - Drop multiple columns. For example, `ALTER TABLE t DROP COLUMN c1, DROP COLUMN c2;`.
  - Drop multiple indexes. For example, `ALTER TABLE t DROP INDEX i1, DROP INDEX i2;`.
  - Drop a column covered by a single-column index. For example, `ALTER TABLE t DROP COLUMN c1`, in which the schema contains `INDEX idx(c1)`.

14.4.1.93 tidb_enable_cascades_planner

> **Warning:**
>
> Currently, cascades planner is an experimental feature. It is not recommended that you use it in production environments.

- Scope: SESSION | GLOBAL

- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to control whether to enable the cascades planner.

14.4.1.94 tidb_enable_chunk_rpc New in v4.0

- Scope: SESSION
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable the `Chunk` data encoding format in Coprocessor.

14.4.1.95 tidb_enable_clustered_index New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `INT_ONLY`
- Possible values: `OFF`, `ON`, `INT_ONLY`
- This variable is used to control whether to create the primary key as a clustered index by default. "By default" here means that the statement does not explicitly specify the keyword `CLUSTERED`/`NONCLUSTERED`. Supported values are `OFF`, `ON`, and `INT_ONLY`:
    - `OFF` indicates that primary keys are created as non-clustered indexes by default.
    - `ON` indicates that primary keys are created as clustered indexes by default.
    - `INT_ONLY` indicates that the behavior is controlled by the configuration item `alter-primary-key`. If `alter-primary-key` is set to `true`, all primary keys are created as non-clustered indexes by default. If it is set to `false`, only the primary keys which consist of an integer column are created as clustered indexes.

14.4.1.96 tidb_enable_collect_execution_info

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Boolean
- Default value: `ON`
- This variable controls whether to record the execution information of each operator in the slow query log.

14.4.1.97 tidb_enable_column_tracking New in v5.4.0

> **Warning:**
>
> Currently, collecting statistics on `PREDICATE COLUMNS` is an experimental feature. It is not recommended that you use it in production environments.

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls whether to enable TiDB to collect `PREDICATE COLUMNS`. After enabling the collection, if you disable it, the information of previously collected `PREDICATE COLUMNS` is cleared. For details, see Collect statistics on some columns.

14.4.1.98 tidb_enable_enhanced_security

- Scope: NONE
- Type: Boolean

- Default value: `OFF`
- This variable indicates whether the TiDB server you are connected to has the Security Enhanced Mode (SEM) enabled. To change its value, you need to modify the value of `enable-sem` in your TiDB server configuration file and restart the TiDB server.

- SEM is inspired by the design of systems such as Security-Enhanced Linux. It reduces the abilities of users with the MySQL `SUPER` privilege and instead requires `RESTRICTED` fine-grained privileges to be granted as a replacement. These fine-grained privileges include:
  - `RESTRICTED_TABLES_ADMIN`: The ability to write data to system tables in the `mysql` schema and to see sensitive columns on `information_schema` tables.
  - `RESTRICTED_STATUS_ADMIN`: The ability to see sensitive variables in the command `SHOW STATUS`.
  - `RESTRICTED_VARIABLES_ADMIN`: The ability to see and set sensitive variables in `SHOW [GLOBAL] VARIABLES` and `SET`.
  - `RESTRICTED_USER_ADMIN`: The ability to prevent other users from making changes or dropping a user account.

14.4.1.99 tidb_enable_exchange_partition

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls whether to enable the `exchange partitions with tables` feature. The default value is `OFF`, that is, `exchange partitions with tables` is disabled by default.

14.4.1.100 tidb_enable_extended_stats

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable indicates whether TiDB can collect the extended statistic to guide the optimizer. See Introduction to Extended Statistics for more information.

14.4.1.101 tidb_enable_fast_analyze

> **Warning:**
>
> Currently, `Fast Analyze` is an experimental feature. It is not recommended that you use it in production environments.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether to enable the statistics `Fast Analyze` feature.
- If the statistics `Fast Analyze` feature is enabled, TiDB randomly samples about 10,000 rows of data as statistics. When the data is distributed unevenly or the data size is small, the statistics accuracy is low. This might lead to a non-optimal execution plan, for example, selecting a wrong index. If the execution time of the regular `Analyze` statement is acceptable, it is recommended to disable the `Fast Analyze` feature.

14.4.1.102 tidb_enable_historical_stats

- Scope: GLOBAL

- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used for an unreleased feature. **Do not change the variable value**.

14.4.1.103 tidb_enable_index_merge New in v4.0

> **Note:**
>
> - After upgrading a TiDB cluster from versions earlier than v4.0.0 to v5.4.0 or later, this variable is disabled by default to prevent performance regression due to changes of execution plans.
> - After upgrading a TiDB cluster from v4.0.0 or later to v5.4.0 or later, this variable remains the setting before the upgrade.
> - Since v5.4.0, for a newly deployed TiDB cluster, this variable is enabled by default.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable the index merge feature.

14.4.1.104 tidb_enable_index_merge_join

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- Specifies whether to enable the `IndexMergeJoin` operator.
- This variable is used only for the internal operation of TiDB. It is **NOT recommended** to adjust it. Otherwise, data correctness might be affected.

14.4.1.105 tidb_enable_legacy_instance_scope New in v6.0.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`

- This variable permits `INSTANCE` scoped variables to be set using the `SET SESSION` as well as `SET GLOBAL` syntax.
- This option is enabled by default for compatibility with earlier versions of TiDB.

14.4.1.106 tidb_enable_list_partition New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to set whether to enable the `LIST (COLUMNS)TABLE PARTITION` feature.

14.4.1.107 tidb_enable_local_txn

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used for an unreleased feature. **Do not change the variable value**.

14.4.1.108 tidb_enable_mutation_checker New in v6.0.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable TiDB mutation checker, which is a tool used to check consistency between data and indexes during the execution of DML statements. If the checker returns an error for a statement, TiDB rolls back the execution of the statement. Enabling this variable causes a slight increase in CPU usage. For more information, see Troubleshoot Inconsistency Between Data and Indexes.
- For new clusters of v6.0.0 or later versions, the default value is `ON`. For existing clusters that upgrade from versions earlier than v6.0.0, the default value is `OFF`.

14.4.1.109 tidb_enable_new_cost_interface

This variable is associated with a feature available since v6.2.0 and not usable in v6.1. For details, see the v6.2 documentation `tidb_enable_new_cost_interface`.

14.4.1.110 tidb_enable_new_only_full_group_by_check New in v6.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `OFF`
- Value options: `OFF` and `ON`
- This variable controls the behavior when TiDB performs the `ONLY_FULL_GROUP_BY` ↪ check. For detailed information about `ONLY_FULL_GROUP_BY`, see the MySQL documentation. In v6.1.0, TiDB handles this check more strictly and correctly.
- To avoid potential compatibility issues caused by version upgrades, the default value of this variable is `OFF` in v6.1.0.

14.4.1.111 tidb_enable_noop_functions New in v4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `OFF`
- Possible values: `OFF`, `ON`, `WARN`
- By default, TiDB returns an error when you attempt to use the syntax for functionality that is not yet implemented. When the variable value is set to `ON`, TiDB silently ignores such cases of unavailable functionality, which is helpful if you cannot make changes to the SQL code.
- Enabling `noop` functions controls the following behaviors:
  - `LOCK IN SHARE MODE` syntax
  - `SQL_CALC_FOUND_ROWS` syntax
  - `START TRANSACTION READ ONLY` and `SET TRANSACTION READ ONLY` syntax
  - The `tx_read_only`, `transaction_read_only`, `offline_mode`, `super_read_only` ↪ , `read_only` and `sql_auto_is_null` system variables
  - `GROUP BY <expr> ASC|DESC` syntax

> **Warning:**
>
> Only the default value of `OFF` can be considered safe. Setting `tidb_enable_noop_functions=1` might lead to unexpected behaviors in your application, because it permits TiDB to ignore certain syntax without providing an error. For example, the syntax `START TRANSACTION READ ONLY` is permitted, but the transaction remains in read-write mode.

14.4.1.112 tidb_enable_outer_join_reorder New in v6.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: In v6.1.0, the default value is `ON`. In v6.1.1 and later 6.1 patch versions, the default value is `OFF`.
- Since v6.1.0, the Join Reorder algorithm of TiDB supports Outer Join. This variable controls the support behavior. The default value is `OFF`, which means the Join Reorder's support for Outer Join is disabled by default.
- If your TiDB cluster is upgraded from an earlier version to v6.1.0, the default value of this variable after the upgrade is `ON`. If your TiDB cluster is upgraded from an earlier version to v6.1.1 or a later 6.1 patch version, the default value of this variable after the upgrade is `OFF`.

14.4.1.113 tidb_enable_inl_join_inner_multi_pattern New in v6.1.6

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls whether Index Join is supported when the inner table has `Selection` or `Projection` operators on it. The default value `OFF` means that Index Join is not supported in this scenario.

14.4.1.114 tidb_enable_ordered_result_mode

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `OFF`
- Specifies whether to sort the final output result automatically.
- For example, with this variable enabled, TiDB processes `SELECT a, MAX(b)FROM t GROUP BY a` as `SELECT a, MAX(b)FROM t GROUP BY a ORDER BY a, MAX(b)`.

14.4.1.115 tidb_enable_paging New in v5.4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls whether to use the method of paging to send coprocessor requests in `IndexLookUp` operator.

- User scenarios: For read queries that use `IndexLookup` and `Limit` and that `Limit` cannot be pushed down to `IndexScan`, there might be high latency for the read queries and high CPU usage for TiKV's `unified read pool`. In such cases, because the `Limit` operator only requires a small set of data, if you set `tidb_enable_paging` to `ON`, TiDB processes less data, which reduces query latency and resource consumption.
- When `tidb_enable_paging` is enabled, for the `IndexLookUp` requests with `Limit` that cannot be pushed down and are fewer than `960`, TiDB uses the method of paging to send coprocessor requests. The fewer `Limit`, the more obvious the optimization.

14.4.1.116 tidb_enable_parallel_apply New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls whether to enable concurrency for the `Apply` operator. The number of concurrencies is controlled by the `tidb_executor_concurrency` variable. The `Apply` operator processes correlated subqueries and has no concurrency by default, so the execution speed is slow. Setting this variable value to `1` can increase concurrency and speed up execution. Currently, concurrency for `Apply` is disabled by default.

14.4.1.117 tidb_enable_pipelined_window_function

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variables specifies whether to use the pipeline execution algorithm for window functions.

14.4.1.118 tidb_enable_prepared_plan_cache New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- Determines whether to enable Prepared Plan Cache. When it is enabled, the execution plans of `Prepare` and `Execute` are cached so that the subsequent executions skip optimizing the execution plans, which brings performance improvement.
- This setting was previously a `tidb.toml` option (`prepared-plan-cache.enabled`), but changed to a system variable starting from TiDB v6.1.0.

14.4.1.119 tidb_enable_pseudo_for_outdated_stats New in v5.3.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls the behavior of the optimizer on using statistics of a table when the statistics are outdated.

- The optimizer determines whether the statistics of a table is outdated in this way: since the last time `ANALYZE` is executed on a table to get the statistics, if 80% of the table rows are modified (the modified row count divided by the total row count), the optimizer determines that the statistics of this table is outdated. You can change this ratio using the `pseudo-estimate-ratio` configuration.

- By default (with the variable value `ON`), when the statistics of a table is outdated, the optimizer determines that the statistics of the table is no longer reliable except for the total row count. Then, the optimizer uses the pseudo statistics. If you set the variable value to `OFF`, even if the statistics of a table are outdated, the optimizer still keeps using the statistics.
- If the data on a table is frequently modified without executing `ANALYZE` on this table in time, to keep the execution plan stable, you can set the variable value to `OFF`.

14.4.1.120 tidb_enable_rate_limit_action

> **Note:**
>
> This variable is enabled by default, which makes the memory usage not under the control of `tidb_mem_quota_query` in some cases. Therefore, it is recommended to set the value of `tidb_enable_rate_limit_action` to `OFF`.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls whether to enable the dynamic memory control feature for the operator that reads data. By default, this operator enables the maximum number of threads that `tidb_distsql_scan_concurrency` allows to read data. When the memory usage of a single SQL statement exceeds `tidb_mem_quota_query` each time, the operator that reads data stops one thread.

- When the operator that reads data has only one thread left and the memory usage of a single SQL statement continues to exceed `tidb_mem_quota_query`, this SQL statement triggers other memory control behaviors, such as spilling data to disk.

14.4.1.121 tidb_enable_slow_log

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable the slow log feature.

14.4.1.122 tidb_enable_stmt_summary New in v3.0.4

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable the statement summary feature. If enabled, SQL execution information like time consumption is recorded to the `information_schema.STATEMENTS_SUMMARY` system table to identify and troubleshoot SQL performance issues.

14.4.1.123 tidb_enable_strict_double_type_check New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control if tables can be created with invalid definitions of type `DOUBLE`. This setting is intended to provide an upgrade path from earlier versions of TiDB, which were less strict in validating types.
- The default value of `ON` is compatible with MySQL.

For example, the type `DOUBLE(10)` is now considered invalid because the precision of floating point types is not guaranteed. After changing `tidb_enable_strict_double_type_check` ↪ to `OFF`, the table is created:

```
mysql> CREATE TABLE t1 (id int, c double(10));
ERROR 1149 (42000): You have an error in your SQL syntax; check the manual
    ↪ that corresponds to your MySQL server version for the right syntax to
    ↪  use
```

```
mysql> SET tidb_enable_strict_double_type_check = 'OFF';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id int, c double(10));
Query OK, 0 rows affected (0.09 sec)
```

> **Note:**
>
> This setting only applies to the type `DOUBLE` since MySQL permits precision for `FLOAT` types. This behavior is deprecated starting with MySQL 8.0.17, and it is not recommended to specify precision for either `FLOAT` or `DOUBLE` types.

14.4.1.124 tidb_enable_table_partition

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `ON`
- Possible values: `OFF`, `ON`, `AUTO`
- This variable is used to set whether to enable the `TABLE PARTITION` feature:
  - `ON` indicates enabling Range partitioning, Hash partitioning, and Range column partitioning with one single column.
  - `AUTO` functions the same way as `ON` does.
  - `OFF` indicates disabling the `TABLE PARTITION` feature. In this case, the syntax that creates a partition table can be executed, but the table created is not a partitioned one.

14.4.1.125 tidb_enable_telemetry New in v4.0.2

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON` for v6.1.0 ~ v6.1.4; `OFF` for v6.1.5 and later v6.1.x versions

- This variable is used to dynamically control whether the telemetry collection in TiDB is enabled. If the TiDB `enable-telemetry` configuration item is set to `false` on all TiDB instances, the telemetry collection is always disabled and this system variable will not take effect. See Telemetry for details.

14.4.1.126 tidb_enable_top_sql New in v5.4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`

- This variable is used to control whether to enable the Top SQL feature.

14.4.1.127 tidb_enable_tso_follower_proxy New in v5.3.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to enable the TSO Follower Proxy feature. When the value is `OFF`, TiDB only gets TSO from the PD leader. After this feature is enabled, TiDB gets TSO by evenly sending requests to all PD nodes and forwarding TSO requests through PD followers. This helps reduce the CPU pressure of PD leader.
- Scenarios for enabling TSO Follower Proxy:
  - Due to the high pressure of TSO requests, the CPU of the PD leader reaches a bottleneck, which causes high latency of TSO RPC requests.
  - The TiDB cluster has many TiDB instances, and increasing the value of `tidb_tso_client_batch_max_wait_time` cannot alleviate the high latency issue of TSO RPC requests.

> **Note:**
>
> Suppose that the TSO RPC latency increases for reasons other than a CPU usage bottleneck of the PD leader (such as network issues). In this case, enabling the TSO Follower Proxy might increase the execution latency in TiDB and affect the QPS performance of the cluster.

14.4.1.128 tidb_enable_vectorized_expression New in v4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable vectorized execution.

14.4.1.129 tidb_enable_window_function

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable the support for window functions. Note that window functions may use reserved keywords. This might cause SQL statements that could be executed normally cannot be parsed after upgrading TiDB. In this case, you can set `tidb_enable_window_function` to `OFF`.

14.4.1.130 tidb_enforce_mpp New in v5.1

- Scope: SESSION
- Type: Boolean
- Default value: `OFF`

- To change this default value, modify the `performance.enforce-mpp` configuration value.

- Controls whether to ignore the optimizer's cost estimation and to forcibly use TiFlash's MPP mode for query execution. The value options are as follows:
  - `0` or `OFF`, which means that the MPP mode is not forcibly used (by default).
  - `1` or `ON`, which means that the cost estimation is ignored and the MPP mode is forcibly used. Note that this setting only takes effect when `tidb_allow_mpp=true`.

MPP is a distributed computing framework provided by the TiFlash engine, which allows data exchange between nodes and provides high-performance, high-throughput SQL algorithms. For details about the selection of the MPP mode, refer to Control whether to select the MPP mode.

14.4.1.131 tidb_evolve_plan_baselines New in v4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to control whether to enable the baseline evolution feature. For detailed introduction or usage , see Baseline Evolution.
- To reduce the impact of baseline evolution on the cluster, use the following configurations:

- Set `tidb_evolve_plan_task_max_time` to limit the maximum execution time of each execution plan. The default value is 600s.
- Set `tidb_evolve_plan_task_start_time` and `tidb_evolve_plan_task_end_time` ↪ to limit the time window. The default values are respectively `00:00 +0000` and `23:59 +0000`.

14.4.1.132 tidb_evolve_plan_task_end_time New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Time
- Default value: `23:59 +0000`
- This variable is used to set the end time of baseline evolution in a day.

14.4.1.133 tidb_evolve_plan_task_max_time New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `600`
- Range: `[-1, 9223372036854775807]`
- Unit: Seconds
- This variable is used to limit the maximum execution time of each execution plan in the baseline evolution feature.

14.4.1.134 tidb_evolve_plan_task_start_time New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Time
- Default value: `00:00 +0000`
- This variable is used to set the start time of baseline evolution in a day.

14.4.1.135 tidb_executor_concurrency New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `5`
- Range: `[1, 256]`
- Unit: Threads

This variable is used to set the concurrency of the following SQL operators (to one value):

- `index lookup`
- `index lookup join`
- `hash join`
- `hash aggregation` (the `partial` and `final` phases)
- `window`
- `projection`

`tidb_executor_concurrency` incorporates the following existing system variables as a whole for easier management:

- `tidb_index_lookup_concurrency`
- `tidb_index_lookup_join_concurrency`
- `tidb_hash_join_concurrency`
- `tidb_hashagg_partial_concurrency`
- `tidb_hashagg_final_concurrency`
- `tidb_projection_concurrency`
- `tidb_window_concurrency`

Since v5.0, you can still separately modify the system variables listed above (with a deprecation warning returned) and your modification only affects the corresponding single operators. After that, if you use `tidb_executor_concurrency` to modify the operator concurrency, the separately modified operators will not be affected. If you want to use `tidb_executor_concurrency` to modify the concurrency of all operators, you can set the values of all variables listed above to `-1`.

For a system upgraded to v5.0 from an earlier version, if you have not modified any value of the variables listed above (which means that the `tidb_hash_join_concurrency` value is `5` and the values of the rest are `4`), the operator concurrency previously managed by these variables will automatically be managed by `tidb_executor_concurrency`. If you have modified any of these variables, the concurrency of the corresponding operators will still be controlled by the modified variables.

14.4.1.136 tidb_expensive_query_time_threshold

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Integer
- Default value: `60`
- Range: `[10, 2147483647]`
- Unit: Seconds
- This variable is used to set the threshold value that determines whether to print expensive query logs. The difference between expensive query logs and slow query logs is:

- Slow logs are printed after the statement is executed.
- Expensive query logs print the statements that are being executed, with execution time exceeding the threshold value, and their related information.

14.4.1.137 tidb_force_priority

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Default value: `NO_PRIORITY`
- This variable is used to change the default priority for statements executed on a TiDB server. A use case is to ensure that a particular user that is performing OLAP queries receives lower priority than users performing OLTP queries.
- The default value `NO_PRIORITY` means that the priority for statements is not forced to change. Other options are `LOW_PRIORITY`, `DELAYED`, and `HIGH_PRIORITY` in ascending order.

14.4.1.138 tidb_gc_concurrency New in v5.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[1, 256]`
- Unit: Threads
- Specifies the number of threads in the Resolve Locks step of GC. A value of `-1` means that TiDB will automatically decide the number of garbage collection threads to use.

14.4.1.139 tidb_gc_enable New in v5.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- Enables garbage collection for TiKV. Disabling garbage collection will reduce system performance, as old versions of rows will no longer be purged.

14.4.1.140 tidb_gc_life_time New in v5.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Duration

- Default value: `10m0s`
- Range: `[10m0s, 8760h0m0s]`
- The time limit during which data is retained for each GC, in the format of Go Duration. When a GC happens, the current time minus this value is the safe point.

Note:

- In scenarios of frequent updates, a large value (days or even months) for `tidb_gc_life_time` may cause potential issues, such as:
  - Larger storage use
  - A large amount of history data may affect performance to a certain degree, especially for range queries such as `select count(*)from t`
- If there is any transaction that has been running longer than `tidb_gc_life_time`, during GC, the data since `start_ts` is retained for this transaction to continue execution. For example, if `tidb_gc_life_time` is configured to 10 minutes, among all transactions being executed, the transaction that starts earliest has been running for 15 minutes, GC will retain data of the recent 15 minutes.

14.4.1.141 tidb_gc_max_wait_time New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Default value: `86400`
- Range: `[600, 31536000]`
- Unit: Seconds
- This variable is used to set the maximum time that active transactions block the GC safe point. During each time of GC, the safe point does not exceed the start time of the ongoing transactions by default. If the runtime of active transactions does not exceed this variable value, the GC safe point will be blocked until the runtime exceeds this value. This variable value is an integer type.

14.4.1.142 tidb_gc_run_interval New in v5.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Duration
- Default value: `10m0s`
- Range: `[10m0s, 8760h0m0s]`
- Specifies the GC interval, in the format of Go Duration, for example, `"1h30m"`, and `"15m"`

14.4.1.143 tidb_gc_scan_lock_mode New in v5.0

> **Warning:**
>
> Currently, Green GC is an experimental feature. It is not recommended that you use it in production environments.

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `LEGACY`
- Possible values: `PHYSICAL`, `LEGACY`
    - `LEGACY`: Uses the old way of scanning, that is, disable Green GC.
    - `PHYSICAL`: Uses the physical scanning method, that is, enable Green GC.

- This variable specifies the way of scanning locks in the Resolve Locks step of GC. When the variable value is set to `LEGACY`, TiDB scans locks by Regions. When the value `PHYSICAL` is used, it enables each TiKV node to bypass the Raft layer and directly scan data, which can effectively mitigate the impact of GC wakening up all Regions when the Hibernate Region feature is enabled, thus improving the execution speed in the Resolve Locks step.

14.4.1.144 tidb_general_log

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Boolean
- Default value: `OFF`

- This variable is used to set whether to record all SQL statements in the log. This feature is disabled by default. If maintenance personnel needs to trace all SQL statements when locating issues, they can enable this feature.

- To see all records of this feature in the log, you need to set the TiDB configuration item `log.level` to `"info"` or `"debug"` and then query the `"GENERAL_LOG"` string. The following information is recorded:
    - `conn`: The ID of the current session.
    - `user`: The current session user.
    - `schemaVersion`: The current schema version.

- `txnStartTS`: The timestamp at which the current transaction starts.
- `forUpdateTS`: In the pessimistic transactional mode, `forUpdateTS` is the current timestamp of the SQL statement. When a write conflict occurs in the pessimistic transaction, TiDB retries the SQL statement currently being executed and updates this timestamp. You can configure the number of retries via `max-retry-count`. In the optimistic transactional model, `forUpdateTS` is equivalent to `txnStartTS`.
- `isReadConsistency`: Indicates whether the current transactional isolation level is Read Committed (RC).
- `current_db`: The name of the current database.
- `txn_mode`: The transactional mode. Value options are `OPTIMISTIC` and `PESSIMISTIC`.
- `sql`: The SQL statement corresponding to the current query.

14.4.1.145 tidb_guarantee_linearizability New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls the way commit TS is calculated for async commit. By default (with the `ON` value), the two-phase commit requests a new TS from the PD server and uses the TS to calculate the final commit TS. In this situation, linearizability is guaranteed for all the concurrent transactions.
- If you set this variable to `OFF`, the process of fetching TS from the PD server is skipped, with the cost that only causal consistency is guaranteed but not linearizability. For more details, see the blog post Async Commit, the Accelerator for Transaction Commit in TiDB 5.0.
- For scenarios that require only causal consistency, you can set this variable to `OFF` to improve performance.

14.4.1.146 tidb_hash_exchange_with_new_collation

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls whether the MPP hash partition exchange operator is generated in a cluster with new collation enabled. `true` means to generate the operator, and `false` means not to generate it.
- This variable is used for the internal operation of TiDB. It is **NOT recommended** to set this variable.

14.4.1.147 tidb_hash_join_concurrency

> **Warning:**
>
> Since v5.0, this variable is deprecated. Instead, use `tidb_executor_concurrency` for setting.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of the `hash join` algorithm.
- A value of `-1` means that the value of `tidb_executor_concurrency` will be used instead.

14.4.1.148 tidb_hashagg_final_concurrency

> **Warning:**
>
> Since v5.0, this variable is deprecated. Instead, use `tidb_executor_concurrency` for setting.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of executing the concurrent `hash aggregation` algorithm in the `final` phase.
- When the parameter of the aggregate function is not distinct, `HashAgg` is run concurrently and respectively in two phases - the `partial` phase and the `final` phase.
- A value of `-1` means that the value of `tidb_executor_concurrency` will be used instead.

14.4.1.149 tidb_hashagg_partial_concurrency

> **Warning:**
>
> Since v5.0, this variable is deprecated. Instead, use `tidb_executor_concurrency` for setting.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of executing the concurrent `hash aggregation` algorithm in the `partial` phase.
- When the parameter of the aggregate function is not distinct, `HashAgg` is run concurrently and respectively in two phases - the `partial` phase and the `final` phase.
- A value of `-1` means that the value of `tidb_executor_concurrency` will be used instead.

14.4.1.150 tidb_ignore_prepared_cache_close_stmt New in v6.0.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether to ignore the commands for closing prepared statement cache.
- When this variable is set to `ON`, the `COM_STMT_CLOSE` command of the Binary protocol and the `DEALLOCATE PREPARE` statement of the text protocol are ignored. For details, see Ignore the `COM_STMT_CLOSE` command and the `DEALLOCATE PREPARE` statement.

14.4.1.151 tidb_index_join_batch_size

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `25000`
- Range: `[1, 2147483647]`
- Unit: Rows
- This variable is used to set the batch size of the `index lookup join` operation.
- Use a bigger value in OLAP scenarios, and a smaller value in OLTP scenarios.

14.4.1.152 tidb_index_lookup_concurrency

> **Warning:**
>
> Since v5.0, this variable is deprecated. Instead, use `tidb_executor_concurrency` for setting.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of the `index lookup` operation.
- Use a bigger value in OLAP scenarios, and a smaller value in OLTP scenarios.
- A value of `-1` means that the value of `tidb_executor_concurrency` will be used instead.

14.4.1.153 tidb_index_lookup_join_concurrency

> **Warning:**
>
> Since v5.0, this variable is deprecated. Instead, use `tidb_executor_concurrency` for setting.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of the `index lookup join` algorithm.
- A value of `-1` means that the value of `tidb_executor_concurrency` will be used instead.

14.4.1.154 tidb_index_lookup_size

- Scope: SESSION | GLOBAL

- Persists to cluster: Yes
- Type: Integer
- Default value: `20000`
- Range: `[1, 2147483647]`
- Unit: Rows
- This variable is used to set the batch size of the `index lookup` operation.
- Use a bigger value in OLAP scenarios, and a smaller value in OLTP scenarios.

14.4.1.155 tidb_index_serial_scan_concurrency

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of the `serial scan` operation.
- Use a bigger value in OLAP scenarios, and a smaller value in OLTP scenarios.

14.4.1.156 tidb_init_chunk_size

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `32`
- Range: `[1, 32]`
- Unit: Rows
- This variable is used to set the number of rows for the initial chunk during the execution process. The number of rows for a chunk directly affects the amount of memory required for a single query. You can roughly estimate the memory needed for a single chunk by considering the total width of all columns in the query and the number of rows for the chunk. Combining this with the concurrency of the executor, you can make a rough estimation of the total memory required for a single query. It is recommended that the total memory for a single chunk does not exceed 16 MiB.

14.4.1.157 tidb_isolation_read_engines New in v4.0

- Scope: SESSION
- Default value: `tikv,tiflash,tidb`
- This variable is used to set the storage engine list that TiDB can use when reading data.

14.4.1.158 tidb_last_ddl_info New in v6.0.0

- Scope: SESSION
- Persists to cluster: No
- Default value: “”
- Type: String
- This is a read-only variable. It is internally used in TiDB to get the information of the last DDL operation within the current session.
    - “query”: The last DDL query string.
    - “seq_num”: The sequence number for each DDL operation. It is used to identify the order of DDL operations.

14.4.1.159 tidb_last_query_info New in v4.0.14

- Scope: SESSION
- Persists to cluster: No
- Default value: “”
- This is a read-only variable. It is internally used in TiDB to query the transaction information of the last DML statement. The information includes:
    - `txn_scope`: The scope of the transaction, which can be `global` or `local`.
    - `start_ts`: The start timestamp of the transaction.
    - `for_update_ts`: The `for_update_ts` of the previously executed DML statement. This is an internal term of TiDB used for tests. Usually, you can ignore this information.
    - `error`: The error message, if any.

14.4.1.160 tidb_last_txn_info New in v4.0.9

- Scope: SESSION
- Persists to cluster: No
- Type: String
- This variable is used to get the last transaction information within the current session. It is a read-only variable. The transaction information includes:
    - The transaction scope.
    - The start and commit TS.
    - The transaction commit mode, which might be a two-phase, one-phase, or async commit.
    - The information of transaction fallback from async commit or one-phase commit to two-phase commit.
    - The error encountered.

14.4.1.161 tidb_log_file_max_days New in v5.3.0

- Scope: SESSION
- Type: Integer
- Default value: `0`
- Range: `[0, 2147483647]`

- This variable is used to set the maximum days that the log is retained on the current TiDB instance. Its value defaults to the value of the `max-days` configuration in the configuration file. Changing the variable value only affects the current TiDB instance. After TiDB is restarted, the variable value is reset and the configuration value is not affected.

14.4.1.162 tidb_low_resolution_tso

- Scope: SESSION
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether to enable the low precision TSO feature. After this feature is enabled, new transactions use a timestamp updated every 2 seconds to read data.
- The main applicable scenario is to reduce the overhead of acquiring TSO for small read-only transactions when reading old data is acceptable.

14.4.1.163 tidb_max_chunk_size

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1024`
- Range: `[32, 2147483647]`
- Unit: Rows
- This variable is used to set the maximum number of rows in a chunk during the execution process. Setting to too large of a value may cause cache locality issues. The recommended value for this variable is no larger than 65536. The number of rows for a chunk directly affects the amount of memory required for a single query. You can roughly estimate the memory needed for a single chunk by considering the total width of all columns in the query and the number of rows for the chunk. Combining this with the concurrency of the executor, you can make a rough estimation of the total memory required for a single query. It is recommended that the total memory for a single chunk does not exceed 16 MiB. When the query involves a large amount of data and a single chunk is insufficient to handle all the data, TiDB processes it multiple times, doubling the chunk size with each processing iteration, starting from `tidb_init_chunk_size` until the chunk size reaches the value of `tidb_max_chunk_size`.

14.4.1.164 tidb_max_delta_schema_count New in v2.1.18 and v3.0.5

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1024`
- Range: `[100, 16384]`
- This variable is used to set the maximum number of schema versions (the table IDs modified for corresponding versions) allowed to be cached. The value range is 100 ~ 16384.

14.4.1.165 tidb_mem_oom_action New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `CANCEL`
- Possible values: `CANCEL`, `LOG`

- Specifies what operation TiDB performs when a single SQL statement exceeds the memory quota specified by `tidb_mem_quota_query` and cannot be spilled over to disk. See TiDB Memory Control for details.

- The default value is `CANCEL`, but in TiDB v4.0.2 and earlier versions, the default value is `LOG`.
- This setting was previously a `tidb.toml` option (`oom-action`), but changed to a system variable starting from TiDB v6.1.0.

14.4.1.166 tidb_mem_quota_analyze New in v6.1.0

> **Warning:**
>
> Currently, the `ANALYZE` memory quota is an experimental feature, and the memory statistics might be inaccurate in production environments.

- Scope: GLOBAL
- Persists to cluster: Yes
- Default value: `0`
- Unit: Bytes

- This variable controls the maximum memory usage of TiDB updating statistics. Such a memory usage occurs when you manually execute `ANALYZE TABLE` and when TiDB automatically analyzes tasks in the background. When the total memory usage exceeds this threshold, user-executed `ANALYZE` will exit, and an error message is reported that reminds you to try a lower sampling rate or retry later. If the automatic task in the TiDB background exits because the memory threshold is exceeded, and the sampling rate used is higher than the default value, TiDB will retry the update using the default sampling rate. When this variable value is negative or zero, TiDB does not limit the memory usage of both the manual and automatic update tasks.

> **Note:**
>
> `auto_analyze` will be triggered in a TiDB cluster only when `run-auto-analyze` is enabled in the TiDB startup configuration file.

14.4.1.167 tidb_mem_quota_apply_cache New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `33554432` (32 MiB)
- Range: `[0, 9223372036854775807]`
- Unit: Bytes
- This variable is used to set the memory usage threshold of the local cache in the `Apply` operator.
- The local cache in the `Apply` operator is used to speed up the computation of the `Apply` operator. You can set the variable to `0` to disable the `Apply` cache feature.

14.4.1.168 tidb_mem_quota_binding_cache New in v6.0.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `67108864`
- Range: `[0, 2147483647]`
- Unit: Bytes
- This variable is used to set the threshold of the memory used for caching bindings.
- If a system creates or captures excessive bindings, resulting in overuse of memory space, TiDB returns a warning in the log. In this case, the cache cannot hold all available bindings or determine which bindings to store. For this reason, some queries might miss their bindings. To address this problem, you can increase the value of this

variable, which increases the memory used for caching bindings. After modifying this parameter, you need to run `admin reload bindings` to reload bindings and validate the modification.

14.4.1.169 tidb_mem_quota_query

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1073741824` (1 GiB)
- Range: `[-1, 9223372036854775807]`
- Unit: Bytes
- This variable is used to set the threshold value of memory quota for a query.
- If the memory quota of a query during execution exceeds the threshold value, TiDB performs the operation designated by `tidb_mem_oom_action`.

- For versions earlier than TiDB v6.1.0, this is a session scope variable and uses the value of `mem-quota-query` from `tidb.toml` as an initial value. Starting from v6.1.0, `tidb_mem_quota_query` is a `SESSION | GLOBAL` scope variable.

14.4.1.170 tidb_memory_usage_alarm_ratio

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Float
- Default value: `0.8`
- Range: `[0, 1]`

- TiDB triggers an alarm when the percentage of the memory it takes exceeds a certain threshold. For the detailed usage description of this feature, see memory-usage-alarm-ratio.
- You can set the initial value of this variable by configuring memory-usage-alarm-ratio.

14.4.1.171 tidb_merge_join_concurrency

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `1`
- This variable sets the concurrency of the `MergeJoin` operator when a query is executed.
- It is **NOT recommended** to set this variable. Modifying the value of this variable might cause data correctness issues.

14.4.1.172 tidb_metric_query_range_duration New in v4.0

- Scope: SESSION
- Type: Integer
- Default value: `60`
- Range: `[10, 216000]`
- Unit: Seconds
- This variable is used to set the range duration of the Prometheus statement generated when querying `METRICS_SCHEMA`.

14.4.1.173 tidb_metric_query_step New in v4.0

- Scope: SESSION
- Type: Integer
- Default value: `60`
- Range: `[10, 216000]`
- Unit: Seconds
- This variable is used to set the step of the Prometheus statement generated when querying `METRICS_SCHEMA`.

14.4.1.174 tidb_multi_statement_mode New in v4.0.11

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `OFF`
- Possible values: `OFF`, `ON`, `WARN`
- This variable controls whether to allow multiple queries to be executed in the same `COM_QUERY` call.
- To reduce the impact of SQL injection attacks, TiDB now prevents multiple queries from being executed in the same `COM_QUERY` call by default. This variable is intended to be used as part of an upgrade path from earlier versions of TiDB. The following behaviors apply:

Client setting	`tidb_multi_statement_mode` value	Multiple statements permitted?
Multiple Statements = ON	OFF	Yes
Multiple Statements = ON	ON	Yes
Multiple Statements = ON	WARN	Yes
Multiple Statements = OFF	OFF	No
Multiple Statements = OFF	ON	Yes
Multiple Statements = OFF	WARN	Yes (+warning returned)

Note:

Only the default value of `OFF` can be considered safe. Setting `tidb_multi_statement_mode=ON` might be required if your application was specifically designed for an earlier version of TiDB. If your application requires multiple statement support, it is recommended to use the setting provided by your client library instead of the `tidb_multi_statement_mode` option. For example:

- go-sql-driver (`multiStatements`)
- Connector/J (`allowMultiQueries`)
- PHP mysqli (`mysqli_multi_query`)

14.4.1.175 tidb_nontransactional_ignore_error New in v6.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `OFF`
- This variable specifies whether to return an error immediately when the error occurs in a non-transactional DML statement.
- When the value is set to `OFF`, the non-transactional DML statement stops immediately at the first error and returns the error. All the following batches are canceled.
- When the value is set to `ON` and an error occurs in a batch, the following batches will continue to be executed until all batches are executed. All errors occurred during the execution process are returned together in the result.

14.4.1.176 tidb_opt_agg_push_down

- Scope: SESSION
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether the optimizer executes the optimization operation of pushing down the aggregate function to the position before Join, Projection, and UnionAll.
- When the aggregate operation is slow in query, you can set the variable value to ON.

14.4.1.177 tidb_opt_broadcast_cartesian_join

- Scope: SESSION | GLOBAL
- Persists to cluster: YES

- Type: Integer
- Default value: `1`
- Range: `[0, 2]`
- Indicates whether to allow the Broadcast Cartesian Join.
- `0` means that the Broadcast Cartesian Join is not allowed. `1` means that it is allowed based on `tidb_broadcast_join_threshold_count`. `2` means that it is always allowed even if the table size exceeds the threshold.
- This variable is internally used in TiDB, and it is **NOT** recommended to modify its value.

14.4.1.178 tidb_opt_concurrency_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `3.0`
- Indicates the CPU cost of starting a Golang goroutine in TiDB. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.179 tidb_opt_copcpu_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `3.0`
- Indicates the CPU cost for TiKV Coprocessor to process one row. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.180 tidb_opt_correlation_exp_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1`
- Range: `[0, 2147483647]`
- When the method that estimates the number of rows based on column order correlation is not available, the heuristic estimation method is used. This variable is used to control the behavior of the heuristic method.
    - When the value is 0, the heuristic method is not used.
    - When the value is greater than 0:

- A larger value indicates that an index scan will probably be used in the heuristic method.
- A smaller value indicates that a table scan will probably be used in the heuristic method.

14.4.1.181 tidb_opt_correlation_threshold

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Float
- Default value: `0.9`
- Range: `[0, 1]`
- This variable is used to set the threshold value that determines whether to enable estimating the row count by using column order correlation. If the order correlation between the current column and the `handle` column exceeds the threshold value, this method is enabled.

14.4.1.182 tidb_opt_cpu_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `3.0`
- Indicates the CPU cost for TiDB to process one row. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.183 tidb_opt_desc_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `3.0`
- Indicates the cost for TiKV to scan one row from the disk in descending order. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.184 tidb_opt_disk_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES

- Type: Float
- Range: `[0, 2147483647]`
- Default value: `1.5`
- Indicates the I/O cost for TiDB to read or write one byte of data from or to the temporary disk. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.185 tidb_opt_distinct_agg_push_down

- Scope: SESSION
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether the optimizer executes the optimization operation of pushing down the aggregate function with `distinct` (such as `select count(distinct a)from t`) to Coprocessor.
- When the aggregate function with the `distinct` operation is slow in the query, you can set the variable value to `1`.

In the following example, before `tidb_opt_distinct_agg_push_down` is enabled, TiDB needs to read all data from TiKV and execute `distinct` on the TiDB side. After `tidb_opt_distinct_agg_push_down` is enabled, `distinct a` is pushed down to Coprocessor, and a `group by` column `test.t.a` is added to `HashAgg_5`.

```
mysql> desc select count(distinct a) from test.t;
+--
   ↪ ----------------------+----------+-----------+---------------+----------------
   ↪
| id                    | estRows | task      | access object | operator info
   ↪                     |
+--
   ↪ ----------------------+----------+-----------+---------------+----------------
   ↪
| StreamAgg_6           | 1.00     | root      |               | funcs:count(
   ↪ distinct test.t.a)->Column#4 |
| ─TableReader_10       | 10000.00 | root      |               | data:
   ↪ TableFullScan_9                |
|   ─TableFullScan_9    | 10000.00 | cop[tikv] | table:t       | keep order:false
   ↪ , stats:pseudo      |
+--
   ↪ ----------------------+----------+-----------+---------------+----------------
   ↪
3 rows in set (0.01 sec)

mysql> set session tidb_opt_distinct_agg_push_down = 1;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> desc select count(distinct a) from test.t;
+--
   ↪ ------------------------+---------+----------+--------------+--------------
   ↪
| id                     | estRows | task     | access object | operator info
   ↪                      |
+--
   ↪ ------------------------+---------+----------+--------------+--------------
   ↪
| HashAgg_8              | 1.00    | root     |               | funcs:count(
   ↪ distinct test.t.a)->Column#3 |
| ─TableReader_9         | 1.00    | root     |               | data:HashAgg_5
   ↪                        |
|   ─HashAgg_5           | 1.00    | cop[tikv] |              | group by:test.
   ↪ t.a,                   |
|     ─TableFullScan_7   | 10000.00 | cop[tikv] | table:t   | keep order:
   ↪ false, stats:pseudo      |
+--
   ↪ ------------------------+---------+----------+--------------+--------------
   ↪
4 rows in set (0.00 sec)
```

14.4.1.186 tidb_opt_enable_correlation_adjustment

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether the optimizer estimates the number of rows based on column order correlation

14.4.1.187 tidb_opt_insubq_to_join_and_agg

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to set whether to enable the optimization rule that converts a subquery to join and aggregation.

- For example, after you enable this optimization rule, the subquery is converted as follows:

```
select * from t where t.a in (select aa from t1);
```

The subquery is converted to join as follows:

```
select t.* from t, (select aa from t1 group by aa) tmp_t where t.a =
    ↪ tmp_t.aa;
```

If `t1` is limited to be `unique` and `not null` in the `aa` column. You can use the following statement, without aggregation.

```
select t.* from t, t1 where t.a=t1.aa;
```

14.4.1.188 tidb_opt_join_reorder_threshold

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `0`
- Range: `[0, 2147483647]`
- This variable is used to control the selection of the TiDB Join Reorder algorithm. When the number of nodes participating in Join Reorder is greater than this threshold, TiDB selects the greedy algorithm, and when it is less than this threshold, TiDB selects the dynamic programming algorithm.
- Currently, for OLTP queries, it is recommended to keep the default value. For OLAP queries, it is recommended to set the variable value to 10~15 to get better connection orders in OLAP scenarios.

14.4.1.189 tidb_opt_limit_push_down_threshold

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `100`
- Range: `[0, 2147483647]`
- This variable is used to set the threshold that determines whether to push the Limit or TopN operator down to TiKV.
- If the value of the Limit or TopN operator is smaller than or equal to this threshold, these operators are forcibly pushed down to TiKV. This variable resolves the issue that the Limit or TopN operator cannot be pushed down to TiKV partly due to wrong estimation.

14.4.1.190 tidb_opt_memory_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `0.001`
- Indicates the memory cost for TiDB to store one row. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.191 tidb_opt_mpp_outer_join_fixed_build_side New in v5.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- When the variable value is `ON`, the left join operator always uses inner table as the build side and the right join operator always uses outer table as the build side. If you set the value to `OFF`, the outer join operator can use either side of the tables as the build side.

14.4.1.192 tidb_opt_network_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `1.0`
- Indicates the net cost of transferring 1 byte of data through the network. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.193 tidb_opt_prefer_range_scan New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- After you set the value of this variable to `ON`, the optimizer always prefers range scans over full table scans.
- In the following example, before you enable `tidb_opt_prefer_range_scan`, the TiDB optimizer performs a full table scan. After you enable `tidb_opt_prefer_range_scan`, the optimizer selects an index range scan.

```
explain select * from t where age=5;
+--
   ↪ ----------------------+-----------+----------+--------------+--------------
   ↪
| id                    | estRows   | task     | access object | operator info
   ↪  |
+--
   ↪ ----------------------+-----------+----------+--------------+--------------
   ↪
| TableReader_7         | 1048576.00 | root   |              | data:
   ↪ Selection_6 |
| └─Selection_6         | 1048576.00 | cop[tikv] |           | eq(test.t.age,
   ↪  5) |
|   └─TableFullScan_5   | 1048576.00 | cop[tikv] | table:t  | keep order:
   ↪ false |
+--
   ↪ ----------------------+-----------+----------+--------------+--------------
   ↪
3 rows in set (0.00 sec)

set session tidb_opt_prefer_range_scan = 1;

explain select * from t where age=5;
+--
   ↪ ----------------------------+-----------+----------+-----------------------
   ↪
| id                          | estRows   | task     | access object
   ↪           | operator info              |
+--
   ↪ ----------------------------+-----------+----------+-----------------------
   ↪
| IndexLookUp_7               | 1048576.00 | root   |
   ↪                           |                          |
| ├─IndexRangeScan_5(Build)  | 1048576.00 | cop[tikv] | table:t, index:
   ↪ idx_age(age) | range:[5,5], keep order:false |
| └─TableRowIDScan_6(Probe)  | 1048576.00 | cop[tikv] | table:t
   ↪               | keep order:false          |
+--
   ↪ ----------------------------+-----------+----------+-----------------------
   ↪
3 rows in set (0.00 sec)
```

14.4.1.194 tidb_opt_projection_push_down New in v6.1.0

- Scope: SESSION
- Persists to cluster: No
- Type: Boolean
- Default value: `OFF`
- Specifies whether to allow the optimizer to push `Projection` down to the TiKV or TiFlash coprocessor.

14.4.1.195 tidb_opt_scan_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `1.5`
- Indicates the cost for TiKV to scan one row of data from the disk in ascending order. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.196 tidb_opt_seek_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `20`
- Indicates the start-up cost for TiDB to request data from TiKV. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.197 tidb_opt_tiflash_concurrency_factor

- Scope: SESSION | GLOBAL
- Persists to cluster: YES
- Type: Float
- Range: `[0, 2147483647]`
- Default value: `24.0`
- Indicates the concurrency number of TiFlash computation. This variable is internally used in the Cost Model, and it is **NOT** recommended to modify its value.

14.4.1.198 tidb_opt_write_row_id

- Scope: SESSION
- Default value: `OFF`
- This variable is used to control whether to allow `INSERT`, `REPLACE`, and `UPDATE` statements to operate on the `_tidb_rowid` column. This variable can be used only when you import data using TiDB tools.

14.4.1.199 tidb_optimizer_selectivity_level

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `1`
- Value options: `1` and `2` (not recommended)
- This variable controls the iteration of the optimizer's estimation logic. After changing the value of this variable, the estimation logic of the optimizer will change greatly. Currently, `1` is the only valid value. It is not recommended to set the value to `2`.

14.4.1.200 tidb_partition_prune_mode New in v5.1

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `static`
- Specifies whether to enable `dynamic` mode for partitioned tables. For details about the dynamic pruning mode, see Dynamic Pruning Mode for Partitioned Tables.

14.4.1.201 tidb_persist_analyze_options New in v5.4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls whether to enable the ANALYZE configuration persistence feature.

14.4.1.202 tidb_placement_mode New in v6.0.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `STRICT`
- Possible values: `STRICT`, `IGNORE`

- This variable controls whether DDL statements ignore the placement rules specified in SQL. When the variable value is `IGNORE`, all placement rule options are ignored.

- It is intended to be used by logical dump/restore tools to ensure that tables can always be created even if invalid placement rules are assigned. This is similar to how mysqldump writes `SET FOREIGN_KEY_CHECKS=0;` to the start of every dump file.

14.4.1.203 tidb_pprof_sql_cpu New in v4.0

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Integer
- Default value: `0`
- Range: `[0, 1]`
- This variable is used to control whether to mark the corresponding SQL statement in the profile output to identify and troubleshoot performance issues.

14.4.1.204 tidb_prepared_plan_cache_memory_guard_ratio New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Float
- Default value: `0.1`
- Range: `[0, 1]`
- The threshold at which the prepared plan cache triggers a memory protection mechanism. For details, see Memory management of Prepared Plan Cache.
- This setting was previously a `tidb.toml` option (`prepared-plan-cache.memory-guard-ratio`), but changed to a system variable starting from TiDB v6.1.0.

14.4.1.205 tidb_prepared_plan_cache_size New in v6.1.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `100`
- Range: `[1, 100000]`
- The maximum number of plans that can be cached in a session. For details, see Memory management of Prepared Plan Cache.
- This setting was previously a `tidb.toml` option (`prepared-plan-cache.capacity`), but changed to a system variable starting from TiDB v6.1.0.

14.4.1.206 tidb_projection_concurrency

> **Warning:**
>
> Since v5.0, this variable is deprecated. Instead, use `tidb_executor_concurrency` for setting.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[-1, 256]`
- Unit: Threads
- This variable is used to set the concurrency of the `Projection` operator.
- A value of `-1` means that the value of `tidb_executor_concurrency` will be used instead.

14.4.1.207 tidb_query_log_max_len

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `4096` (4 KiB)
- Range: `[0, 1073741824]`
- Unit: Bytes
- The maximum length of the SQL statement output. When the output length of a statement is larger than the `tidb_query_log_max_len` value, the statement is truncated to output.
- This setting was previously also available as a `tidb.toml` option (`log.query-log-max-len`), but is only a system variable starting from TiDB v6.1.0.

14.4.1.208 tidb_rc_read_check_ts New in v6.0.0

Warning:

- This feature is incompatible with `replica-read`. Do not enable `tidb_rc_read_check_ts` and `replica-read` at the same time.
- If your client uses a cursor, it is not recommended to enable `tidb_rc_read_check_ts` in case that the previous batch of returned data has already been used by the client and the statement eventually fails.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`

- This variable is used to optimize the timestamp acquisition, which is suitable for scenarios with read-committed isolation level where read-write conflicts are rare. Enabling this variable can avoid the latency and cost of getting the global timestamp, and can optimize the transaction-level read latency.
- If read-write conflicts are severe, enabling this feature will increase the cost and latency of getting the global timestamp, and might cause performance regression. For details, see Read Committed isolation level.

14.4.1.209 tidb_read_consistency New in v5.4.0

- Scope: SESSION
- Persists to cluster: No
- Type: String
- Default value: `strict`
- This variable is used to control the read consistency for an auto-commit read statement.
- If the variable value is set to `weak`, the locks encountered by the read statement are skipped directly and the read execution might be faster, which is the weak consistency read mode. However, the transaction semantics (such as atomicity) and distributed consistency (such as linearizability) are not guaranteed.
- For user scenarios where the auto-commit read needs to return fast and weak consistency read results are acceptable, you can use the weak consistency read mode.

14.4.1.210 tidb_read_staleness New in v5.4.0

- Scope: SESSION
- Type: Integer
- Default value: `0`
- Range: `[-2147483648, 0]`
- This variable is used to set the time range of historical data that TiDB can read in the current session. After setting the value, TiDB selects a timestamp as new as possible from the range allowed by this variable, and all subsequent read operations are performed against this timestamp. For example, if the value of this variable is set to `-5`, on the condition that TiKV has the corresponding historical version's data, TiDB selects a timestamp as new as possible within a 5-second time range.

14.4.1.211 tidb_record_plan_in_slow_log

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to include the execution plan of slow queries in the slow log.

14.4.1.212 tidb_redact_log

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls whether to hide user information in the SQL statement being recorded into the TiDB log and slow log.
- When you set the variable to `1`, user information is hidden. For example, if the executed SQL statement is `insert into t values (1,2)`, the statement is recorded as `insert into t values (?,?)` in the log.

14.4.1.213 tidb_regard_null_as_point New in v5.4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls whether the optimizer can use a query condition including null equivalence as a prefix condition for index access.
- This variable is enabled by default. When it is enabled, the optimizer can reduce the volume of index data to be accessed, which accelerates query execution. For example, if a query involves multiple-column indexes `index(a, b)` and the query condition contains `a<=>null and b=1`, the optimizer can use both `a<=>null` and `b=1` in the query condition for index access. If the variable is disabled, because `a<=>null and b=1` includes the null equivalence condition, the optimizer does not use `b=1` for index access.

14.4.1.214 tidb_remove_orderby_in_subquery New in v6.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- Specifies whether to remove `ORDER BY` clause in a subquery.

14.4.1.215 tidb_replica_read New in v4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `leader`
- Possible values: `leader`, `follower`, `leader-and-follower`, `closest-replicas`

- This variable is used to control where TiDB reads data. Here are three options:
  - leader: Read only from leader node
  - follower: Read only from follower node
  - leader-and-follower: Read from leader or follower node
- See follower reads for additional details.

14.4.1.216 tidb_restricted_read_only New in v5.2.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- `tidb_restricted_read_only` and `tidb_super_read_only` behave similarly. In most cases, you should use `tidb_super_read_only` only.
- Users with the `SUPER` or `SYSTEM_VARIABLES_ADMIN` privilege can modify this variable. However, if the Security Enhanced Mode is enabled, the additional `RESTRICTED_VARIABLES_ADMIN` privilege is required to read or modify this variable.
- `tidb_restricted_read_only` affects `tidb_super_read_only` in the following cases:
  - Setting `tidb_restricted_read_only` to `ON` will update `tidb_super_read_only` to `ON`.
  - Setting `tidb_restricted_read_only` to `OFF` leaves `tidb_super_read_only` unchanged.
  - If `tidb_restricted_read_only` is `ON`, `tidb_super_read_only` cannot be set to `OFF`.
- For DBaaS providers of TiDB, if a TiDB cluster is a downstream database of another database, to make the TiDB cluster read-only, you might need to use `tidb_restricted_read_only` with Security Enhanced Mode enabled, which prevents your customers from using `tidb_super_read_only` to make the cluster writable. To achieve this, you need to enable Security Enhanced Mode, use an admin user with the `SYSTEM_VARIABLES_ADMIN` and `RESTRICTED_VARIABLES_ADMIN` privileges to control `tidb_restricted_read_only`, and let your database users use the root user with the `SUPER` privilege to control `tidb_super_read_only` only.
- This variable controls the read-only status of the entire cluster. When the variable is `ON`, all TiDB servers in the entire cluster are in the read-only mode. In this case, TiDB only executes the statements that do not modify data, such as `SELECT`, `USE`, and `SHOW`. For other statements such as `INSERT` and `UPDATE`, TiDB rejects executing those statements in the read-only mode.
- Enabling the read-only mode using this variable only ensures that the entire cluster finally enters the read-only status. If you have changed the value of this variable in a TiDB cluster but the change has not yet propagated to other TiDB servers, the un-updated TiDB servers are still **not** in the read-only mode.

- When this variable is enabled, the SQL statements being executed are not affected. TiDB only performs the read-only check for the SQL statements **to be** executed.
- When this variable is enabled, TiDB handles the uncommitted transactions in the following ways:
    - For uncommitted read-only transactions, you can commit the transactions normally.
    - For uncommitted transactions that are not read-only, SQL statements that perform write operations in these transactions are rejected.
    - For uncommitted read-only transactions with modified data, the commit of these transactions is rejected.
- After the read-only mode is enabled, all users (including the users with the `SUPER` privilege) cannot execute the SQL statements that might write data unless the user is explicitly granted the `RESTRICTED_REPLICA_WRITER_ADMIN` privilege.

14.4.1.217 tidb_retry_limit

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `10`
- Range: `[-1, 9223372036854775807]`
- This variable is used to set the maximum number of the retries for optimistic transactions. When a transaction encounters retryable errors (such as transaction conflicts, very slow transaction commit, or table schema changes), this transaction is re-executed according to this variable. Note that setting `tidb_retry_limit` to `0` disables the automatic retry. This variable only applies to optimistic transactions, not to pessimistic transactions.

14.4.1.218 tidb_row_format_version

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `2`
- Range: `[1, 2]`
- Controls the format version of the newly saved data in the table. In TiDB v4.0, the new storage row format version `2` is used by default to save new data.
- If you upgrade from a TiDB version earlier than 4.0.0 to 4.0.0, the format version is not changed, and TiDB continues to use the old format of version `1` to write data to the table, which means that **only newly created clusters use the new data format by default**.
- Note that modifying this variable does not affect the old data that has been saved, but applies the corresponding version format only to the newly written data after modifying this variable.

14.4.1.219 tidb_scatter_region

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- By default, Regions are split for a new table when it is being created in TiDB. After this variable is enabled, the newly split Regions are scattered immediately during the execution of the `CREATE TABLE` statement. This applies to the scenario where data need to be written in batches right after the tables are created in batches, because the newly split Regions can be scattered in TiKV beforehand and do not have to wait to be scheduled by PD. To ensure the continuous stability of writing data in batches, the `CREATE TABLE` statement returns success only after the Regions are successfully scattered. This makes the statement's execution time multiple times longer than that when you disable this variable.
- Note that if `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` have been set when a table is created, the specified number of Regions are evenly split after the table creation.

14.4.1.220 tidb_shard_allocate_step New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `9223372036854775807`
- Range: `[1, 9223372036854775807]`
- This variable controls the maximum number of continuous IDs to be allocated for the `AUTO_RANDOM` or `SHARD_ROW_ID_BITS` attribute. Generally, `AUTO_RANDOM` IDs or the `SHARD_ROW_ID_BITS` annotated row IDs are incremental and continuous in one transaction. You can use this variable to solve the hotspot issue in large transaction scenarios.

14.4.1.221 tidb_skip_ascii_check New in v5.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether to skip ASCII validation.
- Validating ASCII characters affects the performance. When you are sure that the input characters are valid ASCII characters, you can set the variable value to `ON`.

14.4.1.222 tidb_skip_isolation_level_check

- Scope: SESSION | GLOBAL

- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- After this switch is enabled, if an isolation level unsupported by TiDB is assigned to `tx_isolation`, no error is reported. This helps improve compatibility with applications that set (but do not depend on) a different isolation level.

```
tidb> set tx_isolation='serializable';
ERROR 8048 (HY000): The isolation level 'serializable' is not supported.
    ↪ Set tidb_skip_isolation_level_check=1 to skip this error
tidb> set tidb_skip_isolation_level_check=1;
Query OK, 0 rows affected (0.00 sec)

tidb> set tx_isolation='serializable';
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

14.4.1.223 tidb_skip_utf8_check

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to set whether to skip UTF-8 validation.
- Validating UTF-8 characters affects the performance. When you are sure that the input characters are valid UTF-8 characters, you can set the variable value to `ON`.

> **Note:**
>
> If the character check is skipped, TiDB might fail to detect illegal UTF-8 characters written by the application, cause decoding errors when `ANALYZE` is executed, and introduce other unknown encoding issues. If your application cannot guarantee the validity of the written string, it is not recommended to skip the character check.

14.4.1.224 tidb_slow_log_threshold

- Scope: GLOBAL
- Persists to cluster: No, only applicable to the current TiDB instance that you are connecting to.
- Type: Integer

- Default value: `300`
- Range: `[-1, 9223372036854775807]`
- Unit: Milliseconds
- This variable outputs the threshold value of the time consumed by the slow log, and is set to 300 milliseconds by default. When the time consumed by a query is larger than this value, this query is considered as a slow query and its log is output to the slow query log. Note that when the output level of `log.level` is `"debug"`, all queries are recorded in the slow query log, regardless of the setting of this variable.

14.4.1.225 tidb_max_tiflash_threads New in v6.1.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `-1`
- Range: `[-1, 256]`
- This variable is used to set the maximum concurrency for TiFlash to execute a request. The default value is `-1`, indicating that this system variable is invalid and the maximum concurrency depends on the setting of the TiFlash configuration `profiles.default.max_threads`. When the value is `0`, the maximum number of threads is automatically configured by TiFlash.

14.4.1.226 tidb_mpp_store_fail_ttl

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Duration
- Default value: `60s`
- The newly started TiFlash does not provide services. To prevent queries from failing, TiDB limits the tidb-server sending queries to the newly started TiFlash nodes. This variable indicates the time range in which the newly started TiFlash is not sent requests.

14.4.1.227 tidb_slow_query_file

- Scope: SESSION
- Default value: “”
- When `INFORMATION_SCHEMA.SLOW_QUERY` is queried, only the slow query log name set by `slow-query-file` in the configuration file is parsed. The default slow query log name is “tidb-slow.log”. To parse other logs, set the `tidb_slow_query_file` session variable to a specific file path, and then query `INFORMATION_SCHEMA.SLOW_QUERY` to parse the slow query log based on the set file path.

For details, see Identify Slow Queries.

14.4.1.228 tidb_snapshot

- Scope: SESSION
- Default value: “”
- This variable is used to set the time point at which the data is read by the session. For example, when you set the variable to “2017-11-11 20:20:20” or a TSO number like “400036290571534337”, the current session reads the data of this moment.

14.4.1.229 tidb_stats_cache_mem_quota New in v6.1.0

> **Warning:**
>
> This variable is an experimental feature. It is not recommended to use it in production environments.

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `0`
- Range: `[0, 1099511627776]`
- This variable sets the memory quota for the TiDB statistics cache.

14.4.1.230 tidb_stats_load_pseudo_timeout New in v5.4.0

> **Warning:**
>
> Currently, synchronously loading statistics is an experimental feature. It is not recommended that you use it in production environments.

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable controls how TiDB behaves when the waiting time of SQL optimization reaches the timeout to synchronously load complete column statistics. The default value `OFF` means that SQL execution fails after the timeout. If you set this variable to `ON`, the SQL optimization gets back to using pseudo statistics after the timeout.

14.4.1.231 tidb_stats_load_sync_wait New in v5.4.0

> **Warning:**
>
> Currently, synchronously loading statistics is an experimental feature. It is not recommended that you use it in production environments.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `0`
- Range: `[0, 2147483647]`
- Unit: Milliseconds
- This variable controls whether to enable the synchronously loading statistics feature. The default value `0` means that the feature is disabled. To enable the feature, you can set this variable to a timeout (in milliseconds) that SQL optimization can wait for at most to synchronously load complete column statistics. For details, see Load statistics.

14.4.1.232 tidb_stmt_summary_history_size New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `24`
- Range: `[0, 255]`
- This variable is used to set the history capacity of statement summary tables.

14.4.1.233 tidb_stmt_summary_internal_query New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- This variable is used to control whether to include the SQL information of TiDB in statement summary tables.

14.4.1.234 tidb_stmt_summary_max_sql_length New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes

- Type: Integer
- Default value: `4096`
- Range: `[0, 2147483647]`
- This variable is used to control the length of the SQL string in statement summary tables.

14.4.1.235 tidb_stmt_summary_max_stmt_count New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `3000`
- Range: `[1, 32767]`
- This variable is used to set the maximum number of statements that statement summary tables store in memory.

14.4.1.236 tidb_stmt_summary_refresh_interval New in v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `1800`
- Range: `[1, 2147483647]`
- Unit: Seconds
- This variable is used to set the refresh time of statement summary tables.

14.4.1.237 tidb_store_limit New in v3.0.4 and v4.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `0`
- Range: `[0, 9223372036854775807]`
- This variable is used to limit the maximum number of requests TiDB can send to TiKV at the same time. 0 means no limit.

14.4.1.238 tidb_streamagg_concurrency

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `1`
- This variable sets the concurrency of the `StreamAgg` operator when queries are executed.
- It is **NOT recommended** to set this variable. Modifying the variable value might cause data correctness issues.

14.4.1.239 tidb_super_read_only New in v5.3.1

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`
- `tidb_super_read_only` aims to be implemented as a replacement of the MySQL variable `super_read_only`. However, because TiDB is a distributed database, `tidb_super_read_only` does not make the database read-only immediately after execution, but eventually.
- Users with the `SUPER` or `SYSTEM_VARIABLES_ADMIN` privilege can modify this variable.
- This variable controls the read-only status of the entire cluster. When the variable is `ON`, all TiDB servers in the entire cluster are in the read-only mode. In this case, TiDB only executes the statements that do not modify data, such as `SELECT`, `USE`, and `SHOW`. For other statements such as `INSERT` and `UPDATE`, TiDB rejects executing those statements in the read-only mode.
- Enabling the read-only mode using this variable only ensures that the entire cluster finally enters the read-only status. If you have changed the value of this variable in a TiDB cluster but the change has not yet propagated to other TiDB servers, the un-updated TiDB servers are still **not** in the read-only mode.
- When this variable is enabled, the SQL statements being executed are not affected. TiDB only performs the read-only check for the SQL statements **to be** executed.
- When this variable is enabled, TiDB handles the uncommitted transactions in the following ways:
    - For uncommitted read-only transactions, you can commit the transactions normally.
    - For uncommitted transactions that are not read-only, SQL statements that perform write operations in these transactions are rejected.
    - For uncommitted read-only transactions with modified data, the commit of these transactions is rejected.
- After the read-only mode is enabled, all users (including the users with the `SUPER` privilege) cannot execute the SQL statements that might write data unless the user is explicitly granted the `RESTRICTED_REPLICA_WRITER_ADMIN` privilege.
- When the `tidb_restricted_read_only` system variable is set to `ON`, `tidb_super_read_only` ↪ is affected by `tidb_restricted_read_only` in some cases. For detailed impact, see the description of `tidb_restricted_read_only`.

14.4.1.240 tidb_sysdate_is_now New in v6.0.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `OFF`

- This variable is used to control whether the `SYSDATE` function can be replaced by the `NOW` function. This configuration item has the same effect as the MySQL option `sysdate-is-now`.

14.4.1.241 tidb_table_cache_lease New in v6.0.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `3`
- Range: `[1, 10]`
- Unit: Seconds
- This variable is used to control the lease time of cached tables with a default value of `3`. The value of this variable affects the modification to cached tables. After a modification is made to cached tables, the longest waiting time might be `tidb_table_cache_lease` seconds. If the table is read-only or can accept a high write latency, you can increase the value of this variable to increase the valid time for caching tables and to reduce the frequency of lease renewal.

14.4.1.242 tidb_tmp_table_max_size New in v5.3.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `67108864`
- Range: `[1048576, 137438953472]`
- Unit: Bytes
- This variable is used to set the maximum size of a single temporary table. Any temporary table with a size larger than this variable value causes error.

14.4.1.243 tidb_top_sql_max_meta_count New in v6.0.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `5000`
- Range: `[1, 10000]`

- This variable is used to control the maximum number of SQL statement types collected by Top SQL per minute.

14.4.1.244 tidb_top_sql_max_time_series_count New in v6.0.0

> **Note:**
>
> Currently, the Top SQL page in TiDB Dashboard only displays the top 5 types of SQL queries that contribute the most to the load, which is irrelevant with the configuration of `tidb_top_sql_max_time_series_count`.

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `100`
- Range: `[1, 5000]`

- This variable is used to control how many SQL statements that contribute the most to the load (that is, top N) can be recorded by Top SQL per minute.

14.4.1.245 tidb_track_aggregate_memory_usage

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls whether TiDB tracks the memory usage of aggregate functions.

> **Warning:**
>
> If you disable this variable, TiDB might not accurately track the memory usage and cannot control the memory usage of the corresponding SQL statements.

14.4.1.246 tidb_tso_client_batch_max_wait_time New in v5.3.0

- Scope: GLOBAL
- Persists to cluster: Yes
- Type: Float
- Default value: `0`

- Range: `[0, 10]`
- Unit: Milliseconds
- This variable is used to set the maximum waiting time for a batch operation when TiDB requests TSO from PD. The default value is `0`, which means no extra waiting time.
- When obtaining TSO requests from PD each time, PD Client, used by TiDB, collects as many TSO requests received at the same time as possible. Then, PD Client merges the collected requests in batch into one RPC request and sends the request to PD. This helps reduce the pressure on PD.
- After setting this variable to a value greater than `0`, TiDB waits for the maximum duration of this value before the end of each batch merge. This is to collect more TSO requests and improve the effect of batch operations.
- Scenarios for increasing the value of this variable:
    - Due to the high pressure of TSO requests, the CPU of the PD leader reaches a bottleneck, which causes high latency of TSO RPC requests.
    - There are not many TiDB instances in the cluster, but every TiDB instance is in high concurrency.
- It is recommended to set this variable to a value as small as possible.

> **Note:**
>
> Suppose that the TSO RPC latency increases for reasons other than a CPU usage bottleneck of the PD leader (such as network issues). In this case, increasing the value of `tidb_tso_client_batch_max_wait_time` might increase the execution latency in TiDB and affect the QPS performance of the cluster.

14.4.1.247 tidb_txn_assertion_level New in v6.0.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `FAST`
- Possible values: `OFF`, `FAST`, `STRICT`
- This variable is used to control the assertion level. Assertion is a consistency check between data and indexes, which checks whether a key being written exists in the transaction commit process. For more information, see Troubleshoot Inconsistency Between Data and Indexes.

- `OFF`: Disable this check.
- `FAST`: Enable most of the check items, with almost no impact on performance.
- `STRICT`: Enable all check items, with a minor impact on pessimistic transaction performance when the system workload is high.

- For new clusters of v6.0.0 or later versions, the default value is `FAST`. For existing clusters that upgrade from versions earlier than v6.0.0, the default value is `OFF`.

14.4.1.248 tidb_txn_mode

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `pessimistic`
- Possible values: `pessimistic`, `optimistic`
- This variable is used to set the transaction mode. TiDB 3.0 supports the pessimistic transactions. Since TiDB 3.0.8, the pessimistic transaction mode is enabled by default.
- If you upgrade TiDB from v3.0.7 or earlier versions to v3.0.8 or later versions, the default transaction mode does not change. **Only the newly created clusters use the pessimistic transaction mode by default**.
- If this variable is set to "optimistic" or "", TiDB uses the optimistic transaction mode.

14.4.1.249 tidb_use_plan_baselines New in v4.0

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable is used to control whether to enable the execution plan binding feature. It is enabled by default, and can be disabled by assigning the `OFF` value. For the use of the execution plan binding, see Execution Plan Binding.

14.4.1.250 tidb_wait_split_region_finish

- Scope: SESSION
- Type: Boolean
- Default value: `ON`
- It usually takes a long time to scatter Regions, which is determined by PD scheduling and TiKV loads. This variable is used to set whether to return the result to the client after all Regions are scattered completely when the `SPLIT REGION` statement is being executed:
    - `ON` requires that the `SPLIT REGIONS` statement waits until all Regions are scattered.

    - `OFF` permits the `SPLIT REGIONS` statement to return before finishing scattering all Regions.
- Note that when scattering Regions, the write and read performances for the Region that is being scattered might be affected. In batch-write or data importing scenarios, it is recommended to import data after Regions scattering is finished.

14.4.1.251 tidb_wait_split_region_timeout

- Scope: SESSION
- Type: Integer
- Default value: `300`
- Range: `[1, 2147483647]`
- Unit: Seconds
- This variable is used to set the timeout for executing the `SPLIT REGION` statement. If a statement is not executed completely within the specified time value, a timeout error is returned.

14.4.1.252 tidb_window_concurrency New in v4.0

> **Warning:**
>
> Since v5.0, this variable is deprecated. Instead, use `tidb_executor_concurrency` for setting.

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `-1`
- Range: `[1, 256]`
- Unit: Threads
- This variable is used to set the concurrency degree of the window operator.
- A value of `-1` means that the value of `tidb_executor_concurrency` will be used instead.

14.4.1.253 time_zone

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Default value: `SYSTEM`

- This variable returns the current time zone. Values can be specified as either an offset such as '-8:00' or a named zone 'America/Los_Angeles'.
- The value `SYSTEM` means that the time zone should be the same as the system host, which is available via the `system_time_zone` variable.

14.4.1.254 timestamp

- Scope: SESSION
- Type: Float
- Default value: `0`
- Range: `[0, 2147483647]`
- A non-empty value of this variable indicates the UNIX epoch that is used as the timestamp for `CURRENT_TIMESTAMP()`, `NOW()`, and other functions. This variable might be used in data restore or replication.

14.4.1.255 transaction_isolation

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Enumeration
- Default value: `REPEATABLE-READ`
- Possible values: `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, `SERIALIZABLE`
- This variable sets the transaction isolation. TiDB advertises `REPEATABLE-READ` for compatibility with MySQL, but the actual isolation level is Snapshot Isolation. See transaction isolation levels for further details.

14.4.1.256 tx_isolation

This variable is an alias for `transaction_isolation`.

14.4.1.257 tx_isolation_one_shot

> **Note:**
>
> This variable is internally used in TiDB. You are not expected to use it.

Internally, the TiDB parser transforms the `SET TRANSACTION ISOLATION LEVEL [READ COMMITTED| REPEATABLE READ | ...]` statements to `SET @@SESSION.TX_ISOLATION_ONE_SHOT = [READ COMMITTED| REPEATABLE READ | ...]`.

14.4.1.258 tx_read_ts

- Scope: SESSION
- Persists to cluster: No
- Default value: `0`
- In the Stale Read scenarios, this session variable is used to help record the Stable Read timestamp value.
- This variable is used for the internal operation of TiDB. It is **NOT recommended** to set this variable.

14.4.1.259 txn_scope

- Scope: SESSION
- Persists to cluster: No
- Default value: `global`
- Value options: `global` and `local`
- This variable is used to set whether the current session transaction is a global transaction or a local transaction.
- This variable is used for the internal operation of TiDB. It is **NOT recommended** to set this variable.

14.4.1.260 version

- Scope: NONE
- Default value: `5.7.25-TiDB-`(tidb version)
- This variable returns the MySQL version, followed by the TiDB version. For example '5.7.25-TiDB-v4.0.0-beta.2-716-g25e003253'.

14.4.1.261 version_comment

- Scope: NONE
- Default value: (string)
- This variable returns additional details about the TiDB version. For example, 'TiDB Server (Apache License 2.0) Community Edition, MySQL 5.7 compatible'.

14.4.1.262 version_compile_machine

- Scope: NONE
- Default value: (string)
- This variable returns the name of the CPU architecture on which TiDB is running.

14.4.1.263 version_compile_os

- Scope: NONE
- Default value: (string)
- This variable returns the name of the OS on which TiDB is running.

14.4.1.264 wait_timeout

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Integer
- Default value: `28800`
- Range: `[0, 31536000]`
- Unit: Seconds
- This variable controls the idle timeout of user sessions. A zero-value means unlimited.

14.4.1.265 warning_count

- Scope: SESSION
- Default value: `0`
- This read-only variable indicates the number of warnings that occurred in the statement that was previously executed.

14.4.1.266 windowing_use_high_precision

- Scope: SESSION | GLOBAL
- Persists to cluster: Yes
- Type: Boolean
- Default value: `ON`
- This variable controls whether to use the high precision mode when computing the window functions.

14.5 Configuration File Parameters

14.5.1 TiDB Configuration File

The TiDB configuration file supports more options than command-line parameters. You can download the default configuration file `config.toml.example` and rename it to `config.toml`. This document describes only the options that are not involved in command line options.

Tip:

If you need to adjust the value of a configuration item, refer to Modify the configuration.

14.5.1.0.1 `split-table`

- Determines whether to create a separate Region for each table.
- Default value: `true`
- It is recommended to set it to `false` if you need to create a large number of tables (for example, more than 100 thousand tables).

14.5.1.0.2 `token-limit`

- The number of sessions that can execute requests concurrently.
- Type: Integer
- Default value: `1000`
- Minimum value: `1`
- Maximum Value (64-bit platforms): `18446744073709551615`
- Maximum Value (32-bit platforms): `4294967295`

14.5.1.0.3 `oom-use-tmp-storage`

- Controls whether to enable the temporary storage for some operators when a single SQL statement exceeds the memory quota specified by the system variable `tidb_mem_quota_query`.
- Default value: `true`

14.5.1.0.4 `tmp-storage-path`

- Specifies the temporary storage path for some operators when a single SQL statement exceeds the memory quota specified by the system variable `tidb_mem_quota_query`.
- Default value: `<temporary directory of OS>/<OS user ID>_tidb/MC4wLjAuMDo0MDAwLzAuMC4wL` `↪ =/tmp-storage`. `MC4wLjAuMDo0MDAwLzAuMC4wLjA6MTAwODA=` is the `Base64` encoding result of `<host>:<port>/<statusHost>:<statusPort>`.
- This configuration takes effect only when `oom-use-tmp-storage` is `true`.

14.5.1.0.5 `tmp-storage-quota`

- Specifies the quota for the storage in `tmp-storage-path`. The unit is byte.
- When a single SQL statement uses a temporary disk and the total volume of the temporary disk of the TiDB server exceeds this configuration value, the current SQL operation is cancelled and the `Out of Global Storage Quota!` error is returned.
- When the value of this configuration is smaller than `0`, the above check and limit do not apply.
- Default value: `-1`
- When the remaining available storage in `tmp-storage-path` is lower than the value defined by `tmp-storage-quota`, the TiDB server reports an error when it is started, and exits.

14.5.1.0.6 `lease`

- The timeout of the DDL lease.
- Default value: `45s`
- Unit: second

14.5.1.0.7 `compatible-kill-query`

- Determines whether to set the `KILL` statement to be MySQL compatible.
- Default value: `false`
- `compatible-kill-query` takes effect only when `enable-global-kill` is set to `false`.
- When `enable-global-kill` is `false`, `compatible-kill-query` controls whether you need to append the `TIDB` keyword when killing a query.
    - When `compatible-kill-query` is `false`, the behavior of `KILL xxx` in TiDB is different from that in MySQL. To kill a query in TiDB, you need to append the `TIDB` keyword, such as `KILL TIDB xxx`.
    - When `compatible-kill-query` is `true`, to kill a query in TiDB, there is no need to append the `TIDB` keyword. It is **STRONGLY NOT RECOMMENDED** to set `compatible-kill-query` to `true` in your configuration file UNLESS you are certain that clients will be always connected to the same TiDB instance. This is because pressing Control+C in the default MySQL client opens a new connection in which `KILL` is executed. If there is a proxy between the client and the TiDB cluster, the new connection might be routed to a different TiDB instance, which possibly kills a different session by mistake.
- When `enable-global-kill` is `true`, `KILL xxx` and `KILL TIDB xxx` have the same effect, but using Control+C to kill a query is not supported.
- For more information about the `KILL` statement, see KILL [TIDB].

14.5.1.0.8 `check-mb4-value-in-utf8`

- Determines whether to enable the `utf8mb4` character check. When this feature is enabled, if the character set is `utf8` and the `mb4` characters are inserted in `utf8`, an error is returned.
- Default value: `false`

14.5.1.0.9 `treat-old-version-utf8-as-utf8mb4`

- Determines whether to treat the `utf8` character set in old tables as `utf8mb4`.
- Default value: `true`

14.5.1.0.10 `alter-primary-key` (Deprecated)

- Determines whether to add or remove the primary key constraint to or from a column.
- Default value: `false`
- With this default setting, adding or removing the primary key constraint is not supported. You can enable this feature by setting `alter-primary-key` to `true`. However, if a table already exists before the switch is on, and the data type of its primary key column is an integer, dropping the primary key from the column is not possible even if you set this configuration item to `true`.

> **Note:**
>
> This configuration item has been deprecated, and currently takes effect only when the value of `@tidb_enable_clustered_index` is `INT_ONLY`. If you need to add or remove the primary key, use the `NONCLUSTERED` keyword instead when creating the table. For more details about the primary key of the `CLUSTERED` type, refer to clustered index.

14.5.1.0.11 `server-version`

- Modifies the version string returned by TiDB in the following situations:
  - When the built-in `VERSION()` function is used.
  - When TiDB establishes the initial connection to the client and returns the initial handshake packet with version string of the server. For details, see MySQL Initial Handshake Packet.
- Default value: ""
- By default, the format of the TiDB version string is `5.7.${mysql_latest_minor_version}-TiDB-${tidb_version}`.

14.5.1.0.12 `repair-mode`

- Determines whether to enable the untrusted repair mode. When the `repair-mode` is set to `true`, bad tables in the `repair-table-list` cannot be loaded.
- Default value: `false`
- The `repair` syntax is not supported by default. This means that all tables are loaded when TiDB is started.

14.5.1.0.13 `repair-table-list`

- `repair-table-list` is only valid when repair-mode is set to `true`. `repair-table-list` is a list of bad tables that need to be repaired in an instance. An example of the list is: [“db.table1”,“db.table2”…].
- Default value: []
- The list is empty by default. This means that there are no bad tables that need to be repaired.

14.5.1.0.14 `new_collations_enabled_on_first_bootstrap`

- Enables or disables the new collation support.
- Default value: `true`
- Note: This configuration takes effect only for the TiDB cluster that is first initialized. After the initialization, you cannot use this configuration item to enable or disable the new collation support.

14.5.1.0.15 `max-server-connections`

- The maximum number of concurrent client connections allowed in TiDB. It is used to control resources.
- Default value: `0`
- By default, TiDB does not set limit on the number of concurrent client connections. When the value of this configuration item is greater than `0` and the number of actual client connections reaches this value, the TiDB server rejects new client connections.

14.5.1.0.16 `max-index-length`

- Sets the maximum allowable length of the newly created index.
- Default value: `3072`
- Unit: byte
- Currently, the valid value range is `[3072, 3072*4]`. MySQL and TiDB (version < v3.0.11) do not have this configuration item, but both limit the length of the newly created index. This limit in MySQL is `3072`. In TiDB (version =< 3.0.7), this limit is `3072*4`. In TiDB (3.0.7 < version < 3.0.11), this limit is `3072`. This configuration is added to be compatible with MySQL and earlier versions of TiDB.

14.5.1.0.17 `table-column-count-limit` New in v5.0

- Sets the limit on the number of columns in a single table.
- Default value: `1017`
- Currently, the valid value range is `[1017, 4096]`.

14.5.1.0.18 `index-limit` New in v5.0

- Sets the limit on the number of indexes in a single table.
- Default value: `64`
- Currently, the valid value range is `[64, 512]`.

14.5.1.0.19 `enable-telemetry` New in v4.0.2

- Enables or disables the telemetry collection in TiDB.
- Default value: `true` for v6.1.0 ~ v6.1.4; `false` for v6.1.5 and later v6.1.x versions
- When this configuration is set to `true` on a TiDB instance and the `tidb_enable_telemetry` ↪ system variable is set to `ON`, the telemetry collection in this TiDB instance is enabled.
- When this configuration is set to `false` on all TiDB instances, the telemetry collection in TiDB is disabled and the `tidb_enable_telemetry` system variable does not take effect. See Telemetry for details.

14.5.1.0.20 `enable-tcp4-only` New in v5.0

- Enables or disables listening on TCP4 only.
- Default value: `false`
- Enabling this option is useful when TiDB is used with LVS for load balancing because the real client IP from the TCP header can be correctly parsed by the "tcp4" protocol.

14.5.1.0.21 `enable-enum-length-limit` New in v5.0

- Determines whether to limit the maximum length of a single `ENUM` element and a single `SET` element.
- Default value: `true`
- When this configuration value is `true`, the maximum length of a single `ENUM` element and a single `SET` element is 255 characters, which is compatible with MySQL 8.0. When this configuration value is `false`, there is no limit on the length of a single element, which is compatible with TiDB (earlier than v5.0).

14.5.1.0.22 `graceful-wait-before-shutdown` New in v5.0

- Specifies the number of seconds that TiDB waits when you shut down the server, which allows the clients to disconnect.
- Default value: `0`
- When TiDB is waiting for shutdown (in the grace period), the HTTP status will indicate a failure, which allows the load balancers to reroute traffic.

14.5.1.0.23 `enable-global-kill` New in v6.1.0

- Controls whether to enable the Global Kill (terminating queries or connections across instances) feature.
- Default value: `true`
- When the value is `true`, both `KILL` and `KILL TIDB` statements can terminate queries or connections across instances so you do not need to worry about erroneously terminating queries or connections. When you use a client to connect to any TiDB instance and execute the `KILL` or `KILL TIDB` statement, the statement will be forwarded to the target TiDB instance. If there is a proxy between the client and the TiDB cluster, the `KILL` and `KILL TIDB` statements will also be forwarded to the target TiDB instance for execution. Currently, using the MySQL command line ctrl+c to terminate a query or connection in TiDB is not supported when `enable-global-kill` is `true`. For more information on the `KILL` statement, see KILL.

14.5.1.0.24 `enable-forwarding` New in v5.0.0

- Controls whether the PD client and TiKV client in TiDB forward requests to the leader via the followers in the case of possible network isolation.
- Default value: `false`
- If the environment might have isolated network, enabling this parameter can reduce the window of service unavailability.
- If you cannot accurately determine whether isolation, network interruption, or downtime has occurred, using this mechanism has the risk of misjudgment and causes reduced availability and performance. If network failure has never occurred, it is not recommended to enable this parameter.

14.5.1.0.25 `enable-table-lock` New in v4.0.0

> **Warning:**
>
> The table lock is an experimental feature. It is not recommended that you use it in the production environment.

- Controls whether to enable the table lock feature.
- Default value: `false`
- The table lock is used to coordinate concurrent access to the same table among multiple sessions. Currently, the `READ`, `WRITE`, and `WRITE LOCAL` lock types are supported. When the configuration item is set to `false`, executing the `LOCK TABLES` or `UNLOCK TABLES` statement does not take effect and returns the "LOCK/UNLOCK TABLES is not supported" warning. For more information, see `LOCK TABLES` and `UNLOCK TABLES`.

14.5.1.1 Log

Configuration items related to log.

14.5.1.1.1 `level`

- Specifies the log output level.
- Value options: `debug`, `info`, `warn`, `error`, and `fatal`.
- Default value: `info`

14.5.1.1.2 `format`

- Specifies the log output format.
- Value options: `json` and `text`.
- Default value: `text`

14.5.1.1.3 `enable-timestamp`

- Determines whether to enable timestamp output in the log.
- Default value: `null`
- If you set the value to `false`, the log does not output timestamp.

Note:

- To be backward compatible, the initial `disable-timestamp` configuration item remains valid. But if the value of `disable-timestamp` semantically conflicts with the value of `enable-timestamp` (for example, if both `enable-timestamp` and `disable-timestamp` are set to `true`), TiDB ignores the value for `disable-timestamp`.
- Currently, TiDB use `disable-timestamp` to determine whether to output timestamps in the log. In this situation, the value of `enable-timestamp` is `null`.

- In later versions, the `disable-timestamp` configuration will be removed. Discard `disable-timestamp` and use `enable-timestamp` which is semantically easier to understand.

14.5.1.1.4 `enable-slow-log`

- Determines whether to enable the slow query log.
- Default value: `true`
- To enable the slow query log, set `enable-slow-log` to `true`. Otherwise, set it to `false`.

14.5.1.1.5 `slow-query-file`

- The file name of the slow query log.
- Default value: `tidb-slow.log`
- The format of the slow log is updated in TiDB v2.1.8, so the slow log is output to the slow log file separately. In versions before v2.1.8, this variable is set to "" by default.
- After you set it, the slow query log is output to this file separately.

14.5.1.1.6 `slow-threshold`

- Outputs the threshold value of consumed time in the slow log.
- Default value: `300`
- Unit: Milliseconds
- When the time consumed by a query is larger than this value, this query is considered as a slow query and its log is output to the slow query log. Note that when the output level of `log.level` is `"debug"`, all queries are recorded in the slow query log, regardless of the setting of this parameter.
- Since v6.1.0, the threshold value of consumed time in the slow log is specified by the TiDB configuration item `instance.tidb_slow_log_threshold` or the system variable `tidb_slow_log_threshold`. `slow-threshold` still takes effect. But if `slow-threshold` and `instance.tidb_slow_log_threshold` are set at the same time, the latter takes effect.

14.5.1.1.7 `record-plan-in-slow-log`

- Determines whether to record execution plans in the slow log.
- Default value: `1`
- `0` means to disable, and `1` (by default) means to enable. The value of this parameter is the initial value of the `tidb_record_plan_in_slow_log` system variable.

14.5.1.1.8 `expensive-threshold`

> **Warning:**
>
> Starting from v5.4.0, the `expensive-threshold` configuration item is deprecated and replaced by the system variable `tidb_expensive_query_time_threshold`.

- Outputs the threshold value of the number of rows for the `expensive` operation.
- Default value: `10000`
- When the number of query rows (including the intermediate results based on statistics) is larger than this value, it is an `expensive` operation and outputs log with the `[ ↪ EXPENSIVE_QUERY]` prefix.

14.5.1.2 log.file

Configuration items related to log files.

`filename`

- The file name of the general log file.
- Default value: ""
- If you set it, the log is output to this file.

`max-size`

- The size limit of the log file.
- Default value: 300
- Unit: MB
- The maximum value is 4096.

`max-days`

- The maximum number of days that the log is retained.
- Default value: `0`
- The log is retained by default. If you set the value, the expired log is cleaned up after `max-days`.

`max-backups`

- The maximum number of retained logs.
- Default value: `0`
- All the log files are retained by default. If you set it to `7`, seven log files are retained at maximum.

14.5.1.3 Security

Configuration items related to security.

14.5.1.3.1 `enable-sem`

- Enables the Security Enhanced Mode (SEM).
- Default value: `false`
- The status of SEM is available via the system variable `tidb_enable_enhanced_security` ↪ .

14.5.1.3.2 `ssl-ca`

- The file path of the trusted CA certificate in the PEM format.
- Default value: “”
- If you set this option and `--ssl-cert`, `--ssl-key` at the same time, TiDB authenticates the client certificate based on the list of trusted CAs specified by this option when the client presents the certificate. If the authentication fails, the connection is terminated.
- If you set this option but the client does not present the certificate, the secure connection continues without client certificate authentication.

14.5.1.3.3 `ssl-cert`

- The file path of the SSL certificate in the PEM format.
- Default value: “”
- If you set this option and `--ssl-key` at the same time, TiDB allows (but not forces) the client to securely connect to TiDB using TLS.
- If the specified certificate or private key is invalid, TiDB starts as usual but cannot receive secure connection.

14.5.1.3.4 `ssl-key`

- The file path of the SSL certificate key in the PEM format, that is, the private key of the certificate specified by `--ssl-cert`.
- Default value: “”
- Currently, TiDB does not support loading the private keys protected by passwords.

14.5.1.3.5 `cluster-ssl-ca`

- The CA root certificate used to connect TiKV or PD with TLS.
- Default value: “”

14.5.1.3.6 `cluster-ssl-cert`

- The path of the SSL certificate file used to connect TiKV or PD with TLS.
- Default value: “”

14.5.1.3.7 `cluster-ssl-key`

- The path of the SSL private key file used to connect TiKV or PD with TLS.
- Default value: “”

14.5.1.3.8 `cluster-verify-cn`

- A list of acceptable X.509 Common Names in certificates presented by clients. Requests are permitted only when the presented Common Name is an exact match with one of the entries in the list.
- Default value: [], which means that the client certificate CN check is disabled.

14.5.1.3.9 `spilled-file-encryption-method`

- Determines the encryption method used for saving the spilled files to disk.
- Default value: `"plaintext"`, which disables encryption.
- Optional values: `"plaintext"` and `"aes128-ctr"`

14.5.1.3.10 `auto-tls`

- Determines whether to automatically generate the TLS certificates on startup.
- Default value: `false`

14.5.1.3.11 `tls-version`

- Set the minimum TLS version for MySQL Protocol connections.
- Default value: “”, which allows TLSv1.1 or higher.
- Optional values: `"TLSv1.0"`, `"TLSv1.1"`, `"TLSv1.2"` and `"TLSv1.3"`

14.5.1.4 Performance

Configuration items related to performance.

14.5.1.4.1 `max-procs`

- The number of CPUs used by TiDB.
- Default value: `0`
- The default `0` indicates using all the CPUs on the machine. You can also set it to n, and then TiDB uses n CPUs.

14.5.1.4.2 `server-memory-quota` New in v4.0.9

> **Warning:**
>
> `server-memory-quota` is still an experimental feature. It is **NOT** recommended that you use it in a production environment.

- The memory usage limit of tidb-server instances.
- Default value: `0` (in bytes), which means no memory limit.

14.5.1.4.3 `memory-usage-alarm-ratio` New in v4.0.9

- TiDB triggers an alarm when the memory usage of tidb-server instance exceeds a certain threshold. The valid value for this configuration item ranges from `0` to `1`. If it is configured as `0` or `1`, this alarm feature is disabled.
- Default value: `0.8`
- When the memory usage alarm is enabled, if `server-memory-quota` is not set, then the threshold of memory usage is `the `memory-usage-alarm-ratio` value * the system memory size`; if `server-memory-quota` is set to a value greater than 0, then the threshold of memory usage is `the `memory-usage-alarm-ratio` value * the `server-memory-quota` value`.
- When TiDB detects that the memory usage of the tidb-server instance exceeds the threshold, it considers that there might be a risk of OOM. Therefore, it records ten SQL statements with the highest memory usage, ten SQL statements with the longest running time, and the heap profile among all SQL statements currently being executed to the directory `tmp-storage-path/record` and outputs a log containing the keyword `tidb-server has the risk of OOM`.
- The value of this configuration item is the initial value of the system variable `tidb_memory_usage_alarm_ratio`.

14.5.1.4.4 `max-txn-ttl`

- The longest time that a single transaction can hold locks. If this time is exceeded, the locks of a transaction might be cleared by other transactions so that this transaction cannot be successfully committed.
- Default value: `3600000`
- Unit: Millisecond
- The transaction that holds locks longer than this time can only be committed or rolled back. The commit might not be successful.

14.5.1.4.5 `stmt-count-limit`

- The maximum number of statements allowed in a single TiDB transaction.
- Default value: `5000`
- If a transaction does not roll back or commit after the number of statements exceeds `stmt-count-limit`, TiDB returns the `statement count 5001 exceeds the transaction limitation, autocommit = false` error. This configuration takes effect **only** in the retryable optimistic transaction. If you use the pessimistic transaction or have disabled the transaction retry, the number of statements in a transaction is not limited by this configuration.

14.5.1.4.6 `txn-entry-size-limit` New in v4.0.10 and v5.0.0

- The size limit of a single row of data in TiDB.
- Default value: `6291456` (in bytes)
- The size limit of a single key-value record in a transaction. If the size limit is exceeded, TiDB returns the `entry too large` error. The maximum value of this configuration item does not exceed `125829120` (120 MB).
- Note that TiKV has a similar limit. If the data size of a single write request exceeds `raft-entry-max-size`, which is 8 MB by default, TiKV refuses to process this request. When a table has a row of large size, you need to modify both configurations at the same time.
- The default value of `max_allowed_packet` (the maximum size of a packet for the MySQL protocol) is 67108864 (64 MiB). If a row is larger than `max_allowed_packet`, the row gets truncated.
- The default value of `txn-total-size-limit` (the size limit of a single transaction in TiDB) is 100 MiB. If you increase the `txn-entry-size-limit` value to be over 100 MiB, you need to increase the `txn-total-size-limit` value accordingly.

14.5.1.4.7 `txn-total-size-limit`

- The size limit of a single transaction in TiDB.
- Default value: `104857600` (in bytes)
- In a single transaction, the total size of key-value records cannot exceed this value. The maximum value of this parameter is `1099511627776` (1 TB). Note that if you have used the binlog to serve the downstream consumer Kafka (such as the `arbiter` cluster), the value of this parameter must be no more than `1073741824` (1 GB). This is because 1 GB is the upper limit of a single message size that Kafka can process. Otherwise, an error is returned if this limit is exceeded.

14.5.1.4.8 `tcp-keep-alive`

- Determines whether to enable `keepalive` in the TCP layer.
- Default value: `true`

14.5.1.4.9 `tcp-no-delay`

- Determines whether to enable TCP_NODELAY at the TCP layer. After it is enabled, TiDB disables the Nagle algorithm in the TCP/IP protocol and allows sending small data packets to reduce network latency. This is suitable for latency-sensitive applications with a small transmission volume of data.
- Default value: `true`

14.5.1.4.10 `cross-join`

- Default value: `true`
- TiDB supports executing the `JOIN` statement without any condition (the `WHERE` field) of both sides tables by default; if you set the value to `false`, the server refuses to execute when such a `JOIN` statement appears.

14.5.1.4.11 `stats-lease`

- The time interval of reloading statistics, updating the number of table rows, checking whether it is needed to perform the automatic analysis, using feedback to update statistics and loading statistics of columns.
- Default value: `3s`
    - At intervals of `stats-lease` time, TiDB checks the statistics for updates and updates them to the memory if updates exist.
    - At intervals of `20 * stats-lease` time, TiDB updates the total number of rows generated by DML and the number of modified rows to the system table.
    - At intervals of `stats-lease`, TiDB checks for tables and indexes that need to be automatically analyzed.
    - At intervals of `stats-lease`, TiDB checks for column statistics that need to be loaded to the memory.
    - At intervals of `200 * stats-lease`, TiDB writes the feedback cached in the memory to the system table.
    - At intervals of `5 * stats-lease`, TiDB reads the feedback in the system table, and updates the statistics cached in the memory.
- When `stats-lease` is set to 0s, TiDB periodically reads the feedback in the system table, and updates the statistics cached in the memory every three seconds. But TiDB no longer automatically modifies the following statistics-related system tables:
    - `mysql.stats_meta`: TiDB no longer automatically records the number of table rows that are modified by the transaction and updates it to this system table.
    - `mysql.stats_histograms`/`mysql.stats_buckets` and `mysql.stats_top_n`: TiDB no longer automatically analyzes and proactively updates statistics.
    - `mysql.stats_feedback`: TiDB no longer updates the statistics of the tables and indexes according to a part of statistics returned by the queried data.

14.5.1.4.12 `feedback-probability`

> **Warning:**
>
> This feature has been deprecated since v5.4. It is not recommended to enable this feature.

- The probability that TiDB collects the feedback statistics of each query.
- Default value: `0`
- This feature is disabled by default, and it is not recommended to enable this feature. If it is enabled, TiDB collects the feedback of each query at the probability of `feedback -probability`, to update statistics.

14.5.1.4.13 `query-feedback-limit`

- The maximum pieces of query feedback that can be cached in memory. Extra pieces of feedback that exceed this limit are discarded.
- Default value: `1024`

14.5.1.4.14 `pseudo-estimate-ratio`

- The ratio of (number of modified rows)/(total number of rows) in a table. If the value is exceeded, the system assumes that the statistics have expired and the pseudo statistics will be used.
- Default value: `0.8`
- The minimum value is `0` and the maximum value is `1`.

14.5.1.4.15 `force-priority`

- Sets the priority for all statements.
- Default value: `NO_PRIORITY`
- Value options: The default value `NO_PRIORITY` means that the priority for statements is not forced to change. Other options are `LOW_PRIORITY`, `DELAYED`, and `HIGH_PRIORITY` in ascending order.

14.5.1.4.16 `distinct-agg-push-down`

- Determines whether the optimizer executes the operation that pushes down the aggregation function with `Distinct` (such as `select count(distinct a)from t`) to Coprocessors.
- Default: `false`
- This variable is the initial value of the system variable `tidb_opt_distinct_agg_push_down`.

14.5.1.4.17 `enforce-mpp`

- Determines whether to ignore the optimizer's cost estimation and to forcibly use TiFlash's MPP mode for query execution.
- Default value: `false`
- This configuration item controls the initial value of `tidb_enforce_mpp`. For example, when this configuration item is set to `true`, the default value of `tidb_enforce_mpp` is `ON`.

14.5.1.4.18 `enable-stats-cache-mem-quota` New in v6.1.0

> **Warning:**
>
> This variable is an experimental feature. It is not recommended to use it in production environments.

- Controls whether to enable the memory quota for the statistics cache.
- Default value: `false`

14.5.1.4.19 `stats-load-concurrency` New in v5.4.0

> **Warning:**
>
> Currently, synchronously loading statistics is an experimental feature. It is not recommended that you use it in production environments.

- The maximum number of columns that the TiDB synchronously loading statistics feature can process concurrently.
- Default value: `5`
- Currently, the valid value range is `[1, 128]`.

14.5.1.4.20 `stats-load-queue-size` New in v5.4.0

> **Warning:**
>
> Currently, synchronously loading statistics is an experimental feature. It is not recommended that you use it in production environments.

- The maximum number of column requests that the TiDB synchronously loading statistics feature can cache.
- Default value: `1000`
- Currently, the valid value range is `[1, 100000]`.

14.5.1.5 opentracing

Configuration items related to opentracing.

14.5.1.5.1 `enable`

- Enables opentracing to trace the call overhead of some TiDB components. Note that enabling opentracing causes some performance loss.
- Default value: `false`

14.5.1.5.2 `rpc-metrics`

- Enables RPC metrics.
- Default value: `false`

14.5.1.6 opentracing.sampler

Configuration items related to opentracing.sampler.

14.5.1.6.1 `type`

- Specifies the type of the opentracing sampler. The string value is case-insensitive.
- Default value: `"const"`
- Value options: `"const"`, `"probabilistic"`, `"ratelimiting"`, `"remote"`

14.5.1.6.2 `param`

- The parameter of the opentracing sampler.
    - For the `const` type, the value can be `0` or `1`, which indicates whether to enable the `const` sampler.
    - For the `probabilistic` type, the parameter specifies the sampling probability, which can be a float number between `0` and `1`.
    - For the `ratelimiting` type, the parameter specifies the number of spans sampled per second.
    - For the `remote` type, the parameter specifies the sampling probability, which can be a float number between `0` and `1`.
- Default value: `1.0`

14.5.1.6.3 `sampling-server-url`

- The HTTP URL of the jaeger-agent sampling server.
- Default value: `""`

14.5.1.6.4 `max-operations`

- The maximum number of operations that the sampler can trace. If an operation is not traced, the default probabilistic sampler is used.
- Default value: `0`

14.5.1.6.5 `sampling-refresh-interval`

- Controls the frequency of polling the jaeger-agent sampling policy.
- Default value: `0`

14.5.1.7 opentracing.reporter

Configuration items related to opentracing.reporter.

14.5.1.7.1 `queue-size`

- The queue size with which the reporter records spans in memory.
- Default value: `0`

14.5.1.7.2 `buffer-flush-interval`

- The interval at which the reporter flushes the spans in memory to the storage.
- Default value: `0`

14.5.1.7.3 `log-spans`

- Determines whether to print the log for all submitted spans.
- Default value: `false`

14.5.1.7.4 `local-agent-host-port`

- The address at which the reporter sends spans to the jaeger-agent.
- Default value: `""`

14.5.1.8 tikv-client

14.5.1.8.1 `grpc-connection-count`

- The maximum number of connections established with each TiKV.
- Default value: `4`

14.5.1.8.2 `grpc-keepalive-time`

- The `keepalive` time interval of the RPC connection between TiDB and TiKV nodes. If there is no network packet within the specified time interval, the gRPC client executes `ping` command to TiKV to see if it is alive.
- Default: `10`
- Unit: second

14.5.1.8.3 `grpc-keepalive-timeout`

- The timeout of the RPC `keepalive` check between TiDB and TiKV nodes.
- Default value: `3`
- Unit: second

14.5.1.8.4 `grpc-compression-type`

- Specifies the compression type used for data transfer between TiDB and TiKV nodes. The default value is `"none"`, which means no compression. To enable the gzip compression, set this value to `"gzip"`.
- Default value: `"none"`
- Value options: `"none"`, `"gzip"`

14.5.1.8.5 `commit-timeout`

- The maximum timeout when executing a transaction commit.
- Default value: `41s`
- It is required to set this value larger than twice of the Raft election timeout.

14.5.1.8.6 `max-batch-size`

- The maximum number of RPC packets sent in batch. If the value is not `0`, the `BatchCommands` API is used to send requests to TiKV, and the RPC latency can be reduced in the case of high concurrency. It is recommended that you do not modify this value.
- Default value: `128`

14.5.1.8.7 `max-batch-wait-time`

- Waits for `max-batch-wait-time` to encapsulate the data packets into a large packet in batch and send it to the TiKV node. It is valid only when the value of `tikv-client.max-batch-size` is greater than `0`. It is recommended not to modify this value.
- Default value: `0`
- Unit: nanoseconds

14.5.1.8.8 `batch-wait-size`

- The maximum number of packets sent to TiKV in batch. It is recommended not to modify this value.
- Default value: `8`
- If the value is `0`, this feature is disabled.

14.5.1.8.9 `overload-threshold`

- The threshold of the TiKV load. If the TiKV load exceeds this threshold, more `batch` packets are collected to relieve the pressure of TiKV. It is valid only when the value of `tikv-client.max-batch-size` is greater than `0`. It is recommended not to modify this value.
- Default value: `200`

14.5.1.9 tikv-client.copr-cache New in v4.0.0

This section introduces configuration items related to the Coprocessor Cache feature.

14.5.1.9.1 `capacity-mb`

- The total size of the cached data. When the cache space is full, old cache entries are evicted. When the value is `0.0`, the Coprocessor Cache feature is disabled.
- Default value: `1000.0`
- Unit: MB
- Type: Float

14.5.1.10 txn-local-latches

Configuration related to the transaction latch. It is recommended to enable it when many local transaction conflicts occur.

14.5.1.10.1 `enabled`

- Determines whether to enable the memory lock of transactions.
- Default value: `false`

14.5.1.10.2 `capacity`

- The number of slots corresponding to Hash, which automatically adjusts upward to an exponential multiple of 2. Each slot occupies 32 Bytes of memory. If set too small, it might result in slower running speed and poor performance in the scenario where data writing covers a relatively large range (such as importing data).
- Default value: `2048000`

14.5.1.11 binlog

Configurations related to TiDB Binlog.

14.5.1.11.1 `enable`

- Enables or disables binlog.
- Default value: `false`

14.5.1.11.2 `write-timeout`

- The timeout of writing binlog into Pump. It is not recommended to modify this value.
- Default: `15s`
- unit: second

14.5.1.11.3 `ignore-error`

- Determines whether to ignore errors occurred in the process of writing binlog into Pump. It is not recommended to modify this value.
- Default value: `false`
- When the value is set to `true` and an error occurs, TiDB stops writing binlog and add `1` to the count of the `tidb_server_critical_error_total` monitoring item. When the value is set to `false`, the binlog writing fails and the entire TiDB service is stopped.

14.5.1.11.4 `binlog-socket`

- The network address to which binlog is exported.
- Default value: “”

14.5.1.11.5 `strategy`

- The strategy of Pump selection when binlog is exported. Currently, only the `hash` and `range` methods are supported.
- Default value: `range`

14.5.1.12 status

Configuration related to the status of TiDB service.

14.5.1.12.1 `report-status`

- Enables or disables the HTTP API service.
- Default value: `true`

14.5.1.12.2 `record-db-qps`

- Determines whether to transmit the database-related QPS metrics to Prometheus.
- Default value: `false`

14.5.1.13 pessimistic-txn

For pessimistic transaction usage, refer to TiDB Pessimistic Transaction Mode.

14.5.1.13.1 max-retry-count

- The maximum number of retries of each statement in pessimistic transactions. If the number of retries exceeds this limit, an error occurs.
- Default value: `256`

14.5.1.13.2 deadlock-history-capacity

- The maximum number of deadlock events that can be recorded in the `INFORMATION_SCHEMA.DEADLOCKS` table of a single TiDB server. If this table is in full volume and an additional deadlock event occurs, the earliest record in the table will be removed to make place for the newest error.
- Default value: `10`
- Minimum value: `0`
- Maximum value: `10000`

14.5.1.13.3 deadlock-history-collect-retryable

- Controls whether the `INFORMATION_SCHEMA.DEADLOCKS` table collects the information of retryable deadlock errors. For the description of retryable deadlock errors, see Retryable deadlock errors.
- Default value: `false`

14.5.1.13.4 pessimistic-auto-commit New in v6.0.0

- Determines the transaction mode that the auto-commit transaction uses when the pessimistic transaction mode is globally enabled (`tidb_txn_mode='pessimistic'`). By default, even if the pessimistic transaction mode is globally enabled, the auto-commit transaction still uses the optimistic transaction mode. After enabling `pessimistic-auto-commit` (set to `true`), the auto-commit transaction also uses pessimistic mode, which is consistent with the other explicitly committed pessimistic transactions.
- For scenarios with conflicts, after enabling this configuration, TiDB includes auto-commit transactions into the global lock-waiting management, which avoids deadlocks and mitigates the latency spike brought by deadlock-causing conflicts.
- For scenarios with no conflicts, if there are many auto-commit transactions (the specific number is determined by the real scenarios. For example, the number of auto-commit transactions accounts for more than half of the total number of applications), and a single transaction operates a large data volume, enabling this configuration causes performance regression. For example, the auto-commit `INSERT INTO SELECT` statement.
- Default value: `false`

14.5.1.14 isolation-read

Configuration items related to read isolation.

14.5.1.14.1 engines

- Controls from which engine TiDB allows to read data.
- Default value: ["tikv", "tiflash", "tidb"], indicating that the engine is automatically selected by the optimizer.
- Value options: Any combinations of "tikv", "tiflash", and "tidb", for example, ["tikv", "tidb"] or ["tiflash", "tidb"]

14.5.1.15 instance

14.5.1.15.1 tidb_enable_collect_execution_info

- This configuration controls whether to record the execution information of each operator in the slow query log.
- Default value: `true`
- Before v6.1.0, this configuration is set by `enable-collect-execution-info`.

14.5.1.15.2 tidb_enable_slow_log

- This configuration is used to control whether to enable the slow log feature.
- Default value: `true`
- Value options: `true` or `false`
- Before v6.1.0, this configuration is set by `enable-slow-log`.

14.5.1.15.3 `tidb_slow_log_threshold`

- Outputs the threshold value of the time consumed by the slow log.
- Default value: `300`
- Range: `[-1, 9223372036854775807]`
- Unit: Milliseconds
- When the time consumed by a query is larger than this value, this query is considered as a slow query and its log is output to the slow query log. Note that when the output level of `log.level` is `"debug"`, all queries are recorded in the slow query log, regardless of the setting of this parameter.
- Before v6.1.0, this configuration is set by `slow-threshold`.

14.5.1.15.4 `tidb_expensive_query_time_threshold`

- This configuration is used to set the threshold value that determines whether to print expensive query logs. The difference between expensive query logs and slow query logs is:
  - Slow logs are printed after the statement is executed.
  - Expensive query logs print the statements that are being executed, with execution time exceeding the threshold value, and their related information.
- Default value: `60`
- Range: `[10, 2147483647]`
- Unit: Seconds
- Before v5.4.0, this configuration is set by `expensive-threshold`.

14.5.1.15.5 `tidb_record_plan_in_slow_log`

- This configuration is used to control whether to include the execution plan of slow queries in the slow log.
- Default value: `1`
- Value options: `1` (enabled, default) or `0` (disabled).
- The value of this configuration will initialize the value of system variable `tidb_record_plan_in_slow`
  ↪
- Before v6.1.0, this configuration is set by `record-plan-in-slow-log`.

14.5.1.15.6 `tidb_force_priority`

- This configuration is used to change the default priority for statements executed on a TiDB server.
- Default value: `NO_PRIORITY`
- The default value `NO_PRIORITY` means that the priority for statements is not forced to change. Other options are `LOW_PRIORITY`, `DELAYED`, and `HIGH_PRIORITY` in ascending order.
- Before v6.1.0, this configuration is set by `force-priority`.

14.5.1.16 proxy-protocol

Configuration items related to the PROXY protocol.

14.5.1.16.1 `networks`

- The list of proxy server's IP addresses allowed to connect to TiDB using the PROXY protocol
- Default value: ""
- In general cases, when you access TiDB behind a reverse proxy, TiDB takes the IP address of the reverse proxy server as the IP address of the client. By enabling the PROXY protocol, reverse proxies that support this protocol, such as HAProxy, can pass the real client IP address to TiDB.
- After configuring this parameter, TiDB allows the configured source IP address to connect to TiDB using the PROXY protocol; if a protocol other than PROXY is used, this connection will be denied. If this parameter is left empty, no IP address can connect to TiDB using the PROXY protocol. The value can be an IP address (192.168.1.50) or CIDR (192.168.1.0/24) with `,` as the separator. `*` means any IP addresses.

> **Warning:**
>
> Use `*` with caution because it might introduce security risks by allowing a client of any IP address to report its IP address. In addition, using `*` might also cause the internal component that directly connects to TiDB (such as TiDB Dashboard) to be unavailable.

14.5.1.17 experimental

The `experimental` section, introduced in v3.1.0, describes the configurations related to the experimental features of TiDB.

14.5.1.17.1 `allow-expression-index` New in v4.0.0

- Controls whether an expression index can be created. Since TiDB v5.2.0, if the function in an expression is safe, you can create an expression index directly based on this function without enabling this configuration. If you want to create an expression index based on other functions, you can enable this configuration, but correctness issues might exist. By querying the `tidb_allow_function_for_expression_index` variable, you can get the functions that are safe to be directly used for creating an expression.
- Default value: `false`

14.5.2 TiKV Configuration File

The TiKV configuration file supports more options than command-line parameters. You can find the default configuration file in etc/config-template.toml and rename it to `config.toml`.

This document only describes the parameters that are not included in command-line parameters. For more details, see command-line parameter.

> **Tip:**
>
> If you need to adjust the value of a configuration item, refer to Modify the configuration.

14.5.2.1 Global configuration

14.5.2.1.1 `abort-on-panic`

- Sets whether to call `abort()` to exit the process when TiKV panics. This option affects whether TiKV allows the system to generate core dump files.
  - If the value of this configuration item is `false`, when TiKV panics, it calls `exit()` to exit the process.
  - If the value of this configuration item is `true`, when TiKV panics, TiKV calls `abort()` to exit the process. At this time, TiKV allows the system to generate core dump files when exiting. To generate the core dump file, you also need to perform the system configuration related to core dump (for example, setting the size limit of the core dump file via `ulimit -c` command, and configure the core dump path. Different operating systems have different related configurations). To avoid the core dump files occupying too much disk space and causing insufficient TiKV disk space, it is recommended to set the core dump generation path to a disk partition different to that of TiKV data.
- Default value: `false`

14.5.2.1.2 `slow-log-file`

- The file that stores slow logs
- If this configuration item is not set, but `log.file.filename` is set, slow logs are output to the log file specified by `log.file.filename`.
- If neither `slow-log-file` nor `log.file.filename` are set, all logs are output to "stderr" by default.

- If both configuration items are set, ordinary logs are output to the log file specified by `log.file.filename`, and slow logs are output to the log file set by `slow-log-file`.
- Default value: `""`

14.5.2.1.3 `slow-log-threshold`

- The threshold for outputing slow logs. If the processing time is longer than this threshold, slow logs are output.
- Default value: `"1s"`

14.5.2.2 log New in v5.4.0

- Configuration items related to the log.
- From v5.4.0, to make the log configuration items of TiKV and TiDB consistent, TiKV deprecates the former configuration item `log-rotation-timespan` and changes `log-level`, `log-format`, `log-file`, `log-rotation-size` to the following ones. If you only set the old configuration items, and their values are set to non-default values, the old items remain compatible with the new items. If both old and new configuration items are set, the new items take effect.

14.5.2.2.1 `level` New in v5.4.0

- The log level
- Optional values: `"debug"`, `"info"`, `"warn"`, `"error"`, `"fatal"`
- Default value: `"info"`

14.5.2.2.2 `format` New in v5.4.0

- The log format
- Optional values: `"json"`, `"text"`
- Default value: `"text"`

14.5.2.2.3 `enable-timestamp` New in v5.4.0

- Determines whether to enable or disable the timestamp in the log
- Optional values: `true`, `false`
- Default value: `true`

14.5.2.3 log.file New in v5.4.0

- Configuration items related to the log file.

14.5.2.3.1 `filename` New in v5.4.0

- The log file. If this configuration item is not set, logs are output to "stderr" by default. If this configuration item is set, logs are output to the corresponding file.
- Default value: `""`

14.5.2.3.2 `max-size` New in v5.4.0

- The maximum size of a single log file. When the file size is larger than the value set by this configuration item, the system automatically splits the single file into multiple files.
- Default value: `300`
- Maximum value: `4096`
- Unit: MiB

14.5.2.3.3 `max-days` New in v5.4.0

- The maximum number of days that TiKV keeps log files.
    - If the configuration item is not set, or the value of it is set to the default value `0`, TiKV does not clean log files.
    - If the parameter is set to a value other than `0`, TiKV cleans up the expired log files after `max-days`.
- Default value: `0`

14.5.2.3.4 `max-backups` New in v5.4.0

- The maximum number of log files that TiKV keeps.
    - If the configuration item is not set, or the value of it is set to the default value `0`, TiKV keeps all log files.
    - If the configuration item is set to a value other than `0`, TiKV keeps at most the number of old log files specified by `max-backups`. For example, if the value is set to `7`, TiKV keeps up to 7 old log files.
- Default value: `0`

14.5.2.4 server

- Configuration items related to the server.

14.5.2.4.1 status-thread-pool-size

- The number of worker threads for the HTTP API service
- Default value: 1
- Minimum value: 1

14.5.2.4.2 grpc-compression-type

- The compression algorithm for gRPC messages
- Optional values: "none", "deflate", "gzip"
- Default value: "none"

14.5.2.4.3 grpc-concurrency

- The number of gRPC worker threads. When you modify the size of the gRPC thread pool, refer to Performance tuning for TiKV thread pools.
- Default value: 5
- Minimum value: 1

14.5.2.4.4 grpc-concurrent-stream

- The maximum number of concurrent requests allowed in a gRPC stream
- Default value: 1024
- Minimum value: 1

14.5.2.4.5 grpc-memory-pool-quota

- Limits the memory size that can be used by gRPC
- Default value: No limit
- Limit the memory in case OOM is observed. Note that limit the usage can lead to potential stall

14.5.2.4.6 grpc-raft-conn-num

- The maximum number of connections between TiKV nodes for Raft communication
- Default value: 1
- Minimum value: 1

14.5.2.4.7 max-grpc-send-msg-len

- Sets the maximum length of a gRPC message that can be sent
- Default value: 10485760
- Unit: Bytes
- Maximum value: 2147483647

14.5.2.4.8 `grpc-stream-initial-window-size`

- The window size of the gRPC stream
- Default value: `2MB`
- Unit: KB|MB|GB
- Minimum value: `"1KB"`

14.5.2.4.9 `grpc-keepalive-time`

- The time interval at which that gRPC sends `keepalive` Ping messages
- Default value: `"10s"`
- Minimum value: `"1s"`

14.5.2.4.10 `grpc-keepalive-timeout`

- Disables the timeout for gRPC streams
- Default value: `"3s"`
- Minimum value: `"1s"`

14.5.2.4.11 `concurrent-send-snap-limit`

- The maximum number of snapshots sent at the same time
- Default value: `32`
- Minimum value: `1`

14.5.2.4.12 `concurrent-recv-snap-limit`

- The maximum number of snapshots received at the same time
- Default value: `32`
- Minimum value: `1`

14.5.2.4.13 `end-point-recursion-limit`

- The maximum number of recursive levels allowed when TiKV decodes the Coprocessor DAG expression
- Default value: `1000`
- Minimum value: `1`

14.5.2.4.14 `end-point-request-max-handle-duration`

- The longest duration allowed for a TiDB's push down request to TiKV for processing tasks
- Default value: `"60s"`
- Minimum value: `"1s"`

14.5.2.4.15 `snap-max-write-bytes-per-sec`

- The maximum allowable disk bandwidth when processing snapshots
- Default value: `"100MB"`
- Unit: KB|MB|GB
- Minimum value: `"1KB"`

14.5.2.4.16 `end-point-slow-log-threshold`

- The time threshold for a TiDB's push-down request to output slow log. If the processing time is longer than this threshold, the slow logs are output.
- Default value: `"1s"`
- Minimum value: `0`

14.5.2.4.17 `raft-client-queue-size`

- Specifies the queue size of the Raft messages in TiKV. If too many messages not sent in time result in a full buffer, or messages discarded, you can specify a greater value to improve system stability.
- Default value: `8192`

14.5.2.4.18 `simplify-metrics` New in v6.1.1

- Specifies whether to simplify the returned monitoring metrics. After you set the value to `true`, TiKV reduces the amount of data returned for each request by filtering out some metrics.
- Default value: `false`

14.5.2.4.19 `forward-max-connections-per-address` New in v5.0.0

- Sets the size of the connection pool for service and forwarding requests to the server. Setting it to too small a value affects the request latency and load balancing.
- Default value: `4`

14.5.2.5 readpool.unified

Configuration items related to the single thread pool serving read requests. This thread pool supersedes the original storage thread pool and coprocessor thread pool since the 4.0 version.

14.5.2.5.1 `min-thread-count`

- The minimal working thread count of the unified read pool
- Default value: `1`

14.5.2.5.2 `max-thread-count`

- The maximum working thread count of the unified read pool or the UnifyReadPool thread pool. When you modify the size of this thread pool, refer to Performance tuning for TiKV thread pools.
- Value range: `[min-thread-count, MAX(4, CPU)]`. In `MAX(4, CPU)`, `CPU` means the number of your CPU cores. `MAX(4, CPU)` takes the greater value out of `4` and the `CPU`.
- Default value: MAX(4, CPU * 0.8)

14.5.2.5.3 `stack-size`

- The stack size of the threads in the unified thread pool
- Type: Integer + Unit
- Default value: `"10MB"`
- Unit: KB|MB|GB
- Minimum value: `"2MB"`
- Maximum value: The number of Kbytes output in the result of the `ulimit -sH` command executed in the system.

14.5.2.5.4 `max-tasks-per-worker`

- The maximum number of tasks allowed for a single thread in the unified read pool. `Server Is Busy` is returned when the value is exceeded.
- Default value: `2000`
- Minimum value: `2`

14.5.2.6 readpool.storage

Configuration items related to storage thread pool.

14.5.2.6.1 `use-unified-pool`

- Determines whether to use the unified thread pool (configured in `readpool.unified`) for storage requests. If the value of this parameter is `false`, a separate thread pool is used, which is configured through the rest parameters in this section (`readpool.storage`).
- Default value: If this section (`readpool.storage`) has no other configurations, the default value is `true`. Otherwise, for the backward compatibility, the default value is `false`. Change the configuration in `readpool.unified` as needed before enabling this option.

14.5.2.6.2 high-concurrency

- The allowable number of concurrent threads that handle high-priority `read` requests
- When `8 cpu num 16`, the default value is `cpu_num * 0.5`; when `cpu num` is smaller than `8`, the default value is `4`; when `cpu num` is greater than `16`, the default value is `8`.
- Minimum value: `1`

14.5.2.6.3 normal-concurrency

- The allowable number of concurrent threads that handle normal-priority `read` requests
- When `8 cpu num 16`, the default value is `cpu_num * 0.5`; when `cpu num` is smaller than `8`, the default value is `4`; when `cpu num` is greater than `16`, the default value is `8`.
- Minimum value: `1`

14.5.2.6.4 low-concurrency

- The allowable number of concurrent threads that handle low-priority `read` requests
- When `8 cpu num 16`, the default value is `cpu_num * 0.5`; when `cpu num` is smaller than `8`, the default value is `4`; when `cpu num` is greater than `16`, the default value is `8`.
- Minimum value: `1`

14.5.2.6.5 max-tasks-per-worker-high

- The maximum number of tasks allowed for a single thread in a high-priority thread pool. `Server Is Busy` is returned when the value is exceeded.
- Default value: `2000`
- Minimum value: `2`

14.5.2.6.6 max-tasks-per-worker-normal

- The maximum number of tasks allowed for a single thread in a normal-priority thread pool. `Server Is Busy` is returned when the value is exceeded.
- Default value: `2000`
- Minimum value: `2`

14.5.2.6.7 max-tasks-per-worker-low

- The maximum number of tasks allowed for a single thread in a low-priority thread pool. `Server Is Busy` is returned when the value is exceeded.
- Default value: `2000`
- Minimum value: `2`

14.5.2.6.8 `stack-size`

- The stack size of threads in the Storage read thread pool
- Type: Integer + Unit
- Default value: `"10MB"`
- Unit: KB|MB|GB
- Minimum value: `"2MB"`
- Maximum value: The number of Kbytes output in the result of the `ulimit -sH` command executed in the system.

14.5.2.7 `readpool.coprocessor`

Configuration items related to the Coprocessor thread pool.

14.5.2.7.1 `use-unified-pool`

- Determines whether to use the unified thread pool (configured in `readpool.unified` ↪) for coprocessor requests. If the value of this parameter is `false`, a separate thread pool is used, which is configured through the rest parameters in this section (`readpool.coprocessor`).
- Default value: If none of the parameters in this section (`readpool.coprocessor`) are set, the default value is `true`. Otherwise, the default value is `false` for the backward compatibility. Adjust the configuration items in `readpool.unified` before enabling this parameter.

14.5.2.7.2 `high-concurrency`

- The allowable number of concurrent threads that handle high-priority Coprocessor requests, such as checkpoints
- Default value: `CPU * 0.8`
- Minimum value: `1`

14.5.2.7.3 `normal-concurrency`

- The allowable number of concurrent threads that handle normal-priority Coprocessor requests
- Default value: `CPU * 0.8`
- Minimum value: `1`

14.5.2.7.4 `low-concurrency`

- The allowable number of concurrent threads that handle low-priority Coprocessor requests, such as table scan
- Default value: `CPU * 0.8`
- Minimum value: `1`

14.5.2.7.5 `max-tasks-per-worker-high`

- The number of tasks allowed for a single thread in a high-priority thread pool. When this number is exceeded, `Server Is Busy` is returned.
- Default value: `2000`
- Minimum value: `2`

14.5.2.7.6 `max-tasks-per-worker-normal`

- The number of tasks allowed for a single thread in a normal-priority thread pool. When this number is exceeded, `Server Is Busy` is returned.
- Default value: `2000`
- Minimum value: `2`

14.5.2.7.7 `max-tasks-per-worker-low`

- The number of tasks allowed for a single thread in a low-priority thread pool. When this number is exceeded, `Server Is Busy` is returned.
- Default value: `2000`
- Minimum value: `2`

14.5.2.7.8 `stack-size`

- The stack size of the thread in the Coprocessor thread pool
- Type: Integer + Unit
- Default value: `"10MB"`
- Unit: KB|MB|GB
- Minimum value: `"2MB"`
- Maximum value: The number of Kbytes output in the result of the `ulimit -sH` command executed in the system.

14.5.2.8 storage

Configuration items related to storage.

14.5.2.8.1 `scheduler-concurrency`

- A built-in memory lock mechanism to prevent simultaneous operations on a key. Each key has a hash in a different slot.
- Default value: `524288`
- Minimum value: `1`

14.5.2.8.2 `scheduler-worker-pool-size`

- The number of threads in the Scheduler thread pool. Scheduler threads are mainly used for checking transaction consistency before data writing. If the number of CPU cores is greater than or equal to `16`, the default value is `8`; otherwise, the default value is `4`. When you modify the size of the Scheduler thread pool, refer to Performance tuning for TiKV thread pools.
- Default value: `4`
- Value range: `[1, MAX(4, CPU)]`. In `MAX(4, CPU)`, `CPU` means the number of your CPU cores. `MAX(4, CPU)` takes the greater value out of `4` and the `CPU`.

14.5.2.8.3 `scheduler-pending-write-threshold`

- The maximum size of the write queue. A `Server Is Busy` error is returned for a new write to TiKV when this value is exceeded.
- Default value: `"100MB"`
- Unit: MB|GB

14.5.2.8.4 `reserve-space`

- When TiKV is started, some space is reserved on the disk as disk protection. When the remaining disk space is less than the reserved space, TiKV restricts some write operations. The reserved space is divided into two parts: 80% of the reserved space is used as the extra disk space required for operations when the disk space is insufficient, and the other 20% is used to store the temporary file. In the process of reclaiming space, if the storage is exhausted by using too much extra disk space, this temporary file serves as the last protection for restoring services.
- The name of the temporary file is `space_placeholder_file`, located in the `storage.` ↪ `data-dir` directory. When TiKV goes offline because its disk space ran out, if you restart TiKV, the temporary file is automatically deleted and TiKV tries to reclaim the space.
- When the remaining space is insufficient, TiKV does not create the temporary file. The effectiveness of the protection is related to the size of the reserved space. The size of the reserved space is the larger value between 5% of the disk capacity and this configuration value. When the value of this configuration item is `"0MB"`, TiKV disables this disk protection feature.
- Default value: `"5GB"`
- Unit: MB|GB

14.5.2.8.5 `enable-ttl`

Warning:

- Set `enable-ttl` to `true` or `false` **ONLY WHEN** deploying a new TiKV cluster. **DO NOT** modify the value of this configuration item in an existing TiKV cluster. TiKV clusters with different `enable-ttl` values use different data formats. Therefore, if you modify the value of this item in an existing TiKV cluster, the cluster will store data in different formats, which causes the "can't enable TTL on a non-ttl" error when you restart the TiKV cluster.
- Use `enable-ttl` **ONLY IN** a TiKV cluster. **DO NOT** use this configuration item in a cluster that has TiDB nodes (which means setting `enable-ttl` to `true` in such clusters) unless `storage.api-version = 2` is configured. Otherwise, critical issues such as data corruption and the upgrade failure of TiDB clusters will occur.

- TTL is short for "Time to live". If this item is enabled, TiKV automatically deletes data that reaches its TTL. To set the value of TTL, you need to specify it in the requests when writing data via the client. If the TTL is not specified, it means that TiKV does not automatically delete the corresponding data.
- Default value: `false`

14.5.2.8.6 `ttl-check-poll-interval`

- The interval of checking data to reclaim physical spaces. If data reaches its TTL, TiKV forcibly reclaims its physical space during the check.
- Default value: `"12h"`
- Minimum value: `"0s"`

14.5.2.8.7 `background-error-recovery-window` New in v6.1.0

- The maximum allowable time for TiKV to recover after RocksDB detects a recoverable background error. If some background SST files are damaged, RocksDB will report to PD via heartbeat after locating the Peer to which the damaged SST files belong. PD then performs scheduling operations to remove this Peer. Finally, the damaged SST files are deleted directly, and the TiKV background will work as normal again.
- The damaged SST files still exist before the recovery finishes. During such a period, RocksDB can continue writing data, but an error will be reported when the damaged part of the data is read.
- If the recovery fails to finish within this time window, TiKV will panic.
- Default value: 1h

14.5.2.8.8 `api-version` New in v6.1.0

- The storage format and interface version used by TiKV when TiKV serves as the raw key-value store.
- Value options:
  - `1`: Uses API V1, does not encode the data passed from the client, and stores data as it is. In versions earlier than v6.1.0, TiKV uses API V1 by default.
  - `2`: Uses API V2:
    - The data is stored in the Multi-Version Concurrency Control (MVCC) format, where the timestamp is obtained from PD (which is TSO) by tikv-server.
    - When API V2 is used, you are expected to set `storage.enable-ttl = true` at the same time. Because API V2 supports the TTL feature, you must turn on `enable-ttl` explicitly. Otherwise, it will be in conflict because `storage.enable-ttl` defaults to `false`.
    - When API V2 is enabled, you need to deploy at least one tidb-server instance to reclaim expired data. Note that this tidb-server instance cannot provide read or write services. To ensure high availability, you can deploy multiple tidb-server instances.
    - Client support is required for API V2. For details, see the corresponding instruction of the client for the API V2.
- Default value: `1`

Warning:

- TiKV API V2 is still an experimental feature. It is not recommended to use it in production environments.
- You can set the value of `api-version` to `2` **only when** deploying a new TiKV cluster. **Do not** modify the value of this configuration item in an existing TiKV cluster. TiKV clusters with different `api-version` values use different data formats. Therefore, if you modify the value of this item in an existing TiKV cluster, the cluster will store data in different formats and causes data corruption. It will raise the "unable to switch storage.api_version" error when you start the TiKV cluster.
- After API V2 is enabled, you **cannot** downgrade the TiKV cluster to a version earlier than v6.1.0. Otherwise, data corruption might occur.

14.5.2.9 storage.block-cache

Configuration items related to the sharing of block cache among multiple RocksDB Column Families (CF). When these configuration items are enabled, block cache separately configured for each column family is disabled.

14.5.2.9.1 `shared`

- Enables or disables the sharing of block cache.
- Default value: `true`

14.5.2.9.2 `capacity`

- The size of the shared block cache.
- Default value: 45% of the size of total system memory
- Unit: KB|MB|GB

14.5.2.10 storage.flow-control

Configuration items related to the flow control mechanism in TiKV. This mechanism replaces the write stall mechanism in RocksDB and controls flow at the scheduler layer, which avoids secondary disasters caused by the stuck Raftstore or Apply threads.

14.5.2.10.1 `enable`

- Determines whether to enable the flow control mechanism. After it is enabled, TiKV automatically disables the write stall mechanism of KvDB and the write stall mechanism of RaftDB (excluding memtable).
- Default value: `true`

14.5.2.10.2 `memtables-threshold`

- When the number of kvDB memtables reaches this threshold, the flow control mechanism starts to work. When `enable` is set to `true`, this configuration item overrides `rocksdb.(defaultcf|writecf|lockcf).max-write-buffer-number`.
- Default value: `5`

14.5.2.10.3 `l0-files-threshold`

- When the number of kvDB L0 files reaches this threshold, the flow control mechanism starts to work. When `enable` is set to `true`, this configuration item overrides `rocksdb` ↪ `.(defaultcf|writecf|lockcf).level0-slowdown-writes-trigger`.
- Default value: `20`

14.5.2.10.4 `soft-pending-compaction-bytes-limit`

- When the pending compaction bytes in KvDB reach this threshold, the flow control mechanism starts to reject some write requests and reports the `ServerIsBusy` error. When `enable` is set to `true`, this configuration item overrides `rocksdb.(defaultcf|writecf|lockcf).soft-pending-compaction-bytes-limit`.
- Default value: `"192GB"`

14.5.2.10.5 `hard-pending-compaction-bytes-limit`

- When the pending compaction bytes in KvDB reach this threshold, the flow control mechanism rejects all write requests and reports the `ServerIsBusy` error. When `enable` is set to `true`, this configuration item overrides `rocksdb.(defaultcf|writecf|lockcf).hard-pending-compaction-bytes-limit`.
- Default value: `"1024GB"`

14.5.2.11 storage.io-rate-limit

Configuration items related to the I/O rate limiter.

14.5.2.11.1 `max-bytes-per-sec`

- Limits the maximum I/O bytes that a server can write to or read from the disk (determined by the `mode` configuration item below) in one second. When this limit is reached, TiKV prefers throttling background operations over foreground ones. The value of this configuration item should be set to the disk's optimal I/O bandwidth, for example, the maximum I/O bandwidth specified by your cloud disk vendor. When this configuration value is set to zero, disk I/O operations are not limited.
- Default value: `"0MB"`

14.5.2.11.2 `mode`

- Determines which types of I/O operations are counted and restrained below the `max-bytes-per-sec` threshold. Currently, only the write-only mode is supported.
- Optional value: `"write-only"`
- Default value: `"write-only"`

14.5.2.12 pd

14.5.2.12.1 `enable-forwarding` New in v5.0.0

- Controls whether the PD client in TiKV forwards requests to the leader via the followers in the case of possible network isolation.
- Default value: `false`
- If the environment might have isolated network, enabling this parameter can reduce the window of service unavailability.
- If you cannot accurately determine whether isolation, network interruption, or downtime has occurred, using this mechanism has the risk of misjudgment and causes reduced availability and performance. If network failure has never occurred, it is not recommended to enable this parameter.

14.5.2.12.2 `endpoints`

- The endpoints of PD. When multiple endpoints are specified, you need to separate them using commas.
- Default value: `["127.0.0.1:2379"]`

14.5.2.12.3 `retry-interval`

- The interval for retrying the PD connection.
- Default value: `"300ms"`

14.5.2.12.4 `retry-log-every`

- Specified the frequency at which the PD client skips reporting errors when the client observes errors. For example, when the value is `5`, after the PD client observes errors, the client skips reporting errors every 4 times and reports errors every 5th time.
- To disable this feature, set the value to `1`.
- Default value: `10`

14.5.2.12.5 `retry-max-count`

- The maximum number of times to retry to initialize PD connection
- To disable the retry, set its value to `0`. To release the limit on the number of retries, set the value to `-1`.
- Default value: `-1`

14.5.2.13 raftstore

Configuration items related to Raftstore.

14.5.2.13.1 `prevote`

- Enables or disables `prevote`. Enabling this feature helps reduce jitter on the system after recovery from network partition.
- Default value: `true`

14.5.2.13.2 `capacity`

- The storage capacity, which is the maximum size allowed to store data. If `capacity` is left unspecified, the capacity of the current disk prevails. To deploy multiple TiKV instances on the same physical disk, add this parameter to the TiKV configuration. For details, see Key parameters of the hybrid deployment.
- Default value: `0`
- Unit: KB|MB|GB

14.5.2.13.3 `raftdb-path`

- The path to the Raft library, which is `storage.data-dir/raft` by default
- Default value: `""`

14.5.2.13.4 `raft-base-tick-interval`

> **Note:**
>
> This configuration item cannot be queried via SQL statements but can be configured in the configuration file.

- The time interval at which the Raft state machine ticks
- Default value: `"1s"`
- Minimum value: greater than `0`

14.5.2.13.5 `raft-heartbeat-ticks`

> **Note:**
>
> This configuration item cannot be queried via SQL statements but can be configured in the configuration file.

- The number of passed ticks when the heartbeat is sent. This means that a heartbeat is sent at the time interval of `raft-base-tick-interval` * `raft-heartbeat-ticks`.
- Default value: `2`
- Minimum value: greater than `0`

14.5.2.13.6 `raft-election-timeout-ticks`

> **Note:**
>
> This configuration item cannot be queried via SQL statements but can be configured in the configuration file.

- The number of passed ticks when Raft election is initiated. This means that if Raft group is missing the leader, a leader election is initiated approximately after the time interval of `raft-base-tick-interval` * `raft-election-timeout-ticks`.
- Default value: `10`
- Minimum value: `raft-heartbeat-ticks`

14.5.2.13.7 `raft-min-election-timeout-ticks`

> **Note:**
>
> This configuration item cannot be queried via SQL statements but can be configured in the configuration file.

- The minimum number of ticks during which the Raft election is initiated. If the number is `0`, the value of `raft-election-timeout-ticks` is used. The value of this parameter must be greater than or equal to `raft-election-timeout-ticks`.
- Default value: `0`
- Minimum value: `0`

14.5.2.13.8 `raft-max-election-timeout-ticks`

> **Note:**
>
> This configuration item cannot be queried via SQL statements but can be configured in the configuration file.

- The maximum number of ticks during which the Raft election is initiated. If the number is `0`, the value of `raft-election-timeout-ticks` * `2` is used.
- Default value: `0`
- Minimum value: `0`

14.5.2.13.9 `raft-max-size-per-msg`

- The soft limit on the size of a single message packet
- Default value: `"1MB"`
- Minimum value: greater than `0`
- Maximum value: `3GB`
- Unit: KB|MB|GB

14.5.2.13.10 `raft-max-inflight-msgs`

- The number of Raft logs to be confirmed. If this number is exceeded, the Raft state machine slows down log sending.
- Default value: `256`
- Minimum value: greater than `0`
- Maximum value: `16384`

14.5.2.13.11 `raft-entry-max-size`

- The hard limit on the maximum size of a single log
- Default value: `"8MB"`
- Minimum value: `0`
- Unit: MB|GB

14.5.2.13.12 `raft-log-compact-sync-interval` New in v5.3

- The time interval to compact unnecessary Raft logs
- Default value: `"2s"`
- Minimum value: `"0s"`

14.5.2.13.13 `raft-log-gc-tick-interval`

- The time interval at which the polling task of deleting Raft logs is scheduled. `0` means that this feature is disabled.
- Default value: `"3s"`
- Minimum value: `"0s"`

14.5.2.13.14 raft-log-gc-threshold

- The soft limit on the maximum allowable count of residual Raft logs
- Default value: 50
- Minimum value: 1

14.5.2.13.15 raft-log-gc-count-limit

- The hard limit on the allowable number of residual Raft logs
- Default value: the log number that can be accommodated in the 3/4 Region size (calculated as 1MB for each log)
- Minimum value: 0

14.5.2.13.16 raft-log-gc-size-limit

- The hard limit on the allowable size of residual Raft logs
- Default value: 3/4 of the Region size
- Minimum value: greater than 0

14.5.2.13.17 raft-log-reserve-max-ticks New in v5.3

- After the number of ticks set by this configuration item passes, even if the number of residual Raft logs does not reach the value set by raft-log-gc-threshold, TiKV still performs garbage collection (GC) to these logs.
- Default value: 6
- Minimum value: greater than 0

14.5.2.13.18 raft-entry-cache-life-time

- The maximum remaining time allowed for the log cache in memory.
- Default value: "30s"
- Minimum value: 0

14.5.2.13.19 hibernate-regions

- Enables or disables Hibernate Region. When this option is enabled, a Region idle for a long time is automatically set as hibernated. This reduces the extra overhead caused by heartbeat messages between the Raft leader and the followers for idle Regions. You can use peer-stale-state-check-interval to modify the heartbeat interval between the leader and the followers of hibernated Regions.
- Default value: true in v5.0.2 and later versions; false in versions before v5.0.2

14.5.2.13.20 `split-region-check-tick-interval`

- Specifies the interval at which to check whether the Region split is needed. `0` means that this feature is disabled.
- Default value: `"10s"`
- Minimum value: `0`

14.5.2.13.21 `region-split-check-diff`

- The maximum value by which the Region data is allowed to exceed before Region split
- Default value: 1/16 of the Region size.
- Minimum value: `0`

14.5.2.13.22 `region-compact-check-interval`

- The time interval at which to check whether it is necessary to manually trigger RocksDB compaction. `0` means that this feature is disabled.
- Default value: `"5m"`
- Minimum value: `0`

14.5.2.13.23 `region-compact-check-step`

- The number of Regions checked at one time for each round of manual compaction
- Default value: `100`
- Minimum value: `0`

14.5.2.13.24 `region-compact-min-tombstones`

- The number of tombstones required to trigger RocksDB compaction
- Default value: `10000`
- Minimum value: `0`

14.5.2.13.25 `region-compact-tombstones-percent`

- The proportion of tombstone required to trigger RocksDB compaction
- Default value: `30`
- Minimum value: `1`
- Maximum value: `100`

14.5.2.13.26 `report-region-buckets-tick-interval` New in v6.1.0

Warning:

`report-region-buckets-tick-interval` is an experimental feature introduced in TiDB v6.1.0. It is not recommended that you use it in production environments.

- The interval at which TiKV reports bucket information to PD when `enable-region-bucket` is true.
- Default value: `10s`

14.5.2.13.27 pd-heartbeat-tick-interval

- The time interval at which a Region's heartbeat to PD is triggered. `0` means that this feature is disabled.
- Default value: `"1m"`
- Minimum value: `0`

14.5.2.13.28 pd-store-heartbeat-tick-interval

- The time interval at which a store's heartbeat to PD is triggered. `0` means that this feature is disabled.
- Default value: `"10s"`
- Minimum value: `0`

14.5.2.13.29 snap-mgr-gc-tick-interval

- The time interval at which the recycle of expired snapshot files is triggered. `0` means that this feature is disabled.
- Default value: `"1m"`
- Minimum value: `0`

14.5.2.13.30 snap-gc-timeout

- The longest time for which a snapshot file is saved
- Default value: `"4h"`
- Minimum value: `0`

14.5.2.13.31 `snap-generator-pool-size` New in v5.4.0

- Configures the size of the `snap-generator` thread pool.
- To make Regions generate snapshot faster in TiKV in recovery scenarios, you need to increase the count of the `snap-generator` threads of the corresponding worker. You can use this configuration item to increase the size of the `snap-generator` thread pool.
- Default value: `2`
- Minimum value: `1`

14.5.2.13.32 `lock-cf-compact-interval`

- The time interval at which TiKV triggers a manual compaction for the Lock Column Family
- Default value: `"10m"`
- Minimum value: `0`

14.5.2.13.33 `lock-cf-compact-bytes-threshold`

- The size out of which TiKV triggers a manual compaction for the Lock Column Family
- Default value: `"256MB"`
- Minimum value: `0`
- Unit: MB

14.5.2.13.34 `notify-capacity`

- The longest length of the Region message queue.
- Default value: `40960`
- Minimum value: `0`

14.5.2.13.35 `messages-per-tick`

- The maximum number of messages processed per batch
- Default value: `4096`
- Minimum value: `0`

14.5.2.13.36 `max-peer-down-duration`

- The longest inactive duration allowed for a peer. A peer with timeout is marked as `down`, and PD tries to delete it later.
- Default value: `"10m"`
- Minimum value: When Hibernate Region is enabled, the minimum value is `peer-stale-state-check-interval * 2`; when Hibernate Region is disabled, the minimum value is `0`.

14.5.2.13.37 `max-leader-missing-duration`

- The longest duration allowed for a peer to be in the state where a Raft group is missing the leader. If this value is exceeded, the peer verifies with PD whether the peer has been deleted.
- Default value: `"2h"`
- Minimum value: greater than `abnormal-leader-missing-duration`

14.5.2.13.38 `abnormal-leader-missing-duration`

- The longest duration allowed for a peer to be in the state where a Raft group is missing the leader. If this value is exceeded, the peer is seen as abnormal and marked in metrics and logs.
- Default value: `"10m"`
- Minimum value: greater than `peer-stale-state-check-interval`

14.5.2.13.39 `peer-stale-state-check-interval`

- The time interval to trigger the check for whether a peer is in the state where a Raft group is missing the leader.
- Default value: `"5m"`
- Minimum value: greater than `2 * election-timeout`

14.5.2.13.40 `leader-transfer-max-log-lag`

- The maximum number of missing logs allowed for the transferee during a Raft leader transfer
- Default value: `128`
- Minimum value: `10`

14.5.2.13.41 `max-snapshot-file-raw-size` New in v6.1.0

- When the size of a snapshot file exceeds this configuration value, this file will be split into multiple files.
- Default value: `100MiB`
- Minimum value: `100MiB`

14.5.2.13.42 `snap-apply-batch-size`

- The memory cache size required when the imported snapshot file is written into the disk
- Default value: `"10MB"`
- Minimum value: `0`
- Unit: MB

14.5.2.13.43 `consistency-check-interval`

> **Warning:**
>
> It is **NOT** recommended to enable the consistency check in production environments, because it affects cluster performance and is incompatible with the garbage collection in TiDB.

- The time interval at which the consistency check is triggered. `0` means that this feature is disabled.
- Default value: `"0s"`
- Minimum value: `0`

14.5.2.13.44 `raft-store-max-leader-lease`

- The longest trusted period of a Raft leader
- Default value: `"9s"`
- Minimum value: `0`

14.5.2.13.45 `right-derive-when-split`

- Specifies the start key of the new Region when a Region is split. When this configuration item is set to `true`, the start key is the maximum split key. When this configuration item is set to `false`, the start key is the original Region's start key.
- Default value: `true`

14.5.2.13.46 `merge-max-log-gap`

- The maximum number of missing logs allowed when `merge` is performed
- Default value: `10`
- Minimum value: greater than `raft-log-gc-count-limit`

14.5.2.13.47 `merge-check-tick-interval`

- The time interval at which TiKV checks whether a Region needs merge
- Default value: `"2s"`
- Minimum value: greater than `0`

14.5.2.13.48 use-delete-range

- Determines whether to delete data from the `rocksdb delete_range` interface
- Default value: `false`

14.5.2.13.49 cleanup-import-sst-interval

- The time interval at which the expired SST file is checked. `0` means that this feature is disabled.
- Default value: `"10m"`
- Minimum value: `0`

14.5.2.13.50 local-read-batch-size

- The maximum number of read requests processed in one batch
- Default value: `1024`
- Minimum value: greater than `0`

14.5.2.13.51 apply-max-batch-size

- Raft state machines process data write requests in batches by the BatchSystem. This configuration item specifies the maximum number of Raft state machines that can process the requests in one batch.
- Default value: `256`
- Minimum value: greater than `0`
- Maximum value: `10240`

14.5.2.13.52 apply-pool-size

- The allowable number of threads in the pool that flushes data to the disk, which is the size of the Apply thread pool. When you modify the size of this thread pool, refer to Performance tuning for TiKV thread pools.
- Default value: `2`
- Value ranges: `[1, CPU * 10]`. `CPU` means the number of your CPU cores.

14.5.2.13.53 store-max-batch-size

- Raft state machines process requests for flushing logs into the disk in batches by the BatchSystem. This configuration item specifies the maximum number of Raft state machines that can process the requests in one batch.
- If `hibernate-regions` is enabled, the default value is `256`. If `hibernate-regions` is disabled, the default value is `1024`.
- Minimum value: greater than `0`
- Maximum value: `10240`

14.5.2.13.54 `store-pool-size`

- The allowable number of threads in the pool that processes Raft, which is the size of the Raftstore thread pool. When you modify the size of this thread pool, refer to Performance tuning for TiKV thread pools.
- Default value: `2`
- Value ranges: `[1, CPU * 10]`. `CPU` means the number of your CPU cores.

14.5.2.13.55 `store-io-pool-size` New in v5.3.0

- The allowable number of threads that process Raft I/O tasks, which is the size of the StoreWriter thread pool. When you modify the size of this thread pool, refer to Performance tuning for TiKV thread pools.
- Default value: `0`
- Minimum value: `0`

14.5.2.13.56 `future-poll-size`

- The allowable number of threads that drive `future`
- Default value: `1`
- Minimum value: greater than `0`

14.5.2.13.57 `cmd-batch`

- Controls whether to enable batch processing of the requests. When it is enabled, the write performance is significantly improved.
- Default value: `true`

14.5.2.13.58 `inspect-interval`

- At a certain interval, TiKV inspects the latency of the Raftstore component. This parameter specifies the interval of the inspection. If the latency exceeds this value, this inspection is marked as timeout.
- Judges whether the TiKV node is slow based on the ratio of timeout inspection.
- Default value: `"500ms"`
- Minimum value: `"1ms"`

14.5.2.13.59 `raft-write-size-limit` New in v5.3.0

- Determines the threshold at which Raft data is written into the disk. If the data size is larger than the value of this configuration item, the data is written to the disk. When the value of `store-io-pool-size` is `0`, this configuration item does not take effect.
- Default value: `1MB`
- Minimum value: `0`

14.5.2.13.60 `report-min-resolved-ts-interval` New in v6.0.0

- Determines the interval at which the minimum resolved timestamp is reported to the PD leader. If this value is set to `0`, it means that the reporting is disabled.
- Default value: `"0s"`
- Minimum value: `0`
- Unit: second

14.5.2.14 coprocessor

Configuration items related to Coprocessor.

14.5.2.14.1 `split-region-on-table`

- Determines whether to split Region by table. It is recommended for you to use the feature only in TiDB mode.
- Default value: `false`

14.5.2.14.2 `batch-split-limit`

- The threshold of Region split in batches. Increasing this value speeds up Region split.
- Default value: `10`
- Minimum value: `1`

14.5.2.14.3 `region-max-size`

- The maximum size of a Region. When the value is exceeded, the Region splits into many.
- Default value: `region-split-size / 2 * 3`
- Unit: KiB|MiB|GiB

14.5.2.14.4 `region-split-size`

- The size of the newly split Region. This value is an estimate.
- Default value: `"96MiB"`
- Unit: KiB|MiB|GiB

14.5.2.14.5 `region-max-keys`

- The maximum allowable number of keys in a Region. When this value is exceeded, the Region splits into many.
- Default value: `region-split-keys / 2 * 3`

14.5.2.14.6 `region-split-keys`

- The number of keys in the newly split Region. This value is an estimate.
- Default value: `960000`

14.5.2.14.7 `enable-region-bucket` New in v6.1.0

- Determines whether to divide a Region into smaller ranges called buckets. The bucket is used as the unit of the concurrent query to improve the scan concurrency. For more about the design of the bucket, refer to Dynamic size Region.
- Default value: false

> **Warning:**
>
> - `enable-region-bucket` is an experimental feature introduced in TiDB v6.1.0. It is not recommended that you use it in production environments.
> - This configuration makes sense only when `region-split-size` is twice of `region-bucket-size` or above; otherwise, no bucket is actually generated.
> - Adjusting `region-split-size` to a larger value might have the risk of performance regression and slow scheduling.

14.5.2.14.8 `region-bucket-size` New in v6.1.0

- The size of a bucket when `enable-region-bucket` is true.
- Default value: `96MiB`

> **Warning:**
>
> `region-bucket-size` is an experimental feature introduced in TiDB v6.1.0. It is not recommended that you use it in production environments.

14.5.2.15 rocksdb

Configuration items related to RocksDB

14.5.2.15.1 `max-background-jobs`

- The number of background threads in RocksDB. When you modify the size of the RocksDB thread pool, refer to Performance tuning for TiKV thread pools.
- Default value:
  - When the number of CPU cores is 10, the default value is `9`.
  - When the number of CPU cores is 8, the default value is `7`.
  - When the number of CPU cores is `N`, the default value is `max(2, min(N - 1, 9))`.
- Minimum value: `2`

14.5.2.15.2 `max-background-flushes`

- The maximum number of concurrent background memtable flush jobs
- Default value:
  - When the number of CPU cores is 10, the default value is `3`.
  - When the number of CPU cores is 8, the default value is `2`.
  - When the number of CPU cores is `N`, the default value is `[(max-background-jobs + 3)/ 4]`.
- Minimum value: `1`

14.5.2.15.3 `max-sub-compactions`

- The number of sub-compaction operations performed concurrently in RocksDB
- Default value: `3`
- Minimum value: `1`

14.5.2.15.4 `max-open-files`

- The total number of files that RocksDB can open
- Default value: `40960`
- Minimum value: `-1`

14.5.2.15.5 `max-manifest-file-size`

- The maximum size of a RocksDB Manifest file
- Default value: `"128MB"`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.6 `create-if-missing`

- Determines whether to automatically create a DB switch
- Default value: `true`

14.5.2.15.7 `wal-recovery-mode`

- WAL recovery mode
- Optional values:
  - `"tolerate-corrupted-tail-records"`: tolerates and discards the records that have incomplete trailing data on all logs
  - `"absolute-consistency"`: abandons recovery when corrupted logs are found
  - `"point-in-time"`: recovers logs sequentially until the first corrupted log is encountered
  - `"skip-any-corrupted-records"`: post-disaster recovery. The data is recovered as much as possible, and corrupted records are skipped.
- Default value: `"point-in-time"`

14.5.2.15.8 `wal-dir`

- The directory in which WAL files are stored. If not specified, the WAL files will be stored in the same directory as the data.
- Default value: `""`

14.5.2.15.9 `wal-ttl-seconds`

- The living time of the archived WAL files. When the value is exceeded, the system deletes these files.
- Default value: `0`
- Minimum value: `0`
- unit: second

14.5.2.15.10 `wal-size-limit`

- The size limit of the archived WAL files. When the value is exceeded, the system deletes these files.
- Default value: `0`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.11 `enable-statistics`

- Determines whether to enable the statistics of RocksDB
- Default value: `true`

14.5.2.15.12 `stats-dump-period`

- The interval at which statistics are output to the log.
- Default value: `10m`

14.5.2.15.13 `compaction-readahead-size`

- Enables the readahead feature during RocksDB compaction and specifies the size of readahead data. If you are using mechanical disks, it is recommended to set the value to 2MB at least.
- Default value: `0`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.14 `writable-file-max-buffer-size`

- The maximum buffer size used in WritableFileWrite
- Default value: `"1MB"`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.15 `use-direct-io-for-flush-and-compaction`

- Determines whether to use `O_DIRECT` for both reads and writes in the background flush and compactions. The performance impact of this option: enabling `O_DIRECT` bypasses and prevents contamination of the OS buffer cache, but the subsequent file reads require re-reading the contents to the buffer cache.
- Default value: `false`

14.5.2.15.16 `rate-bytes-per-sec`

- The maximum rate permitted by RocksDB's compaction rate limiter
- Default value: `10GB`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.17 `rate-limiter-mode`

- RocksDB's compaction rate limiter mode
- Optional values: `"read-only"`, `"write-only"`, `"all-io"`
- Default value: `"write-only"`

14.5.2.15.18 `rate-limiter-auto-tuned` New in v5.0

- Determines whether to automatically optimize the configuration of the RocksDB's compaction rate limiter based on recent workload. When this configuration is enabled, compaction pending bytes will be slightly higher than usual.
- Default value: `true`

14.5.2.15.19 `enable-pipelined-write`

- Controls whether to enable Pipelined Write. When this configuration is enabled, the previous Pipelined Write is used. When this configuration is disabled, the new Pipelined Commit mechanism is used.
- Default value: `false`

14.5.2.15.20 `bytes-per-sync`

- The rate at which OS incrementally synchronizes files to disk while these files are being written asynchronously
- Default value: `"1MB"`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.21 `wal-bytes-per-sync`

- The rate at which OS incrementally synchronizes WAL files to disk while the WAL files are being written
- Default value: `"512KB"`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.22 `info-log-max-size`

> **Warning:**
>
> Starting from v5.4.0, RocksDB logs are managed by the logging module of TiKV. Therefore, this configuration item is deprecated, and its function is replaced by the configuration item `log.file.max-size`.

- The maximum size of Info log
- Default value: `"1GB"`
- Minimum value: `0`
- Unit: B|KB|MB|GB

14.5.2.15.23 `info-log-roll-time`

> **Warning:**
>
> Starting from v5.4.0, RocksDB logs are managed by the logging module of TiKV. Therefore, this configuration item is deprecated. TiKV no longer supports automatic log splitting based on time. Instead, you can use the configuration item `log.file.max-size` to set the threshold for automatic log splitting based on file size.

- The time interval at which Info logs are truncated. If the value is `0s`, logs are not truncated.
- Default value: `"0s"`

14.5.2.15.24 `info-log-keep-log-file-num`

> **Warning:**
>
> Starting from v5.4.0, RocksDB logs are managed by the logging module of TiKV. Therefore, this configuration item is deprecated, and its function is replaced by the configuration item `log.file.max-backups`.

- The maximum number of kept log files
- Default value: `10`
- Minimum value: `0`

14.5.2.15.25 `info-log-dir`

- The directory in which logs are stored
- Default value: `""`

14.5.2.15.26 `info-log-level`

> **Warning:**
>
> Starting from v5.4.0, RocksDB logs are managed by the logging module of TiKV. Therefore, this configuration item is deprecated, and its function is replaced by the configuration item `log.level`.

- Log levels of RocksDB
- Default value: `"info"`

14.5.2.16 rocksdb.titan

Configuration items related to Titan.

14.5.2.16.1 `enabled`

> **Warning**
>
> When disabling Titan for TiDB versions earlier than v8.5.0, it is not recommended to modify this configuration item to `false`, as this might cause TiKV to crash. To disable Titan, refer to the steps in Disable Titan (experimental).

- Enables or disables Titan
- Default value: `false`

14.5.2.16.2 `dirname`

- The directory in which the Titan Blob file is stored
- Default value: `"titandb"`

14.5.2.16.3 `disable-gc`

- Determines whether to disable Garbage Collection (GC) that Titan performs to Blob files
- Default value: `false`

14.5.2.16.4 `max-background-gc`

- The maximum number of GC threads in Titan
- Default value: `4`
- Minimum value: `1`

14.5.2.17 rocksdb.defaultcf | rocksdb.writecf | rocksdb.lockcf

Configuration items related to `rocksdb.defaultcf`, `rocksdb.writecf`, and `rocksdb.lockcf`.

14.5.2.17.1 `block-size`

- The default size of a RocksDB block
- Default value for `defaultcf` and `writecf`: `"64KB"`
- Default value for `lockcf`: `"16KB"`
- Minimum value: `"1KB"`
- Unit: KB|MB|GB

14.5.2.17.2 `block-cache-size`

- The cache size of a RocksDB block
- Default value for `defaultcf`: `Total machine memory * 25%`
- Default value for `writecf`: `Total machine memory * 15%`
- Default value for `lockcf`: `Total machine memory * 2%`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.17.3 `disable-block-cache`

- Enables or disables block cache
- Default value: `false`

14.5.2.17.4 `cache-index-and-filter-blocks`

- Enables or disables caching index and filter
- Default value: `true`

14.5.2.17.5 `pin-l0-filter-and-index-blocks`

- Determines whether to pin the index and filter blocks of the level 0 SST files in memory.
- Default value: `true`

14.5.2.17.6 use-bloom-filter

- Enables or disables bloom filter
- Default value: true

14.5.2.17.7 optimize-filters-for-hits

- Determines whether to optimize the hit ratio of filters
- Default value for defaultcf: true
- Default value for writecf and lockcf: false

14.5.2.17.8 whole-key-filtering

- Determines whether to put the entire key to bloom filter
- Default value for defaultcf and lockcf: true
- Default value for writecf: false

14.5.2.17.9 bloom-filter-bits-per-key

- The length that bloom filter reserves for each key
- Default value: 10
- Unit: byte

14.5.2.17.10 block-based-bloom-filter

- Determines whether each block creates a bloom filter
- Default value: false

14.5.2.17.11 read-amp-bytes-per-bit

- Enables or disables statistics of read amplification.
- Optional values: 0 (disabled), > 0 (enabled).
- Default value: 0
- Minimum value: 0

14.5.2.17.12 compression-per-level

- The default compression algorithm for each level
- Default value for defaultcf: [“no”, “no”, “lz4”, “lz4”, “lz4”, “zstd”, “zstd”]
- Default value for writecf: [“no”, “no”, “lz4”, “lz4”, “lz4”, “zstd”, “zstd”]
- Default value for lockcf: [“no”, “no”, “no”, “no”, “no”, “no”, “no”]

14.5.2.17.13 `bottommost-level-compression`

- Sets the compression algorithm of the bottommost layer. This configuration item overrides the `compression-per-level` setting.
- Ever since data is written to LSM-tree, RocksDB does not directly adopt the last compression algorithm specified in the `compression-per-level` array for the bottommost layer. `bottommost-level-compression` enables the bottommost layer to use the compression algorithm of the best compression effect from the beginning.
- If you do not want to set the compression algorithm for the bottommost layer, set the value of this configuration item to `disable`.
- Default value: `"zstd"`

14.5.2.17.14 `write-buffer-size`

- Memtable size
- Default value for `defaultcf` and `writecf`: `"128MB"`
- Default value for `lockcf`: `"32MB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.17.15 `max-write-buffer-number`

- The maximum number of memtables. When `storage.flow-control.enable` is set to `true`, `storage.flow-control.memtables-threshold` overrides this configuration item.
- Default value: `5`
- Minimum value: `0`

14.5.2.17.16 `min-write-buffer-number-to-merge`

- The minimum number of memtables required to trigger flush
- Default value: `1`
- Minimum value: `0`

14.5.2.17.17 `max-bytes-for-level-base`

- The maximum number of bytes at base level (level-1). Generally, it is set to 4 times the size of a memtable. When the level-1 data size reaches the limit value of `max-bytes-for-level-base`, the SST files of level-1 and their overlapping SST files of level-2 will be compacted.
- Default value for `defaultcf` and `writecf`: `"512MB"`
- Default value for `lockcf`: `"128MB"`

- Minimum value: `0`
- Unit: KB|MB|GB
- It is recommended that the value of `max-bytes-for-level-base` is set approximately equal to the data volume in L0 to reduce unnecessary compaction. For example, if the compression method is "no:no:lz4:lz4:lz4:lz4:lz4", the value of `max-bytes-for-level-base` should be `write-buffer-size * 4`, because there is no compression of L0 and L1 and the trigger condition of compaction for L0 is that the number of the SST files reaches 4 (the default value). When L0 and L1 both adopt compaction, you need to analyze RocksDB logs to understand the size of an SST file compressed from a memtable. For example, if the file size is 32 MB, it is recommended to set the value of `max-bytes-for-level-base` to 128 MB (`32 MB * 4`).

14.5.2.17.18 `target-file-size-base`

- The size of the target file at base level. This value is overridden by `compaction-guard-max-output-file-size` when the `enable-compaction-guard` value is `true`.
- Default value: `"8MB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.17.19 `level0-file-num-compaction-trigger`

- The maximum number of files at L0 that trigger compaction
- Default value for `defaultcf` and `writecf`: `4`
- Default value for `lockcf`: `1`
- Minimum value: `0`

14.5.2.17.20 `level0-slowdown-writes-trigger`

- The maximum number of files at L0 that trigger write stall. When `storage.flow-control.enable` is set to `true`, `storage.flow-control.l0-files-threshold` overrides this configuration item.
- Default value: `20`
- Minimum value: `0`

14.5.2.17.21 `level0-stop-writes-trigger`

- The maximum number of files at L0 required to completely block write
- Default value: `36`
- Minimum value: `0`

14.5.2.17.22 `max-compaction-bytes`

- The maximum number of bytes written into disk per compaction
- Default value: `"2GB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.17.23 `compaction-pri`

- The priority type of compaction
- Optional values:
    - `"by-compensated-size"`: compact files in order of file size and large files are compacted with higher priority.
    - `"oldest-largest-seq-first"`: prioritize compaction on files with the oldest update time. Use this value **only** when updating hot keys in small ranges.
    - `"oldest-smallest-seq-first"`: prioritize compaction on files with ranges that are not compacted to the next level for a long time. If you randomly update hot keys across the key space, this value can slightly reduce write amplification.
    - `"min-overlapping-ratio"`: prioritize compaction on files with a high overlap ratio. When a file is small in different levels (the result of `the file size in the next level ÷ the file size in this level` is small), TiKV compacts this file first. In many cases, this value can effectively reduce write amplification.
- Default value for `defaultcf` and `writecf`: `"min-overlapping-ratio"`
- Default value for `lockcf`: `"by-compensated-size"`

14.5.2.17.24 `dynamic-level-bytes`

- Determines whether to optimize dynamic level bytes
- Default value: `true`

14.5.2.17.25 `num-levels`

- The maximum number of levels in a RocksDB file
- Default value: `7`

14.5.2.17.26 `max-bytes-for-level-multiplier`

- The default amplification multiple for each layer
- Default value: `10`

14.5.2.17.27 `compaction-style`

- Compaction method
- Optional values: `"level"`, `"universal"`, `"fifo"`
- Default value: `"level"`

14.5.2.17.28 `disable-auto-compactions`

- Determines whether to disable auto compaction.
- Default value: `false`

14.5.2.17.29 `soft-pending-compaction-bytes-limit`

- The soft limit on the pending compaction bytes. When `storage.flow-control.enable` is set to `true`, `storage.flow-control.soft-pending-compaction-bytes-limit` overrides this configuration item.
- Default value: `"192GB"`
- Unit: KB|MB|GB

14.5.2.17.30 `hard-pending-compaction-bytes-limit`

- The hard limit on the pending compaction bytes. When `storage.flow-control.enable` is set to `true`, `storage.flow-control.hard-pending-compaction-bytes-limit` overrides this configuration item.
- Default value: `"256GB"`
- Unit: KB|MB|GB

14.5.2.17.31 `enable-compaction-guard`

- Enables or disables the compaction guard, which is an optimization to split SST files at TiKV Region boundaries. This optimization can help reduce compaction I/O and allows TiKV to use larger SST file size (thus less SST files overall) and at the time efficiently clean up stale data when migrating Regions.
- Default value for `defaultcf` and `writecf`: `true`
- Default value for `lockcf`: `false`

14.5.2.17.32 `compaction-guard-min-output-file-size`

- The minimum SST file size when the compaction guard is enabled. This configuration prevents SST files from being too small when the compaction guard is enabled.
- Default value: `"8MB"`
- Unit: KB|MB|GB

14.5.2.17.33 `compaction-guard-max-output-file-size`

- The maximum SST file size when the compaction guard is enabled. The configuration prevents SST files from being too large when the compaction guard is enabled. This configuration overrides `target-file-size-base` for the same column family.
- Default value: `"128MB"`
- Unit: KB|MB|GB

14.5.2.18 rocksdb.defaultcf.titan

Configuration items related to `rocksdb.defaultcf.titan`.

14.5.2.18.1 `min-blob-size`

- The smallest value stored in a Blob file. Values smaller than the specified size are stored in the LSM-Tree.
- Default value: `"1KB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.18.2 `blob-file-compression`

- The compression algorithm used in a Blob file
- Optional values: `"no"`, `"snappy"`, `"zlib"`, `"bzip2"`, `"lz4"`, `"lz4hc"`, `"zstd"`
- Default value: `"lz4"`

14.5.2.18.3 `blob-cache-size`

- The cache size of a Blob file
- Default value: `"0GB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.18.4 `min-gc-batch-size`

- The minimum total size of Blob files required to perform GC for one time
- Default value: `"16MB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.18.5 `max-gc-batch-size`

- The maximum total size of Blob files allowed to perform GC for one time
- Default value: `"64MB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.18.6 `discardable-ratio`

- The ratio at which GC is triggered for Blob files. The Blob file can be selected for GC only if the proportion of the invalid values in a Blob file exceeds this ratio.
- Default value: `0.5`
- Minimum value: `0`
- Maximum value: `1`

14.5.2.18.7 `sample-ratio`

- The ratio of (data read from a Blob file/the entire Blob file) when sampling the file during GC
- Default value: `0.1`
- Minimum value: `0`
- Maximum value: `1`

14.5.2.18.8 `merge-small-file-threshold`

- When the size of a Blob file is smaller than this value, the Blob file might still be selected for GC. In this situation, `discardable-ratio` is ignored.
- Default value: `"8MB"`
- Minimum value: `0`
- Unit: KB|MB|GB

14.5.2.18.9 `blob-run-mode`

- Specifies the running mode of Titan.
- Optional values:
  - `normal`: Writes data to the blob file when the value size exceeds `min-blob-size`.
  - `read_only`: Refuses to write new data to the blob file, but still reads the original data from the blob file.
  - `fallback`: Writes data in the blob file back to LSM.
- Default value: `normal`

14.5.2.18.10 `level-merge`

- Determines whether to optimize the read performance. When `level-merge` is enabled, there is more write amplification.
- Default value: `false`

14.5.2.18.11 `gc-merge-rewrite`

- Determines whether to use the merge operator to write back blob indexes for Titan GC. When `gc-merge-rewrite` is enabled, it reduces the effect of Titan GC on the writes in the foreground.
- Default value: `false`

14.5.2.19 raftdb

Configuration items related to `raftdb`

14.5.2.19.1 `max-background-jobs`

- The number of background threads in RocksDB. When you modify the size of the RocksDB thread pool, refer to Performance tuning for TiKV thread pools.
- Default value: `4`
- Minimum value: `2`

14.5.2.19.2 `max-sub-compactions`

- The number of concurrent sub-compaction operations performed in RocksDB
- Default value: `2`
- Minimum value: `1`

14.5.2.19.3 `wal-dir`

- The directory in which WAL files are stored
- Default value: `"/tmp/tikv/store"`

14.5.2.20 raft-engine

Configuration items related to Raft Engine.

Note:

- When you enable Raft Engine for the first time, TiKV transfers its data from RocksDB to Raft Engine. Therefore, you need to wait extra tens of seconds for TiKV to start.
- The data format of Raft Engine in TiDB v5.4.0 is not compatible with earlier TiDB versions. Therefore, if you need to downgrade a TiDB cluster from v5.4.0 to an earlier version, **before** downgrading, disable Raft Engine by setting `enable` to `false` and restart TiKV for the configuration to take effect.

14.5.2.20.1 `enable`

- Determines whether to use Raft Engine to store Raft logs. When it is enabled, configurations of `raftdb` are ignored.
- Default value: `true`

14.5.2.20.2 `dir`

- The directory at which raft log files are stored. If the directory does not exist, it will be created when TiKV is started.
- When this configuration is not set, `{data-dir}/raft-engine` is used.
- If there are multiple disks on your machine, it is recommended to store the data of Raft Engine on a different disk to improve TiKV performance.
- Default value: `""`

14.5.2.20.3 `batch-compression-threshold`

- Specifies the threshold size of a log batch. A log batch larger than this configuration is compressed. If you set this configuration item to `0`, compression is disabled.
- Default value: `"8KB"`

14.5.2.20.4 `bytes-per-sync`

- Specifies the maximum accumulative size of buffered writes. When this configuration value is exceeded, buffered writes are flushed to the disk.
- If you set this configuration item to `0`, incremental sync is disabled.
- Default value: `"4MB"`

14.5.2.20.5 `target-file-size`

- Specifies the maximum size of log files. When a log file is larger than this value, it is rotated.
- Default value: `"128MB"`

14.5.2.20.6 `purge-threshold`

- Specifies the threshold size of the main log queue. When this configuration value is exceeded, the main log queue is purged.
- This configuration can be used to adjust the disk space usage of Raft Engine.
- Default value: `"10GB"`

14.5.2.20.7 `recovery-mode`

- Determines how to deal with file corruption during recovery.
- Value options: `"absolute-consistency"`, `"tolerate-tail-corruption"`, `"tolerate-any-corruption"`
- Default value: `"tolerate-tail-corruption"`

14.5.2.20.8 `recovery-read-block-size`

- The minimum I/O size for reading log files during recovery.
- Default value: `"16KB"`
- Minimum value: `"512B"`

14.5.2.20.9 `recovery-threads`

- The number of threads used to scan and recover log files.
- Default value: `4`
- Minimum value: `1`

14.5.2.20.10 `memory-limit`

- Specifies the limit on the memory usage of Raft Engine.
- When this configuration value is not set, 15% of the available system memory is used.
- Default value: `Total machine memory * 15%`

14.5.2.21 security

Configuration items related to security.

14.5.2.21.1 `ca-path`

- The path of the CA file
- Default value: `""`

14.5.2.21.2 `cert-path`

- The path of the Privacy Enhanced Mail (PEM) file that contains the X.509 certificate
- Default value: `""`

14.5.2.21.3 `key-path`

- The path of the PEM file that contains the X.509 key
- Default value: `""`

14.5.2.21.4 `cert-allowed-cn`

- A list of acceptable X.509 Common Names in certificates presented by clients. Requests are permitted only when the presented Common Name is an exact match with one of the entries in the list.
- Default value: `[]`. This means that the client certificate CN check is disabled by default.

14.5.2.21.5 `redact-info-log` New in v4.0.8

- This configuration item enables or disables log redaction. If the configuration value is set to `true`, all user data in the log will be replaced by `?`.
- Default value: `false`

14.5.2.22 security.encryption

Configuration items related to encryption at rest (TDE).

14.5.2.22.1 `data-encryption-method`

- The encryption method for data files
- Value options: “plaintext”, “aes128-ctr”, “aes192-ctr”, and “aes256-ctr”
- A value other than “plaintext” means that encryption is enabled, in which case the master key must be specified.
- Default value: `"plaintext"`

14.5.2.22.2 `data-key-rotation-period`

- Specifies how often TiKV rotates the data encryption key.
- Default value: `7d`

14.5.2.22.3 enable-file-dictionary-log

- Enables the optimization to reduce I/O and mutex contention when TiKV manages the encryption metadata.
- To avoid possible compatibility issues when this configuration parameter is enabled (by default), see Encryption at Rest - Compatibility between TiKV versions for details.
- Default value: `true`

14.5.2.22.4 master-key

- Specifies the master key if encryption is enabled. To learn how to configure a master key, see Encryption at Rest - Configure encryption.

14.5.2.22.5 previous-master-key

- Specifies the old master key when rotating the new master key. The configuration format is the same as that of `master-key`. To learn how to configure a master key, see Encryption at Rest - Configure encryption.

14.5.2.23 import

Configuration items related to TiDB Lightning import and BR restore.

14.5.2.23.1 num-threads

- The number of threads to process RPC requests
- Default value: `8`
- Minimum value: `1`

14.5.2.24 gc

14.5.2.24.1 enable-compaction-filter New in v5.0

- Controls whether to enable the GC in Compaction Filter feature
- Default value: `true`

14.5.2.25 backup

Configuration items related to BR backup.

14.5.2.25.1 `num-threads`

- The number of worker threads to process backup
- Default value: `MIN(CPU * 0.5, 8)`
- Value range: `[1, CPU]`
- Minimum value: `1`

14.5.2.25.2 `enable-auto-tune` New in v5.4.0

- Controls whether to limit the resources used by backup tasks to reduce the impact on the cluster when the cluster resource utilization is high. For more information, refer to BR Auto-Tune.
- Default value: `true`

14.5.2.25.3 `s3-multi-part-size` New in v5.3.2

> **Note:**
>
> This configuration is introduced to address backup failures caused by S3 rate limiting. This problem has been fixed by refining the backup data storage structure. Therefore, this configuration is deprecated from v6.1.1 and is no longer recommended.

- The part size used when you perform multipart upload to S3 during backup. You can adjust the value of this configuration to control the number of requests sent to S3.
- If data is backed up to S3 and the backup file is larger than the value of this configuration item, multipart upload is automatically enabled. Based on the compression ratio, the backup file generated by a 96-MiB Region is approximately 10 MiB to 30 MiB.
- Default value: 5MiB

14.5.2.26 cdc

Configuration items related to TiCDC.

14.5.2.26.1 `min-ts-interval`

- The interval at which Resolved TS is calculated and forwarded.
- Default value: `"1s"`

14.5.2.26.2 `old-value-cache-memory-quota`

- The upper limit of memory usage by TiCDC old values.
- Default value: `512MB`

14.5.2.26.3 `sink-memory-quota`

- The upper limit of memory usage by TiCDC data change events.
- Default value: `512MB`

14.5.2.26.4 `incremental-scan-speed-limit`

- The maximum speed at which historical data is incrementally scanned.
- Default value: `"128MB"`, which means 128 MB per second.

14.5.2.26.5 `incremental-scan-threads`

- The number of threads for the task of incrementally scanning historical data.
- Default value: `4`, which means 4 threads.

14.5.2.26.6 `incremental-scan-concurrency`

- The maximum number of concurrent executions for the tasks of incrementally scanning historical data.
- Default value: `6`, which means 6 tasks can be concurrent executed at most.
- Note: The value of `incremental-scan-concurrency` must be greater than or equal to that of `incremental-scan-threads`; otherwise, TiKV will report an error at startup.

14.5.2.27 resolved-ts

Configuration items related to maintaining the Resolved TS to serve Stale Read requests.

14.5.2.27.1 `enable`

- Determines whether to maintain the Resolved TS for all Regions.
- Default value: `true`

14.5.2.27.2 `advance-ts-interval`

- The interval at which Resolved TS is calculated and forwarded.
- Default value: `"1s"`

14.5.2.27.3 `scan-lock-pool-size`

- The number of threads that TiKV uses to scan the MVCC (multi-version concurrency control) lock data when initializing the Resolved TS.
- Default value: `2`, which means 2 threads.

14.5.2.28 pessimistic-txn

For pessimistic transaction usage, refer to TiDB Pessimistic Transaction Mode.

14.5.2.28.1 `wait-for-lock-timeout`

- The longest time that a pessimistic transaction in TiKV waits for other transactions to release the lock. If the time is out, an error is returned to TiDB, and TiDB retries to add a lock. The lock wait timeout is set by `innodb_lock_wait_timeout`.
- Default value: `"1s"`
- Minimum value: `"1ms"`

14.5.2.28.2 `wake-up-delay-duration`

- When pessimistic transactions release the lock, among all the transactions waiting for lock, only the transaction with the smallest `start_ts` is woken up. Other transactions will be woken up after `wake-up-delay-duration`.
- Default value: `"20ms"`

14.5.2.28.3 `pipelined`

- This configuration item enables the pipelined process of adding the pessimistic lock. With this feature enabled, after detecting that data can be locked, TiKV immediately notifies TiDB to execute the subsequent requests and write the pessimistic lock asynchronously, which reduces most of the latency and significantly improves the performance of pessimistic transactions. But there is a still low probability that the asynchronous write of the pessimistic lock fails, which might cause the failure of pessimistic transaction commits.
- Default value: `true`

14.5.2.28.4 `in-memory` New in v6.0.0

- Enables the in-memory pessimistic lock feature. With this feature enabled, pessimistic transactions try to store their locks in memory, instead of writing the locks to disk or replicating the locks to other replicas. This improves the performance of pessimistic transactions. However, there is a still low probability that the pessimistic lock gets lost and causes the pessimistic transaction commits to fail.
- Default value: `true`
- Note that `in-memory` takes effect only when the value of `pipelined` is `true`.

14.5.2.29 quota

Configuration items related to Quota Limiter.

Suppose that your machine on which TiKV is deployed has limited resources, for example, with only 4v CPU and 16 G memory. In this situation, the foreground of TiKV might process too many read and write requests so that the CPU resources used by the background are occupied to help process such requests, which affects the performance stability of TiKV. To avoid this situation, you can use the quota-related configuration items to limit the CPU resources to be used by the foreground. When a request triggers Quota Limiter, the request is forced to wait for a while for TiKV to free up CPU resources. The exact waiting time depends on the number of requests, and the maximum waiting time is no longer than the value of `max-delay-duration`.

> **Warning:**
>
> - Quota Limiter is an experimental feature introduced in TiDB v6.0.0, and it is **NOT** recommended to use it in the production environment.
> - This feature is only suitable for environments with limited resources to ensure that TiKV can run stably in those environments. If you enable this feature in an environment with rich resources, performance degradation might occur when the amount of requests reaches a peak.

14.5.2.29.1 `foreground-cpu-time` (new in v6.0.0)

- The soft limit on the CPU resources used by TiKV foreground to process read and write requests.
- Default value: `0` (which means no limit)
- Unit: millicpu (for example, `1500` means that foreground requests consume 1.5v CPU)

14.5.2.29.2 `foreground-write-bandwidth` (new in v6.0.0)

- The soft limit on the bandwidth with which transactions write data.
- Default value: `0KB` (which means no limit)

14.5.2.29.3 `foreground-read-bandwidth` (new in v6.0.0)

- The soft limit on the bandwidth with which transactions and the Coprocessor read data.
- Default value: `0KB` (which means no limit)

14.5.2.29.4 `max-delay-duration` (new in v6.0.0)

- The maximum time that a single read or write request is forced to wait before it is processed in the foreground.
- Default value: `500ms`

14.5.2.30 causal-ts New in v6.1.0

Configuration items related to getting the timestamp when TiKV API V2 is enabled (`storage.api-version = 2`).

To reduce write latency and avoid frequent access to PD, TiKV periodically fetches and caches a batch of timestamps in the local. When the locally cached timestamps are exhausted, TiKV immediately makes a timestamp request. In this situation, the latency of some write requests are increased. To reduce the occurrence of this situation, TiKV dynamically adjusts the size of the locally cached timestamps according to the workload. For most of the time, you do not need to adjust the following parameters.

> **Warning:**
>
> TiKV API V2 is still an experimental feature. It is not recommended to use it in production environments.

14.5.2.30.1 `renew-interval`

- The interval at which the locally cached timestamps are refreshed.
- At an interval of `renew-interval`, TiKV starts a batch of timestamp refresh and adjusts the number of cached timestamps according to the timestamp consumption in the previous period. If you set this parameter to too large a value, the latest TiKV workload changes are not reflected in time. If you set this parameter to too small a value, the load of PD increases. If the write traffic is strongly fluctuating, if timestamps are frequently exhausted, and if write latency increases, you can set this parameter to a smaller value. At the same time, you should also consider the load of PD.
- Default value: `"100ms"`

14.5.2.30.2 `renew-batch-min-size`

- The minimum number of locally cached timestamps.
- TiKV adjusts the number of cached timestamps according to the timestamp consumption in the previous period. If the usage of locally cached timestamps is low, TiKV gradually reduces the number of cached timestamps until it reaches `renew-batch-min-size`. If large bursty write traffic often occurs in your application, you can set this

parameter to a larger value as appropriate. Note that this parameter is the cache size for a single tikv-server. If you set the parameter to too large a value and the cluster contains many tikv-servers, the TSO consumption will be too fast.

- In the **TiKV-RAW > Causal timestamp** panel in Grafana, **TSO batch size** is the number of locally cached timestamps that has been dynamically adjusted according to the application workload. You can refer to this metric to adjust `renew-batch-min-size`.
- Default value: `100`

14.5.3 Configure TiFlash

This document introduces the configuration parameters related to the deployment and use of TiFlash.

14.5.3.1 PD scheduling parameters

You can adjust the PD scheduling parameters using pd-ctl. Note that you can use `tiup ctl:<cluster-version> pd` to replace `pd-ctl -u <pd_ip:pd_port>` when using tiup to deploy and manage your cluster.

- `replica-schedule-limit`: determines the rate at which the replica-related operator is generated. The parameter affects operations such as making nodes offline and add replicas.

> **Note:**
>
> The value of this parameter should be less than that of `region-schedule-limit`. Otherwise, the normal Region scheduling among TiKV nodes is affected.

- `store-balance-rate`: limits the rate at which Regions of each TiKV/TiFlash store are scheduled. Note that this parameter takes effect only when the stores have newly joined the cluster. If you want to change the setting for existing stores, use the following command.

> **Note:**
>
> Since v4.0.2, the `store-balance-rate` parameter has been deprecated and changes have been made to the `store limit` command. See store-limit for details.

```
- Execute the `pd-ctl -u <pd_ip:pd_port> store limit <store_id> <value>`
    ↪ command to set the scheduling rate of a specified store. (To get `
    ↪ store_id`, you can execute the `pd-ctl -u <pd_ip:pd_port> store`
    ↪ command.
- If you do not set the scheduling rate for Regions of a specified store,
    ↪ this store inherits the setting of `store-balance-rate`.
- You can execute the `pd-ctl -u <pd_ip:pd_port> store limit` command to
    ↪ view the current setting value of `store-balance-rate`.
```

- `replication.location-labels`: indicates the topological relationship of TiKV instances. The order of the keys indicates the layering relationship of different labels. If TiFlash is enabled, you need to use `pd-ctl config placement-rules` to set the default value. For details, see geo-distributed-deployment-topology.

14.5.3.2 TiFlash configuration parameters

This section introduces the configuration parameters of TiFlash.

> **Tip:**
>
> If you need to adjust the value of a configuration item, refer to Modify the configuration.

14.5.3.2.1 Configure the `tiflash.toml` file

```
#### The listening host for supporting services such as TPC/HTTP. It is
    ↪ recommended to configure it as "0.0.0.0", which means to listen on
    ↪ all IP addresses of this machine.
listen_host = "0.0.0.0"
#### The TiFlash TCP service port.
tcp_port = 9000
#### The TiFlash HTTP service port.
http_port = 8123
#### The cache size limit of the metadata of a data block. Generally, you do
    ↪  not need to change this value.
mark_cache_size = 5368709120
#### The cache size limit of the min-max index of a data block. Generally,
    ↪ you do not need to change this value.
minmax_index_cache_size = 5368709120
#### The cache size limit of the DeltaIndex. The default value is 0, which
    ↪ means no limit.
```

```
delta_index_cache_size = 0

#### The storage path of TiFlash data. If there are multiple directories,
   ↪ separate each directory with a comma.
#### path and path_realtime_mode are deprecated since v4.0.9. Use the
   ↪ configurations
#### in the [storage] section to get better performance in the multi-disk
   ↪ deployment scenarios
#### Since TiDB v5.2.0, if you need to use the storage.io_rate_limit
   ↪ configuration, you need to set the storage path of TiFlash data to
   ↪ storage.main.dir at the same time.
#### When the [storage] configurations exist, both path and
   ↪ path_realtime_mode configurations are ignored.
### path = "/tidb-data/tiflash-9000"
#### or
### path = "/ssd0/tidb-data/tiflash,/ssd1/tidb-data/tiflash,/ssd2/tidb-data/
   ↪ tiflash"
#### The default value is false. If you set it to true and multiple
   ↪ directories
#### are set in the path, the latest data is stored in the first directory
   ↪ and older
#### data is stored in the rest directories.
### path_realtime_mode = false

#### The path in which the TiFlash temporary files are stored. By default it
   ↪  is the first directory in path
#### or in storage.latest.dir appended with "/tmp".
### tmp_path = "/tidb-data/tiflash-9000/tmp"

#### Storage paths settings take effect starting from v4.0.9
[storage]

   ## DTFile format
   ## * format_version = 2, the default format for versions < v6.0.0.
   ## * format_version = 3, the default format for versions >= v6.0.0,
      ↪ which provides more data validation features.
   # format_version = 3

   [storage.main]
   ## The list of directories to store the main data. More than 90% of the
      ↪ total data is stored in
   ## the directory list.
   dir = [ "/tidb-data/tiflash-9000" ]
   ## or
   # dir = [ "/ssd0/tidb-data/tiflash", "/ssd1/tidb-data/tiflash" ]
```

```
## The maximum storage capacity of each directory in storage.main.dir.
## If it is not set, or is set to multiple 0, the actual disk (the disk
   ↪ where the directory is located) capacity is used.
## Note that human-readable numbers such as "10GB" are not supported yet
   ↪ .
## Numbers are specified in bytes.
## The size of the capacity list should be the same with the dir size.
## For example:
# capacity = [ 10737418240, 10737418240 ]

[storage.latest]
## The list of directories to store the latest data. About 10% of the
   ↪ total data is stored in
## the directory list. The directories (or directory) listed here
   ↪ require higher IOPS
## metrics than those in storage.main.dir.
## If it is not set (by default), the values of storage.main.dir are
   ↪ used.
# dir = [ ]
## The maximum storage capacity of each directory in storage.latest.dir.
## If it is not set, or is set to multiple 0, the actual disk (the disk
   ↪ where the directory is located) capacity is used.
# capacity = [ 10737418240, 10737418240 ]

## [storage.io_rate_limit] settings are new in v5.2.0.
[storage.io_rate_limit]
## This configuration item determines whether to limit the I/O traffic,
   ↪ which is disabled by default. This traffic limit in TiFlash is
   ↪ suitable for cloud storage that has the disk bandwidth of a small
   ↪ and specific size.
## The total I/O bandwidth for disk reads and writes. The unit is bytes
   ↪ and the default value is 0, which means the I/O traffic is not
   ↪ limited by default.
# max_bytes_per_sec = 0
## max_read_bytes_per_sec and max_write_bytes_per_sec have similar
   ↪ meanings to max_bytes_per_sec. max_read_bytes_per_sec means the
   ↪ total I/O bandwidth for disk reads, and max_write_bytes_per_sec
   ↪ means the total I/O bandwidth for disk writes.
## These configuration items limit I/O bandwidth for disk reads and
   ↪ writes separately. You can use them for cloud storage that
   ↪ calculates the limit of I/O bandwidth for disk reads and writes
   ↪ separately, such as the Persistent Disk provided by Google Cloud
   ↪ Platform.
## When the value of max_bytes_per_sec is not 0, max_bytes_per_sec is
```

```
    ↪ prioritized.
  # max_read_bytes_per_sec = 0
  # max_write_bytes_per_sec = 0

  ## The following parameters control the bandwidth weights assigned to
    ↪ different I/O traffic types. Generally, you do not need to adjust
    ↪ these parameters.
  ## TiFlash internally divides I/O requests into four types: foreground
    ↪ writes, background writes, foreground reads, background reads.
  ## When the I/O traffic limit is initialized, TiFlash assigns the
    ↪ bandwidth according to the following weight ratio.
  ## The following default configurations indicate that each type of
    ↪ traffic gets a weight of 25% (25 / (25 + 25 + 25 + 25) = 25%).
  ## If the weight is configured to 0, the corresponding I/O traffic is
    ↪ not limited.
  # foreground_write_weight = 25
  # background_write_weight = 25
  # foreground_read_weight = 25
  # background_read_weight = 25
  ## TiFlash supports automatically tuning the traffic limit for different
    ↪  I/O types according to the current I/O load. Sometimes, the tuned
    ↪  bandwidth might exceed the weight ratio set above.
  ## auto_tune_sec indicates the interval of automatic tuning. The unit is
    ↪  seconds. If the value of auto_tune_sec is 0, the automatic tuning
    ↪  is disabled.
  # auto_tune_sec = 5

[flash]
  ## The listening address of TiFlash coprocessor services.
  service_addr = "0.0.0.0:3930"

#### Multiple TiFlash nodes elect a master to add or delete placement rules
  ↪ to PD,
#### and the configurations in flash.flash_cluster control this process.
[flash.flash_cluster]
  refresh_interval = Master regularly refreshes the valid period.
  update_rule_interval = Master regularly gets the status of TiFlash
    ↪ replicas and interacts with PD.
  master_ttl = The valid period of the elected master.
  cluster_manager_path = The absolute path of the pd buddy directory.
  log = The pd buddy log path.

[flash.proxy]
  ## The listening address of proxy. If it is left empty, 127.0.0.1:20170
    ↪ is used by default.
```

```
    addr = "127.0.0.1:20170"
    ## The external access address of addr. If it is left empty, "addr" is
       ↪ used by default.
    ## Should guarantee that other nodes can access through `advertise-addr`
       ↪  when you deploy the cluster on multiple nodes.
    advertise-addr = ""
    ## The listening address from which the proxy pulls metrics or status
       ↪ information. If it is left empty, 127.0.0.1:20292 is used by
       ↪ default.
    status-addr = "127.0.0.1:20292"
    ## The external access address of status-addr. If it is left empty, the
       ↪ value of "status-addr" is used by default.
    ## Should guarantee that other nodes can access through `advertise-
       ↪ status-addr` when you deploy the cluster on multiple nodes.
    advertise-status-addr = ""
    ## The data storage path of proxy.
    data-dir = "/tidb-data/tiflash-9000/flash"
    ## The configuration file path of proxy.
    config = "/tidb-deploy/tiflash-9000/conf/tiflash-learner.toml"
    ## The log path of proxy.
    log-file = "/tidb-deploy/tiflash-9000/log/tiflash_tikv.log"
    ## The log level of proxy (available options: "trace", "debug", "info",
       ↪ "warn", "error"). The default value is "info"
    # log-level = "info"

[logger]
    ## log level (available options: "trace", "debug", "info", "warn", "
       ↪ error"). The default value is "debug".
    level = "debug"
    log = "/tidb-deploy/tiflash-9000/log/tiflash.log"
    errorlog = "/tidb-deploy/tiflash-9000/log/tiflash_error.log"
    ## Size of a single log file. The default value is "100M".
    size = "100M"
    ## Maximum number of log files to save. The default value is 10.
    count = 10

[raft]
    ## PD service address. Multiple addresses are separated with commas.
    pd_addr = "10.0.1.11:2379,10.0.1.12:2379,10.0.1.13:2379"

[status]
    ## The port through which Prometheus pulls metrics information. The
       ↪ default value is 8234.
    metrics_port = 8234
```

```
[profiles]

[profiles.default]
   ## The default value is false. This parameter determines whether the
      ↪ segment
   ## of DeltaTree Storage Engine uses logical split.
   ## Using the logical split can reduce the write amplification.
   ## However, these are at the cost of disk space waste.
   ## Modifying the default value is not recommended.
   # dt_enable_logical_split = false

   ## The memory usage limit for the generated intermediate data when a
      ↪ single
   ## coprocessor query is executed. The default value is 0, which means no
      ↪  limit.
   max_memory_usage = 0

   ## The memory usage limit for the generated intermediate data when all
      ↪ queries
   ## are executed. The default value is 0 (in bytes), which means no limit
      ↪ .
   max_memory_usage_for_all_queries = 0

   ## New in v5.0. This item specifies the maximum number of cop requests
      ↪ that TiFlash Coprocessor executes at the same time. If the number
      ↪ of requests exceeds the specified value, the exceeded requests
      ↪ will queue. If the configuration value is set to 0 or not set, the
      ↪  default value is used, which is twice the number of physical
      ↪ cores.
   cop_pool_size = 0

   ## New in v5.0. This item specifies the maximum number of cop requests
      ↪ that TiFlash Coprocessor handles at the same time, including the
      ↪ requests being executed and the requests waiting in the queue. If
      ↪ the number of requests exceeds the specified value, the error "
      ↪ TiFlash Server is Busy" is returned. -1 indicates no limit; 0
      ↪ indicates using the default value, which is 10 * cop_pool_size.
   cop_pool_handle_limit = 0

   ## New in v5.0. This item specifies the maximum time that a cop request
      ↪ can queue in TiFlash. If a cop request waits in the queue for a
      ↪ time longer than the value specified by this configuration, the
      ↪ error "TiFlash Server is Busy" is returned. A value less than or
      ↪ equal to 0 indicates no limit.
   cop_pool_max_queued_seconds = 15
```

```
## New in v5.0. This item specifies the maximum number of batch requests
    ↪  that TiFlash Coprocessor executes at the same time. If the number
    ↪  of requests exceeds the specified value, the exceeded requests
    ↪ will queue. If the configuration value is set to 0 or not set, the
    ↪  default value is used, which is twice the number of physical
    ↪ cores.
batch_cop_pool_size = 0
## New in v6.1.0. This item specifies the number of requests that
    ↪ TiFlash can concurrently process when it receives ALTER TABLE ...
    ↪ COMPACT from TiDB.
## If the value is set to 0, the default value 1 prevails.
manual_compact_pool_size = 1
## New in v5.4.0. This item enables or disables the elastic thread pool
    ↪ feature, which significantly improves CPU utilization in high
    ↪ concurrency scenarios of TiFlash. The default value is true.
enable_elastic_threadpool = true
# Compression algorithm of the TiFlash storage engine. The value can be
    ↪ LZ4, zstd, or LZ4HC, and is case-insensitive. By default, LZ4 is
    ↪ used.
dt_compression_method = "LZ4"
## Compression level of the TiFlash storage engine. The default value is
    ↪  1.
## It is recommended that you set this value to 1 if
    ↪ dt_compression_method is LZ4.
## It is recommended that you set this value to -1 (smaller compression
    ↪ rate, but better read performance) or 1 if dt_compression_method
    ↪ is zstd.
## It is recommended that you set this value to 9 if
    ↪ dt_compression_method is LZ4HC.
dt_compression_level = 1

## New in v6.0.0. This item is used for the MinTSO scheduler. It
    ↪ specifies the maximum number of threads that one resource group
    ↪ can use. The default value is 5000. For details about the MinTSO
    ↪ scheduler, see https://docs.pingcap.com/tidb/v6.1/tiflash-mintso-
    ↪ scheduler.
task_scheduler_thread_soft_limit = 5000

## New in v6.0.0. This item is used for the MinTSO scheduler. It
    ↪ specifies the maximum number of threads in the global scope. The
    ↪ default value is 10000. For details about the MinTSO scheduler,
    ↪ see https://docs.pingcap.com/tidb/v6.1/tiflash-mintso-scheduler.
task_scheduler_thread_hard_limit = 10000
```

```
#### Security settings take effect starting from v4.0.5.
[security]
    ## New in v5.0. This configuration item enables or disables log
       ↪ redaction. If the configuration value
    ## is set to true, all user data in the log will be replaced by ?.
    ## Note that you also need to set security.redact-info-log for tiflash-
       ↪ learner's logging in tiflash-learner.toml.
    # redact_info_log = false

    ## Path of the file that contains a list of trusted SSL CAs. If set, the
       ↪  following settings
    ## cert_path and key_path are also needed.
    # ca_path = "/path/to/ca.pem"
    ## Path of the file that contains X509 certificate in PEM format.
    # cert_path = "/path/to/tiflash-server.pem"
    ## Path of the file that contains X509 key in PEM format.
    # key_path = "/path/to/tiflash-server-key.pem"
```

14.5.3.2.2 Configure the `tiflash-learner.toml` file

```
[server]
    engine-addr = The external access address of the TiFlash coprocessor
       ↪ service.
[raftstore]
    ## The allowable number of threads in the pool that flushes Raft data to
       ↪  storage.
    apply-pool-size = 4

    ## The allowable number of threads that process Raft, which is the size
       ↪ of the Raftstore thread pool.
    store-pool-size = 4

    ## The number of threads that handle snapshots.
    ## The default number is 2.
    ## If you set it to 0, the multi-thread optimization is disabled.
    snap-handle-pool-size = 2

[security]
    ## New in v5.0. This configuration item enables or disables log
       ↪ redaction.
    ## If the configuration value is set to true,
    ## all user data in the log will be replaced by ?. The default value is
       ↪ false.
    redact-info-log = false
```

In addition to the items above, other parameters are the same as those of TiKV. Note that the `label` whose key is `engine` is reserved and cannot be configured manually.

14.5.3.2.3 Schedule replicas by topology labels

See Set available zones.

14.5.3.2.4 Multi-disk deployment

TiFlash supports multi-disk deployment. If there are multiple disks in your TiFlash node, you can make full use of those disks by configuring the parameters described in the following sections. For TiFlash's configuration template to be used for TiUP, see The complex template for the TiFlash topology.

Multi-disk deployment with TiDB version earlier than v4.0.9

For TiDB clusters earlier than v4.0.9, TiFlash only supports storing the main data of the storage engine on multiple disks. You can set up a TiFlash node on multiple disks by specifying the `path` (`data_dir` in TiUP) and `path_realtime_mode` configuration.

If there are multiple data storage directories in `path`, separate each with a comma. For example, `/nvme_ssd_a/data/tiflash,/sata_ssd_b/data/tiflash,/sata_ssd_c/data/tiflash`. If there are multiple disks in your environment, it is recommended that each directory corresponds to one disk and you put disks with the best performance at the front to maximize the performance of all disks.

If there are multiple disks with similar I/O metrics on your TiFlash node, you can leave the `path_realtime_mode` parameter to the default value (or you can explicitly set it to `false`). It means that data will be evenly distributed among all storage directories. However, the latest data is written only to the first directory, so the corresponding disk is busier than other disks.

If there are multiple disks with different I/O metrics on your TiFlash node, it is recommended to set `path_realtime_mode` to `true` and put disks with the best I/O metrics at the front of `path`. It means that the first directory only stores the latest data, and the older data are evenly distributed among the other directories. Note that in this case, the capacity of the first directory should be planned as 10% of the total capacity of all directories.

Multi-disk deployment with TiDB v4.0.9 or later

For TiDB clusters with v4.0.9 or later versions, TiFlash supports storing the main data and the latest data of the storage engine on multiple disks. If you want to deploy a TiFlash node on multiple disks, it is recommended to specify your storage directories in the `[storage]` section to make full use of your node. Note that the configurations earlier than v4.0.9 (`path` and `path_realtime_mode`) are still supported.

If there are multiple disks with similar I/O metrics on your TiFlash node, it is recommended to specify corresponding directories in the `storage.main.dir` list and leave `storage.latest.dir` empty. TiFlash will distribute I/O pressure and data among all directories.

If there are multiple disks with different I/O metrics on your TiFlash node, it is recommended to specify directories with higher metrics in the `storage.latest.dir` list, and specify directories with lower metrics in the `storage.main.dir` list. For example, for one NVMe-SSD and two SATA-SSDs, you can set `storage.latest.dir` to `["/nvme_ssd_a/data/tiflash"]` and `storage.main.dir` to `["/sata_ssd_b/data/tiflash", "/sata_ssd_c/data/tiflash"]`. TiFlash will distribute I/O pressure and data among these two directories list respectively. Note that in this case, the capacity of `storage.latest.dir` should be planned as 10% of the total planned capacity.

> **Warning:**
>
> The `[storage]` configuration is supported in TiUP since v1.2.5. If your TiDB cluster version is v4.0.9 or later, make sure that your TiUP version is v1.2.5 or later. Otherwise, the data directories defined in `[storage]` will not be managed by TiUP.

14.5.4 PD Configuration File

The PD configuration file supports more options than command-line parameters. You can find the default configuration file here.

This document only describes parameters that are not included in command-line parameters. Check here for the command line parameters.

> **Tip:**
>
> If you need to adjust the value of a configuration item, refer to Modify the configuration.

14.5.4.0.1 `name`

- The unique name of a PD node
- Default value: `"pd"`
- To start multiply PD nodes, use a unique name for each node.

14.5.4.0.2 `data-dir`

- The directory in which PD stores data
- Default value: `default.${name}"`

14.5.4.0.3 `client-urls`

- The list of client URLs to be listened to by PD
- Default value: `"http://127.0.0.1:2379"`
- When you deploy a cluster, you must specify the IP address of the current host as `client-urls` (for example, `"http://192.168.100.113:2379"`). If the cluster runs on Docker, specify the IP address of Docker as `"http://0.0.0.0:2379"`.

14.5.4.0.4 `advertise-client-urls`

- The list of advertise URLs for the client to access PD
- Default value: `"${client-urls}"`
- In some situations such as in the Docker or NAT network environment, if a client cannot access PD through the default client URLs listened to by PD, you must manually set the advertise client URLs.
- For example, the internal IP address of Docker is `172.17.0.1`, while the IP address of the host is `192.168.100.113` and the port mapping is set to `-p 2380:2380`. In this case, you can set `advertise-client-urls` to `"http://192.168.100.113:2380"`. The client can find this service through `"http://192.168.100.113:2380"`.

14.5.4.0.5 `peer-urls`

- The list of peer URLs to be listened to by a PD node
- Default value: `"http://127.0.0.1:2380"`
- When you deploy a cluster, you must specify `peer-urls` as the IP address of the current host, such as `"http://192.168.100.113:2380"`. If the cluster runs on Docker, specify the IP address of Docker as `"http://0.0.0.0:2380"`.

14.5.4.0.6 `advertise-peer-urls`

- The list of advertise URLs for other PD nodes (peers) to access a PD node
- Default: `"${peer-urls}"`
- In some situations such as in the Docker or NAT network environment, if the other nodes (peers) cannot access the PD node through the default peer URLs listened to by this PD node, you must manually set the advertise peer URLs.
- For example, the internal IP address of Docker is `172.17.0.1`, while the IP address of the host is `192.168.100.113` and the port mapping is set to `-p 2380:2380`. In this case, you can set `advertise-peer-urls` to `"http://192.168.100.113:2380"`. The other PD nodes can find this service through `"http://192.168.100.113:2380"`.

14.5.4.0.7 `initial-cluster`

- The initial cluster configuration for bootstrapping
- Default value: `"{name}=http://{advertise-peer-url}"`
- For example, if `name` is "pd", and `advertise-peer-urls` is `"http://192.168.100.113:2380"` ↪ , the `initial-cluster` is `"pd=http://192.168.100.113:2380"`.
- If you need to start three PD servers, the `initial-cluster` might be:

```
pd1=http://192.168.100.113:2380, pd2=http://192.168.100.114:2380, pd3
    ↪ =192.168.100.115:2380
```

14.5.4.0.8 `initial-cluster-state`

- The initial state of the cluster
- Default value: `"new"`

14.5.4.0.9 `initial-cluster-token`

- Identifies different clusters during the bootstrap phase
- Default value: `"pd-cluster"`
- If multiple clusters that have nodes with same configurations are deployed successively, you must specify different tokens to isolate different cluster nodes.

14.5.4.0.10 `lease`

- The timeout of the PD Leader Key lease. After the timeout, the system re-elects a Leader.
- Default value: `3`
- Unit: second

14.5.4.0.11 `quota-backend-bytes`

- The storage size of the meta-information database, which is 8GiB by default
- Default value: `8589934592`

14.5.4.0.12 `auto-compaction-mod`

- The automatic compaction modes of the meta-information database
- Available options: `periodic` (by cycle) and `revision` (by version number).
- Default value: `periodic`

14.5.4.0.13 `auto-compaction-retention`

- The time interval for automatic compaction of the meta-information database when `auto-compaction-retention` is `periodic`. When the compaction mode is set to `revision`, this parameter indicates the version number for the automatic compaction.
- Default value: 1h

14.5.4.0.14 `force-new-cluster`

- Determines whether to force PD to start as a new cluster and modify the number of Raft members to `1`
- Default value: `false`

14.5.4.1 pd-server

Configuration items related to pd-server

14.5.4.1.1 `flow-round-by-digit` New in TiDB 5.1

- Default value: 3
- PD rounds the lowest digits of the flow number, which reduces the update of statistics caused by the changes of the Region flow information. This configuration item is used to specify the number of lowest digits to round for the Region flow information. For example, the flow `100512` will be rounded to `101000` because the default value is `3`. This configuration replaces `trace-region-flow`.

> **Note:**
>
> If you have upgraded your cluster from a TiDB 4.0 version to the current version, the behavior of `flow-round-by-digit` after the upgrading and the behavior of `trace-region-flow` before the upgrading are consistent by default. This means that if the value of `trace-region-flow` is false before the upgrading, the value of `flow-round-by-digit` after the upgrading is 127; if the value of `trace-region-flow` is `true` before the upgrading, the value of `flow-round-by-digit` after the upgrading is `3`.

14.5.4.1.2 `min-resolved-ts-persistence-interval` New in v6.0.0

- Determines the interval at which the minimum resolved timestamp is persistent to the PD. If this value is set to `0`, it means that the persistence is disabled.
- Default value: `"0s"`
- Minimum value: `0`
- Unit: second

14.5.4.2 security

Configuration items related to security

14.5.4.2.1 `cacert-path`

- The path of the CA file
- Default value: “”

14.5.4.2.2 `cert-path`

- The path of the Privacy Enhanced Mail (PEM) file that contains the X509 certificate
- Default value: “”

14.5.4.2.3 `key-path`

- The path of the PEM file that contains the X509 key
- Default value: “”

14.5.4.2.4 `redact-info-log` New in v5.0

- Controls whether to enable log redaction in the PD log
- When you set the configuration value to `true`, user data is redacted in the PD log.
- Default value: `false`

14.5.4.3 `log`

Configuration items related to log

14.5.4.3.1 `level`

- Specifies the level of the output log
- Optional value: `"debug"`, `"info"`, `"warn"`, `"error"`, `"fatal"`
- Default value: `"info"`

14.5.4.3.2 `format`

- The log format
- Optional value: `"text"`, `"json"`
- Default value: `"text"`

14.5.4.3.3 `disable-timestamp`

- Whether to disable the automatically generated timestamp in the log
- Default value: `false`

14.5.4.4 `log.file`

Configuration items related to the log file

14.5.4.4.1 `max-size`

- The maximum size of a single log file. When this value is exceeded, the system automatically splits the log into several files.
- Default value: `300`
- Unit: MiB
- Minimum value: `1`

14.5.4.4.2 `max-days`

- The maximum number of days in which a log is kept
- If the configuration item is not set, or the value of it is set to the default value 0, PD does not clean log files.
- Default value: `0`

14.5.4.4.3 `max-backups`

- The maximum number of log files to keep
- If the configuration item is not set, or the value of it is set to the default value 0, PD keeps all log files.
- Default value: `0`

14.5.4.5 `metric`

Configuration items related to monitoring

14.5.4.5.1 `interval`

- The interval at which monitoring metric data is pushed to Prometheus
- Default value: `15s`

14.5.4.6 `schedule`

Configuration items related to scheduling

Note:

To modify these PD configuration items related to `schedule`, choose one of the following methods based on your cluster status:

- For clusters to be newly deployed, you can modify the PD configuration file directly.
- For existing clusters, use the command-line tool PD Control to make changes instead. Direct modifications to these PD configuration items related to `schedule` in the configuration file do not take effect on existing clusters.

14.5.4.6.1 `max-merge-region-size`

- Controls the size limit of `Region Merge`. When the Region size is greater than the specified value, PD does not merge the Region with the adjacent Regions.
- Default value: `20`
- Unit: MiB

14.5.4.6.2 `max-merge-region-keys`

- Specifies the upper limit of the `Region Merge` key. When the Region key is greater than the specified value, the PD does not merge the Region with its adjacent Regions.
- Default value: `200000`

14.5.4.6.3 `patrol-region-interval`

- Controls the running frequency at which `replicaChecker` checks the health state of a Region. The smaller this value is, the faster `replicaChecker` runs. Normally, you do not need to adjust this parameter.
- Default value: `10ms`

14.5.4.6.4 `split-merge-interval`

- Controls the time interval between the `split` and `merge` operations on the same Region. That means a newly split Region will not be merged for a while.
- Default value: `1h`

14.5.4.6.5 `max-snapshot-count`

- Controls the maximum number of snapshots that a single store receives or sends at the same time. PD schedulers depend on this configuration to prevent the resources used for normal traffic from being preempted.
- Default value value: `64`

14.5.4.6.6 `max-pending-peer-count`

- Controls the maximum number of pending peers in a single store. PD schedulers depend on this configuration to prevent too many Regions with outdated logs from being generated on some nodes.
- Default value: `64`

14.5.4.6.7 `max-store-down-time`

- The downtime after which PD judges that the disconnected store cannot be recovered. When PD fails to receive the heartbeat from a store after the specified period of time, it adds replicas at other nodes.
- Default value: `30m`

14.5.4.6.8 `max-store-preparing-time` New in v6.1.0

- Controls the maximum waiting time for the store to go online. During the online stage of a store, PD can query the online progress of the store. When the specified time is exceeded, PD assumes that the store has been online and cannot query the online progress of the store again. But this does not prevent Regions from transferring to the new online store. In most scenarios, you do not need to adjust this parameter.
- Default value: `48h`

14.5.4.6.9 `leader-schedule-limit`

- The number of Leader scheduling tasks performed at the same time
- Default value: `4`

14.5.4.6.10 `region-schedule-limit`

- The number of Region scheduling tasks performed at the same time
- Default value: `2048`

14.5.4.6.11 `hot-region-schedule-limit`

- Controls the hot Region scheduling tasks that are running at the same time. It is independent of the Region scheduling.
- Default value: `4`

14.5.4.6.12 `hot-region-cache-hits-threshold`

- The threshold used to set the number of minutes required to identify a hot Region. PD can participate in the hotspot scheduling only after the Region is in the hotspot state for more than this number of minutes.
- Default value: `3`

14.5.4.6.13 `replica-schedule-limit`

- The number of Replica scheduling tasks performed at the same time
- Default value: `64`

14.5.4.6.14 `merge-schedule-limit`

- The number of the `Region Merge` scheduling tasks performed at the same time. Set this parameter to `0` to disable `Region Merge`.
- Default value: `8`

14.5.4.6.15 `high-space-ratio`

- The threshold ratio below which the capacity of the store is sufficient. If the space occupancy ratio of the store is smaller than this threshold value, PD ignores the remaining space of the store when performing scheduling, and balances load mainly based on the Region size. This configuration takes effect only when `region-score-formula-version` is set to `v1`.
- Default value: `0.7`
- Minimum value: greater than `0`
- Maximum value: less than `1`

14.5.4.6.16 `low-space-ratio`

- The threshold ratio above which the capacity of the store is insufficient. If the space occupancy ratio of a store exceeds this threshold value, PD avoids migrating data to this store as much as possible. Meanwhile, to avoid the disk space of the corresponding store being exhausted, PD performs scheduling mainly based on the remaining space of the store.
- Default value: `0.8`
- Minimum value: greater than `0`
- Maximum value: less than `1`

14.5.4.6.17 `tolerant-size-ratio`

- Controls the `balance` buffer size
- Default value: `0` (automatically adjusts the buffer size)
- Minimum value: `0`

14.5.4.6.18 `enable-cross-table-merge`

- Determines whether to enable the merging of cross-table Regions
- Default value: `true`

14.5.4.6.19 `region-score-formula-version` New in v5.0

- Controls the version of the Region score formula
- Default value: `v2`
- Optional values: `v1` and `v2`. Compared to v1, the changes in v2 are smoother, and the scheduling jitter caused by space reclaim is improved.

> **Note:**
>
> If you have upgraded your cluster from a TiDB 4.0 version to the current version, the new formula version is automatically disabled by default to ensure consistent PD behavior before and after the upgrading. If you want to change the formula version, you need to manually switch through the `pd-ctl` setting. For details, refer to PD Control.

14.5.4.6.20 `enable-joint-consensus` New in v5.0

- Controls whether to use Joint Consensus for replica scheduling. If this configuration is disabled, PD schedules one replica at a time.
- Default value: `true`

14.5.4.6.21 `hot-regions-write-interval` New in v5.4.0

- The time interval at which PD stores hot Region information.
- Default value: `10m`

> **Note:**
>
> The information about hot Regions is updated every three minutes. If the interval is set to less than three minutes, updates during the interval might be meaningless.

14.5.4.6.22 `hot-regions-reserved-days` New in v5.4.0

- Specifies how many days the hot Region information is retained.
- Default value: `7`

14.5.4.7 `replication`

Configuration items related to replicas

14.5.4.7.1 `max-replicas`

- The number of replicas, that is, the sum of the number of leaders and followers. The default value `3` means 1 leader and 2 followers. When this configuration is modified dynamically, PD will schedule Regions in the background so that the number of replicas matches this configuration.
- Default value: `3`

14.5.4.7.2 `location-labels`

- The topology information of a TiKV cluster
- Default value: `[]`
- Cluster topology configuration

14.5.4.7.3 `isolation-level`

- The minimum topological isolation level of a TiKV cluster
- Default value: `""`
- Cluster topology configuration

14.5.4.7.4 `strictly-match-label`

- Enables the strict check for whether the TiKV label matches PD's `location-labels`.
- Default value: `false`

14.5.4.7.5 `enable-placement-rules`

- Enables `placement-rules`.
- Default value: `true`
- See Placement Rules.

14.5.4.8 `label-property` (deprecated)

Configuration items related to labels, which only support the `reject-leader` type.

> **Note:**
>
> Starting from v5.2, the configuration items related to labels are deprecated. It is recommended to use Placement Rules to configure the replica policy.

14.5.4.8.1 `key` (deprecated)

- The label key for the store that rejected the Leader
- Default value: `""`

14.5.4.8.2 `value` (deprecated)

- The label value for the store that rejected the Leader
- Default value: `""`

14.5.4.9 `dashboard`

Configuration items related to the TiDB Dashboard built in PD.

14.5.4.9.1 `tidb-cacert-path`

- The path of the root CA certificate file. You can configure this path when you connect to TiDB's SQL services using TLS.
- Default value: `""`

14.5.4.9.2 `tidb-cert-path`

- The path of the SSL certificate file. You can configure this path when you connect to TiDB's SQL services using TLS.
- Default value: `""`

14.5.4.9.3 `tidb-key-path`

- The path of the SSL private key file. You can configure this path when you connect to TiDB's SQL services using TLS.
- Default value: `""`

14.5.4.9.4 `public-path-prefix`

- When TiDB Dashboard is accessed behind a reverse proxy, this item sets the public URL path prefix for all web resources.
- Default value: `/dashboard`
- Do **not** modify this configuration item when TiDB Dashboard is accessed not behind a reverse proxy; otherwise, access issues might occur. See Use TiDB Dashboard behind a Reverse Proxy for details.

14.5.4.9.5 `enable-telemetry`

- Determines whether to enable the telemetry collection feature in TiDB Dashboard.
- Default value: `true` for v6.1.0 ~ v6.1.4; `false` for v6.1.5 and later v6.1.x versions
- See Telemetry for details.

14.5.4.10 `replication-mode`

Configuration items related to the replication mode of all Regions. See Enable the DR Auto-Sync mode for details.

14.6 CLI

14.6.1 TiKV Control User Guide

TiKV Control (`tikv-ctl`) is a command line tool of TiKV, used to manage the cluster. Its installation directory is as follows:

- If the cluster is deployed using TiUP, `tikv-ctl` directory is in the in `~/.tiup/components/ctl/{VERSION}/` directory.

14.6.1.1 Use TiKV Control in TiUP

> **Note:**
>
> It is recommended that the version of the Control tool you use is consistent with the version of the cluster.

`tikv-ctl` is also integrated in the `tiup` command. Execute the following command to call the `tikv-ctl` tool:

```
tiup ctl:<cluster-version> tikv
```

```
Starting component `ctl`: /home/tidb/.tiup/components/ctl/v4.0.8/ctl tikv
TiKV Control (tikv-ctl)
Release Version: 4.0.8
Edition:         Community
Git Commit Hash: 83091173e960e5a0f5f417e921a0801d2f6635ae
Git Commit Branch: heads/refs/tags/v4.0.8
UTC Build Time:  2020-10-30 08:40:33
Rust Version:    rustc 1.42.0-nightly (0de96d37f 2019-12-19)
Enable Features: jemalloc mem-profiling portable sse protobuf-codec
Profile:         dist_release

A tool for interacting with TiKV deployments.
USAGE:
    TiKV Control (tikv-ctl) [FLAGS] [OPTIONS] [SUBCOMMAND]
FLAGS:
    -h, --help                 Prints help information
        --skip-paranoid-checks Skip paranoid checks when open rocksdb
    -V, --version              Prints version information
OPTIONS:
        --ca-path <ca-path>          Set the CA certificate path
        --cert-path <cert-path>      Set the certificate path
        --config <config>            TiKV config path, by default it's <
            ↪ deploy-dir>/conf/tikv.toml
        --data-dir <data-dir>        TiKV data directory path, check <deploy-
            ↪ dir>/scripts/run.sh to get it
        --decode <decode>            Decode a key in escaped format
        --encode <encode>            Encode a key in escaped format
        --to-hex <escaped-to-hex>    Convert an escaped key to hex key
        --to-escaped <hex-to-escaped> Convert a hex key to escaped key
        --host <host>                Set the remote host
        --key-path <key-path>        Set the private key path
        --log-level <log-level>      Set the log level [default: warn]
        --pd <pd>                    Set the address of pd
SUBCOMMANDS:
    bad-regions        Get all regions with corrupt raft
    cluster            Print the cluster id
    compact            Compact a column family in a specified range
    compact-cluster    Compact the whole cluster in a specified range in one
        ↪  or more column families
    consistency-check  Force a consistency-check for a specified region
```

```
decrypt-file       Decrypt an encrypted file
diff               Calculate difference of region keys from different
    ↪ dbs
dump-snap-meta     Dump snapshot meta file
encryption-meta    Dump encryption metadata
fail               Inject failures to TiKV and recovery
help               Prints this message or the help of the given
    ↪ subcommand(s)
metrics            Print the metrics
modify-tikv-config Modify tikv config, eg. tikv-ctl --host ip:port
    ↪ modify-tikv-config -n
                   rocksdb.defaultcf.disable-auto-compactions -v true
mvcc               Print the mvcc value
print              Print the raw value
raft               Print a raft log entry
raw-scan           Print all raw keys in the range
recover-mvcc       Recover mvcc data on one node by deleting corrupted
    ↪ keys
recreate-region    Recreate a region with given metadata, but alloc new
    ↪ id for it
region-properties  Show region properties
scan               Print the range db range
size               Print region size
split-region       Split the region
store              Print the store id
tombstone          Set some regions on the node to tombstone by manual
unsafe-recover     Unsafely recover the cluster when the majority
    ↪ replicas are failed
```

You can add corresponding parameters and subcommands after `tiup ctl:<cluster-version> tikv`.

14.6.1.2 General options

`tikv-ctl` provides two operation modes:

- Remote mode: use the `--host` option to accept the service address of TiKV as the argument

  For this mode, if SSL is enabled in TiKV, `tikv-ctl` also needs to specify the related certificate file. For example:

  ```
  $ tikv-ctl --ca-path ca.pem --cert-path client.pem --key-path client-
      ↪ key.pem --host 127.0.0.1:20160 <subcommands>
  ```

  However, sometimes `tikv-ctl` communicates with PD instead of TiKV. In this case, you need to use the `--pd` option instead of `--host`. Here is an example:

```
$ tikv-ctl --pd 127.0.0.1:2379 compact-cluster
store:"127.0.0.1:20160" compact db:KV cf:default range:([], []) success
    ↪ !
```

- Local mode:
  - Use the `--data-dir` option to specify the local TiKV data directory path.
  - Use the `--config` option to specify the local TiKV configuration file path.

In this mode, you need to stop the running TiKV instance.

Unless otherwise noted, all commands support both the remote mode and the local mode.

Additionally, `tikv-ctl` has two simple commands `--to-hex` and `--to-escaped`, which are used to make simple changes to the form of the key.

Generally, use the `escaped` form of the key. For example:

```
$ tikv-ctl --to-escaped 0xaaff
\252\377
$ tikv-ctl --to-hex "\252\377"
AAFF
```

> **Note:**
>
> When you specify the `escaped` form of the key in a command line, it is required to enclose it in double quotes. Otherwise, bash eats the backslash and a wrong result is returned.

14.6.1.3 Subcommands, some options and flags

This section describes the subcommands that `tikv-ctl` supports in detail. Some subcommands support a lot of options. For all details, run `tikv-ctl --help <subcommand>`.

14.6.1.3.1 View information of the Raft state machine

Use the `raft` subcommand to view the status of the Raft state machine at a specific moment. The status information includes two parts: three structs (**RegionLocalState**, **RaftLocalState**, and **RegionApplyState**) and the corresponding Entries of a certain piece of log.

Use the `region` and `log` subcommands to obtain the above information respectively. The two subcommands both support the remote mode and the local mode at the same time. Their usage and output are as follows:

```
$ tikv-ctl --host 127.0.0.1:20160 raft region -r 2
region id: 2
region state key: \001\003\000\000\000\000\000\000\000\002\001
region state: Some(region {id: 2 region_epoch {conf_ver: 3 version: 1} peers
    ↪ {id: 3 store_id: 1} peers {id: 5 store_id: 4} peers {id: 7 store_id:
    ↪ 6}})
raft state key: \001\002\000\000\000\000\000\000\000\002\002
raft state: Some(hard_state {term: 307 vote: 5 commit: 314617} last_index:
    ↪ 314617)
apply state key: \001\002\000\000\000\000\000\000\000\002\003
apply state: Some(applied_index: 314617 truncated_state {index: 313474 term:
    ↪ 151})
```

14.6.1.3.2 View the Region size

Use the `size` command to view the Region size:

```
$ tikv-ctl --data-dir /path/to/tikv size -r 2
region id: 2
cf default region size: 799.703 MB
cf write region size: 41.250 MB
cf lock region size: 27616
```

14.6.1.3.3 Scan to view MVCC of a specific range

The `--from` and `--to` options of the `scan` command accept two escaped forms of raw key, and use the `--show-cf` flag to specify the column families that you need to view.

```
$ tikv-ctl --data-dir /path/to/tikv scan --from 'zm' --limit 2 --show-cf
    ↪ lock,default,write
key: zmBootstr\377a\377pKey
    ↪ \000\000\377\000\000\373\000\000\000\000\000\377\000\000s
    ↪ \000\000\000\000\000\372
        write cf value: start_ts: 399650102814441473 commit_ts:
            ↪ 399650102814441475 short_value: "20"
key: zmDB:29\000\000\377\000\374\000\000\000\000\000\000\377\000H
    ↪ \000\000\000\000\000\000\371
        write cf value: start_ts: 399650105239273474 commit_ts:
            ↪ 399650105239273475 short_value: "
            ↪ \000\000\000\000\000\000\000\002"
        write cf value: start_ts: 399650105199951882 commit_ts:
            ↪ 399650105213059076 short_value: "
            ↪ \000\000\000\000\000\000\000\001"
```

14.6.1.3.4 View MVCC of a given key

Similar to the `scan` command, the `mvcc` command can be used to view MVCC of a given key.

```
$ tikv-ctl --data-dir /path/to/tikv mvcc -k "zmDB
    ↪ :29\000\000\377\000\374\000\000\000\000\000\000\377\000H
    ↪ \000\000\000\000\000\000\371" --show-cf=lock,write,default
key: zmDB:29\000\000\377\000\374\000\000\000\000\000\000\377\000H
    ↪ \000\000\000\000\000\000\371
        write cf value: start_ts: 399650105239273474 commit_ts:
            ↪ 399650105239273475 short_value: "
            ↪ \000\000\000\000\000\000\000\002"
        write cf value: start_ts: 399650105199951882 commit_ts:
            ↪ 399650105213059076 short_value: "
            ↪ \000\000\000\000\000\000\000\001"
```

In this command, the key is also the escaped form of raw key.

14.6.1.3.5 Scan raw keys

The `raw-scan` command scans directly from the RocksDB. Note that to scan data keys you need to add a `'z'` prefix to keys.

Use `--from` and `--to` options to specify the range to scan (unbounded by default). Use `--limit` to limit at most how many keys to print out (30 by default). Use `--cf` to specify which cf to scan (can be `default`, `write` or `lock`).

```
$ ./tikv-ctl --data-dir /var/lib/tikv raw-scan --from 'zt' --limit 2 --cf
    ↪ default
key: "zt\200\000\000\000\000\000\000\377\005_r
    ↪ \200\000\000\000\000\377\000\000\001\000\000\000\000\000\372\372b2
    ↪ ,^\033\377\364", value: "\010\002\002\002%\010\004\002\010root
    ↪ \010\006\002\000\010\010\t\002\010\n\t\002\010\014\t\002\010\016\t
    ↪ \002\010\020\t\002\010\022\t\002\010\024\t\002\010\026\t\002\010\030\
    ↪ t\002\010\032\t\002\010\034\t\002\010\036\t\002\010 \t\002\010\"\t
    ↪ \002\010s\t\002\010&\t\002\010(\t\002\010*\t\002\010,\t\002\010.\t
    ↪ \002\0100\t\002\0102\t\002\0104\t\002"
key: "zt\200\000\000\000\000\000\000\377\025_r
    ↪ \200\000\000\000\000\377\000\000\023\000\000\000\000\000\372\372b2
    ↪ ,^\033\377\364", value: "\010\002\002&slow_query_log_file\010\004\002
    ↪ P/usr/local/mysql/data/localhost-slow.log"

Total scanned keys: 2
```

14.6.1.3.6 Print a specific key value

To print the value of a key, use the `print` command.

14.6.1.3.7 Print some properties about Region

In order to record Region state details, TiKV writes some statistics into the SST files of Regions. To view these properties, run `tikv-ctl` with the `region-properties` sub-command:

```
$ tikv-ctl --host localhost:20160 region-properties -r 2
num_files: 0
num_entries: 0
num_deletes: 0
mvcc.min_ts: 18446744073709551615
mvcc.max_ts: 0
mvcc.num_rows: 0
mvcc.num_puts: 0
mvcc.num_versions: 0
mvcc.max_row_versions: 0
middle_key_by_approximate_size:
```

The properties can be used to check whether the Region is healthy or not. If not, you can use them to fix the Region. For example, splitting the Region manually by `middle_key_approximate_size`.

14.6.1.3.8 Compact data of each TiKV manually

Use the `compact` command to manually compact data of each TiKV.

- Use the `--from` and `--to` options to specify the compaction range in the form of escaped raw key. If not set, the whole range will be compacted.
- Use the `--region` option to compact the range of a specific region. If set, `--from` and `--to` will be ignored.
- Use the `-c` option to specify the column family name. The default value is `default`. The optional values are `default`, `lock`, and `write`.
- Use the `-d` option to specify the RocksDB that performs compaction. The default value is `kv`. The optional values are `kv` and `raft`.
- Use the `--threads` option allows you to specify the concurrency for the TiKV compaction and its default value is `8`. Generally, a higher concurrency comes with a faster compaction speed, which might yet affect the service. You need to choose an appropriate concurrency count based on your scenario.
- Use the `--bottommost` option to include or exclude the bottommost files when TiKV performs compaction. The value options are `default`, `skip`, and `force`. The default value is `default`.

- `default` means that the bottommost files are included only when the Compaction Filter feature is enabled.
- `skip` means that the bottommost files are excluded when TiKV performs compaction.
- `force` means that the bottommost files are always included when TiKV performs compaction.

- To compact data in the local mode, use the following command:

```
tikv-ctl --data-dir /path/to/tikv compact -d kv
```

- To compact data in the remote mode, use the following command:

```
tikv-ctl --host ip:port compact -d kv
```

14.6.1.3.9 Compact data of the whole TiKV cluster manually

Use the `compact-cluster` command to manually compact data of the whole TiKV cluster. The flags of this command have the same meanings and usage as those of the `compact` command. The only difference is as follows:

- For the `compact-cluster` command, use `--pd` to specify the address of the PD, so that `tikv-ctl` can locate all TiKV nodes in the cluster as the compact target.
- For the `compact` command, use `--data-dir` or `--host` to specify a single TiKV as the compact target.

14.6.1.3.10 Set a Region to tombstone

The `tombstone` command is usually used in circumstances where the sync-log is not enabled, and some data written in the Raft state machine is lost caused by power down.

In a TiKV instance, you can use this command to set the status of some Regions to tombstone. Then when you restart the instance, those Regions are skipped to avoid the restart failure caused by damaged Raft state machines of those Regions. Those Regions need to have enough healthy replicas in other TiKV instances to be able to continue the reads and writes through the Raft mechanism.

In general cases, you can remove the corresponding Peer of this Region using the `remove -peer` command:

```
pd-ctl operator add remove-peer <region_id> <store_id>
```

Then use the `tikv-ctl` tool to set a Region to tombstone on the corresponding TiKV instance to skip the health check for this Region at startup:

```
tikv-ctl --data-dir /path/to/tikv tombstone -p 127.0.0.1:2379 -r <region_id>
```

```
success!
```

However, in some cases, you cannot easily remove this Peer of this Region from PD, so you can specify the `--force` option in `tikv-ctl` to forcibly set the Peer to tombstone:

```
tikv-ctl --data-dir /path/to/tikv tombstone -p 127.0.0.1:2379 -r <region_id
    ↪ >,<region_id> --force
```

```
success!
```

Note:

- The `tombstone` command only supports the local mode.
- The argument of the `-p` option specifies the PD endpoints without the `http` prefix. Specifying the PD endpoints is to query whether PD can safely switch to Tombstone.

14.6.1.3.11 Send a `consistency-check` request to TiKV

Use the `consistency-check` command to execute a consistency check among replicas in the corresponding Raft of a specific Region. If the check fails, TiKV itself panics. If the TiKV instance specified by `--host` is not the Region leader, an error is reported.

```
$ tikv-ctl --host 127.0.0.1:20160 consistency-check -r 2
success!
$ tikv-ctl --host 127.0.0.1:20161 consistency-check -r 2
DebugClient::check_region_consistency: RpcFailure(RpcStatus { status:
    ↪ Unknown, details: Some("StringError(\"Leader is on store 1\")") })
```

Note:

- It is **NOT** recommended to use the `consistency-check` command, because it is incompatible with the garbage collection in TiDB and might mistakenly report an error.
- This command only supports the remote mode.
- Even if this command returns `success!`, you need to check whether TiKV panics. This is because this command is only a proposal that requests a consistency check for the leader, and you cannot know from the client whether the whole check process is successful or not.

14.6.1.3.12 Dump snapshot meta

This sub-command is used to parse a snapshot meta file at given path and print the result.

14.6.1.3.13 Print the Regions where the Raft state machine corrupts

To avoid checking the Regions while TiKV is started, you can use the `tombstone` command to set the Regions where the Raft state machine reports an error to Tombstone. Before running this command, use the `bad-regions` command to find out the Regions with errors, so as to combine multiple tools for automated processing.

```
$ tikv-ctl --data-dir /path/to/tikv bad-regions
all regions are healthy
```

If the command is successfully executed, it prints the above information. If the command fails, it prints the list of bad Regions. Currently, the errors that can be detected include the mismatches between `last index`, `commit index` and `apply index`, and the loss of Raft log. Other conditions like the damage of snapshot files still need further support.

14.6.1.3.14 View Region properties

- To view in local the properties of Region 2 on the TiKV instance that is deployed in `/path/to/tikv`:

```
$ tikv-ctl --data-dir /path/to/tikv/data region-properties -r 2
```

- To view online the properties of Region 2 on the TiKV instance that is running on `127.0.0.1:20160`:

```
$ tikv-ctl --host 127.0.0.1:20160 region-properties -r 2
```

14.6.1.3.15 Modify the TiKV configuration dynamically

You can use the `modify-tikv-config` command to dynamically modify the configuration arguments. Currently, the TiKV configuration items that can be dynamically modified and the detailed modification are consistent with modifying configuration using SQL statements. For details, see Modify TiKV configuration dynamically.

- `-n` is used to specify the full name of the configuration item. For the list of configuration items that can be modified dynamically, see Modify TiKV configuration dynamically.
- `-v` is used to specify the configuration value.

Set the size of `shared block cache`:

```
tikv-ctl --host ip:port modify-tikv-config -n storage.block-cache.capacity -
    ↪ v 10GB
```

```
success
```

When `shared block cache` is disabled, set `block cache size` for the `write` CF:

```
tikv-ctl --host ip:port modify-tikv-config -n rocksdb.writecf.block-cache-
    ↪ size -v 256MB
```

```
success
```

```
tikv-ctl --host ip:port modify-tikv-config -n raftdb.defaultcf.disable-auto-
    ↪ compactions -v true
```

```
success
```

```
tikv-ctl --host ip:port modify-tikv-config -n raftstore.sync-log -v false
```

```
success
```

When the compaction rate limit causes accumulated compaction pending bytes, disable the `rate-limiter-auto-tuned` mode or set a higher limit for the compaction flow:

```
tikv-ctl --host ip:port modify-tikv-config -n rocksdb.rate-limiter-auto-
    ↪ tuned -v false
```

```
success
```

```
tikv-ctl --host ip:port modify-tikv-config -n rocksdb.rate-bytes-per-sec -v
    ↪ "1GB"
```

```
success
```

14.6.1.3.16 Force Regions to recover services from failure of multiple replicas (deprecated)

Warning:

It is not recommended to use this feature. Instead, you can use Online Unsafe Recovery in `pd-ctl` which provides one-stop automatic recovery capabilities. Extra operations such as stopping services are not needed. For detailed introduction, see Online Unsafe Recovery.

You can use the `unsafe-recover remove-fail-stores` command to remove the failed machines from the peer list of Regions. Before running this command, you need to stop the service of the target TiKV store to release file locks.

The `-s` option accepts multiple `store_id` separated by comma and uses the `-r` flag to specify involved Regions. If you need to perform this operation on all Regions in a specific store, you can simply specify `--all-regions`.

Warning:

- If any misoperation is performed, it might be hard to recover the cluster. Be aware of the potential risks and avoid using this feature in a production environment.
- If the `--all-regions` option is used, you are expected to run this command on all the remaining stores connected to the cluster. You need to ensure that these healthy stores stop providing services before recovering the damaged stores. Otherwise, the inconsistent peer lists in Region replicas will cause errors when you run `split-region` or `remove-peer`. This further causes inconsistency between other metadata, and finally, the Regions will become unavailable.
- Once you have run `remove-fail-stores`, you cannot restart the removed nodes or add these nodes to the cluster. Otherwise, the metadata will be inconsistent, and finally, the Regions will be unavailable.

```
tikv-ctl --data-dir /path/to/tikv unsafe-recover remove-fail-stores -s 3 -r
    ↪ 1001,1002
```

```
success!
```

```
tikv-ctl --data-dir /path/to/tikv unsafe-recover remove-fail-stores -s 4,5
    ↪ --all-regions
```

Then, after you restart TiKV, the Regions can continue providing services with the remaining healthy replicas. This command is commonly used when multiple TiKV stores are damaged or deleted.

Note:

- You are expected to run this command for all stores where the specified Regions' peers are located.

- This command only supports the local mode. It prints `success!` when successfully run.

14.6.1.3.17 Recover from MVCC data corruption

Use the `recover-mvcc` command in circumstances where TiKV cannot run normally caused by MVCC data corruption. It cross-checks 3 CFs ("default", "write", "lock") to recover from various kinds of inconsistency.

- Use the `-r` option to specify involved Regions by `region_id`.
- Use the `-p` option to specify PD endpoints.

```
$ tikv-ctl --data-dir /path/to/tikv recover-mvcc -r 1001,1002 -p
    ↪ 127.0.0.1:2379
success!
```

Note:

- This command only supports the local mode. It prints `success!` when successfully run.
- The argument of the `-p` option specifies the PD endpoints without the `http` prefix. Specifying the PD endpoints is to query whether the specified `region_id` is validated or not.
- You need to run this command for all stores where specified Regions' peers are located.

14.6.1.3.18 Ldb Command

The `ldb` command line tool offers multiple data access and database administration commands. Some examples are listed below. For more information, refer to the help message displayed when running `tikv-ctl ldb` or check the documents from RocksDB.

Examples of data access sequence:

To dump an existing RocksDB in HEX:

```
$ tikv-ctl ldb --hex --db=/tmp/db dump
```

To dump the manifest of an existing RocksDB:

```
$ tikv-ctl ldb --hex manifest_dump --path=/tmp/db/MANIFEST-000001
```

You can specify the column family that your query is against using the `--column_family =<string>` command line.

`--try_load_options` loads the database options file to open the database. It is recommended to always keep this option on when the database is running. If you open the database with default options, the LSM-tree might be messed up, which cannot be recovered automatically.

14.6.1.3.19 Dump encryption metadata

Use the `encryption-meta` subcommand to dump encryption metadata. The subcommand can dump two types of metadata: encryption info for data files, and the list of data encryption keys used.

To dump encryption info for data files, use the `encryption-meta dump-file` subcommand. You need to create a TiKV config file to specify `data-dir` for the TiKV deployment:

```
### conf.toml
[storage]
data-dir = "/path/to/tikv/data"
```

The `--path` option can be used to specify an absolute or relative path to the data file of interest. The command might give empty output if the data file is not encrypted. If `--path` is not provided, encryption info for all data files will be printed.

```
$ tikv-ctl --config=./conf.toml encryption-meta dump-file --path=/path/to/
    ↪ tikv/data/db/CURRENT
/path/to/tikv/data/db/CURRENT: key_id: 9291156302549018620 iv:
    ↪ E3C2FDBF63FC03BFC28F265D7E78283F method: Aes128Ctr
```

To dump data encryption keys, use the `encryption-meta dump-key` subcommand. In additional to `data-dir`, you also need to specify the current master key used in the config file. For how to config master key, refer to Encryption-At-Rest. Also with this command, the `security.encryption.previous-master-key` config will be ignored, and the master key rotation will not be triggered.

```
### conf.toml
[storage]
data-dir = "/path/to/tikv/data"

[security.encryption.master-key]
type = "kms"
key-id = "0987dcba-09fe-87dc-65ba-ab0987654321"
region = "us-west-2"
```

Note if the master key is a AWS KMS key, `tikv-ctl` needs to have access to the KMS key. Access to a AWS KMS key can be granted to `tikv-ctl` via environment variable, AWS default config file, or IAM role, whichever is suitable. Refer to AWS document for usage.

The `--ids` option can be used to specified a list of comma-separated data encryption key ids to print. If `--ids` is not provided, all data encryption keys will be printed, along with current key id, which is the id of the latest active data encryption key.

When using the command, you will see a prompt warning that the action will expose sensitive information. Type "I consent" to continue.

```
$ ./tikv-ctl --config=./conf.toml encryption-meta dump-key
This action will expose encryption key(s) as plaintext. Do not output the
    ↪ result in file on disk.
Type "I consent" to continue, anything else to exit: I consent
current key id: 9291156302549018620
9291156302549018620: key: 8B6B6B8F83D36BE2467ED55D72AE808B method: Aes128Ctr
    ↪  creation_time: 1592938357
```

```
$ ./tikv-ctl --config=./conf.toml encryption-meta dump-key --ids
    ↪ =9291156302549018620
This action will expose encryption key(s) as plaintext. Do not output the
    ↪ result in file on disk.
Type "I consent" to continue, anything else to exit: I consent
9291156302549018620: key: 8B6B6B8F83D36BE2467ED55D72AE808B method: Aes128Ctr
    ↪  creation_time: 1592938357
```

> **Note:**
>
> The command will expose data encryption keys as plaintext. In production, DO NOT redirect the output to a file. Even deleting the output file afterward may not cleanly wipe out the content from disk.

14.6.1.3.20 Print information related to damaged SST files

Damaged SST files in TiKV might cause TiKV processes to panic. Before TiDB v6.1.0, these files cause TiKV to panic immediately. Since TiDB v6.1.0, TiKV processes panic 1 hour after SST files are damaged.

To clean up the damaged SST files, you can run the `bad-ssts` command in TiKV Control to show the needed information. The following is an example command and output.

> **Note:**
>
> Before running this command, stop the running TiKV instance.

```
tikv-ctl --data-dir </path/to/tikv> bad-ssts --pd <endpoint>
```

```
--------------------------------------------------------
corruption info:
data/tikv-21107/db/000014.sst: Corruption: Bad table magic number: expected
    ↪ 9863518390377041911, found 759105309091689679 in data/tikv-21107/db
    ↪ /000014.sst

sst meta:
14:552997[1 .. 5520]['0101' seq:1, type:1 .. '7
    ↪ A7480000000000000FF0F5F728000000000FF0002160000000000FAFA13AB33020BFFFA
    ↪ ' seq:2032, type:1] at level 0 for Column family "default" (ID 0)
it isn't easy to handle local data, start key:0101

overlap region:
RegionInfo { region: id: 4 end_key: 7480000000000000FF0500000000000000F8
    ↪ region_epoch { conf_ver: 1 version: 2 } peers { id: 5 store_id: 1 },
    ↪ leader: Some(id: 5 store_id: 1) }

refer operations:
tikv-ctl ldb --db=/path/to/tikv/db unsafe_remove_sst_file 000014
tikv-ctl --data-dir=/path/to/tikv tombstone -r 4 --pd <endpoint>
--------------------------------------------------------
corruption analysis has completed
```

From the output above, you can see that the information of the damaged SST file is printed first and then the meta-information is printed.

- In the `sst meta` part, `14` means the SST file number; `552997` means the file size, followed by the smallest and largest sequence numbers and other meta-information.
- The `overlap region` part shows the information of the Region involved. This information is obtained through the PD server.
- The `suggested operations` part provides you suggestion to clean up the damaged SST file. You can take the suggestion to clean up files and restart the TiKV instance.

14.6.2 PD Control User Guide

As a command line tool of PD, PD Control obtains the state information of the cluster and tunes the cluster.

14.6.2.1 Install PD Control

Note:

It is recommended that the version of the Control tool you use is consistent with the version of the cluster.

14.6.2.1.1 Use TiUP command

To use PD Control, execute the `tiup ctl:<cluster-version> pd -u http://<pd_ip>:<pd_port> [-i]` command.

14.6.2.1.2 Download the installation package

To obtain `pd-ctl` of the latest version, download the TiDB server installation package. `pd-ctl` is included in the `ctl-{version}-linux-amd64.tar.gz` package.

Installation package	OS	Architecture	SHA256 checksum
`https://download.pingcap.org/tidb-community-server-{version}-linux-amd64.tar.gz` (pd-ctl)	Linux	amd64	`https://download.pingcap.org/tidb-community-server-{version}-linux-amd64.sha256`

Note:

`{version}` indicates the version number of TiDB. For example, if `{version}` is `v6.1.7`, the package download link is `https://download.pingcap.org/tidb-community-server-v6.1.7-linux-amd64.tar.gz`.

14.6.2.1.3 Compile from source code

1. Go Version 1.19 or later because the Go modules are used.
2. In the root directory of the PD project, use the `make` or `make pd-ctl` command to compile and generate `bin/pd-ctl`.

14.6.2.2 Usage

Single-command mode:

```
tiup ctl:<cluster-version> pd store -u http://127.0.0.1:2379
```

Interactive mode:

```
tiup ctl:<cluster-version> pd -i -u http://127.0.0.1:2379
```

Use environment variables:

```
export PD_ADDR=http://127.0.0.1:2379
tiup ctl:<cluster-version> pd
```

Use TLS to encrypt:

```
tiup ctl:<cluster-version> pd -u https://127.0.0.1:2379 --cacert="path/to/ca
    ↪ " --cert="path/to/cert" --key="path/to/key"
```

14.6.2.3 Command line flags

14.6.2.3.1 `--cacert`

- Specifies the path to the certificate file of the trusted CA in PEM format
- Default: “”

14.6.2.3.2 `--cert`

- Specifies the path to the certificate of SSL in PEM format
- Default: “”

14.6.2.3.3 `--detach` / `-d`

- Uses the single command line mode (not entering readline)
- Default: true

14.6.2.3.4 `--help` / `-h`

- Outputs the help information
- Default: false

14.6.2.3.5 `--interact` / `-i`

- Uses the interactive mode (entering readline)
- Default: false

14.6.2.3.6 `--key`

- Specifies the path to the certificate key file of SSL in PEM format, which is the private key of the certificate specified by `--cert`
- Default: “”

14.6.2.3.7 `--pd` / `-u`

- Specifies the PD address
- Default address: `http://127.0.0.1:2379`
- Environment variable: `PD_ADDR`

14.6.2.3.8 `--version` / `-V`

- Prints the version information and exit
- Default: false

14.6.2.4 Command

14.6.2.4.1 `cluster`

Use this command to view the basic information of the cluster.

Usage:

```
>> cluster                                  // To show the cluster information
{
  "id": 6493707687106161130,
  "max_peer_count": 3
}
```

14.6.2.4.2 `config [show | set <option> <value> | placement-rules]`

Use this command to view or modify the configuration information.

Usage:

```
>> config show                              // Display the config information
   ↪ of the scheduling
{
  "replication": {
    "enable-placement-rules": "true",
    "isolation-level": "",
    "location-labels": "",
    "max-replicas": 3,
```

```
    "strictly-match-label": "false"
  },
  "schedule": {
    "enable-cross-table-merge": "true",
    "high-space-ratio": 0.7,
    "hot-region-cache-hits-threshold": 3,
    "hot-region-schedule-limit": 4,
    "leader-schedule-limit": 4,
    "leader-schedule-policy": "count",
    "low-space-ratio": 0.8,
    "max-merge-region-keys": 200000,
    "max-merge-region-size": 20,
    "max-pending-peer-count": 64,
    "max-snapshot-count": 64,
    "max-store-down-time": "30m0s",
    "merge-schedule-limit": 8,
    "patrol-region-interval": "10ms",
    "region-schedule-limit": 2048,
    "region-score-formula-version": "v2",
    "replica-schedule-limit": 64,
    "scheduler-max-waiting-operator": 5,
    "split-merge-interval": "1h0m0s",
    "tolerant-size-ratio": 0
  }
}
>> config show all                          // Display all config information
>> config show replication                  // Display the config information
   ↪ of replication
{
  "max-replicas": 3,
  "location-labels": "",
  "isolation-level": "",
  "strictly-match-label": "false",
  "enable-placement-rules": "true"
}

>> config show cluster-version              // Display the current version of
   ↪ the cluster, which is the current minimum version of TiKV nodes in
   ↪ the cluster and does not correspond to the binary version.
"5.2.2"
```

- `max-snapshot-count` controls the maximum number of snapshots that a single store receives or sends out at the same time. The scheduler is restricted by this configuration to avoid taking up normal application resources. When you need to improve the speed

of adding replicas or balancing, increase this value.

```
config set max-snapshot-count 64 // Set the maximum number of snapshots
    ↪  to 64
```

- `max-pending-peer-count` controls the maximum number of pending peers in a single store. The scheduler is restricted by this configuration to avoid producing a large number of Regions without the latest log in some nodes. When you need to improve the speed of adding replicas or balancing, increase this value. Setting it to 0 indicates no limit.

```
config set max-pending-peer-count 64 // Set the maximum number of
    ↪ pending peers to 64
```

- `max-merge-region-size` controls the upper limit on the size of Region Merge (the unit is MiB). When `regionSize` exceeds the specified value, PD does not merge it with the adjacent Region. Setting it to 0 indicates disabling Region Merge.

```
config set max-merge-region-size 16 // Set the upper limit on the size
    ↪ of Region Merge to 16 MiB
```

- `max-merge-region-keys` controls the upper limit on the key count of Region Merge. When `regionKeyCount` exceeds the specified value, PD does not merge it with the adjacent Region.

```
config set max-merge-region-keys 50000 // Set the the upper limit on
    ↪ keyCount to 50000
```

- `split-merge-interval` controls the interval between the `split` and `merge` operations on a same Region. This means the newly split Region won't be merged within a period of time.

```
config set split-merge-interval 24h // Set the interval between `split`
    ↪  and `merge` to one day
```

- `enable-one-way-merge` controls whether PD only allows a Region to merge with the next Region. When you set it to `false`, PD allows a Region to merge with the adjacent two Regions.

```
config set enable-one-way-merge true // Enables one-way merging.
```

- `enable-cross-table-merge` is used to enable the merging of cross-table Regions. When you set it to `false`, PD does not merge the Regions from different tables. This option only works when key type is "table".

```
config set enable-cross-table-merge true // Enable cross table merge.
```

- `key-type` specifies the key encoding type used for the cluster. The supported options are ["table", "raw", "txn"], and the default value is "table".

    - If no TiDB instance exists in the cluster, `key-type` will be "raw" or "txn", and PD is allowed to merge Regions across tables regardless of the `enable-cross-table-merge` setting.
    - If any TiDB instance exists in the cluster, `key-type` should be "table". Whether PD can merge Regions across tables is determined by `enable-cross-table-merge`. If `key-type` is "raw", placement rules do not work.

```
config set key-type raw // Enable cross table merge.
```

- `region-score-formula-version` controls the version of the Region score formula. The value options are `v1` and `v2`. The version 2 of the formula helps to reduce redundant balance Region scheduling in some scenarios, such as taking TiKV nodes online or offline.

```
config set region-score-formula-version v2
```

- `patrol-region-interval` controls the execution frequency that `replicaChecker` checks the health status of Regions. A shorter interval indicates a higher execution frequency. Generally, you do not need to adjust it.

```
config set patrol-region-interval 10ms // Set the execution frequency
    ↪ of replicaChecker to 10ms
```

- `max-store-down-time` controls the time that PD decides the disconnected store cannot be restored if exceeded. If PD does not receive heartbeats from a store within the specified period of time, PD adds replicas in other nodes.

```
config set max-store-down-time 30m // Set the time within which PD
    ↪ receives no heartbeats and after which PD starts to add replicas
    ↪ to 30 minutes
```

- `max-store-preparing-time` controls the maximum waiting time for the store to go online. During the online stage of a store, PD can query the online progress of the store. When the specified time is exceeded, PD assumes that the store has been online and cannot query the online progress of the store again. But this does not prevent Regions from transferring to the new online store. In most scenarios, you do not need to adjust this parameter.

  The following command specifies that the maximum waiting time for the store to go online is 4 hours.

```
config set max-store-preparing-time 4h
```

- `leader-schedule-limit` controls the number of tasks scheduling the leader at the same time. This value affects the speed of leader balance. A larger value means a higher speed and setting the value to 0 closes the scheduling. Usually the leader scheduling has a small load, and you can increase the value in need.

```
config set leader-schedule-limit 4    // 4 tasks of leader scheduling
    ↪ at the same time at most
```

- `region-schedule-limit` controls the number of tasks of scheduling Regions at the same time. This value avoids too many Region balance operators being created. The default value is `2048` which is enough for all sizes of clusters, and setting the value to `0` closes the scheduling. Usually, the Region scheduling speed is limited by `store-limit`, but it is recommended that you do not customize this value unless you know exactly what you are doing.

```
config set region-schedule-limit 2    // 2 tasks of Region scheduling
    ↪ at the same time at most
```

- `replica-schedule-limit` controls the number of tasks scheduling the replica at the same time. This value affects the scheduling speed when the node is down or removed. A larger value means a higher speed and setting the value to 0 closes the scheduling. Usually the replica scheduling has a large load, so do not set a too large value. Note that this configuration item is usually kept at the default value. If you want to change the value, you need to try a few values to see which one works best according to the real situation.

```
config set replica-schedule-limit 4   // 4 tasks of replica scheduling
    ↪ at the same time at most
```

- `merge-schedule-limit` controls the number of Region Merge scheduling tasks. Setting the value to 0 closes Region Merge. Usually the Merge scheduling has a large load, so do not set a too large value. Note that this configuration item is usually kept at the default value. If you want to change the value, you need to try a few values to see which one works best according to the real situation.

```
config set merge-schedule-limit 16  // 16 tasks of Merge scheduling at
    ↪ the same time at most
```

- `hot-region-schedule-limit` controls the hot Region scheduling tasks that are running at the same time. Setting its value to `0` means disabling the scheduling. It is not recommended to set a too large value. Otherwise, it might affect the system performance. Note that this configuration item is usually kept at the default value. If you want to change the value, you need to try a few values to see which one works best according to the real situation.

```
config set hot-region-schedule-limit 4 // 4 tasks of hot Region
    ↪ scheduling at the same time at most
```

- `hot-region-cache-hits-threshold` is used to set the number of minutes required to identify a hot Region. PD can participate in the hotspot scheduling only after the Region is in the hotspot state for more than this number of minutes.

- `tolerant-size-ratio` controls the size of the balance buffer area. When the score difference between the leader or Region of the two stores is less than specified multiple times of the Region size, it is considered in balance by PD.

```
config set tolerant-size-ratio 20   // Set the size of the buffer area
    ↪ to about 20 times of the average Region Size
```

- `low-space-ratio` controls the threshold value that is considered as insufficient store space. When the ratio of the space occupied by the node exceeds the specified value, PD tries to avoid migrating data to the corresponding node as much as possible. At the same time, PD mainly schedules the remaining space to avoid using up the disk space of the corresponding node.

```
config set low-space-ratio 0.9         // Set the threshold value of
    ↪ insufficient space to 0.9
```

- `high-space-ratio` controls the threshold value that is considered as sufficient store space. This configuration takes effect only when `region-score-formula-version` is set to `v1`. When the ratio of the space occupied by the node is less than the specified value, PD ignores the remaining space and mainly schedules the actual data volume.

```
config set high-space-ratio 0.5        // Set the threshold value of
    ↪ sufficient space to 0.5
```

- `cluster-version` is the version of the cluster, which is used to enable or disable some features and to deal with the compatibility issues. By default, it is the minimum version of all normally running TiKV nodes in the cluster. You can set it manually only when you need to roll it back to an earlier version.

```
config set cluster-version 1.0.8        // Set the version of the
    ↪ cluster to 1.0.8
```

- `replication-mode` controls the replication mode of Regions in the dual data center scenario. See Enable the DR Auto-Sync mode for details.

- `leader-schedule-policy` is used to select the scheduling strategy for the leader. You can schedule the leader according to `size` or `count`.

- `scheduler-max-waiting-operator` is used to control the number of waiting operators in each scheduler.

- `enable-remove-down-replica` is used to enable the feature of automatically deleting DownReplica. When you set it to `false`, PD does not automatically clean up the downtime replicas.

- `enable-replace-offline-replica` is used to enable the feature of migrating OfflineReplica. When you set it to `false`, PD does not migrate the offline replicas.
- `enable-make-up-replica` is used to enable the feature of making up replicas. When you set it to `false`, PD does not add replicas for Regions without sufficient replicas.
- `enable-remove-extra-replica` is used to enable the feature of removing extra replicas. When you set it to `false`, PD does not remove extra replicas for Regions with redundant replicas.
- `enable-location-replacement` is used to enable the isolation level checking. When you set it to `false`, PD does not increase the isolation level of a Region replica through scheduling.
- `enable-debug-metrics` is used to enable the metrics for debugging. When you set it to `true`, PD enables some metrics such as `balance-tolerant-size`.
- `enable-placement-rules` is used to enable placement rules, which is enabled by default in v5.0 and later versions.
- `store-limit-mode` is used to control the mode of limiting the store speed. The optional modes are `auto` and `manual`. In `auto` mode, the stores are automatically balanced according to the load (experimental).
- PD rounds the lowest digits of the flow number, which reduces the update of statistics caused by the changes of the Region flow information. This configuration item is used to specify the number of lowest digits to round for the Region flow information. For example, the flow `100512` will be rounded to `101000` because the default value is `3`. This configuration replaces `trace-region-flow`.
- For example, set the value of `flow-round-by-digit` to 4:

```
config set flow-round-by-digit 4
```

`config placement-rules [disable | enable | load | save | show | rule-group]`

For the usage of `config placement-rules [disable | enable | load | save | show | rule-group]`, see Configure placement rules.

14.6.2.4.3 health

Use this command to view the health information of the cluster.

Usage:

```
>> health                              // Display the health information
[
  {
    "name": "pd",
```

```
    "member_id": 13195394291058371180,
    "client_urls": [
      "http://127.0.0.1:2379"
      ......
    ],
    "health": true
  }
  ......
]
```

14.6.2.4.4 hot [read | write | store| history <start_time> <end_time> [<key> <value>]]

Use this command to view the hot spot information of the cluster.

Usage:

```
>> hot read                                // Display hot spot for the read
   ↪ operation
>> hot write                               // Display hot spot for the write
   ↪ operation
>> hot store                               // Display hot spot for all the read
   ↪ and write operations
>> hot history 1629294000000 1631980800000 // Display history hot spot for
   ↪ the specified period (milliseconds). 1629294000000 is the start time
   ↪ and 1631980800000 is the end time.
{
  "history_hot_region": [
    {
      "update_time": 1630864801948,
      "region_id": 103,
      "peer_id": 1369002,
      "store_id": 3,
      "is_leader": true,
      "is_learner": false,
      "hot_region_type": "read",
      "hot_degree": 152,
      "flow_bytes": 0,
      "key_rate": 0,
      "query_rate": 305,
      "start_key": "7480000000000000FF5300000000000000F8",
      "end_key": "7480000000000000FF5600000000000000F8"
    },
    ...
  ]
}
```

```
>> hot history 1629294000000 1631980800000 hot_region_type read region_id
   ↪ 1,2,3 store_id 1,2,3 peer_id 1,2,3 is_leader true is_learner true //
   ↪ Display history hotspot for the specified period with more conditions
{
  "history_hot_region": [
    {
      "update_time": 1630864801948,
      "region_id": 103,
      "peer_id": 1369002,
      "store_id": 3,
      "is_leader": true,
      "is_learner": false,
      "hot_region_type": "read",
      "hot_degree": 152,
      "flow_bytes": 0,
      "key_rate": 0,
      "query_rate": 305,
      "start_key": "7480000000000000FF5300000000000000F8",
      "end_key": "7480000000000000FF5600000000000000F8"
    },
    ...
  ]
}
```

14.6.2.4.5 label [store <name> <value>]

Use this command to view the label information of the cluster.

Usage:

```
>> label                              // Display all labels
>> label store zone cn                // Display all stores including the "zone
   ↪ ":"cn" label
```

14.6.2.4.6 member [delete | leader_priority | leader [show | resign | transfer <member_name>]]

Use this command to view the PD members, remove a specified member, or configure the priority of leader.

Usage:

```
>> member                             // Display the information of all members
{
  "header": {......},
  "members": [......],
```

```
  "leader": {......},
  "etcd_leader": {......},
}
>> member delete name pd2          // Delete "pd2"
Success!
>> member delete id 1319539429105371180 // Delete a node using id
Success!
>> member leader show              // Display the leader information
{
  "name": "pd",
  "member_id": 13155432540099656863,
  "peer_urls": [......],
  "client_urls": [......]
}
>> member leader resign // Move leader away from the current member
......
>> member leader transfer pd3 // Migrate leader to a specified member
......
```

14.6.2.4.7 operator [check | show | add | remove]

Use this command to view and control the scheduling operation.

Usage:

```
>> operator show                                // Display all operators
>> operator show admin                          // Display all admin
   ↪ operators
>> operator show leader                         // Display all leader
   ↪ operators
>> operator show region                         // Display all Region
   ↪ operators
>> operator add add-peer 1 2                    // Add a replica of Region
   ↪ 1 on store 2
>> operator add add-learner 1 2                 // Add a learner replica of
   ↪  Region 1 on store 2
>> operator add remove-peer 1 2                 // Remove a replica of
   ↪ Region 1 on store 2
>> operator add transfer-leader 1 2             // Schedule the leader of
   ↪ Region 1 to store 2
>> operator add transfer-region 1 2 3 4         // Schedule Region 1 to
   ↪ stores 2,3,4
>> operator add transfer-peer 1 2 3             // Schedule the replica of
   ↪ Region 1 on store 2 to store 3
>> operator add merge-region 1 2                // Merge Region 1 with
   ↪ Region 2
```

```
>> operator add split-region 1 --policy=approximate // Split Region 1 into
   ↪ two Regions in halves, based on approximately estimated value
>> operator add split-region 1 --policy=scan     // Split Region 1 into two
   ↪ Regions in halves, based on accurate scan value
>> operator remove 1                             // Remove the scheduling
   ↪ operation of Region 1
>> operator check 1                              // Check the status of the
   ↪ operators related to Region 1
```

The splitting of Regions starts from the position as close as possible to the middle. You can locate this position using two strategies, namely "scan" and "approximate". The difference between them is that the former determines the middle key by scanning the Region, and the latter obtains the approximate position by checking the statistics recorded in the SST file. Generally, the former is more accurate, while the latter consumes less I/O and can be completed faster.

14.6.2.4.8 ping

Use this command to view the time that ping PD takes.

Usage:

```
>> ping
time: 43.12698ms
```

14.6.2.4.9 region <region_id> [--jq="<query string>"]

Use this command to view the Region information. For a jq formatted output, see jq-formatted-json-output-usage.

Usage:

```
>> region                           //  Display the information of all
   ↪ Regions
{
  "count": 1,
  "regions": [......]
}

>> region 2                         // Display the information of the Region
   ↪ with the ID of 2
{
  "id": 2,
  "start_key": "7480000000000000FF1D00000000000000F8",
  "end_key": "7480000000000000FF1F00000000000000F8",
  "epoch": {
    "conf_ver": 1,
```

```
    "version": 15
  },
  "peers": [
    {
      "id": 40,
      "store_id": 3
    }
  ],
  "leader": {
    "id": 40,
    "store_id": 3
  },
  "written_bytes": 0,
  "read_bytes": 0,
  "written_keys": 0,
  "read_keys": 0,
  "approximate_size": 1,
  "approximate_keys": 0
}
```

14.6.2.4.10 region key [--format=raw|encode|hex] <key>

Use this command to query the Region that a specific key resides in. It supports the raw, encoding, and hex formats. And you need to use single quotes around the key when it is in the encoding format.

Hex format usage (default):

```
>> region key 7480000000000000FF1300000000000000F8
{
  "region": {
    "id": 2,
    ......
  }
}
```

Raw format usage:

```
>> region key --format=raw abc
{
  "region": {
    "id": 2,
    ......
  }
}
```

Encoding format usage:

```
>> region key --format=encode 't\200\000\000\000\000\000\000\377\035_r
   ↪ \200\000\000\000\000\377\017U\320\000\000\000\000\000\000\372'
{
  "region": {
    "id": 2,
    ......
  }
}
```

14.6.2.4.11 `region scan`

Use this command to get all Regions.

Usage:

```
>> region scan
{
  "count": 20,
  "regions": [......],
}
```

14.6.2.4.12 `region sibling <region_id>`

Use this command to check the adjacent Regions of a specific Region.

Usage:

```
>> region sibling 2
{
  "count": 2,
  "regions": [......],
}
```

14.6.2.4.13 `region keys [--format=raw|encode|hex] <start_key> <end_key> <limit>`

Use this command to query all Regions in a given range `[startkey, endkey)`. Ranges without `endKeys` are supported.

The `limit` parameter limits the number of keys. The default value of `limit` is `16`, and the value of `-1` means unlimited keys.

Usage:

```
>> region keys --format=raw a    // Display all Regions that start from the
   ↪  key a with a default limit count of 16
```

```
{
  "count": 16,
  "regions": [......],
}

>> region keys --format=raw a z // Display all Regions in the range [a, z)
   ↪ with a default limit count of 16
{
  "count": 16,
  "regions": [......],
}

>> region keys --format=raw a z -1 // Display all Regions in the range [a,
   ↪ z) without a limit count
{
  "count": ...,
  "regions": [......],
}

>> region keys --format=raw a "" 20 // Display all Regions that start from
   ↪ the key a with a limit count of 20
{
  "count": 20,
  "regions": [......],
}
```

14.6.2.4.14 region store <store_id>

Use this command to list all Regions of a specific store.

Usage:

```
>> region store 2
{
  "count": 10,
  "regions": [......],
}
```

14.6.2.4.15 region topread [limit]

Use this command to list Regions with top read flow. The default value of the limit is 16.

Usage:

```
>> region topread
```

```
{
  "count": 16,
  "regions": [......],
}
```

14.6.2.4.16 region topwrite [limit]

Use this command to list Regions with top write flow. The default value of the limit is 16.

Usage:

```
>> region topwrite
{
  "count": 16,
  "regions": [......],
}
```

14.6.2.4.17 region topconfver [limit]

Use this command to list Regions with top conf version. The default value of the limit is 16.

Usage:

```
>> region topconfver
{
  "count": 16,
  "regions": [......],
}
```

14.6.2.4.18 region topversion [limit]

Use this command to list Regions with top version. The default value of the limit is 16.

Usage:

```
>> region topversion
{
  "count": 16,
  "regions": [......],
}
```

14.6.2.4.19 region topsize [limit]

Use this command to list Regions with top approximate size. The default value of the limit is 16.

Usage:

```
>> region topsize
{
  "count": 16,
  "regions": [......],
}
```

14.6.2.4.20 region check [miss-peer | extra-peer | down-peer | pending-peer | offline-peer | empty-region | hist-size | hist-keys]

Use this command to check the Regions in abnormal conditions.

Description of various types:

- miss-peer: the Region without enough replicas
- extra-peer: the Region with extra replicas
- down-peer: the Region in which some replicas are Down
- pending-peer: the Region in which some replicas are Pending

Usage:

```
>> region check miss-peer
{
  "count": 2,
  "regions": [......],
}
```

14.6.2.4.21 scheduler [show | add | remove | pause | resume | config]

Use this command to view and control the scheduling policy.

Usage:

```
>> scheduler show                                  // Display all created
   ↪ schedulers
>> scheduler add grant-leader-scheduler 1          // Schedule all the
   ↪ leaders of the Regions on store 1 to store 1
>> scheduler add evict-leader-scheduler 1          // Move all the Region
   ↪ leaders on store 1 out
>> scheduler config evict-leader-scheduler         // Display the stores in
   ↪  which the scheduler is located since v4.0.0
```

```
>> scheduler config evict-leader-scheduler add-store 2 // Add leader
   ↪ eviction scheduling for store 2
>> scheduler config evict-leader-scheduler delete-store 2 // Remove leader
   ↪ eviction scheduling for store 2
>> scheduler add evict-slow-store-scheduler // When there is one and only
   ↪ one slow store, evict all Region leaders of that store
>> scheduler remove grant-leader-scheduler-1 // Remove the corresponding
   ↪ scheduler, and `-1` corresponds to the store ID
>> scheduler pause balance-region-scheduler 10 // Pause the balance-region
   ↪ scheduler for 10 seconds
>> scheduler pause all 10                    // Pause all schedulers for 10
   ↪ seconds
>> scheduler resume balance-region-scheduler // Continue to run the balance-
   ↪ region scheduler
>> scheduler resume all                      // Continue to run all
   ↪ schedulers
>> scheduler config balance-hot-region-scheduler // Display the
   ↪ configuration of the balance-hot-region scheduler
```

14.6.2.4.22 `scheduler config balance-leader-scheduler`

Use this command to view and control the `balance-leader-scheduler` policy.

Since TiDB v6.0.0, PD introduces the `Batch` parameter for `balance-leader-scheduler` to control the speed at which the balance-leader processes tasks. To use this parameter, you can modify the `balance-leader batch` configuration item using pd-ctl.

Before v6.0.0, PD does not have this configuration item, which means `balance-leader batch=1`. In v6.0.0 or later versions, the default value of `balance-leader batch` is `4`. To set this configuration item to a value greater than `4`, you need to set a greater value for `scheduler-max-waiting-operator` (whose default value is `5`) at the same time. You can get the expected acceleration effect only after modifying both configuration items.

```
scheduler config balance-leader-scheduler set batch 3 // Set the size of
   ↪ the operator that the balance-leader scheduler can execute in a batch
   ↪  to 3
```

`scheduler config balance-hot-region-scheduler`

Use this command to view and control the `balance-hot-region-scheduler` policy.

Usage:

```
>> scheduler config balance-hot-region-scheduler // Display all
   ↪ configuration of the balance-hot-region scheduler
{
  "min-hot-byte-rate": 100,
  "min-hot-key-rate": 10,
```

```
  "min-hot-query-rate": 10,
  "max-zombie-rounds": 3,
  "max-peer-number": 1000,
  "byte-rate-rank-step-ratio": 0.05,
  "key-rate-rank-step-ratio": 0.05,
  "query-rate-rank-step-ratio": 0.05,
  "count-rank-step-ratio": 0.01,
  "great-dec-ratio": 0.95,
  "minor-dec-ratio": 0.99,
  "src-tolerance-ratio": 1.05,
  "dst-tolerance-ratio": 1.05,
  "read-priorities": [
    "query",
    "byte"
  ],
  "write-leader-priorities": [
    "key",
    "byte"
  ],
  "write-peer-priorities": [
    "byte",
    "key"
  ],
  "strict-picking-store": "true",
  "enable-for-tiflash": "true"
}
```


- `min-hot-byte-rate` means the smallest number of bytes to be counted, which is usually 100.

```
scheduler config balance-hot-region-scheduler set min-hot-byte-rate 100
```

- `min-hot-key-rate` means the smallest number of keys to be counted, which is usually 10.

```
scheduler config balance-hot-region-scheduler set min-hot-key-rate 10
```

- `min-hot-query-rate` means the smallest number of queries to be counted, which is usually 10.

```
scheduler config balance-hot-region-scheduler set min-hot-query-rate 10
```

- `max-zombie-rounds` means the maximum number of heartbeats with which an operator can be considered as the pending influence. If you set it to a larger value, more operators might be included in the pending influence. Usually, you do not need to

adjust its value. Pending influence refers to the operator influence that is generated during scheduling but still has an effect.

```
scheduler config balance-hot-region-scheduler set max-zombie-rounds 3
```

- `max-peer-number` means the maximum number of peers to be solved, which prevents the scheduler from being too slow.

```
scheduler config balance-hot-region-scheduler set max-peer-number 1000
```

- `byte-rate-rank-step-ratio`, `key-rate-rank-step-ratio`, `query-rate-rank-step-ratio`, and `count-rank-step-ratio` respectively mean the step ranks of byte, key, query, and count. The rank-step-ratio decides the step when the rank is calculated. `great-dec-ratio` and `minor-dec-ratio` are used to determine the `dec` rank. Usually, you do not need to modify these items.

```
scheduler config balance-hot-region-scheduler set byte-rate-rank-step-
    ↪ ratio 0.05
```

- `src-tolerance-ratio` and `dst-tolerance-ratio` are configuration items for the expectation scheduler. The smaller the `tolerance-ratio`, the easier it is for scheduling. When redundant scheduling occurs, you can appropriately increase this value.

```
scheduler config balance-hot-region-scheduler set src-tolerance-ratio
    ↪ 1.1
```

- `read-priorities`, `write-leader-priorities`, and `write-peer-priorities` control which dimension the scheduler prioritizes for hot Region scheduling. Two dimensions are supported for configuration.

    - `read-priorities` and `write-leader-priorities` control which dimensions the scheduler prioritizes for scheduling hot Regions of the read and write-leader types. The dimension options are `query`, `byte`, and `key`.
    - `write-peer-priorities` controls which dimensions the scheduler prioritizes for scheduling hot Regions of the write-peer type. The dimension options are `byte` and `key`.

> **Note:**
>
> If a cluster component is earlier than v5.2, the configuration of `query` dimension does not take effect. If some components are upgraded to v5.2 or later, the `byte` and `key` dimensions still by default have the priority for hot Region scheduling. After all components of the cluster are upgraded to v5.2 or later, such a configuration still takes effect for compatibility. You can view the real-time configuration using the `pd-ctl` command. Usually, you do not need to modify these configurations.

```
scheduler config balance-hot-region-scheduler set read-priorities query
    ↪ ,byte
```

- `strict-picking-store` controls the search space of hot Region scheduling. Usually, it is enabled. When it is enabled, hot Region scheduling ensures hotspot balance on the two configured dimensions. When it is disabled, hot Region scheduling only ensures the balance on the dimension with the first priority, which might reduce balance on other dimensions. Usually, you do not need to modify this configuration.

```
scheduler config balance-hot-region-scheduler set strict-picking-store
    ↪ true
```

- `enable-for-tiflash` controls whether hot Region scheduling takes effect for TiFlash instances. Usually, it is enabled. When it is disabled, the hot Region scheduling between TiFlash instances is not performed.

```
scheduler config balance-hot-region-scheduler set enable-for-tiflash
    ↪ true
```

14.6.2.4.23 `scheduler config evict-leader-scheduler`

Use this command to view and manage the configuration of the `evict-leader-scheduler`.

- When an `evict-leader-scheduler` already exists, use the `add-store` subcommand to add leader eviction scheduling for the specified store:

```
scheduler config evict-leader-scheduler add-store 2 // Add leader
    ↪ eviction scheduling for store 2
```

- When an `evict-leader-scheduler` already exists, use the `delete-store` subcommand to remove leader eviction scheduling for the specified store:

```
scheduler config evict-leader-scheduler delete-store 2 // Remove leader
    ↪  eviction scheduling for store 2
```

  If all store configurations of an `evict-leader-scheduler` are removed, the scheduler itself is automatically removed.

14.6.2.4.24 `service-gc-safepoint`

Use this command to query the current GC safepoint and service GC safepoint. The output is as follows:

```
{
  "service_gc_safe_points": [
    {
      "service_id": "gc_worker",
      "expired_at": 9223372036854775807,
      "safe_point": 439923410637160448
    }
  ],
  "gc_safe_point": 0
}
```

14.6.2.4.25 store [delete | cancel-delete | label | weight | remove-tombstone | limit] <store_id> [--jq="<query string>"]

Use this command to view the store information or remove a specified store. For a jq formatted output, see jq-formatted-json-output-usage.

Usage:

```
>> store                            // Display information of all stores
{
  "count": 3,
  "stores": [...]
}
>> store 1                          // Get the store with the store id of 1
......
>> store delete 1                   // Delete the store with the store id of
   ↪ 1
......
>> store cancel-delete 1            // Cancel the delete operation previously
   ↪  performed on the store with the id of 1 which is in the offline
   ↪ state. After the cancellation, the store will enter the up state.
   ↪ Note that this command cannot make the tombstone store back to the up
   ↪  state. If the PD leader has been switched during the offline process
   ↪ , you need to manually modify the store limit.
>> store label 1 zone cn            // Set the value of the label with the "
   ↪ zone" key to "cn" for the store with the store id of 1
>> store weight 1 5 10              // Set the leader weight to 5 and Region
   ↪ weight to 10 for the store with the store id of 1
>> store remove-tombstone           // Remove stores that are in tombstone
   ↪ state
>> store limit                      // Show the speed limit of adding-peer
   ↪ operations and the limit of removing-peer operations per minute in
   ↪ all stores
```

```
>> store limit add-peer           // Show the speed limit of adding-peer
   ↪ operations per minute in all stores
>> store limit remove-peer        // Show the limit of removing-peer
   ↪ operations per minute in all stores
>> store limit all 5              // Set the limit of adding-peer
   ↪ operations to 5 and the limit of removing-peer operations to 5 per
   ↪ minute for all stores
>> store limit 1 5                // Set the limit of adding-peer
   ↪ operations to 5 and the limit of removing-peer operations to 5 per
   ↪ minute for store 1
>> store limit all 5 add-peer     // Set the limit of adding-peer
   ↪ operations to 5 per minute for all stores
>> store limit 1 5 add-peer       // Set the limit of adding-peer
   ↪ operations to 5 per minute for store 1
>> store limit 1 5 remove-peer    // Set the limit of removing-peer
   ↪ operations to 5 per minute for store 1
>> store limit all 5 remove-peer  // Set the limit of removing-peer
   ↪ operations to 5 per minute for all stores
```

Note:

You can use `pd-ctl` to check the status (Up, Disconnect, Offline, Down, or Tombstone) of a TiKV store. For the relationship between each status, refer to Relationship between each status of a TiKV store.

14.6.2.4.26 `log [fatal | error | warn | info | debug]`

Use this command to set the log level of the PD leader.

Usage:

```
log warn
```

14.6.2.4.27 `tso`

Use this command to parse the physical and logical time of TSO.

Usage:

```
>> tso 395181938313123110    // Parse TSO
system: 2017-10-09 05:50:59 +0800 CST
logic: 120102
```

14.6.2.4.28 `unsafe remove-failed-stores [store-ids | show]`

> **Warning:**
>
> - This feature is a lossy recovery, so TiKV cannot guarantee data integrity and data indexes integrity after using the feature.
> - It is recommended to perform the feature-related operations with the support from the TiDB team. If any misoperation is performed, it might be hard to recover the cluster.

Use this command to perform lossy recovery operations when permanently damaged replicas cause data to be unavailable. See the following example. The details are described in Online Unsafe Recovery

Execute Online Unsafe Recovery to remove permanently damaged stores:

```
unsafe remove-failed-stores 101,102,103
```

```
Success!
```

Show the current or historical state of Online Unsafe Recovery:

```
unsafe remove-failed-stores show
```

```
[
  "Collecting cluster info from all alive stores, 10/12.",
  "Stores that have reports to PD: 1, 2, 3, ...",
  "Stores that have not reported to PD: 11, 12",
]
```

14.6.2.5 Jq formatted JSON output usage

14.6.2.5.1 Simplify the output of `store`

```
>> store --jq=".stores[].store | { id, address, state_name}"
{"id":1,"address":"127.0.0.1:20161","state_name":"Up"}
{"id":30,"address":"127.0.0.1:20162","state_name":"Up"}
...
```

14.6.2.5.2 Query the remaining space of the node

```
>> store --jq=".stores[] | {id: .store.id, available: .status.available}"
{"id":1,"available":"10 GiB"}
{"id":30,"available":"10 GiB"}
...
```

14.6.2.5.3 Query all nodes whose status is not Up

```
store --jq='.stores[].store | select(.state_name!="Up") | { id, address,
    ↪ state_name}'
```

```
{"id":1,"address":"127.0.0.1:20161""state_name":"Offline"}
{"id":5,"address":"127.0.0.1:20162""state_name":"Offline"}
...
```

14.6.2.5.4 Query all TiFlash nodes

```
store --jq='.stores[].store | select(.labels | length>0 and contains([{"key
    ↪ ":"engine","value":"tiflash"}])) | { id, address, state_name}'
```

```
{"id":1,"address":"127.0.0.1:20161""state_name":"Up"}
{"id":5,"address":"127.0.0.1:20162""state_name":"Up"}
...
```

14.6.2.5.5 Query the distribution status of the Region replicas

```
>> region --jq=".regions[] | {id: .id, peer_stores: [.peers[].store_id]}"
{"id":2,"peer_stores":[1,30,31]}
{"id":4,"peer_stores":[1,31,34]}
...
```

14.6.2.5.6 Filter Regions according to the number of replicas

For example, to filter out all Regions whose number of replicas is not 3:

```
>> region --jq=".regions[] | {id: .id, peer_stores: [.peers[].store_id] |
    ↪ select(length != 3)}"
{"id":12,"peer_stores":[30,32]}
{"id":2,"peer_stores":[1,30,31,32]}
```

14.6.2.5.7 Filter Regions according to the store ID of replicas

For example, to filter out all Regions that have a replica on store30:

```
>> region --jq=".regions[] | {id: .id, peer_stores: [.peers[].store_id] |
   ↪ select(any(.==30))}"
{"id":6,"peer_stores":[1,30,31]}
{"id":22,"peer_stores":[1,30,32]}
...
```

You can also find out all Regions that have a replica on store30 or store31 in the same way:

```
>> region --jq=".regions[] | {id: .id, peer_stores: [.peers[].store_id] |
   ↪ select(any(.==(30,31)))}"
{"id":16,"peer_stores":[1,30,34]}
{"id":28,"peer_stores":[1,30,32]}
{"id":12,"peer_stores":[30,32]}
...
```

14.6.2.5.8 Look for relevant Regions when restoring data

For example, when [store1, store30, store31] is unavailable at its downtime, you can find all Regions whose Down replicas are more than normal replicas:

```
>> region --jq=".regions[] | {id: .id, peer_stores: [.peers[].store_id] |
   ↪ select(length as $total | map(if .==(1,30,31) then . else empty end)
   ↪ | length>=$total-length) }"
{"id":2,"peer_stores":[1,30,31,32]}
{"id":12,"peer_stores":[30,32]}
{"id":14,"peer_stores":[1,30,32]}
...
```

Or when [store1, store30, store31] fails to start, you can find Regions where the data can be manually removed safely on store1. In this way, you can filter out all Regions that have a replica on store1 but don't have other DownPeers:

```
>> region --jq=".regions[] | {id: .id, peer_stores: [.peers[].store_id] |
   ↪ select(length>1 and any(.==1) and all(.!=(30,31)))}"
{"id":24,"peer_stores":[1,32,33]}
```

When [store30, store31] is down, find out all Regions that can be safely processed by creating the `remove-peer` Operator, that is, Regions with one and only DownPeer:

```
>> region --jq=".regions[] | {id: .id, remove_peer: [.peers[].store_id] |
   ↪ select(length>1) | map(if .==(30,31) then . else empty end) | select(
   ↪ length==1)}"
{"id":12,"remove_peer":[30]}
```

```
{"id":4,"remove_peer":[31]}
{"id":22,"remove_peer":[30]}
...
```

14.6.3 TiDB Control User Guide

TiDB Control is a command-line tool of TiDB, usually used to obtain the status information of TiDB for debugging. This document introduces the features of TiDB Control and how to use these features.

> **Note:**
>
> TiDB Control is specifically designed for debugging purposes and might not be fully compatible with future capabilities introduced in TiDB. It's not recommended to include this tool in applications or utilities development to get information.

14.6.3.1 Get TiDB Control

You can get TiDB Control by installing it using TiUP or by compiling it from source code.

> **Note:**
>
> It is recommended that the version of the Control tool you use is consistent with the version of the cluster.

14.6.3.1.1 Install TiDB Control using TiUP

After installing TiUP, you can use `tiup ctl:<cluster-version> tidb` command to get and execute TiDB Control.

14.6.3.1.2 Compile from source code

- Compilation environment requirement: Go Version 1.19 or later
- Compilation procedures: Go to the root directory of the TiDB Control project, use the `make` command to compile, and generate `tidb-ctl`.
- Compilation documentation: you can find the help files in the `doc` directory; if the help files are lost or you want to update them, use the `make doc` command to generate the help files.

14.6.3.2 Usage introduction

This section describes how to use commands, subcommands, options, and flags in `tidb-ctl`.

- command: characters without `-` or `--`
- subcommand: characters without `-` or `--` that follow a command
- option: characters with `-` or `--`
- flag: characters exactly following a command/subcommand or option, passing value to the command/subcommand or option

Usage example: `tidb-ctl schema in mysql -n db`

- `schema`: the command
- `in`: the subcommand of `schema`
- `mysql`: the flag of `in`
- `-n`: the option
- `db`: the flag of `-n`

Currently, TiDB Control has the following subcommands:

- `tidb-ctl base64decode`: used for `BASE64` decoding
- `tidb-ctl decoder`: used for `KEY` decoding
- `tidb-ctl etcd`: used for operating etcd
- `tidb-ctl log`: used to format the log file to expand the single-line stack information
- `tidb-ctl mvcc`: used to get the MVCC information
- `tidb-ctl region`: used to get the Region information
- `tidb-ctl schema`: used to get the schema information
- `tidb-ctl table`: used to get the table information

14.6.3.2.1 Get help

Use `tidb-ctl -h/--help` to get usage information.

TiDB Control consists of multiple layers of commands. You can use `-h/--help` after each command/subcommand to get its respective usage information.

The following example shows how to obtain the schema information:

Use `tidb-ctl schema -h` to get usage details. The `schema` command itself has two subcommands: `in` and `tid`.

- `in` is used to obtain the table schema of all tables in the database through the database name.
- `tid` is used to obtain the table schema by using the unique `table_id` in the whole database.

14.6.3.2.2 Global options

`tidb-ctl` has the following connection-related global options:

- `--host`: TiDB Service address (default 127.0.0.1)
- `--port`: TiDB status port (default 10080)
- `--pdhost`: PD Service address (default 127.0.0.1)
- `--pdport`: PD Service port (default 2379)
- `--ca`: The CA file path used for the TLS connection
- `--ssl-key`: The key file path used for the TLS connection
- `--ssl-cert`: The certificate file path used for the TLS connection

`--pdhost` and `--pdport` are mainly used in the `etcd` subcommand. For example, `tidb-ctl etcd ddlinfo`. If you do not specify the address and the port, the following default value is used:

- The default service address of TiDB and PD: `127.0.0.1`. The service address must be an IP address.
- The default service port of TiDB: `10080`.
- The default service port of PD: `2379`.

14.6.3.2.3 The `schema` command

The `in` subcommand

`in` is used to obtain the table schema of all tables in the database through the database name.

```
tidb-ctl schema in <database name>
```

For example, running `tidb-ctl schema in mysql` returns the following result:

```
[
    {
        "id": 13,
        "name": {
            "O": "columns_priv",
            "L": "columns_priv"
        },
              ...
        "update_timestamp": 399494726837600268,
        "ShardRowIDBits": 0,
        "Partition": null
    }
]
```

The result is displayed in the JSON format. (The above output is truncated.)

- If you want to specify the table name, use `tidb-ctl schema in <database> -n <table name>` to filter.

  For example, `tidb-ctl schema in mysql -n db` returns the table schema of the `db` table in the `mysql` database:

```
{
    "id": 9,
    "name": {
        "O": "db",
        "L": "db"
    },
    ...
    "Partition": null
}
```

  (The above output is also truncated.)

  If you do not want to use the default TiDB service address and port, use the `--host` and `--port` options to configure. For example, `tidb-ctl --host 172.16.55.88 --port 8898 schema in mysql -n db`.

The `tid` subcommand

`tid` is used to obtain the table schema by using the unique `table_id` in the whole database. You can use the `in` subcommand to get all table IDs of certain schema and use the `tid` subcommand to get the detailed table information.

For example, the table ID of `mysql.stat_meta` is `21`. You can use `tidb-ctl schema tid -i 21` to obtain the detail of `mysql.stat_meta`.

```
{
 "id": 21,
 "name": {
  "O": "stats_meta",
  "L": "stats_meta"
 },
 "charset": "utf8mb4",
 "collate": "utf8mb4_bin",
  ...
}
```

Like the `in` subcommand, if you do not want to use the default TiDB service address and status port, use the `--host` and `--port` options to specify the host and port.

The `base64decode` command

`base64decode` is used to decode `base64` data.

```
tidb-ctl base64decode [base64_data]
tidb-ctl base64decode [db_name.table_name] [base64_data]
tidb-ctl base64decode [table_id] [base64_data]
```

1. Execute the following SQL statement to prepare the environment:

```
use test;
create table t (a int, b varchar(20),c datetime default
    ↪ current_timestamp , d timestamp default current_timestamp,
    ↪ unique index(a));
insert into t (a,b,c) values(1,"哈哈 hello",NULL);
alter table t add column e varchar(20);
```

2. Obtian MVCC data using the HTTP API interface:

```
$ curl "http://$IP:10080/mvcc/index/test/t/a/1?a=1"
{
 "info": {
  "writes": [
   {
    "start_ts": 407306449994645510,
    "commit_ts": 407306449994645513,
    "short_value": "AAAAAAAAAAE=" # The unique index a stores the
       ↪ handle id of the corresponding row.
   }
  ]
 }
}%

$ curl "http://$IP:10080/mvcc/key/test/t/1"
{
 "info": {
  "writes": [
   {
    "start_ts": 407306588892692486,
    "commit_ts": 407306588892692489,
    "short_value": "CAIIAggEAhjlk4jlk4ggaGVsbG8IBgAICAmAgIDwjYuu0Rk=" #
       ↪  Row data that handle id is 1.
   }
  ]
 }
}%
```

3. Decode `handle id (uint64)using `base64decode``.

```
$ tidb-ctl base64decode AAAAAAAAAAE=
hex: 0000000000000001
uint64: 1
```

4. Decode row data using `base64decode`.

```
$ ./tidb-ctl base64decode test.t
    ↪ CAIIAggEAhjlk4jlk4ggaGVsbG8IBgAICAmAgIDwjYuuORk=
a:      1
b:      哈哈 hello
c is NULL
d:      2019-03-28 05:35:30
e not found in data

# if the table id of test.t is 60, you can also use below command to do
    ↪  the same thing.
$ ./tidb-ctl base64decode 60
    ↪ CAIIAggEAhjlk4jlk4ggaGVsbG8IBgAICAmAgIDwjYuuORk=
a:      1
b:      哈哈 hello
c is NULL
d:      2019-03-28 05:35:30
e not found in data
```

14.6.3.2.4 The `decoder` command

- The following example shows how to decode the row key, similar to decoding the index key.

```
$ ./tidb-ctl decoder "t\x00\x00\x00\x00\x00\x00\x00\x1c_r\x00\x00\x00\
    ↪ x00\x00\x00\x00\xfa"
format: table_row
table_id: -9223372036854775780  table_id: -9223372036854775780
row_id: -9223372036854775558    row_id: -9223372036854775558
```

- The following example shows how to decode `value`.

```
$ ./tidb-ctl decoder AhZoZWxsbyB3b3JsZAiAEA==
format: index_value
type: bigint, value: 1024   index_value[0]: {type: bytes, value: hello
    ↪ world}
index_value[1]: {type: bigint, value: 1024}
```

14.6.3.2.5 The `etcd` command

- `tidb-ctl etcd ddlinfo` is used to obtain DDL information.
- `tidb-ctl etcd putkey KEY VALUE` is used to add KEY VALUE to etcd (All the KEYs are added to the `/tidb/ddl/all_schema_versions/` directory).

```
tidb-ctl etcd putkey "foo" "bar"
```

  In fact, a key-value pair is added to the etcd whose KEY is `/tidb/ddl/` ↪ `all_schema_versions/foo` and VALUE is `bar`.

- `tidb-ctl etcd delkey` deletes the KEY in etcd. Only those KEYs with the `/tidb/` ↪ `ddl/fg/owner/` or `/tidb/ddl/all_schema_versions/` prefix can be deleted.

```
tidb-ctl etcd delkey "/tidb/ddl/fg/owner/foo"
tidb-ctl etcd delkey "/tidb/ddl/all_schema_versions/bar"
```

14.6.3.2.6 The `log` command

The stack information for the TiDB error log is in one line format. You could use `tidb-ctl log` to change its format to multiple lines.

14.6.3.2.7 The `keyrange` command

The `keyrange` subcommand is used to query the global or table-related key range information, which is output in the hexadecimal form.

- Execute the `tidb-ctl keyrange` command to check the global key range information:

```
tidb-ctl keyrange
```

```
global ranges:
  meta: (6d, 6e)
  table: (74, 75)
```

- Add the `--encode` option to display encoded keys (in the same format as in TiKV and PD):

```
tidb-ctl keyrange --encode
```

```
global ranges:
  meta: (6d00000000000000f8, 6e00000000000000f8)
  table: (7400000000000000f8, 7500000000000000f8)
```

- Execute the `tidb-ctl keyrange --database={db} --table={tbl}` command to check the global and table-related key range information:

```
tidb-ctl keyrange --database test --table ttt
```

```
global ranges:
  meta: (6d, 6e)
  table: (74, 75)
table ttt ranges: (NOTE: key range might be changed after DDL)
  table: (748000000000002f, 748000000000000030)
  table indexes: (748000000000002f5f69, 748000000000002f5f72)
    index c2: (748000000000002f5f698000000000000001,
       ↪ 748000000000002f5f698000000000000002)
    index c3: (748000000000002f5f698000000000000002,
       ↪ 748000000000002f5f698000000000000003)
    index c4: (748000000000002f5f698000000000000003,
       ↪ 748000000000002f5f698000000000000004)
  table rows: (748000000000002f5f72, 748000000000000030)
```

14.6.4 PD Recover User Guide

PD Recover is a disaster recovery tool of PD, used to recover the PD cluster which cannot start or provide services normally.

14.6.4.1 Compile from source code

- Go Version 1.19 or later is required because the Go modules are used.
- In the root directory of the PD project, use the `make pd-recover` command to compile and generate `bin/pd-recover`.

> **Note:**
>
> Generally, you do not need to compile source code because the PD Control tool already exists in the released binary or Docker. However, developer users can refer to the instructions above for compiling source code.

14.6.4.2 Download TiDB Toolkit

The PD Recover installation package is included in the TiDB Toolkit. To download the TiDB Toolkit, see Download TiDB Tools.

The following sections introduce two methods to recover a PD cluster: recover from a surviving PD node and rebuild a PD cluster entirely.

14.6.4.3 Method 1: Recover a PD cluster using a surviving PD node

When a majority of PD nodes in a cluster experience an unrecoverable error, the cluster becomes unable to provide services. If there are any surviving PD nodes, you can recover the service by selecting a surviving PD node and forcibly modifying the members of the Raft Group. The steps are as follows:

14.6.4.3.1 Step 1: Stop all nodes

To prevent data corruption or other unrecoverable errors caused by interactions with PD parameters during the recovery process, stop the TiDB, TiKV, and TiFlash processes in the cluster.

14.6.4.3.2 Step 2: Start the surviving PD node

Start the surviving PD node using the `--force-new-cluster` startup parameter. The following is an example:

```
./bin/pd-server --force-new-cluster --name=pd-127.0.0.10-2379 --client-urls=
    ↪ http://0.0.0.0:2379 --advertise-client-urls=http://127.0.0.1:2379 --
    ↪ peer-urls=http://0.0.0.0:2380 --advertise-peer-urls=http
    ↪ ://127.0.0.1:2380 --config=conf/pd.toml
```

14.6.4.3.3 Step 3: Repair metadata using `pd-recover`

Since this method relies on a minority PD node to recover the service, the node might contain outdated data. If the `alloc_id` and `tso` data roll back, the cluster data might be corrupted or unavailable. To prevent this, you need to use `pd-recover` to modify the metadata to ensure that the node can provide correct allocation IDs and TSO services. The following is an example:

```
./bin/pd-recover --from-old-member --endpoints=http://127.0.0.1:2379 #
    ↪ Specify the corresponding PD address
```

> **Note:**
>
> In this step, the `alloc_id` in the storage automatically increases by a safe value of `100000000`. As a result, the subsequent cluster will allocate larger IDs.
>
> Additionally, `pd-recover` does not modify the TSO. Therefore, before performing this step, make sure that the local time is later than the time when the failure occurs, and verify that the NTP clock synchronization service is enabled between the PD components before the failure. If it is not enabled, you need to adjust the local clock to a future time to prevent the TSO from rolling back.

14.6.4.3.4 Step 4: Restart the PD node

Once you see the prompt message `recovery is successful`, restart the PD node.

14.6.4.3.5 Step 5: Scale out PD and start the cluster

Scale out the PD cluster using the deployment tool and start the other components in the cluster. At this point, the PD service is available.

14.6.4.4 Method 2: Entirely rebuild a PD cluster

This method is applicable to scenarios in which all PD data is lost, but the data of other components, such as TiDB, TiKV, and TiFlash, still exists.

14.6.4.4.1 Step 1: Get cluster ID

The cluster ID can be obtained from the log of PD, TiKV, or TiDB. To get the cluster ID, you can view the log directly on the server.

Get cluster ID from PD log (recommended)

To get the cluster ID from the PD log, run the following command:

```
cat {{/path/to}}/pd.log | grep "init cluster id"
```

```
[2019/10/14 10:35:38.880 +00:00] [INFO] [server.go:212] ["init cluster id"]
    ↪ [cluster-id=6747551640615446306]
...
```

Get cluster ID from TiDB log

To get the cluster ID from the TiDB log, run the following command:

```
cat {{/path/to}}/tidb.log | grep "init cluster id"
```

```
2019/10/14 19:23:04.688 client.go:161: [info] [pd] init cluster id
    ↪ 6747551640615446306
...
```

Get cluster ID from TiKV log

To get the cluster ID from the TiKV log, run the following command:

```
cat {{/path/to}}/tikv.log | grep "connect to PD cluster"
```

```
[2019/10/14 07:06:35.278 +00:00] [INFO] [tikv-server.rs:464] ["connect to PD
    ↪  cluster 6747551640615446306"]
...
```

14.6.4.4.2 Step 2: Get allocated ID

The allocated ID value you specify must be larger than the currently largest allocated ID value. To get allocated ID, you can either get it from the monitor, or view the log directly on the server.

Get allocated ID from the monitor (recommended)

To get allocated ID from the monitor, you need to make sure that the metrics you are viewing are the metrics of **the last PD leader**, and you can get the largest allocated ID from the **Current ID allocation** panel in PD dashboard.

Get allocated ID from PD log

To get the allocated ID from the PD log, you need to make sure that the log you are viewing is the log of **the last PD leader**, and you can get the maximum allocated ID by running the following command:

```
cat {{/path/to}}/pd*.log | grep "idAllocator allocates a new id" | awk -F
    ↪ '=' '{print $2}' | awk -F']' '{print $1}' | sort -r -n | head -n 1
```

```
4000
...
```

Or you can simply run the above command in all PD servers to find the largest one.

14.6.4.4.3 Step 3: Deploy a new PD cluster

Before deploying a new PD cluster, you need to stop the the existing PD cluster and then delete the previous data directory or specify a new data directory using `--data-dir`.

14.6.4.4.4 Step 4: Use pd-recover

You only need to run `pd-recover` on one PD node.

```
./pd-recover -endpoints http://10.0.1.13:2379 -cluster-id
    ↪ 6747551640615446306 -alloc-id 10000
```

14.6.4.4.5 Step 5: Restart the whole cluster

When you see the prompted information that the recovery is successful, restart the whole cluster.

14.6.4.5 FAQ

14.6.4.5.1 Multiple cluster IDs are found when getting the cluster ID

When a PD cluster is created, a new cluster ID is generated. You can determine the cluster ID of the old cluster by viewing the log.

14.6.4.5.2 The error `dial tcp 10.0.1.13:2379: connect: connection refused` is returned when executing `pd-recover`

The PD service is required when you execute `pd-recover`. Deploy and start the PD cluster before you use PD Recover.

14.7 Command Line Flags

14.7.1 Configuration Options

When you start the TiDB cluster, you can use command-line options or environment variables to configure it. This document introduces TiDB's command options. The default TiDB ports are `4000` for client requests and `10080` for status report.

14.7.1.1 `--advertise-address`

- The IP address through which to log into the TiDB server
- Default: `""`
- This address must be accessible by the rest of the TiDB cluster and the user.

14.7.1.2 `--config`

- The configuration file
- Default: `""`
- If you have specified the configuration file, TiDB reads the configuration file. If the corresponding configuration also exists in the command line options, TiDB uses the configuration in the command line options to overwrite that in the configuration file. For detailed configuration information, see TiDB Configuration File Description.

14.7.1.3 `--config-check`

- Checks the validity of the configuration file and exits
- Default: `false`

14.7.1.4 `--config-strict`

- Enforces the validity of the configuration file
- Default: `false`

14.7.1.5 `--cors`

- Specifies the `Access-Control-Allow-Origin` value for Cross-Origin Request Sharing (CORS) request of the TiDB HTTP status service
- Default: `""`

14.7.1.6 `--enable-binlog`

- Enables or disables TiDB binlog generation
- Default: `false`

14.7.1.7 `--host`

- The host address that the TiDB server monitors
- Default: `"0.0.0.0"`
- The TiDB server monitors this address.
- The `"0.0.0.0"` address monitors all network cards by default. If you have multiple network cards, specify the network card that provides service, such as `192.168.100.113`.

14.7.1.8 `--initialize-insecure`

- Bootstraps tidb-server in insecure mode
- Default: `true`

14.7.1.9 `--initialize-secure`

- Bootstraps tidb-server in secure mode
- Default: `false`

14.7.1.10 `-L`

- The log level
- Default: `"info"`
- Optional values: `"debug"`, `"info"`, `"warn"`, `"error"`, `"fatal"`

14.7.1.11 `--lease`

- The duration of the schema lease. It is **dangerous** to change the value unless you know what you do.
- Default: `45s`

14.7.1.12 `--log-file`

- The log file
- Default: `""`
- If this option is not set, logs are output to "stderr". If this option is set, logs are output to the corresponding file.

14.7.1.13 `--log-slow-query`

- The directory for the slow query log
- Default: `""`
- If this option is not set, logs are output to the file specified by `--log-file` by default.

14.7.1.14 `--metrics-addr`

- The Prometheus Pushgateway address
- Default: `""`
- Leaving it empty stops the Prometheus client from pushing.
- The format is `--metrics-addr=192.168.100.115:9091`.

14.7.1.15 `--metrics-interval`

- The Prometheus client push interval in seconds
- Default: `15s`
- Setting the value to 0 stops the Prometheus client from pushing.

14.7.1.16 `-P`

- The monitoring port of TiDB services
- Default: `"4000"`
- The TiDB server accepts MySQL client requests from this port.

14.7.1.17 `--path`

- The path to the data directory for local storage engine like "unistore"
- For `--store = tikv`, you must specify the path; for `--store = unistore`, the default value is used if you do not specify the path.
- For the distributed storage engine like TiKV, `--path` specifies the actual PD address. Assuming that you deploy the PD server on 192.168.100.113:2379, 192.168.100.114:2379 and 192.168.100.115:2379, the value of `--path` is "192.168.100.113:2379, 192.168.100.114:2379, 192.168.100.115:2379".
- Default: `"/tmp/tidb"`
- You can use `tidb-server --store=unistore --path=""` to enable a pure in-memory TiDB.

14.7.1.18 `--proxy-protocol-networks`

- The list of proxy server's IP addresses allowed to connect to TiDB using the PROXY protocol.
- Default: `""`
- In general cases, when you access TiDB behind a reverse proxy, TiDB takes the IP address of the reverse proxy server as the IP address of the client. By enabling the PROXY protocol, reverse proxies that support this protocol such as HAProxy can pass the real client IP address to TiDB.
- After configuring this flag, TiDB allows the configured source IP address to connect to TiDB using the PROXY protocol; if a protocol other than PROXY is used, this connection will be denied. If this flag is left empty, no IP address can connect to TiDB using the PROXY protocol. The value can be the IP address (192.168.1.50) or CIDR (192.168.1.0/24) with `,` as the separator. `*` means any IP addresses.

> **Warning:**
>
> Use `*` with caution because it might introduce security risks by allowing a client of any IP address to report its IP address. In addition, using `*` might also cause the internal component that directly connects to TiDB (such as TiDB Dashboard) to be unavailable.

14.7.1.19 `--proxy-protocol-header-timeout`

- Timeout for the PROXY protocol header read
- Default: `5` (seconds)

> **Note:**
>
> Do not set the value to `0`. Use the default value except for special situations.

14.7.1.20 `--report-status`

- Enables (`true`) or disables (`false`) the status report and pprof tool
- Default: `true`
- When set to `true`, this parameter enables metrics and pprof. When set to `false`, this parameter disables metrics and pprof.

14.7.1.21 `--run-ddl`

- To see whether the `tidb-server` runs DDL statements, and set when the number of `tidb-server` is over two in the cluster
- Default: `true`
- The value can be (true) or (false). (true) indicates the `tidb-server` runs DDL itself. (false) indicates the `tidb-server` does not run DDL itself.

14.7.1.22 `--socket string`

- The TiDB services use the unix socket file for external connections.
- Default: `""`
- Use `/tmp/tidb.sock` to open the unix socket file.

14.7.1.23 `--status`

- The status report port for TiDB server
- Default: `"10080"`
- This port is used to get server internal data. The data includes Prometheus metrics and pprof.
- Prometheus metrics can be accessed by `"http://host:status_port/metrics"`.
- pprof data can be accessed by `"http://host:status_port/debug/pprof"`.

14.7.1.24 `--status-host`

- The `HOST` used to monitor the status of TiDB service
- Default: `0.0.0.0`

14.7.1.25 `--store`

- Specifies the storage engine used by TiDB in the bottom layer
- Default: `"unistore"`
- You can choose “unistore” or “tikv”. (“unistore” is the local storage engine; “tikv” is a distributed storage engine)

14.7.1.26 `--temp-dir`

- The temporary directory of TiDB
- Default: `"/tmp/tidb"`

14.7.1.27 `--token-limit`

- The number of sessions allowed to run concurrently in TiDB. It is used for traffic control.
- Default: `1000`
- If the number of the concurrent sessions is larger than `token-limit`, the request is blocked and waiting for the operations which have been finished to release tokens.

14.7.1.28 `-V`

- Outputs the version of TiDB
- Default: `""`

14.7.1.29 `--plugin-dir`

- The storage directory for plugins.
- Default: `"/data/deploy/plugin"`

14.7.1.30 `--plugin-load`

- The names of the plugins to be loaded, each separated by a comma.
- Default: `""`

14.7.1.31 `--affinity-cpus`

- Sets the CPU affinity of TiDB servers, which is separated by commas. For example, "1,2,3".
- Default: `""`

14.7.1.32 `--repair-mode`

- Determines whether to enable the repair mode, which is only used in the data repair scenario.
- Default: `false`

14.7.1.33 `--repair-list`

- The names of the tables to be repaired in the repair mode.
- Default: `""`

14.7.2 TiKV Configuration Flags

TiKV supports some readable unit conversions for command line parameters.

- File size (based on byte): KB, MB, GB, TB, PB (or lowercase)
- Time (based on ms): ms, s, m, h

14.7.2.1 `-A, --addr`

- The address that the TiKV server monitors
- Default: `"127.0.0.1:20160"`
- To deploy a cluster, you must use `--addr` to specify the IP address of the current host, such as `"192.168.100.113:20160"`. If the cluster is run on Docker, specify the IP address of Docker as `"0.0.0.0:20160"`.

14.7.2.2 `--advertise-addr`

- The server advertise address for client traffic from outside
- Default: `${addr}`
- If the client cannot connect to TiKV through the `--addr` address because of Docker or NAT network, you must manually set the `--advertise-addr` address.
- For example, the internal IP address of Docker is 172.17.0.1, while the IP address of the host is 192.168.100.113 and the port mapping is set to `-p 20160:20160`. In this case, you can set `--advertise-addr` to "192.168.100.113:20160". The client can find this service through 192.168.100.113:20160.

14.7.2.3 `--status-addr`

- The port through which the TiKV service status is listened
- Default: `"20180"`
- The Prometheus can access this status information via `http://host:status_port/metrics`.
- The Profile can access this status information via `http://host:status_port/debug/pprof/profile`.

14.7.2.4 `--advertise-status-addr`

- The address through which TiKV accesses service status from outside.
- Default: The value of `--status-addr` is used.
- If the client cannot connect to TiKV through the `--status-addr` address because of Docker or NAT network, you must manually set the `--advertise-status-addr` address.

- For example, the internal IP address of Docker is `172.17.0.1`, while the IP address of the host is `192.168.100.113` and the port mapping is set to `-p 20180:20180`. In this case, set `--advertise-status-addr="192.168.100.113:20180"`. The client can find this service through `192.168.100.113:20180`.

14.7.2.5 `-C, --config`

- The config file
- Default: `""`
- If you set the configuration using the command line, the same setting in the config file will be overwritten.

14.7.2.6 `--capacity`

- The store capacity
- Default: `0` (unlimited)
- PD uses this flag to determine how to balance the TiKV servers. (Tip: you can use 10GB instead of 1073741824)

14.7.2.7 `--data-dir`

- The path to the data directory
- Default: `"/tmp/tikv/store"`

14.7.2.8 `-L`

- The log level
- Default: `"info"`
- Optional values: `"debug"`, `"info"`, `"warn"`, `"error"`, `"fatal"`

14.7.2.9 `--log-file`

- The log file
- Default: `""`
- If this flag is not set, logs will be written to "stderr". If this flag is set, logs are output to the corresponding file.

14.7.2.10 `--pd`

- The address list of PD servers
- Default: `""`
- To make TiKV work, you must use the value of `--pd` to connect the TiKV server to the PD server. Separate multiple PD addresses using comma, for example "192.168.100.113:2379, 192.168.100.114:2379, 192.168.100.115:2379".

14.7.3 TiFlash Command-Line Flags

This document introduces the command-line flags that you can use when you launch TiFlash.

14.7.3.1 `server --config-file`

- Specifies the path of the TiFlash configuration file
- Default: “”
- You must specify the configuration file. For detailed configuration items, refer to TiFlash configuration parameters.

14.7.3.2 `dttool migrate`

- Migrates the file format of DTFile (for testing or downgrading). Data is migrated in the unit of a single DTFile. If you want to migrate the whole table, you need to locate all the paths similar to `<data dir>/t_<table id>/stable/dmf_<file id>` and migrate them one by one. You can use scripts to automate the migration.

- User scenarios:

    - If you need to downgrade TiFlash from a version >= v5.4.0 that has enabled data validation to a version < v5.4.0, you can use this tool to downgrade the data format of the DTFile.
    - If you upgrade TiFlash to a version >= v5.4.0, and if you hope to enable data validation for existing data, you can use this tool to upgrade the data format of the DTFile.
    - Test the space usage and read speed of the DTFile in different configurations.

- Parameters:

    - `--imitative`: When you do not use the encryption feature of the DTFile, you can use this flag to avoid using the configuration file and connecting to PD.
    - `--version`: The version of DTFile. The value options are `1` and `2` (default). `1` is the old version, and `2` is the version corresponding to the new checksum.
    - `--algorithm`: The hash algorithm used for data validation. The value options are `xxh3` (default), `city128`, `crc32`, `crc64`, and `none`. This parameter is effective only when `version` is 2.
    - `--frame`: The size of the validation frame. The default value is `1048576`. This parameter is effective only when `version` is 2.
    - `--compression`: The target compression algorithm. The value options are `LZ4` (default), `LZ4HC`, `zstd`, and `none`.

- `--level`: The target compression level. If not specified, the recommended compression level is used by default according to the compression algorithm. If `compression` is set to `LZ4` or `zstd`, the default level is 1. If `compression` is set to `LZ4HC`, the default level is 9.
- `--config-file`: The configuration file of `dttool migrate` is the same as the configuration file of `server`. When you use the configuration file, exit the local TiFlash server instance. For more information, see `--imitative`.
- `--file-id`: The ID of the DTFile. For example, the ID of the DTFile `dmf_123` is `123`.
- `--workdir`: The parent directory of `dmf_xxx`.
- `--dry`: The dry run mode. Only the migration process is output.
- `--nokeep`: Does not keep the original data. When this option is not enabled, `dmf_xxx.old` files are created.

Warning:

TiFlash can read DTFile that uses custom compression algorithms and compression levels. However, only the `lz4` algorithm with the default compression level is officially supported. Custom compression parameters have not been thoroughly tested and are only experimental.

Note:

For security reasons, DTTool attempts to add a lock to the working directory in the migration mode. Therefore, in the same directory, only one DTTool can perform the migration task at the same time. If you forcibly stop DTTool where the lock is not released, then when you try to rerun DTTool later, it might refuse to perform the migration task.

If you encounter this situation, and if you are aware that removing the LOCK file does not cause any data corruption, you can manually delete the LOCK file in the working directory to release the lock.

14.7.3.3 `dttool bench`

- Provides a basic I/O speed test for the DTFile.
- Parameters:
  - `--version`: The version of DTFile. See `--version` in `dttool migrate`.

- `--algorithm`: The hash algorithm used for data validation. See `--algorithm` in `dttool migrate`.
- `--frame`: The size of the validation frame. See `--frame` in `dttool migrate`.
- `--column`: The columns of the table to be tested. The default value is `100`.
- `--size`: The rows of the table to be tested. The default value is `1000`.
- `--field`: The field length limit of the table to be tested. The default value is `1024`.
- `--random`: The random seed. If you do not specify this parameter, the random seed is drawn from the system entropy pool.
- `--encryption`: Enables the encryption feature.
- `--repeat`: The number of times to repeat the test. The default value is `5`.
- `--workdir`: The temporary data directory, which points to a path in the file system to be tested. The default value is `/tmp/test`.

14.7.3.4 `dttool inspect`

- Checks the integrity of the DTFile. Data validation is performed in the unit of a single DTFile. If you want to validate the whole table, you need to locate all the paths similar to `<data dir>/t_<table id>/stable/dmf_<file id>` and validate them one by one. You can use scripts to automate the validation.

- User scenarios:

    - After you perform a format upgrade or downgrade, you can validate the data integrity of the DTFile.
    - After you migrate the DTFile to a new environment, you can validate the data integrity of the DTFile.

- Parameters:

    - `--config-file`: The configuration file of `dttool bench`. See `--config-file` in `dttool migrate`.
    - `--check`: Performs hash validation.
    - `--file-id`: The ID of the DTFile. See `--file-id` in `dttool migrate`.
    - `--imitative`: Imitates the database context. See `--imitative` in `dttool migrate`.
    - `--workdir`: The data directory. See `--workdir` in `dttool migrate`.

14.7.4 PD Configuration Flags

PD is configurable using command-line flags and environment variables.

14.7.4.1 `--advertise-client-urls`

- The list of advertise URLs for the client to access PD
- Default: `"${client-urls}"`
- In some situations such as in the Docker or NAT network environment, if a client cannot access PD through the default client URLs listened to by PD, you must manually set the advertise client URLs.
- For example, the internal IP address of Docker is `172.17.0.1`, while the IP address of the host is `192.168.100.113` and the port mapping is set to `-p 2379:2379`. In this case, you can set `--advertise-client-urls` to `"http://192.168.100.113:2379"`. The client can find this service through `"http://192.168.100.113:2379"`.

14.7.4.2 `--advertise-peer-urls`

- The list of advertise URLs for other PD nodes (peers) to access a PD node
- Default: `"${peer-urls}"`
- In some situations such as in the Docker or NAT network environment, if the other nodes (peers) cannot access the PD node through the default peer URLs listened to by this PD node, you must manually set the advertise peer URLs.
- For example, the internal IP address of Docker is `172.17.0.1`, while the IP address of the host is `192.168.100.113` and the port mapping is set to `-p 2380:2380`. In this case, you can set `--advertise-peer-urls` to `"http://192.168.100.113:2380"`. The other PD nodes can find this service through `"http://192.168.100.113:2380"`.

14.7.4.3 `--client-urls`

- The list of client URLs to be listened to by PD
- Default: `"http://127.0.0.1:2379"`
- When you deploy a cluster, you must specify the IP address of the current host as `--client-urls` (for example, `"http://192.168.100.113:2379"`). If the cluster runs on Docker, specify the IP address of Docker as `"http://0.0.0.0:2379"`.

14.7.4.4 `--peer-urls`

- The list of peer URLs to be listened to by a PD node
- Default: `"http://127.0.0.1:2380"`
- When you deploy a cluster, you must specify `--peer-urls` as the IP address of the current host, such as `"http://192.168.100.113:2380"`. If the cluster runs on Docker, specify the IP address of Docker as `"http://0.0.0.0:2380"`.

14.7.4.5 `--config`

- The configuration file
- Default: `""`
- If you set the configuration using the command line, the same setting in the configuration file will be overwritten.

14.7.4.6 `--data-dir`

- The path to the data directory
- Default: `"default.${name}"`

14.7.4.7 `--initial-cluster`

- The initial cluster configuration for bootstrapping
- Default: `"{name}=http://{advertise-peer-url}"`
- For example, if `name` is "pd", and `advertise-peer-urls` is `"http://192.168.100.113:2380"` ↪ , the `initial-cluster` is `"pd=http://192.168.100.113:2380"`.
- If you need to start three PD servers, the `initial-cluster` might be:

```
pd1=http://192.168.100.113:2380, pd2=http://192.168.100.114:2380, pd3
    ↪ =192.168.100.115:2380
```

14.7.4.8 `--join`

- Join the cluster dynamically
- Default: `""`
- If you want to join an existing cluster, you can use `--join="${advertise-client-` ↪ `urls}"`, the `advertise-client-url` is any existing PD's, multiply advertise client urls are separated by comma.

14.7.4.9 `-L`

- The log level
- Default: `"info"`
- Optional values: `"debug"`, `"info"`, `"warn"`, `"error"`, `"fatal"`

14.7.4.10 `--log-file`

- The log file
- Default: `""`
- If this flag is not set, logs will be written to "stderr". If this flag is set, logs are output to the corresponding file.

14.7.4.11 `--log-rotate`

- To enable or disable log rotation
- Default: `true`
- When the value is true, follow the `[log.file]` in PD configuration files.

14.7.4.12 `--name`

- The human-readable unique name for this PD member
- Default: `"pd-${hostname}"`
- If you want to start multiply PDs, you must use different name for each one.

14.7.4.13 `--cacert`

- The file path of CA, used to enable TLS
- Default: `""`

14.7.4.14 `--cert`

- The path of the PEM file including the X509 certificate, used to enable TLS
- Default: `""`

14.7.4.15 `--key`

- The path of the PEM file including the X509 key, used to enable TLS
- Default: `""`

14.7.4.16 `--metrics-addr`

- The address of Prometheus Pushgateway, which does not push data to Prometheus by default.
- Default: `""`

14.8 Key Monitoring Metrics

14.8.1 Key Metrics

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For more information, see TiDB Monitoring Framework Overview.

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node_exporter, Disk Performance, and Performance_overview. A lot of metrics are there to help you diagnose.

For routine operations, you can get an overview of the component (PD, TiDB, TiKV) status and the entire cluster from the Overview dashboard, where the key metrics are displayed. This document provides a detailed description of these key metrics.

14.8.1.1 Key metrics description

To understand the key metrics displayed on the Overview dashboard, check the following table:

Service	Panel Name	Description	Normal Range
Services Port Status	Services Up	The online nodes number of each service.	
PD	PD role	The role of the current PD.	
PD	Storage capacity	The total storage capacity of the TiDB cluster.	
PD	Current storage size	The occupied storage capacity of the TiDB cluster, including the space occupied by TiKV replicas.	
PD	Normal stores	The number of nodes in the normal state.	
PD	Abnormal stores	The number of nodes in the abnormal state.	0
PD	Number of Regions	The total number of Regions in the current cluster. Note that the number of Regions has nothing to do with the number of replicas.	
PD	99% completed_cmds_duration_seconds	The 99th percentile duration to complete a pd-server request.	less than 5ms
PD	Handle_requests_duration_seconds	The network duration of a PD request.	

Service	Panel Name	Description	Normal Range
PD	Region health	The state of each Region.	Generally, the number of pending peers is less than 100, and that of the missing peers cannot always be greater than `0`.
PD	Hot write Region's leader distribution	The total number of leaders who are the write hotspots on each TiKV instance.	
PD	Hot read Region's leader distribution	The total number of leaders who are the read hotspots on each TiKV instance.	
PD	Region heartbeat report	The count of heartbeats reported to PD per instance.	
PD	99% Region heartbeat latency	The heartbeat latency per TiKV instance (P99).	
TiDB	Statement OPS	The number of different types of SQL statements executed per second, which is counted according to `SELECT`, `INSERT`, `UPDATE`, and other types of statements.	

Service	Panel Name	Description	Normal Range
TiDB	Duration	The execution time.1. The duration between the time that the client's network request is sent to TiDB and the time that the request is returned to the client after TiDB has executed the request. In general, client requests are sent in the form of SQL statements; however, this duration can include the execution time of commands such as `COM_PING`, `COM_SLEEP`, `COM_STMT_FETCH`, and `COM_SEND_LONG_DATA`.2. Because TiDB supports Multi-Query, TiDB supports sending multiple SQL statements at one time, such as `select 1; select 1; select 1;`. In this case, the total execution time of this query includes the execution time of all SQL statements.	
TiDB	CPS By Instance	CPS By Instance: the command statistics on each TiDB instance, which is classified according to the success or failure of command execution results.	
TiDB	Failed Query OPM	The statistics of error types (such as syntax errors and primary key conflicts) based on the errors occurred when executing SQL statements per second on each TiDB instance. The module in which the error occurs and the error code are included.	
TiDB	Connection Count	The connection number of each TiDB instance.	
TiDB	Memory Usage	The memory usage statistics of each TiDB instance, which is divided into the memory occupied by processes and the memory applied by Golang on the heap.	
TiDB	Transaction OPS	The number of transactions executed per second.	

Service	Panel Name	Description	Normal Range
TiDB	Transaction Duration	The execution time of a transaction	
TiDB	KV Cmd OPS	The number of executed KV commands.	
TiDB	KV Cmd Duration 99	The execution time of the KV command.	
TiDB	PD TSO OPS	The number of TSO that TiDB obtains from PD per second.	
TiDB	PD TSO Wait Duration	The duration that TiDB waits for PD to return TSO.	
TiDB	TiClient Region Error OPS	The number of Region related errors returned by TiKV.	
TiDB	Lock Resolve OPS	The number of TiDB operations that resolve locks. When TiDB's read or write request encounters a lock, it tries to resolve the lock.	
TiDB	KV Backoff OPS	The number of errors returned by TiKV.	
TiKV	leader	The number of leaders on each TiKV node.	
TiKV	region	The number of Regions on each TiKV node.	
TiKV	CPU	The CPU usage ratio on each TiKV node.	
TiKV	Memory	The memory usage on each TiKV node.	
TiKV	store size	The size of storage space used by each TiKV instance.	
TiKV	cf size	The size of each column family (CF for short).	
TiKV	channel full	The number of "channel full" errors on each TiKV instance.	0
TiKV	server report failures	The number of error messages reported by each TiKV instance.	0
TiKV	scheduler pending commands	The number of pending commands on each TiKV instance.	
TiKV	coprocessor executor count	The number of coprocessor operations received by TiKV per second. Each type of coprocessor is counted separately.	
TiKV	coprocessor request duration	The time consumed to process read requests of coprocessor.	

Service	Panel Name	Description	Normal Range
TiKV	raft store CPU	The CPU usage ratio of the raftstore thread	The default number of threads is 2 (configured by `raftstore.store-pool-size`). A value of over 80% for a single thread indicates that the CPU usage ratio is very high.
TiKV	Coprocessor CPU	The CPU usage ratio of the coprocessor thread.	
System Info	Vcores	The number of CPU cores.	
System Info	Memory	The total memory.	
System Info	CPU Usage	The CPU usage ratio, 100% at a maximum.	
System Info	Load [1m]	The overload within 1 minute.	
System Info	Memory Available	The size of the available memory.	
System Info	Network Traffic	The statistics of the network traffic.	
System Info	TCP Retrans	The frequency of the TOC retransmission.	
System Info	IO Util	The disk usage ratio, 100% at a maximum; generally you need to consider adding a new node when the usage ratio is up to 80% ~ 90%.	

14.8.1.2 Interface of the Overview dashboard



Figure 330: overview

14.8.2 Key Metrics on Performance Overview

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For more information, see TiDB Monitoring Framework Overview.

The Grafana dashboard is divided into a series of sub dashboards which include PD, TiDB, TiKV, Node_exporter, Overview, and Performance Overview. A lot of metrics are there to help you diagnose.

The Performance Overview dashboard orchestrates the metrics of TiDB, PD, and TiKV, and presents each of them in the following sections:

- Overview: Database time and SQL execution time summary. By checking different colors in the overview, you can quickly identify the database workload profile and the performance bottleneck.

- Load profile: Key metrics and resource usage, including database QPS, connection information, the MySQL command types the application interacts with TiDB, database internal TSO and KV request OPS, and resource usage of the TiKV and TiDB.
- Top-down latency breakdown: Query latency versus connection idle time ratio, query latency breakdown, TSO/KV request latency during execution, breakdown of write latency within TiKV.

With the Performance Overview Dashboard, you can analyze performance efficiently, and confirm whether the bottleneck of user response time is in the database. If the bottleneck is in the database, you can identify the bottleneck inside the database, with database time overview, workload profile and SQL latency breakdown. For details, see Performance Analysis and Tuning.

The following sections illustrate the metrics on the Performance Overview dashboard.

14.8.2.1 Database Time by SQL Type

- database time: Total database time per second
- sql_type: Database time consumed by each type of SQL statements per second

14.8.2.2 Database Time by SQL Phase

- database time: Total database time per second
- get token/parse/compile/execute: Database time consumed in four SQL processing phases

The SQL execution phase is in green and other phases are in red on general. If non-green areas are large, it means much database time is consumed in other phases than the execution phase and further cause analysis is required.

14.8.2.3 SQL Execute Time Overview

- execute time: Database time consumed during SQL execution per second
- tso_wait: Concurrent TSO waiting time per second during SQL execution
- kv request type: Time waiting for each KV request type per second during SQL execution. The total KV request wait time might exceed SQL execution time, because KV requests are concurrent.

Green metrics stand for common KV write requests (such as prewrite and commit), blue metrics stand for common read requests, and metrics in other colors stand for unexpected situations which you need to pay attention to. For example, pessimistic lock KV requests are marked red and TSO waiting is marked dark brown.

If non-blue or non-green areas are large, it means there is a bottleneck during SQL execution. For example:

- If serious lock conflicts occur, the red area will take a large proportion.
- If excessive time is consumed in waiting TSO, the dark brown area will take a large proportion.

14.8.2.4 QPS

Number of SQL statements executed per second in all TiDB instances, collected by type: such as `SELECT`, `INSERT`, and `UPDATE`

14.8.2.5 CPS By Type

Number of commands processed by all TiDB instances per second based on type

14.8.2.6 Queries Using Plan Cache OPS

Number of queries using plan cache per second in all TiDB instances

14.8.2.7 KV/TSO Request OPS

- kv request total: Total number of KV requests per second in all TiDB instances
- kv request by type: Number of KV requests per second in all TiDB instances based on such types as `Get`, `Prewrite`, and `Commit`.
- tso - cmd: Number of `tso cmd` requests per second in all TiDB instances
- tso - request: Number of `tso request` requests per second in all TiDB instances

Generally, dividing `tso - cmd` by `tso - request` yields the average batch size of requests per second.

14.8.2.8 Connection Count

- total: Number of connections to all TiDB instances
- active connections: Number of active connections to all TiDB instances
- Number of connections to each TiDB instance

14.8.2.9 TiDB CPU

- avg: Average CPU utilization across all TiDB instances
- delta: Maximum CPU utilization of all TiDB instances minus minimum CPU utilization of all TiDB instances
- max: Maximum CPU utilization across all TiDB instances

14.8.2.10 TiKV CPU/IO MBps

- CPU-Avg: Average CPU utilization of all TiKV instances
- CPU-Delta: Maximum CPU utilization of all TiKV instances minus minimum CPU utilization of all TiKV instances
- CPU-MAX: Maximum CPU utilization among all TiKV instances
- IO-Avg: Average MBps of all TiKV instances
- IO-Delt: Maximum MBps of all TiKV instances minus minimum MBps of all TiKV instances
- IO-MAX: Maximum MBps of all TiKV instances

14.8.2.11 Duration

- Duration: Execution time
    - The duration from receiving a request from the client to TiDB till TiDB executing the request and returning the result to the client. In general, client requests are sent in the form of SQL statements; however, this duration can include the execution time of commands such as `COM_PING`, `COM_SLEEP`, `COM_STMT_FETCH`, and `COM_SEND_LONG_DATA`.
    - TiDB supports Multi-Query, which means the client can send multiple SQL statements at one time, such as `select 1; select 1; select 1;`. In this case, the total execution time of this query includes the execution time of all SQL statements.
- avg: Average time to execute all requests
- 99: P99 duration to execute all requests
- avg by type: Average time to execute all requests in all TiDB instances, collected by type: `SELECT`, `INSERT`, and `UPDATE`

14.8.2.12 Connection Idle Duration

Connection Idle Duration indicates the duration of a connection being idle.

- avg-in-txn: Average connection idle duration when the connection is within a transaction
- avg-not-in-txn: Average connection idle duration when the connection is not within a transaction
- 99-in-txn: P99 connection idle duration when the connection is within a transaction
- 99-not-in-txn: P99 connection idle duration when the connection is not within a transaction

14.8.2.13 Parse Duration, Compile Duration, and Execute Duration

- Parse Duration: Time consumed in parsing SQL statements
- Compile Duration: Time consumed in compiling the parsed SQL AST to execution plans
- Execution Duration: Time consumed in executing execution plans of SQL statements

All these three metrics include the average duration and the 99th percentile duration in all TiDB instances.

14.8.2.14 Avg TiDB KV Request Duration

Average time consumed in executing KV requests in all TiDB instances based on the type, including `Get`, `Prewrite`, and `Commit`.

14.8.2.15 Avg TiKV GRPC Duration

Average time consumed in executing gRPC requests in all TiKV instances based on the type, including `kv_get`, `kv_prewrite`, and `kv_commit`.

14.8.2.16 PD TSO Wait/RPC Duration

- wait - avg: Average time in waiting for PD to return TSO in all TiDB instances
- rpc - avg: Average time from sending TSO requests to PD to receiving TSO in all TiDB instances
- wait - 99: P99 time in waiting for PD to return TSO in all TiDB instances
- rpc - 99: P99 time from sending TSO requests to PD to receiving TSO in all TiDB instances

14.8.2.17 Storage Async Write Duration, Store Duration, and Apply Duration

- Storage Async Write Duration: Time consumed in asynchronous write
- Store Duration: Time consumed in store loop during asynchronously write
- Apply Duration: Time consumed in apply loop during asynchronously write

All these three metrics include the average duration and P99 duration in all TiKV instances.

Average storage async write duration = Average store duration + Average apply duration

14.8.2.18 Append Log Duration, Commit Log Duration, and Apply Log Duration

- Append Log Duration: Time consumed by Raft to append logs
- Commit Log Duration: Time consumed by Raft to commit logs
- Apply Log Duration: Time consumed by Raft to apply logs

All these three metrics include the average duration and P99 duration in all TiKV instances.

14.8.2.19 Interface of the Performance Overview dashboard

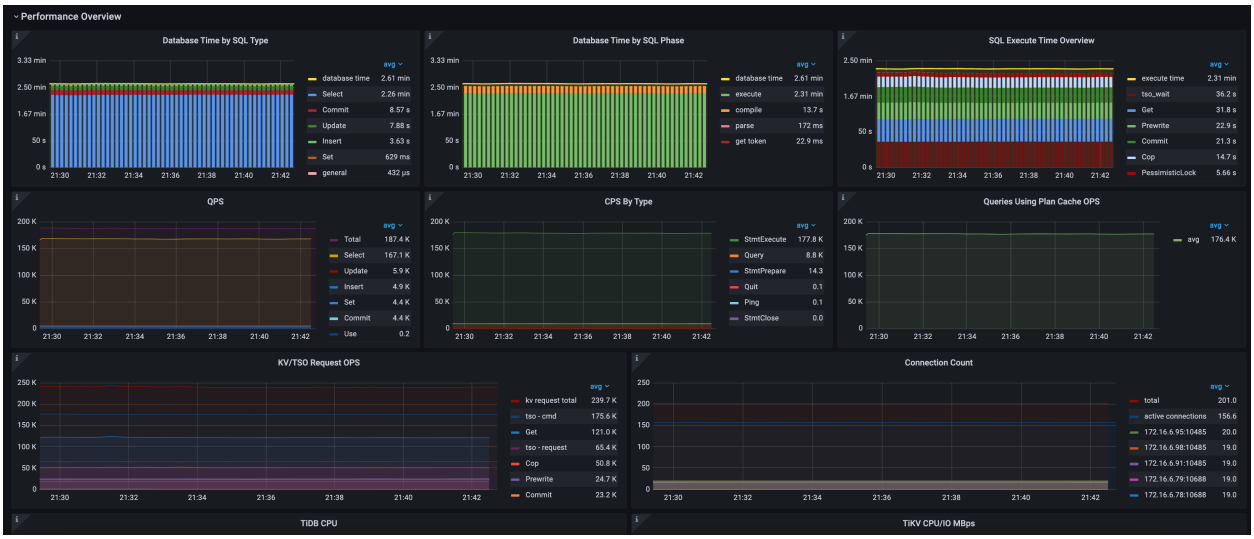


Figure 331: performance overview

14.8.3 TiDB Monitoring Metrics

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For the monitoring architecture, see TiDB Monitoring Framework Overview.

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node_exporter, Disk Performance, and Performance_overview. The TiDB dashboard consists of the TiDB panel and the TiDB Summary panel. The differences between the two panels are different in the following aspects:

- TiDB panel: provides as comprehensive information as possible for troubleshooting cluster anomalies.

- TiDB Summary Panel: extracts parts of the TiDB panel information with which users are most concerned, with some modifications. It provides data (such as QPS, TPS, response delay) that users care about in the daily database operations, which serves as the monitoring information to be displayed or reported.

This document describes some key monitoring metrics displayed on the TiDB dashboard.

14.8.3.1 Key metrics description

To understand the key metrics displayed on the TiDB dashboard, check the following list:

- Query Summary
  - Duration: execution time
    - The duration between the time that the client's network request is sent to TiDB and the time that the request is returned to the client after TiDB has executed it. In general, client requests are sent in the form of SQL statements, but can also include the execution time of commands such as `COM_PING`, `COM_SLEEP`, `COM_STMT_FETCH`, and `COM_SEND_LONG_DATA`
    - Because TiDB supports Multi-Query, it supports sending multiple SQL statements at one time, such as `select 1; select 1; select 1;`. In this case, the total execution time of this query includes the execution time of all SQL statements
  - Command Per Second: the number of commands processed by TiDB per second, which is classified according to the success or failure of command execution results
  - QPS: the number of SQL statements executed per second on all TiDB instances, which is counted according to `SELECT`, `INSERT`, `UPDATE`, and other types of statements
  - CPS By Instance: the command statistics on each TiDB instance, which is classified according to the success or failure of command execution results
  - Failed Query OPM: the statistics of error types (such as syntax errors and primary key conflicts) according to the errors occurred when executing SQL statements per minute on each TiDB instance. It contains the module in which the error occurs and the error code
  - Slow query: the statistics of the processing time of slow queries (the time cost of the entire slow query, the time cost of Coprocessor, and the waiting time for Coprocessor scheduling). Slow queries are classified into internal and general SQL statements
  - Connection Idle Duration: the duration of idle connections
  - 999/99/95/80 Duration: the statistics of the execution time for different types of SQL statements (different percentiles)
- Query Detail

    - Duration 80/95/99/999 By Instance: the statistics of the execution time for SQL statements on each TiDB instance (different percentiles)
    - Failed Query OPM Detail: the statistics of error types (such as syntax errors and primary key conflicts) according to the errors occurred when executing SQL statements per minute on each TiDB instance
    - Internal SQL OPS: the internal SQL statements executed per second in the entire TiDB cluster. The internal SQL statements are internally executed and are generally triggered by user SQL statements or internally scheduled tasks.

- Server

    - Uptime: the runtime of each TiDB instance
    - Memory Usage: the memory usage statistics of each TiDB instance, which is divided into the memory occupied by processes and the memory applied by Golang on the heap
    - CPU Usage: the statistics of CPU usage of each TiDB instance
    - Connection Count: the number of clients connected to each TiDB instance
    - Open FD Count: the statistics of opened file descriptors of each TiDB instance
    - Disconnection Count: the number of clients disconnected to each TiDB instance
    - Events OPM: the statistics of key events, such as “start”, “close”, “graceful-shutdown”,“kill”, and “hang”
    - Goroutine Count: the number of Goroutines on each TiDB instance
    - Prepare Statement Count: the number of `Prepare` statements that are executed on each TiDB instance and the total count of them
    - Keep Alive OPM: the number of times that the metrics are refreshed every minute on each TiDB instance. It usually needs no attention.
    - Panic And Critical Error: the number of panics and critical errors occurred in TiDB
    - Time Jump Back OPS: the number of times that the operating system rewinds every second on each TiDB instance
    - Get Token Duration: the time cost of getting Token on each connection
    - Skip Binlog Count: the number of binlog write failures in TiDB
    - Client Data Traffic: data traffic statistics of TiDB and the client

- Transaction

    - Transaction OPS: the number of transactions executed per second
    - Duration: the execution duration of a transaction
    - Transaction Statement Num: the number of SQL statements in a transaction
    - Transaction Retry Num: the number of times that a transaction retries
    - Session Retry Error OPS: the number of errors encountered during the transaction retry per second. This metric includes two error types: retry failure and exceeding the maximum number of retries
    - Commit Token Wait Duration: the wait duration in the flow control queue during the transaction commit. If the wait duration is long, it means that the transaction to commit is too large and the flow is controlled. If the system still has resources

available, you can speed up the commit process by increasing the system variable `tidb_committer_concurrency`.

- KV Transaction OPS: the number of transactions executed per second within each TiDB instance
  - A user transaction might trigger multiple transaction executions in TiDB, including reading internal metadata and atomic retries of the user transaction
  - TiDB's internally scheduled tasks also operate on the database through transactions, which are also included in this panel
- KV Transaction Duration: the time spent on executing transactions within each TiDB
- Transaction Regions Num: the number of Regions operated in the transaction
- Transaction Write KV Num Rate and Sum: the rate at which KVs are written and the sum of these written KVs in the transaction
- Transaction Write KV Num: the number of KVs operated in the transaction
- Statement Lock Keys: the number of locks for a single statement
- Send HeartBeat Duration: the duration for the transaction to send heartbeats
- Transaction Write Size Bytes Rate and sum: the rate at which bytes are written and the sum of these written bytes in the transaction
- Transaction Write Size Bytes: the size of the data written in the transaction
- Acquire Pessimistic Locks Duration: the time consumed by adding locks
- TTL Lifetime Reach Counter: the number of transactions that reach the upper limit of TTL. The default value of the TTL upper limit is 1 hour. It means that 1 hour has passed since the first lock of a pessimistic transaction or the first prewrite of an optimistic transaction. The default value of the upper limit of TTL is 1 hour. The upper limit of TTL life can be changed by modifying `max-txn-TTL` in the TiDB configuration file
- Load Safepoint OPS: the number of times that `Safepoint` is loaded. `Safepoint` is to ensure that the data before `Safepoint` is not read when the transaction reads data, thus ensuring data safety. The data before `Safepoint` might be cleaned up by the GC
- Pessimistic Statement Retry OPS: the number of retry attempts for pessimistic statements. When the statement tries to add lock, it might encounter a write conflict. At this time, the statement will acquire a new snapshot and add lock again
- Transaction Types Per Seconds: the number of transactions committed per second using the two-phase commit (2PC), async commit, and one-phase commit (1PC) mechanisms, including both success and failure transactions

- Executor
  - Parse Duration: the statistics of the parsing time of SQL statements
  - Compile Duration: the statistics of the time of compiling the parsed SQL AST to the execution plan
  - Execution Duration: the statistics of the execution time for SQL statements
  - Expensive Executor OPS: the statistics of the operators that consume many system resources per second, including `Merge Join`, `Hash Join`, `Index Look Up`

↪ `Join`, `Hash Agg`, `Stream Agg`, `Sort`, and `TopN`
    - Queries Using Plan Cache OPS: the statistics of queries using the Plan Cache per second
    - Plan Cache Miss OPS: the statistics of the number of times that the Plan Cache is missed per second

- Distsql
    - Distsql Duration: the processing time of Distsql statements
    - Distsql QPS: the statistics of Distsql statements
    - Distsql Partial QPS: the number of Partial results every second
    - Scan Keys Num: the number of keys that each query scans
    - Scan Keys Partial Num: the number of keys that each Partial result scans
    - Partial Num: the number of Partial results for each SQL statement

- KV Errors
    - KV Backoff Duration: the total duration that a KV retry request lasts. TiDB might encounter an error when sending a request to TiKV. TiDB has a retry mechanism for every request to TiKV. This `KV Backoff Duration` item records the total time of a request retry.
    - TiClient Region Error OPS: the number of Region related error messages returned by TiKV
    - KV Backoff OPS: the number of error messages returned by TiKV
    - Lock Resolve OPS: the number of TiDB operations to resolve locks. When TiDB's read or write request encounters a lock, it tries to resolve the lock
    - Other Errors OPS: the number of other types of errors, including clearing locks and updating `SafePoint`

- KV Request
    - KV Request OPS: the execution times of a KV request, displayed according to TiKV
    - KV Request Duration 99 by store: the execution time of a KV request, displayed according to TiKV
    - KV Request Duration 99 by type: the execution time of a KV request, displayed according to the request type

- PD Client
    - PD Client CMD OPS: the statistics of commands executed by PD Client per second
    - PD Client CMD Duration: the time it takes for PD Client to execute commands
    - PD Client CMD Fail OPS: the statistics of failed commands executed by PD Client per second
    - PD TSO OPS: the number of TSO that TiDB obtains from PD per second
    - PD TSO Wait Duration: the time that TiDB waits for PD to return TSO
    - PD TSO RPC duration: the duration from the time that TiDB sends request to PD (to get TSO) to the time that TiDB receives TSO

    - Start TSO Wait Duration: the duration from the time that TiDB sends request to PD (to get `start TSO`) to the time that TiDB receives `start TSO`

- Schema Load

    - Load Schema Duration: the time it takes TiDB to obtain the schema from TiKV
    - Load Schema OPS: the statistics of the schemas that TiDB obtains from TiKV per second
    - Schema Lease Error OPM: the Schema Lease errors include two types: `change` and `outdate`. `change` means that the schema has changed, and `outdate` means that the schema cannot be updated, which is a more serious error and triggers an alert.
    - Load Privilege OPS: the statistics of the number of privilege information obtained by TiDB from TiKV per second

- DDL

    - DDL Duration 95: 95% quantile of DDL statement processing time
    - Batch Add Index Duration 100: statistics of the maximum time spent by each Batch on creating an index
    - DDL Waiting Jobs Count: the number of DDL tasks that are waiting
    - DDL META OPM: the number of times that a DDL obtains META every minute
    - DDL Worker Duration 99: 99% quantile of the execution time of each DDL worker
    - Deploy Syncer Duration: the time consumed by Schema Version Syncer initialization, restart, and clearing up operations
    - Owner Handle Syncer Duration: the time that it takes the DDL Owner to update, obtain, and check the Schema Version
    - Update Self Version Duration: the time consumed by updating the version information of Schema Version Syncer
    - DDL OPM: the number of DDL executions per second
    - DDL backfill progress in percentage: the progress of backfilling DDL tasks

- Statistics

    - Auto Analyze Duration 95: the time consumed by automatic `ANALYZE`
    - Auto Analyze QPS: the statistics of automatic `ANALYZE`
    - Stats Inaccuracy Rate: the information of the statistics inaccuracy rate
    - Pseudo Estimation OPS: the number of the SQL statements optimized using pseudo statistics
    - Dump Feedback OPS: the number of stored statistical feedbacks
    - Store Query Feedback QPS: the number of operations per second to store the feedback information of the union query, which is performed in TiDB memory
    - Significant Feedback: the number of significant feedback pieces that update the statistics information
    - Update Stats OPS: the number of operations of updating statistics with feedback
    - Fast Analyze Status 100: the status for quickly collecting statistical information

- Owner

    - New ETCD Session Duration 95: the time it takes to create a new etcd session. TiDB connects to etcd in PD through etcd client to save/read some metadata information. This records the time spent creating the session
    - Owner Watcher OPS: the number of Goroutine operations per second of DDL owner watch PD's etcd metadata

- Meta

    - AutoID QPS: AutoID related statistics, including three operations (global ID allocation, a single table AutoID allocation, a single table AutoID Rebase)
    - AutoID Duration: the time consumed by AutoID related operations
    - Region Cache Error OPS: the number of errors encountered per second by the cached Region information in TiDB
    - Meta Operations Duration 99: the latency of Meta operations

- GC

    - Worker Action OPM: the number of GC related operations, including `run_job`, `resolve_lock`, and `delete_range`
    - Duration 99: the time consumed by GC related operations
    - Config: the configuration of GC data life time and GC running interval
    - GC Failure OPM: the number of failed GC related operations
    - Delete Range Failure OPM: the number of times the `Delete Range` has failed
    - Too Many Locks Error OPM: the number of the error that GC clears up too many locks
    - Action Result OPM: the number of results of GC-related operations
    - Delete Range Task Status: the task status of `Delete Range`, including completion and failure
    - Push Task Duration 95: the time spent pushing GC subtasks to GC workers

- Batch Client

    - Pending Request Count by TiKV: the number of Batch messages that are pending processing
    - Batch Client Unavailable Duration 95: the unavailable time of the Batch client
    - No Available Connection Counter: the number of times the Batch client cannot find an available link

14.8.4 Key Monitoring Metrics of PD

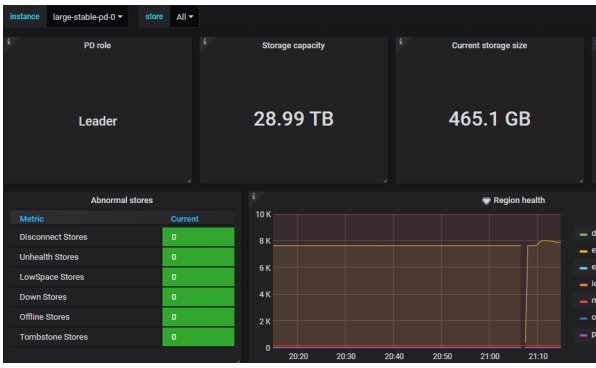
If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For more information, see Overview of the Monitoring Framework.

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node_exporter, Disk Performance, and Performance_overview. A lot of metrics are there to help you diagnose.

You can get an overview of the component PD status from the PD dashboard, where the key metrics are displayed. This document provides a detailed description of these key metrics.

The following is the description of PD Dashboard metrics items:

- PD role: The role of the current PD instance
- Storage capacity: The total storage capacity for this TiDB cluster
- Current storage size: The storage size that is currently used by the TiDB cluster
- Current storage usage: The current storage usage rate
- Normal stores: The count of healthy storage instances
- Number of Regions: The total count of cluster Regions
- Abnormal stores: The count of unhealthy stores. The normal value is `0`. If the number is bigger than `0`, it means at least one instance is abnormal.
- Region health: The health status of Regions indicated via the count of unusual Regions including pending peers, down peers, extra peers, offline peers, missing peers, learner peers and incorrect namespaces. Generally, the number of pending peers should be less than `100`. The missing peers should not be persistently greater than `0`. If many empty Regions exist, enable Region Merge in time.



- Current peer count: The current count of all cluster peers

14.8.4.1 Key metrics description

14.8.4.2 Cluster

- PD scheduler config: The list of PD scheduler configurations
- Cluster ID: The unique identifier of the cluster
- Current TSO: The physical part of current allocated TSO
- Current ID allocation: The maximum allocatable ID for new store/peer
- Region label isolation level: The number of Regions in different label levels
- Label distribution: The distribution status of the labels in the cluster
- Store Limit: The flow control limitation of scheduling on the Store

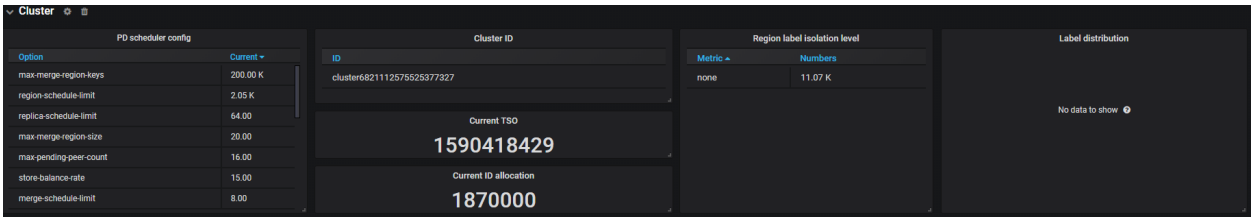


Figure 332: PD Dashboard - Cluster metrics

14.8.4.3 Operator

- Schedule operator create: The number of newly created operators per type
- Schedule operator check: The number of checked operator per type. It mainly checks whether the current step is finished; if yes, it returns the next step to be executed
- Schedule operator finish: The number of finished operators per type
- Schedule operator timeout: The number of timeout operators per type
- Schedule operator replaced or canceled: The number of replaced or canceled operators per type
- Schedule operators count by state: The number of operators per state
- Operator finish duration: The maximum duration of finished operators
- Operator step duration: The maximum duration of finished operator steps

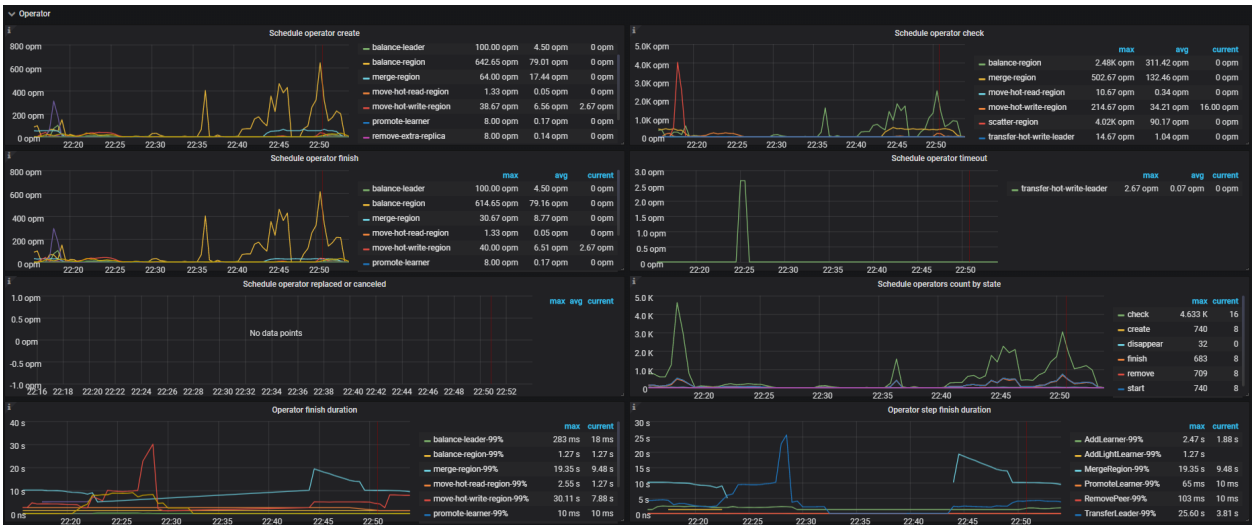


Figure 333: PD Dashboard - Operator metrics

14.8.4.4 Statistics - Balance

- Store capacity: The capacity size per TiKV instance

- Store available: The available capacity size per TiKV instance
- Store used: The used capacity size per TiKV instance
- Size amplification: The size amplification ratio per TiKV instance, which is equal to (Store Region size)/(Store used capacity size)
- Size available ratio: The size availability ratio per TiKV instance, which is equal to (Store available capacity size)/(Store capacity size)
- Store leader score: The leader score per TiKV instance
- Store Region score: The Region score per TiKV instance
- Store leader size: The total leader size per TiKV instance
- Store Region size: The total Region size per TiKV instance
- Store leader count: The leader count per TiKV instance
- Store Region count: The Region count per TiKV instance



Figure 334: PD Dashboard - Balance metrics

14.8.4.5 Statistics - hot write

- Hot Region's leader distribution: The total number of leader Regions that have become write hotspots on each TiKV instance
- Total written bytes on hot leader Regions: The total written bytes by leader Regions that have become write hotspots on each TiKV instance

- Hot write Region's peer distribution: The total number of peer Regions that have become write hotspots on each TiKV instance
- Total written bytes on hot peer Regions: The written bytes of all peer Regions that have become write hotspots on each TiKV instance
- Store Write rate bytes: The total written bytes on each TiKV instance
- Store Write rate keys: The total written keys on each TiKV instance
- Hot cache write entry number: The number of peers on each TiKV instance that are in the write hotspot statistics module
- Selector events: The event count of Selector in the hotspot scheduling module
- Direction of hotspot move leader: The direction of leader movement in the hotspot scheduling. The positive number means scheduling into the instance. The negtive number means scheduling out of the instance
- Direction of hotspot move peer: The direction of peer movement in the hotspot scheduling. The positive number means scheduling into the instance. The negative number means scheduling out of the instance



Figure 335: PD Dashboard - Hot write metrics

14.8.4.6 Statistics - hot read

- Hot Region's peer distribution: The total number of peer Regions that have become read hotspots on each TiKV instance
- Total read bytes on hot peer Regions: The total read bytes of peers that have become read hotspots on each TiKV instance
- Store read rate bytes: The total read bytes of each TiKV instance
- Store read rate keys: The total read keys of each TiKV instance
- Hot cache read entry number: The number of peers that are in the read hotspot statistics module on each TiKV instance



Figure 336: PD Dashboard - Hot read metrics

14.8.4.7 Scheduler

- Scheduler is running: The current running schedulers
- Balance leader movement: The leader movement details among TiKV instances
- Balance Region movement: The Region movement details among TiKV instances
- Balance leader event: The count of balance leader events
- Balance Region event: The count of balance Region events
- Balance leader scheduler: The inner status of balance leader scheduler
- Balance Region scheduler: The inner status of balance Region scheduler
- Replica checker: The replica checker's status
- Rule checker: The rule checker's status
- Region merge checker: The merge checker's status
- Filter target: The number of attempts that the store is selected as the scheduling target but failed to pass the filter
- Filter source: The number of attempts that the store is selected as the scheduling source but failed to pass the filter
- Balance Direction: The number of times that the Store is selected as the target or source of scheduling

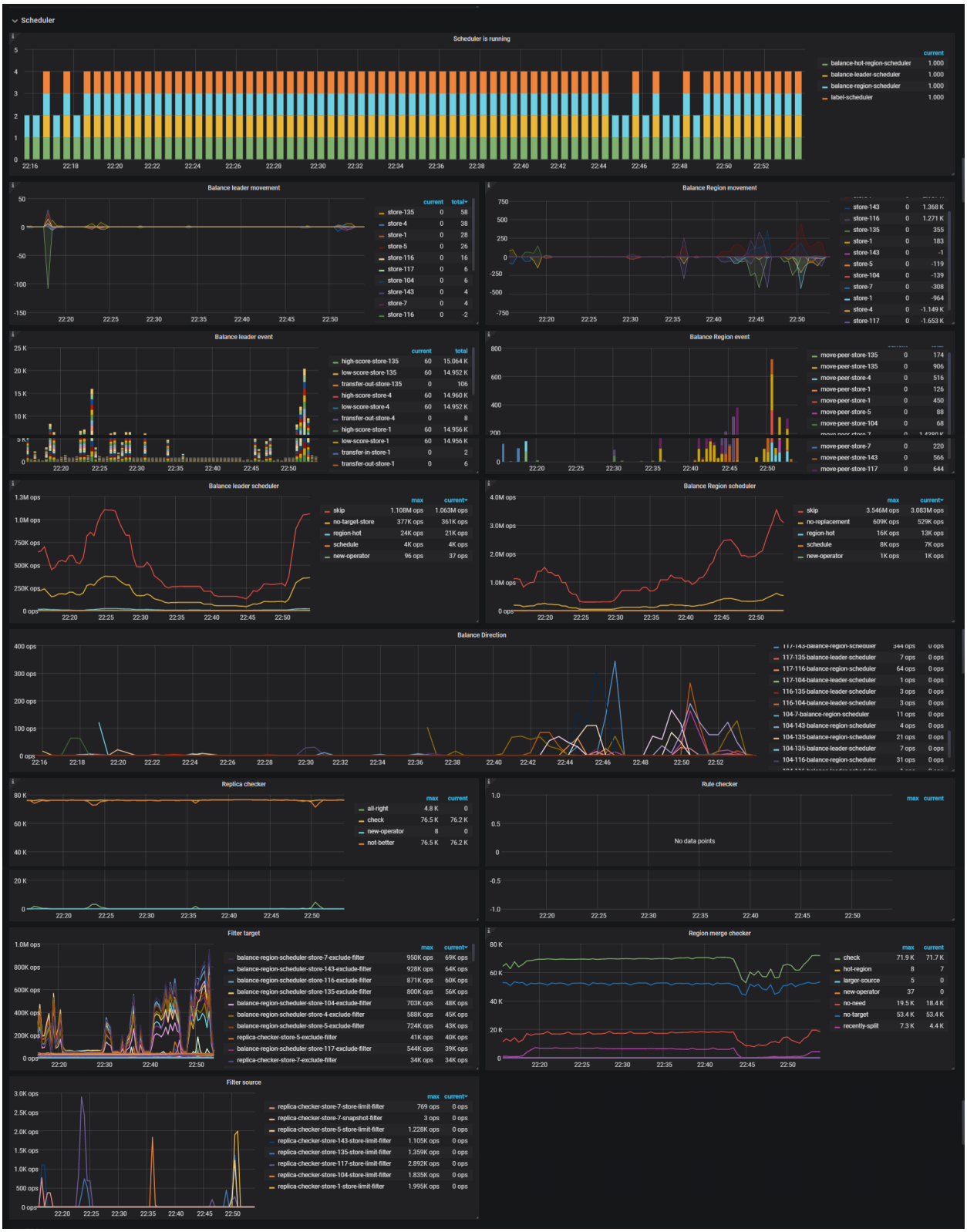


Figure 337: PD Dashboard - Scheduler metrics

14.8.4.8 gRPC

- Completed commands rate: The rate per command type at which gRPC commands are completed
- 99% Completed commands duration: The rate per command type at which gRPC commands are completed (P99)



Figure 338: PD Dashboard - gRPC metrics

14.8.4.9 etcd

- Handle transactions count: The rate at which etcd handles transactions
- 99% Handle transactions duration: The transaction handling rate (P99)
- 99% WAL fsync duration: The time consumed for writing WAL into the persistent storage. It is less than `1s` (P99)
- 99% Peer round trip time seconds: The network latency for etcd (P99) | The value is less than `1s`
- etcd disk WAL fsync rate: The rate of writing WAL into the persistent storage
- Raft term: The current term of Raft
- Raft committed index: The last committed index of Raft
- Raft applied index: The last applied index of Raft



Figure 339: PD Dashboard - etcd metrics

14.8.4.10 TiDB

- PD Server TSO handle time and Client recv time: The duration between PD receiving the TSO request and the PD client getting the TSO response
- Handle requests count: The count of TiDB requests
- Handle requests duration: The time consumed for handling TiDB requests. It should be less than `100ms` (P99)

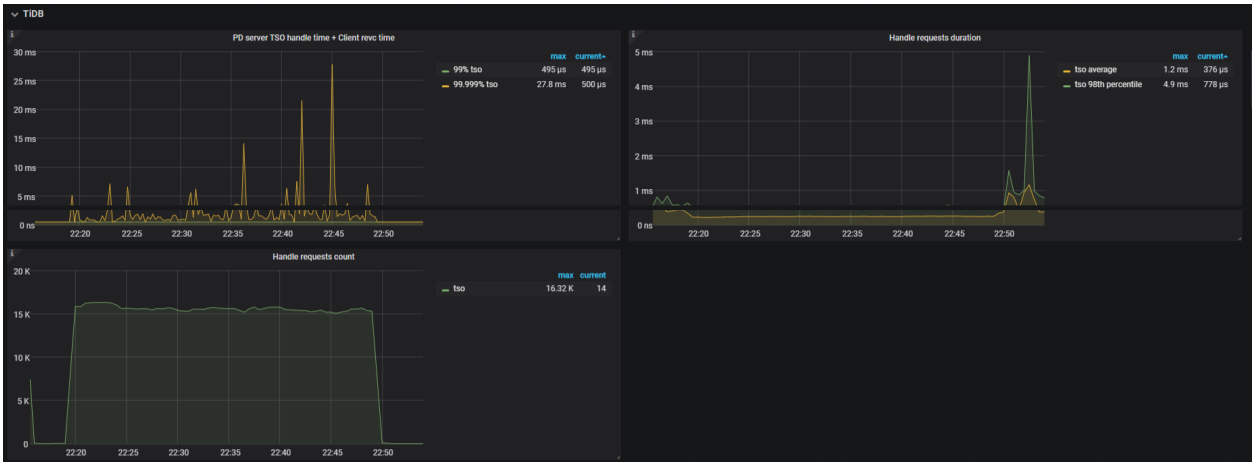


Figure 340: PD Dashboard - TiDB metrics

14.8.4.11 Heartbeat

- Heartbeat region event QPS: The QPS of handling heartbeat messages, including updating the cache and persisting data
- Region heartbeat report: The count of heartbeats reported to PD per instance
- Region heartbeat report error: The count of heartbeats with the `error` status
- Region heartbeat report active: The count of heartbeats with the `ok` status
- Region schedule push: The count of corresponding schedule commands sent from PD per TiKV instance
- 99% Region heartbeat latency: The heartbeat latency per TiKV instance (P99)

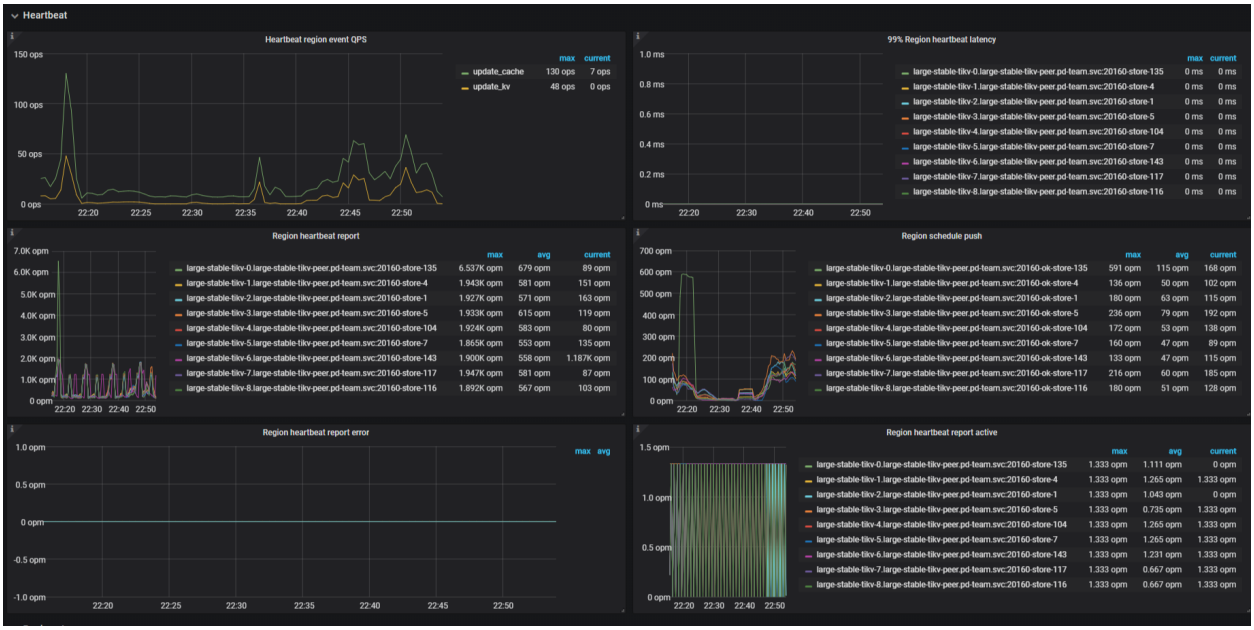


Figure 341: PD Dashboard - Heartbeat metrics

14.8.4.12 Region storage

- Syncer Index: The maximum index in the Region change history recorded by the leader
- history last index: The last index where the Region change history is synchronized successfully with the follower



Figure 342: PD Dashboard - Region storage

14.8.5 Key Monitoring Metrics of TiKV

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus/-Grafana) is deployed at the same time. For more information, see Overview of the Monitoring Framework.

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node_exporter, and Performance_overview. A lot of metrics are there to help you diagnose.

14.8.5.1 TiKV-Details dashboard

You can get an overview of the component TiKV status from the **TiKV-Details** dashboard, where the key metrics are displayed. According to the Performance Map, you can check whether the status of the cluster is as expected.

This section provides a detailed description of these key metrics on the **TiKV-Details** dashboard.

14.8.5.1.1 Cluster

- Store size: The storage size per TiKV instance
- Available size: The available capacity per TiKV instance
- Capacity size: The capacity size per TiKV instance
- CPU: The CPU utilization per TiKV instance
- Memory: The memory usage per TiKV instance
- IO utilization: The I/O utilization per TiKV instance
- MBps: The total bytes of read and write in each TiKV instance
- QPS: The QPS per command in each TiKV instance
- Errps: The rate of gRPC message failures
- leader: The number of leaders per TiKV instance
- Region: The number of Regions per TiKV instance
- Uptime: The runtime of TiKV since last restart

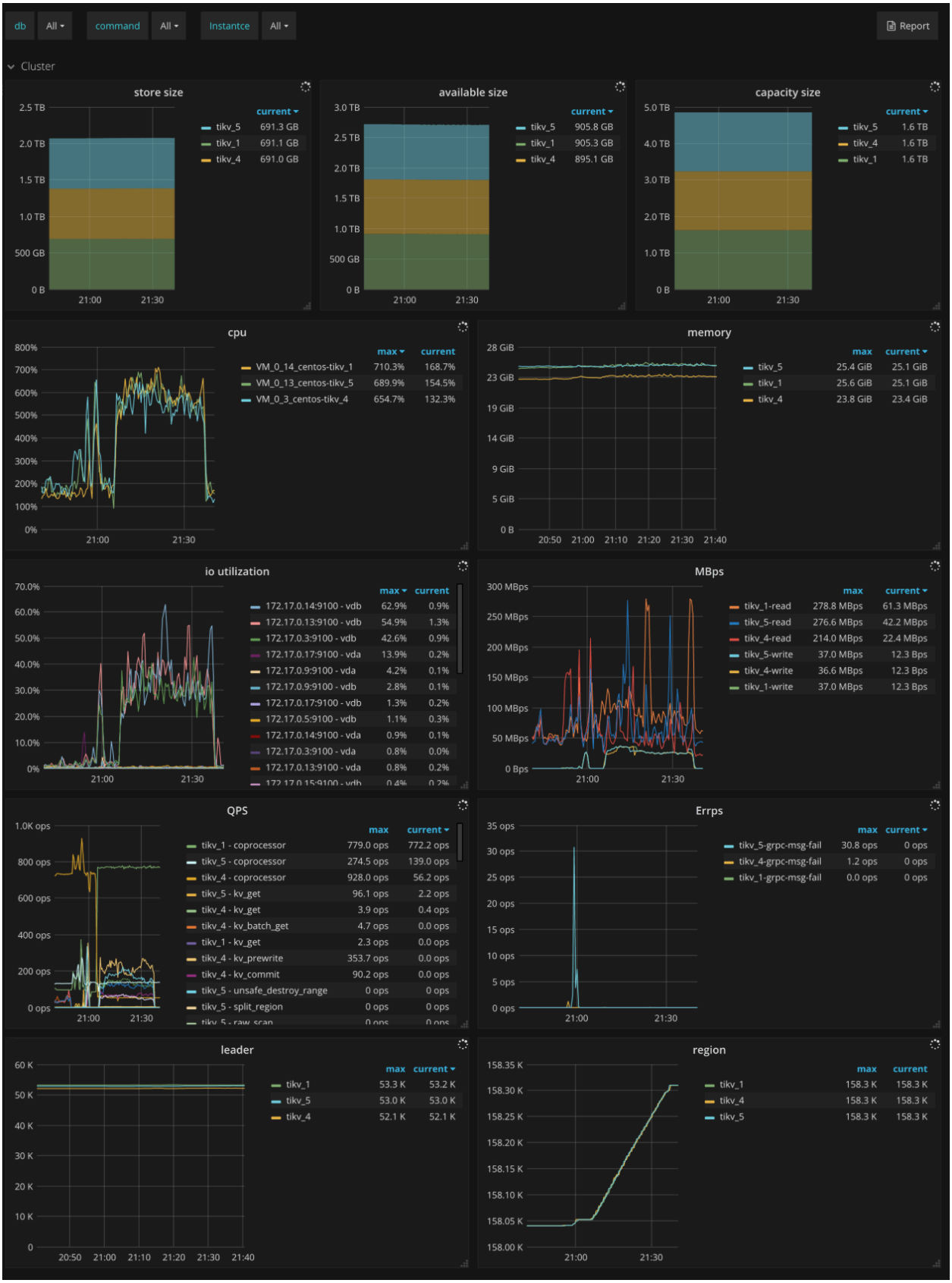


Figure 343: TiKV Dashboard - Cluster metrics

14.8.5.1.2 Errors

- Critical error: The number of critical errors
- Server is busy: Indicates occurrences of events that make the TiKV instance unavailable temporarily, such as Write Stall, and Channel Full. It should be `0` in normal case.
- Server report failures: The number of error messages reported by server. It should be `0` in normal case.
- Raftstore error: The number of Raftstore errors per type on each TiKV instance
- Scheduler error: The number of scheduler errors per type on each TiKV instance
- Coprocessor error: The number of coprocessor errors per type on each TiKV instance
- gRPC message error: The number of gRPC message errors per type on each TiKV instance
- Leader drop: The count of dropped leaders per TiKV instance
- Leader missing: The count of missing leaders per TiKV instance
- Log Replication Reject: The number of logappend messages rejected due to insufficient memory on each TiKV instance



Figure 344: TiKV Dashboard - Errors metrics

14.8.5.1.3 Server

- CF size: The size of each column family
- Store size: The storage size per TiKV instance
- Channel full: The number of Channel Full errors per TiKV instance. It should be `0` in normal case.
- Active written leaders: The number of leaders being written on each TiKV instance

- Approximate Region size: The approximate Region size
- Approximate Region size Histogram: The histogram of each approximate Region size
- Region average written keys: The average number of written keys to Regions per TiKV instance
- Region average written bytes: The average written bytes to Regions per TiKV instance

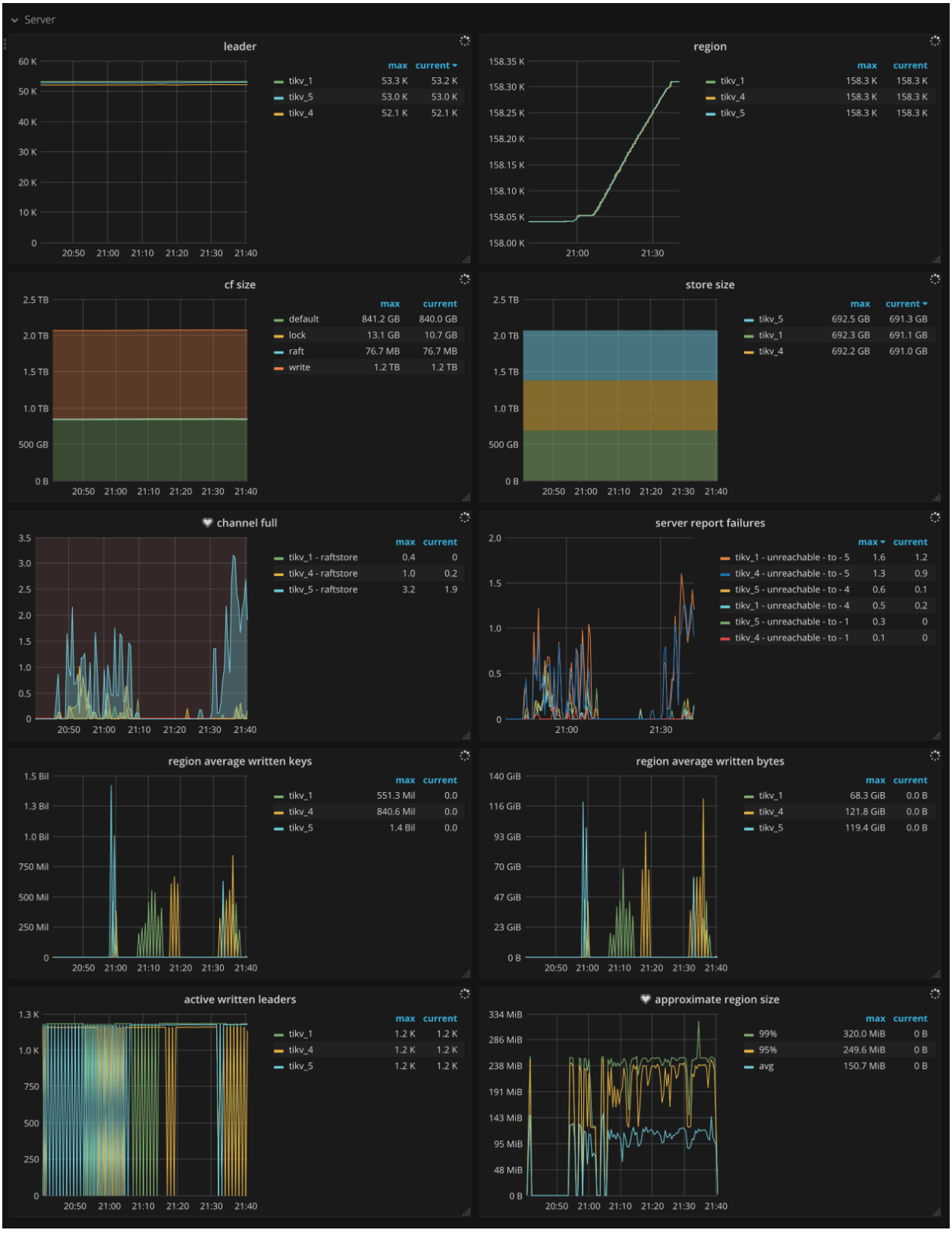


Figure 345: TiKV Dashboard - Server metrics

14.8.5.1.4 gRPC

- gRPC message count: The rate of gRPC messages per type
- gRPC message failed: The rate of failed gRPC messages
- 99% gRPC message duration: The gRPC message duration per message type (P99)
- Average gRPC message duration: The average execution time of gRPC messages
- gRPC batch size: The batch size of gRPC messages between TiDB and TiKV
- Raft message batch size: The batch size of Raft messages between TiKV instances

14.8.5.1.5 Thread CPU

- Raft store CPU: The CPU utilization of the `raftstore` thread. The CPU utilization should be less than 80% * `raftstore.store-pool-size` in normal case.
- Async apply CPU: The CPU utilization of the `async apply` thread. The CPU utilization should be less than 90% * `raftstore.apply-pool-size` in normal cases.
- Scheduler worker CPU: The CPU utilization of the `scheduler worker` thread. The CPU utilization should be less than 90% * `storage.scheduler-worker-pool-size` in normal cases.
- gRPC poll CPU: The CPU utilization of the `gRPC` thread. The CPU utilization should be less than 80% * `server.grpc-concurrency` in normal cases.
- Unified read pool CPU: The CPU utilization of the `unified read pool` thread
- Storage ReadPool CPU: The CPU utilization of the `storage read pool` thread
- Coprocessor CPU: The CPU utilization of the `coprocessor` thread
- RocksDB CPU: The CPU utilization of the RocksDB thread
- GC worker CPU: The CPU utilization of the `GC worker` thread
- BackGround worker CPU: The CPU utilization of the `background worker` thread

14.8.5.1.6 PD

- PD requests: The rate at which TiKV sends to PD
- PD request duration (average): The average duration of processing requests that TiKV sends to PD
- PD heartbeats: The rate at which heartbeat messages are sent from TiKV to PD
- PD validate peers: The rate at which messages are sent from TiKV to PD to validate TiKV peers

14.8.5.1.7 Raft IO

- Apply log duration: The time consumed for Raft to apply logs
- Apply log duration per server: The time consumed for Raft to apply logs per TiKV instance
- Append log duration: The time consumed for Raft to append logs

- Append log duration per server: The time consumed for Raft to append logs per TiKV instance
- Commit log duration: The time consumed by Raft to commit logs
- Commit log duration per server: The time consumed by Raft to commit logs per TiKV instance



Figure 346: TiKV Dashboard - Raft IO metrics

14.8.5.1.8 Raft process

- Ready handled: The number of handled ready operations per type per second
    - count: The number of handled ready operations per second
    - has_ready_region: The number of Regions that have ready per second
    - pending_region: The operations per second of the Regions being checked for whether it has ready. This metric is deprecated since v3.0.0
    - message: The number of messages that the ready operations per second contain
    - append: The number of Raft log entries that the ready operations per second contain
    - commit: The number of committed Raft log entries that the ready operations per second contain
    - snapshot: The number of snapshots that the ready operations per second contains
- 0.99 Duration of Raft store events: The time consumed by Raftstore events (P99)
- Process ready duration: The time consumed for processes to be ready in Raft
- Process ready duration per server: The time consumed for peer processes to be ready in Raft per TiKV instance. It should be less than 2 seconds (P99.99).



Figure 347: TiKV Dashboard - Raft process metrics

14.8.5.1.9 Raft message

- Sent messages per server: The number of Raft messages sent by each TiKV instance per second
- Flush messages per server: The number of Raft messages flushed by the Raft client in each TiKV instance per second
- Receive messages per server: The number of Raft messages received by each TiKV instance per second
- Messages: The number of Raft messages sent per type per second
- Vote: The number of Vote messages sent in Raft per second
- Raft dropped messages: The number of dropped Raft messages per type per second

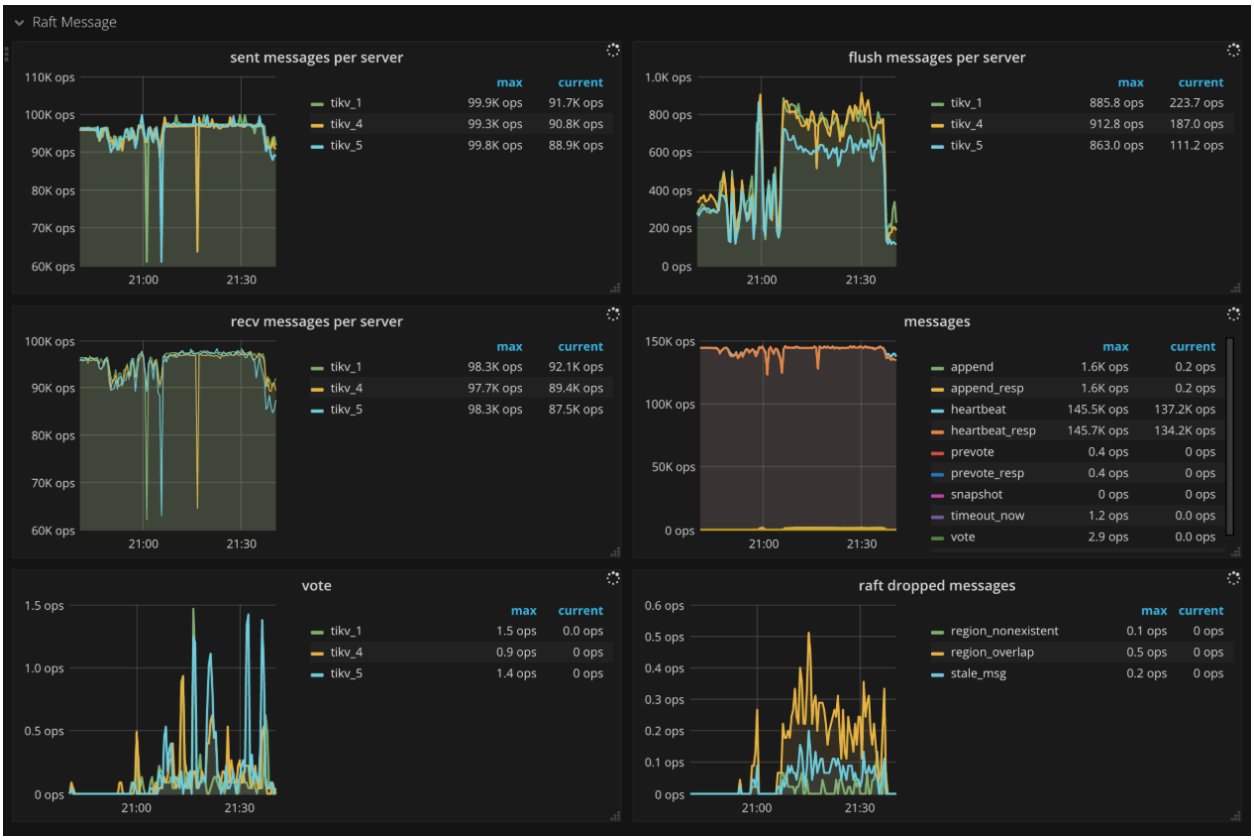


Figure 348: TiKV Dashboard - Raft message metrics

14.8.5.1.10 Raft propose

- Raft apply proposals per ready: The histogram of the number of proposals that each ready operation contains in a batch while applying proposal.
- Raft read/write proposals: The number of proposals per type per second
- Raft read proposals per server: The number of read proposals made by each TiKV instance per second
- Raft write proposals per server: The number of write proposals made by each TiKV instance per second
- Propose wait duration: The histogram of waiting time of each proposal
- Propose wait duration per server: The histogram of waiting time of each proposal per TiKV instance
- Apply wait duration: The histogram of apply time of each proposal
- Apply wait duration per server: The histogram of apply time of each proposal per TiKV instance
- Raft log speed: The average rate at which peers propose logs

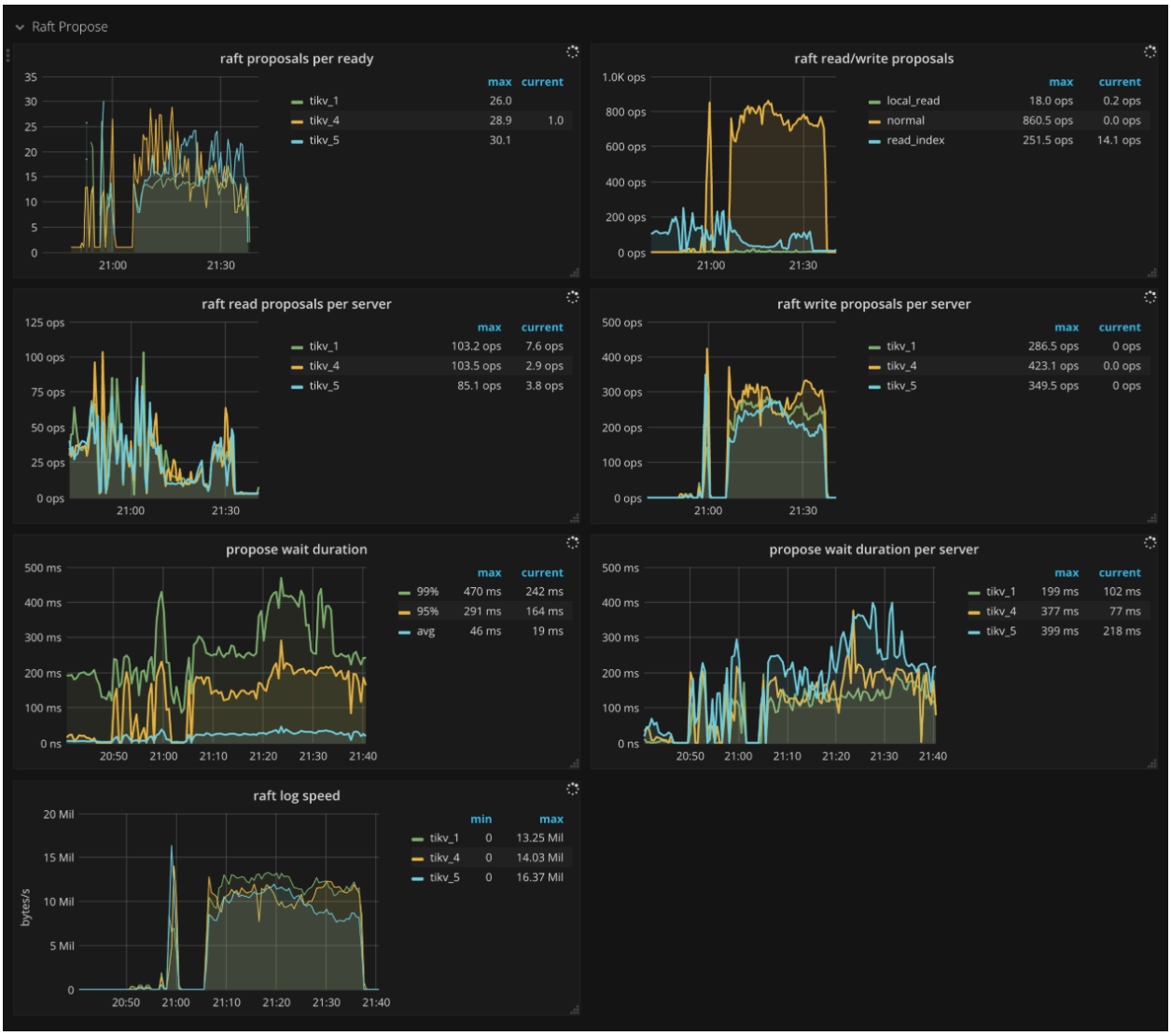


Figure 349: TiKV Dashboard - Raft propose metrics

14.8.5.1.11 Raft admin

- Admin proposals: The number of admin proposals per second
- Admin apply: The number of processed apply commands per second
- Check split: The number of Raftstore split check commands per second
- 99.99% Check split duration: The time consumed when running split check commands (P99.99)

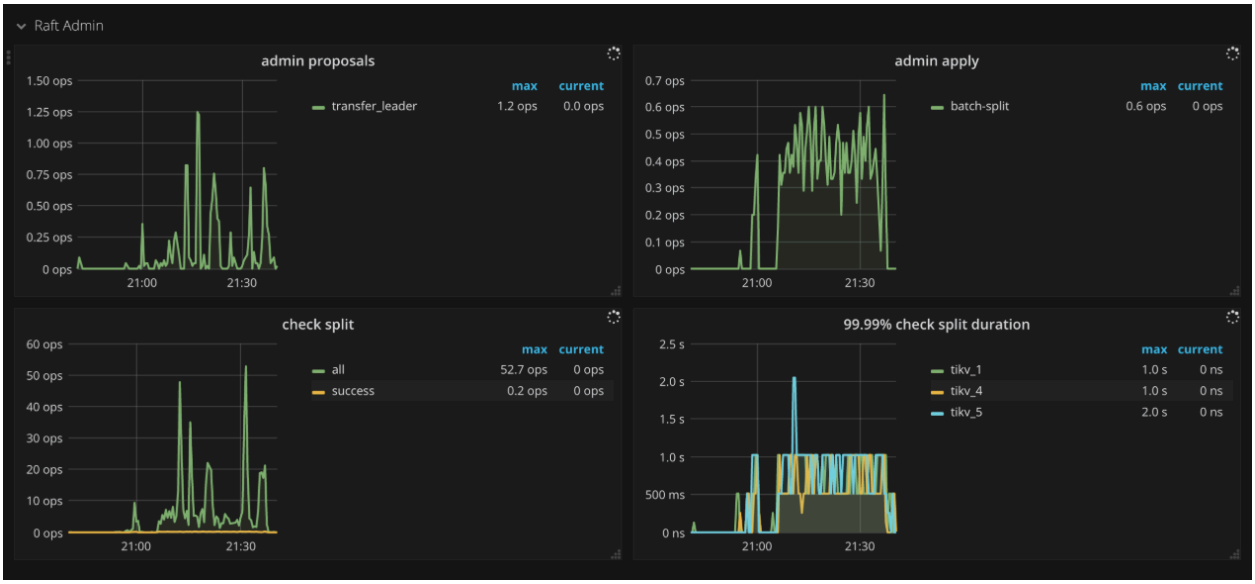


Figure 350: TiKV Dashboard - Raft admin metrics

14.8.5.1.12 Local reader

- Local reader requests: The number of total requests and the number of rejections from the local read thread



Figure 351: TiKV Dashboard - Local reader metrics

14.8.5.1.13 Unified Read Pool

- Time used by level: The time consumed for each level in the unified read pool. Level 0 means small queries.
- Level 0 chance: The proportion of level 0 tasks in unified read pool
- Running tasks: The number of tasks running concurrently in the unified read pool

14.8.5.1.14 Storage

- Storage command total: The number of received command by type per second
- Storage async request error: The number of engine asynchronous request errors per second
- Storage async snapshot duration: The time consumed by processing asynchronous snapshot requests. It should be less than `1s` in `.99`.
- Storage async write duration: The time consumed by processing asynchronous write requests. It should be less than `1s` in `.99`.

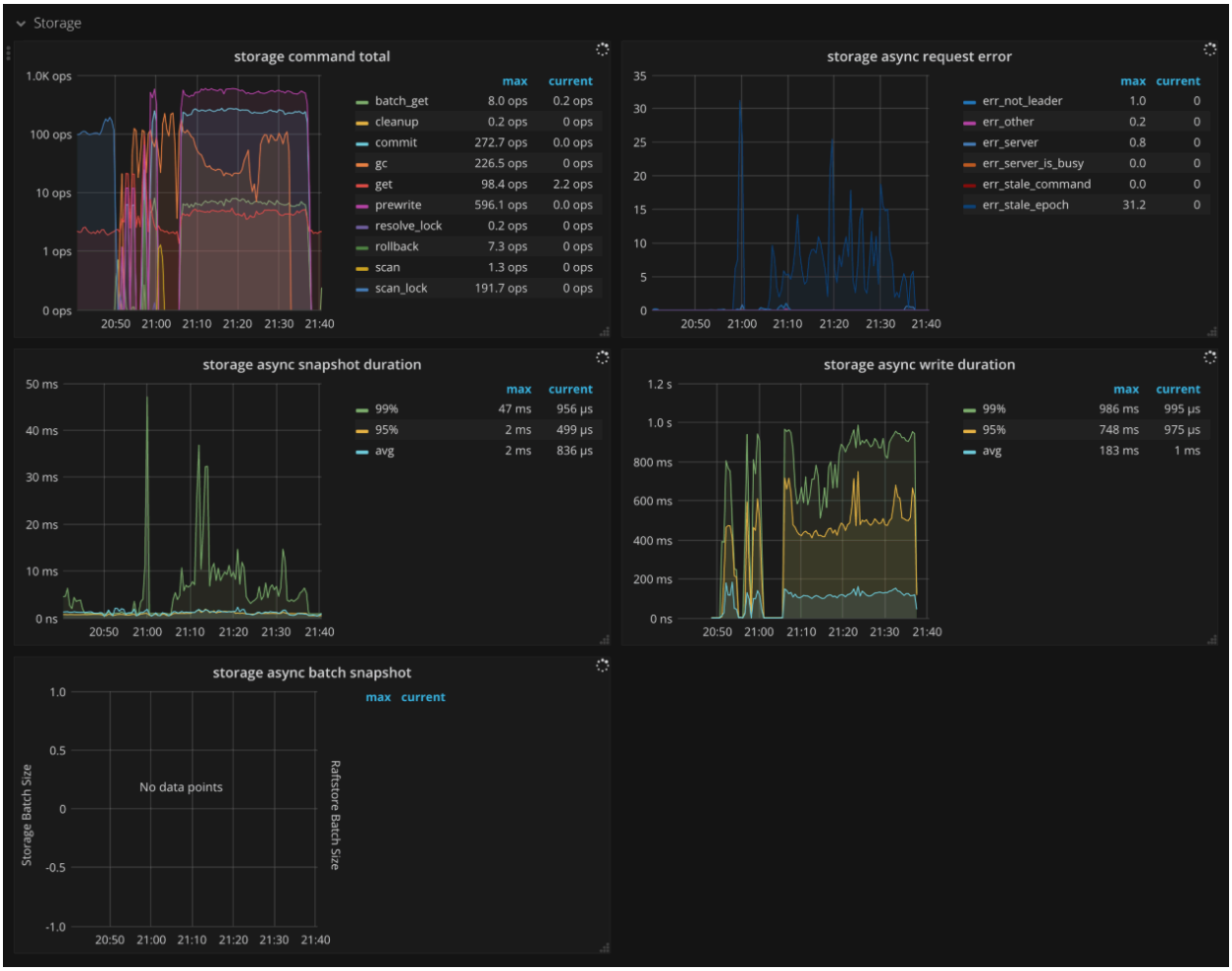


Figure 352: TiKV Dashboard - Storage metrics

14.8.5.1.15 Scheduler

- Scheduler stage total: The number of commands at each stage per second. There should not be a lot of errors in a short time.
- Scheduler writing bytes: The total written bytes by commands processed on each TiKV instance
- Scheduler priority commands: The count of different priority commands per second
- Scheduler pending commands: The count of pending commands per TiKV instance per second

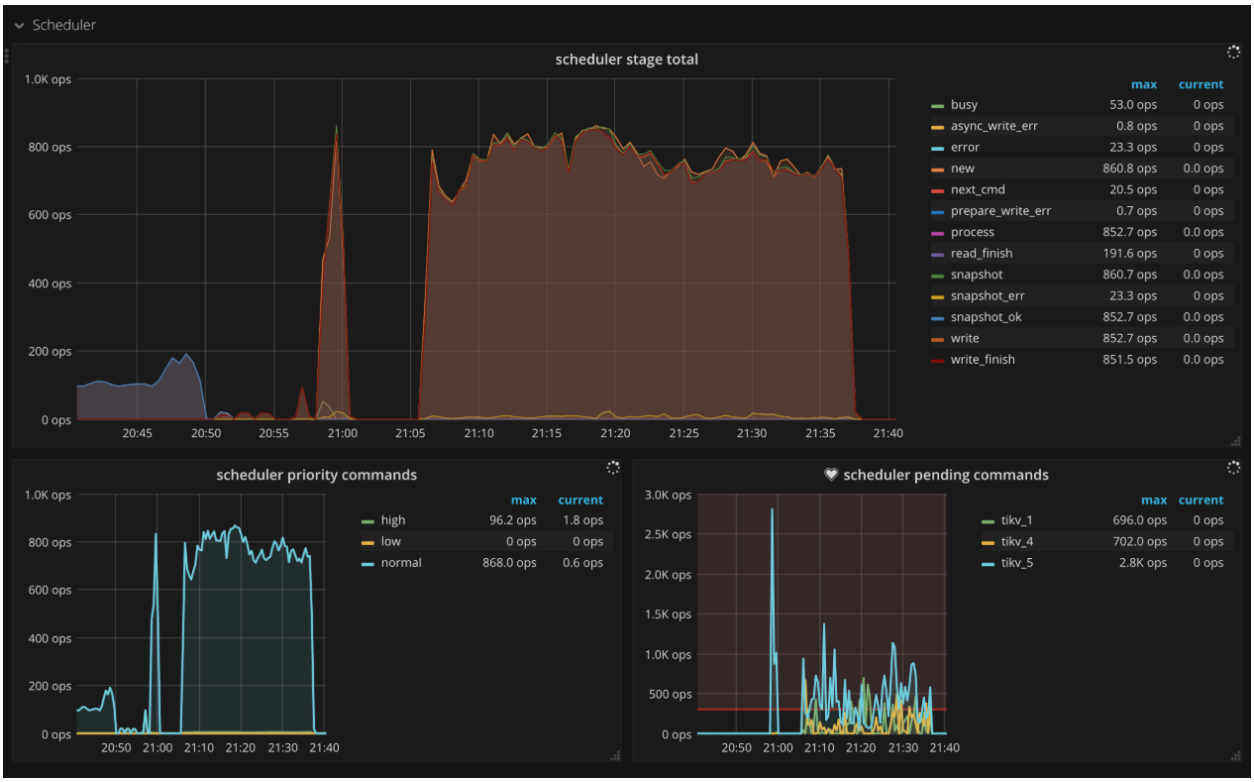


Figure 353: TiKV Dashboard - Scheduler metrics

14.8.5.1.16 Scheduler - commit

- Scheduler stage total: The number of commands at each stage per second when executing the commit command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the commit command. It should be less than `1s`.
- Scheduler latch wait duration: The waiting time caused by latch when executing the commit command. It should be less than `1s`.
- Scheduler keys read: The count of keys read by a commit command
- Scheduler keys written: The count of keys written by a commit command
- Scheduler scan details: The keys scan details of each CF when executing the commit command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the commit command
- Scheduler scan details write: The keys scan details of write CF when executing the commit command
- Scheduler scan details [default]: The keys scan details of default CF when executing the commit command



Figure 354: TiKV Dashboard - Scheduler commit metrics

14.8.5.1.17 Scheduler - pessimistic_rollback

- Scheduler stage total: The number of commands at each stage per second when executing the `pessimistic_rollback` command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the `pessimistic_rollback` command. It should be less than `1s`.

- Scheduler latch wait duration: The waiting time caused by latch when executing the `pessimistic_rollback` command. It should be less than `1s`.
- Scheduler keys read: The count of keys read by a `pessimistic_rollback` command
- Scheduler keys written: The count of keys written by a `pessimistic_rollback` command
- Scheduler scan details: The keys scan details of each CF when executing the `pessimistic_rollback` command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the `pessimistic_rollback` command
- Scheduler scan details write: The keys scan details of write CF when executing the `pessimistic_rollback` command
- Scheduler scan details [default]: The keys scan details of default CF when executing the `pessimistic_rollback` command

14.8.5.1.18 Scheduler - prewrite

- Scheduler stage total: The number of commands at each stage per second when executing the prewrite command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the prewrite command. It should be less than `1s`.
- Scheduler latch wait duration: The waiting time caused by latch when executing the prewrite command. It should be less than `1s`.
- Scheduler keys read: The count of keys read by a prewrite command
- Scheduler keys written: The count of keys written by a prewrite command
- Scheduler scan details: The keys scan details of each CF when executing the prewrite command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the prewrite command
- Scheduler scan details write: The keys scan details of write CF when executing the prewrite command
- Scheduler scan details [default]: The keys scan details of default CF when executing the prewrite command

14.8.5.1.19 Scheduler - rollback

- Scheduler stage total: The number of commands at each stage per second when executing the rollback command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the rollback command. It should be less than `1s`.
- Scheduler latch wait duration: The waiting time caused by latch when executing the rollback command. It should be less than `1s`.
- Scheduler keys read: The count of keys read by a rollback command
- Scheduler keys written: The count of keys written by a rollback command

- Scheduler scan details: The keys scan details of each CF when executing the rollback command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the rollback command
- Scheduler scan details write: The keys scan details of write CF when executing the rollback command
- Scheduler scan details [default]: The keys scan details of default CF when executing the rollback command

14.8.5.1.20 GC

- GC tasks: The count of GC tasks processed by gc_worker
- GC tasks Duration: The time consumed when executing GC tasks
- TiDB GC seconds: The GC duration
- TiDB GC worker actions: The count of TiDB GC worker actions
- ResolveLocks Progress: The progress of the first phase of GC (Resolve Locks)
- TiKV Auto GC Progress: The progress of the second phase of GC
- GC speed: The number of keys deleted by GC per second
- TiKV Auto GC SafePoint: The value of TiKV GC safe point. The safe point is the current GC timestamp
- GC lifetime: The lifetime of TiDB GC
- GC interval: The interval of TiDB GC
- GC in Compaction Filter: The count of filtered versions in the compaction filter of write CF.

14.8.5.1.21 Snapshot

- Rate snapshot message: The rate at which Raft snapshot messages are sent
- 99% Handle snapshot duration: The time consumed to handle snapshots (P99)
- Snapshot state count: The number of snapshots per state
- 99.99% Snapshot size: The snapshot size (P99.99)
- 99.99% Snapshot KV count: The number of KV within a snapshot (P99.99)

14.8.5.1.22 Task

- Worker handled tasks: The number of tasks handled by worker per second
- Worker pending tasks: Current number of pending and running tasks of worker per second. It should be less than `1000` in normal case.
- FuturePool handled tasks: The number of tasks handled by future pool per second
- FuturePool pending tasks: Current number of pending and running tasks of future pool per second

14.8.5.1.23 Coprocessor Overview

- Request duration: The total duration from the time of receiving the coprocessor request to the time of finishing processing the request
- Total Requests: The number of requests by type per second
- Handle duration: The histogram of time spent actually processing coprocessor requests per minute
- Total Request Errors: The number of request errors of Coprocessor per second. There should not be a lot of errors in a short time.
- Total KV Cursor Operations: The total number of the KV cursor operations by type per second, such as `select`, `index`, `analyze_table`, `analyze_index`, `checksum_table` ↪ , and `checksum_index`.
- KV Cursor Operations: The histogram of KV cursor operations by type per second
- Total RocksDB Perf Statistics: The statistics of RocksDB performance
- Total Response Size: The total size of coprocessor response

14.8.5.1.24 Coprocessor Detail

- Handle duration: The histogram of time spent actually processing coprocessor requests per minute
- 95% Handle duration by store: The time consumed to handle coprocessor requests per TiKV instance per second (P95)
- Wait duration: The time consumed when coprocessor requests are waiting to be handled. It should be less than `10s` (P99.99).
- 95% Wait duration by store: The time consumed when coprocessor requests are waiting to be handled per TiKV instance per second (P95)
- Total DAG Requests: The total number of DAG requests per second
- Total DAG Executors: The total number of DAG executors per second
- Total Ops Details (Table Scan): The number of RocksDB internal operations per second when executing select scan in coprocessor
- Total Ops Details (Index Scan): The number of RocksDB internal operations per second when executing index scan in coprocessor
- Total Ops Details by CF (Table Scan): The number of RocksDB internal operations for each CF per second when executing select scan in coprocessor
- Total Ops Details by CF (Index Scan): The number of RocksDB internal operations for each CF per second when executing index scan in coprocessor

14.8.5.1.25 Threads

- Threads state: The state of TiKV threads
- Threads IO: The I/O traffic of each TiKV thread
- Thread Voluntary Context Switches: The number of TiKV threads voluntary context switches
- Thread Nonvoluntary Context Switches: The number of TiKV threads nonvoluntary context switches

14.8.5.1.26 RocksDB - kv/raft

- Get operations: The count of get operations per second
- Get duration: The time consumed when executing get operations
- Seek operations: The count of seek operations per second
- Seek duration: The time consumed when executing seek operations
- Write operations: The count of write operations per second
- Write duration: The time consumed when executing write operations
- WAL sync operations: The count of WAL sync operations per second
- Write WAL duration: The time consumed for writing WAL
- WAL sync duration: The time consumed when executing WAL sync operations
- Compaction operations: The count of compaction and flush operations per second
- Compaction duration: The time consumed when executing the compaction and flush operations
- SST read duration: The time consumed when reading SST files
- Write stall duration: Write stall duration. It should be `0` in normal case.
- Memtable size: The memtable size of each column family
- Memtable hit: The hit rate of memtable
- Block cache size: The block cache size. Broken down by column family if shared block cache is disabled.
- Block cache hit: The hit rate of block cache
- Block cache flow: The flow rate of block cache operations per type
- Block cache operations: The count of block cache operations per type
- Keys flow: The flow rate of operations on keys per type
- Total keys: The count of keys in each column family
- Read flow: The flow rate of read operations per type
- Bytes / Read: The bytes per read operation
- Write flow: The flow rate of write operations per type
- Bytes / Write: The bytes per write operation
- Compaction flow: The flow rate of compaction operations per type
- Compaction pending bytes: The pending bytes to be compacted
- Read amplification: The read amplification per TiKV instance
- Compression ratio: The compression ratio of each level
- Number of snapshots: The number of snapshots per TiKV instance
- Oldest snapshots duration: The time that the oldest unreleased snapshot survivals
- Number files at each level: The number of SST files for different column families in each level
- Ingest SST duration seconds: The time consumed to ingest SST files
- Stall conditions changed of each CF: Stall conditions changed of each column family

14.8.5.1.27 Titan - All

- Blob file count: The number of Titan blob files
- Blob file size: The total size of Titan blob file

- Live blob size: The total size of valid blob record
- Blob cache hit: The hit rate of Titan block cache
- Iter touched blob file count: The number of blob file involved in a single iterator
- Blob file discardable ratio distribution: The ratio distribution of blob record failure of blob files
- Blob key size: The size of Titan blob keys
- Blob value size: The size of Titan blob values
- Blob get operations: The count of get operations in Titan blob
- Blob get duration: The time consumed when executing get operations in Titan blob
- Blob iter operations: The time consumed when executing iter operations in Titan blob
- Blob seek duration: The time consumed when executing seek operations in Titan blob
- Blob next duration: The time consumed when executing next operations in Titan blob
- Blob prev duration: The time consumed when executing prev operations in Titan blob
- Blob keys flow: The flow rate of operations on Titan blob keys
- Blob bytes flow: The flow rate of bytes on Titan blob keys
- Blob file read duration: The time consumed when reading Titan blob file
- Blob file write duration: The time consumed when writing Titan blob file
- Blob file sync operations: The count of blob file sync operations
- Blob file sync duration: The time consumed when synchronizing blob file
- Blob GC action: The count of Titan GC actions
- Blob GC duration: The Titan GC duration
- Blob GC keys flow: The flow rate of keys read and written by Titan GC
- Blob GC bytes flow: The flow rate of bytes read and written by Titan GC
- Blob GC input file size: The size of Titan GC input file
- Blob GC output file size: The size of Titan GC output file
- Blob GC file count: The count of blob files involved in Titan GC

14.8.5.1.28 Pessimistic Locking

- Lock Manager Thread CPU: The CPU utilization of the lock manager thread
- Lock Manager Handled tasks: The number of tasks handled by lock manager
- Waiter lifetime duration: The waiting time of the transaction for the lock to be released
- Wait table: The status information of wait table, including the number of locks and the number of transactions waiting for the lock
- Deadlock detect duration: The time consumed for detecting deadlock
- Detect error: The number of errors encountered when detecting deadlock, including the number of deadlocks
- Deadlock detector leader: The information of the node where the deadlock detector leader is located
- Total pessimistic locks memory size: The memory size occupied by the in-memory pessimistic locks
- In-memory pessimistic locking result: The result of only saving pessimistic locks to memory. `full` means the number of times that the pessimistic lock is not saved to memory because the memory limit is exceeded.

14.8.5.1.29 Memory

- Allocator Stats: The statistics of the memory allocator

14.8.5.1.30 Backup

- Backup CPU: The CPU utilization of the backup thread
- Range Size: The histogram of backup range size
- Backup Duration: The time consumed for backup
- Backup Flow: The total bytes of backup
- Disk Throughput: The disk throughput per instance
- Backup Range Duration: The time consumed for backing up a range
- Backup Errors: The number of errors encountered during a backup

14.8.5.1.31 Encryption

- Encryption data keys: The total number of encrypted data keys
- Encrypted files: The number of encrypted files
- Encryption initialized: Shows whether encryption is enabled. `1` means enabled.
- Encryption meta files size: The size of the encryption meta file
- Encrypt/decrypt data nanos: The histogram of duration on encrypting/decrypting data each time
- Read/write encryption meta duration: The time consumed for reading/writing encryption meta files

14.8.5.1.32 Explanation of Common Parameters

gRPC Message Type

1. Transactional API:

    - kv_get: The command of getting the latest version of data specified by `ts`
    - kv_scan: The command of scanning a range of data
    - kv_prewrite: The command of prewriting the data to be committed at first phase of 2PC
    - kv_pessimistic_lock: The command of adding a pessimistic lock to the key to prevent other transaction from modifying this key
    - kv_pessimistic_rollback: The command of deleting the pessimistic lock on the key
    - kv_txn_heart_beat: The command of updating `lock_ttl` for pessimistic transactions or large transactions to prevent them from rolling back
    - kv_check_txn_status: The command of checking the status of the transaction
    - kv_commit: The command of committing the data written by the prewrite command

- kv_cleanup: The command of rolling back a transaction, which is deprecated in v4.0
- kv_batch_get: The command of getting the value of batch key at once, similar to `kv_get`
- kv_batch_rollback: The command of batch rollback of multiple prewrite transactions
- kv_scan_lock: The command of scanning all locks with a version number before `max_version` to clean up expired transactions
- kv_resolve_lock: The command of committing or rollback the transaction lock, according to the transaction status.
- kv_gc: The command of GC
- kv_delete_range: The command of deleting a range of data from TiKV

2. Raw API:

- raw_get: The command of getting the value of key
- raw_batch_get: The command of getting the value of batch keys
- raw_scan: The command of scanning a range of data
- raw_batch_scan: The command of scanning multiple consecutive data range
- raw_put: The command of writing a key/value pair
- raw_batch_put: The command of writing a batch of key/value pairs
- raw_delete: The command of deleting a key/value pair
- raw_batch_delete: The command of a batch of key/value pairs
- raw_delete_range: The command of deleting a range of data

14.8.5.2 TiKV-FastTune dashboard

If performance issues of TiKV occur, such as QPS jitter, latency jitter, and latency increasing trend, you can check the **TiKV-FastTune** dashboard. This dashboard contains a set of panels that help you with diagnostics, especially when the write workload in your cluster is medium or large.

When write-related performance issues occur, you can first check the TiDB-related dashboards. If the issues are at the storage side, open the **TiKV-FastTune** page, browse and check every panel on it.

In the **TiKV-FastTune** dashboard, you can see a title that suggests a possible cause of the performance issues. To check whether the suggested cause is true, check the graph on the page.

The left-Y-axis of the graph represents the write-RPC QPS of the storage side, and a set of graphs on the right-Y-axis are drawn upside down. If the shape of the left graph matches that of the right graphs, the suggested cause is true.

For detailed metrics and descriptions, see the dashboard user manual.

14.8.6 Monitor the TiFlash Cluster

This document describes the monitoring items of TiFlash.

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For more information, see Overview of the Monitoring Framework.

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, and Node_exporter. A lot of metrics are there to help you diagnose.

TiFlash has three dashboard panels: **TiFlash-Summary**, **TiFlash-Proxy-Summary**, and **TiFlash-Proxy-Details**. The metrics on these panels indicate the current status of TiFlash. The **TiFlash-Proxy-Summary** and **TiFlash-Proxy-Details** panels mainly show the information of the Raft layer and the metrics are detailed in Key Monitoring Metrics of TiKV.

> **Note:**
>
> It is recommended that you use TiDB v4.0.5 or later versions for improved monitor on TiFlash.

The following sections introduce the default monitoring information of **TiFlash-Summary**.

14.8.6.1 Server

- Store size: The storage size used by each TiFlash instance.
- Available size: The storage size available for each TiFlash instance.
- Capacity size: The storage capacity for each TiFlash instance.
- Uptime: The runtime of TiFlash since last restart.
- Memory: The memory usage per TiFlash instance.
- CPU Usage: The CPU utilization per TiFlash instance.
- FSync OPS: The number of fsync operations per TiFlash instance per second.
- File Open OPS: The number of `open` operations per TiFlash instance per second.
- Opened File Count: The number of file descriptors currently opened by each TiFlash instance.

> **Note:**
>
> Store size, FSync OPS, File Open OPS, and Opened File Count currently only cover the monitoring information of the TiFlash storage layer and do not cover that in TiFlash-Proxy.

14.8.6.2 Coprocessor

- Request QPS: The number of coprocessor requests received by all TiFlash instances. `batch` is the number of batch requests. `batch_cop` is the number of coprocessor requests in the batch requests. `cop` is the number of coprocessor requests that are sent directly via the coprocessor interface. `cop_dag` is the number of dag requests in all coprocessor requests. `super_batch` is the number of requests to enable the Super Batch feature.
- Executor QPS: The number of each type of dag executors in the requests received by all TiFlash instances. `table_scan` is the table scan executor. `selection` is the selection executor. `aggregation` is the aggregation executor. `top_n` is the `TopN` executor. `limit` is the limit executor.
- Request Duration: The total duration of all TiFlash instances processing coprocessor requests. The total duration is from the time that the coprocessor request is received to the time that the response to the request is completed.
- Error QPS: The number of errors of all TiFlash instances processing coprocessor requests. `meet_lock` means that the read data is locked. `region_not_found` means that the Region does not exist. `epoch_not_match` means the read Region epoch is inconsistent with the local epoch. `kv_client_error` means that the communication with TiKV returns an error. `internal_error` is the internal system error of TiFlash. `other` is other types of errors.
- Request Handle Duration: The duration of all TiFlash instances processing coprocessor requests. The processing time is from starting to execute the coprocessor request to completing the execution.
- Response Bytes/Seconds: The total bytes of the response from all TiFlash instances.
- Cop task memory usage: The total memory usage of all TiFlash instances processing coprocessor requests.
- Handling Request Number: The total number of all TiFlash instances processing coprocessor requests. The classification of the requests is the same as that of Request QPS.
- Threads of RPC: The real-time number of RPC threads used in each TiFlash instance.
- Max Threads of RPC: The maximum number of RPC threads recently used in each TiFlash instance.
- Threads: The real-time number of threads used in each TiFlash instance.
- Max Threads: The maximum number of threads recently used in each TiFlash instance.

14.8.6.3 Task Scheduler

- Min TSO: The minimum TSO among all queries running on each TiFlash instance. This value ensures that queries with the minimum TSO can be scheduled to run. When no queries are running, this value is the maximum unsigned 64-bit integer.
- Estimated Thread Usage and Limit: The estimated amount of threads used by all queries running on each TiFlash instance, and the soft and hard limits on the amount.

- Active and Waiting Queries Count: The amount of running queries and that of waiting queries on each TiFlash instance.
- Active and Waiting Tasks Count: The amount of running tasks and that of waiting tasks on each TiFlash instance.
- Hard Limit Exceeded Count: Times that the estimated amount of threads used by queries running on each TiFlash instance exceeds the hard limit.
- Task Waiting Duration: The duration from task initialization to task scheduling on each TiFlash instance.

14.8.6.4 DDL

- Schema Version: The version of the schema currently cached in each TiFlash instance.
- Schema Apply OPM: The number of TiDB `schema diff` synchronized in `apply` operations by all TiFlash instances per minute. This item includes the count of three types of `apply`: `diff apply`, `full apply`, and `failed apply`. `diff apply` is the normal process of a single apply. If `diff apply` fails, `failed apply` increases by `1`, and TiFlash rolls back to `full apply` and pulls the latest schema information to update the schema version of TiFlash.
- Schema Internal DDL OPM: The number of specific DDL operations executed per minute in all TiFlash instances.
- Schema Apply Duration: The time used for a single `apply schema` operation in all TiFlash instances.

14.8.6.5 Storage

- Write Command OPS: The number of write requests received per second by the storage layer of all TiFlash instances.
- Write Amplification: Write amplification of each TiFlash instance (the actual bytes of disk writes divided by the written bytes of logical data). `total` is the write amplification since this start, and `5min` is the write amplification in the last 5 minutes.
- Read Tasks OPS: The number of read tasks in the storage layer per second for each TiFlash instance.
- Rough Set Filter Rate: The proportion of the number of packets read by each TiFlash instance in the last minute that are filtered by the rough set index of the storage layer.
- Internal Tasks OPS: The number of times that all TiFlash instances perform internal data sorting tasks per second.
- Internal Tasks Duration: The time consumed by all TiFlash instances for internal data sorting tasks.
- Page GC Tasks OPM: The number of times that all TiFlash instances perform Delta data sorting tasks per minute.
- Page GC Tasks Duration: The distribution of time consumed by all TiFlash instances to perform Delta data sorting tasks.
- Disk Write OPS: The number of disk writes per second by all TiFlash instances.
- Disk Read OPS: The number of disk reads per second by all TiFlash instances.

- Write flow: The traffic of disk writes by all TiFlash instances.
- Read flow: The traffic of disk reads by all TiFlash instances.

> **Note:**
>
> These metrics only cover the monitoring information of the TiFlash storage layer and do not cover that in TiFlash-Proxy.

14.8.6.6 Storage Write Stall

- Write & Delta Management Throughput: The throughput of write and data compaction for all instances.
  - `throughput_write` means the throughput of data synchronization through Raft.
  - `throughput_delta-management` means the throughput of data compaction.
  - `total_write` means the total bytes written since the last start.
  - `total_delta-management` means the total bytes of data compacted since the last start.
- Write Stall Duration: The stall duration of write and removing Region data (deleting ranges) by instance.
- Write Throughput By Instance: The throughput of write by instance. It includes the throughput by applying the Raft write commands and Raft snapshots.
- Write Command OPS By Instance: The total count of different kinds of commands received by instance.
  - `write block` means the data logs synchronized through Raft.
  - `delete_range` means that some Regions are removed from or moved to this instance.
  - `ingest` means some Region snapshots are applied to this instance.

14.8.6.7 Raft

- Read Index OPS: The number of times that each TiFlash instance triggers the `read_index` request per second, which equals to the number of Regions triggered.
- Read Index Duration: The time used by `read_index` for all TiFlash instances. Most time is used for interaction with the Region leader and retry.
- Wait Index Duration: The time used by `wait_index` for all TiFlash instances, namely the time used to wait until local index >= read_index after the `read_index` request is received.

14.8.7 Key Monitoring Metrics of TiCDC

If you use TiUP to deploy the TiDB cluster, you can see a sub-dashboard for TiCDC in the monitoring system which is deployed at the same time. You can get an overview of TiCDC's current status from the TiCDC dashboard, where the key metrics are displayed. This document provides a detailed description of these key metrics.

The metric description in this document is based on the following replication task example, which replicates data to MySQL using the default configuration.

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
    ↪ root:123456@127.0.0.1:3306/" --changefeed-id="simple-replication-task
    ↪ "
```

The TiCDC dashboard contains four monitoring panels. See the following screenshot:

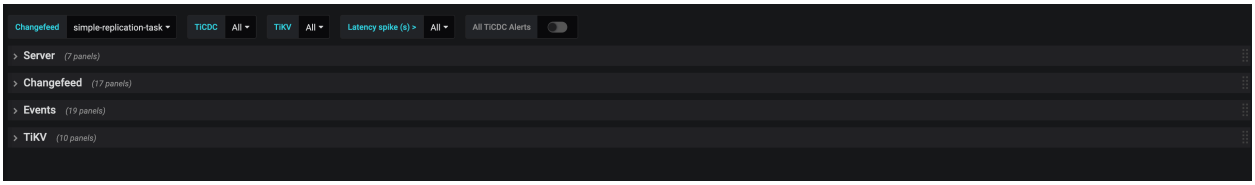


Figure 355: TiCDC Dashboard - Overview

The description of each panel is as follows:

- **Server**: The summary information of TiKV nodes and TiCDC nodes in the TiDB cluster
- **Changefeed**: The detailed information of TiCDC replication tasks
- **Events**: The detail information about the data flow within the TiCDC cluster
- **TiKV**: TiKV information related to TiCDC

14.8.7.1 Server

The following is an example of the **Server** panel:



Figure 356: TiCDC Dashboard - Server metrics

The description of each metric in the **Server** panel is as follows:

- Uptime: The time for which TiKV nodes and TiCDC nodes have been running
- Goroutine count: The number of goroutines of a TiCDC node
- Open FD count: The number of file handles opened by TiCDC nodes
- Ownership: The current status of nodes in the TiCDC cluster
- Ownership history: The ownership history of the TiCDC cluster
- CPU usage: The CPU usage of TiCDC nodes
- Memory usage: The memory usage of TiCDC nodes

14.8.7.2 Changefeed

The following is an example of the **Changefeed** panel:





The description of each metric in the **Changefeed** panel is as follows:

- Changefeed table count: The number of tables that each TiCDC node needs to replicate in the replication task
- Processor resolved ts: The timestamps that have been resolved in the TiCDC cluster
- Table resolved ts: The replication progress of each table in the replication task
- Changefeed checkpoint: The progress of replicating data to the downstream. Normally, the green bars are connected to the yellow line
- PD etcd requests/s: The number of requests that a TiCDC node sends to PD per second
- Exit error count: The number of errors that interrupt the replication task per minute
- Changefeed checkpoint lag: The progress lag of data replication (the unit is second) between the upstream and the downstream
- Changefeed resolved ts lag: The progress lag of data replication (the unit is second) between the upstream and TiCDC nodes
- Flush sink duration: The histogram of the time spent by TiCDC asynchronously flushing data to the downstream
- Flush sink duration percentile: The time (P95, P99, and P999) spent by TiCDC asynchronously flushing data to the downstream within one second
- Sink write duration: The histogram of the time spent by TiCDC writing a transaction change to the downstream
- Sink write duration percentile: The time (P95, P99, and P999) spent by TiCDC writing a transaction change to the downstream within one second
- MySQL sink conflict detect duration: The histogram of the time spent on detecting

MySQL sink conflicts

- MySQL sink conflict detect duration percentile: The time (P95, P99, and P999) spent on detecting MySQL sink conflicts within one second
- MySQL sink worker load: The workload of MySQL sink workers of TiCDC nodes

14.8.7.3 Events

The following is an example of the **Events** panel:



The description of each metric in the **Events** panel is as follows:

- Eventfeed count: The number of Eventfeed RPC requests of TiCDC nodes
- Event size percentile: The event size (P95, P99, and P999) which TiCDC receives from TiKV within one second
- Eventfeed error/m: The number of errors reported by Eventfeed RPC requests of TiCDC nodes per minute
- KV client receive events/s: The number of events that the KV client module of TiCDC nodes receives from TiKV per second
- Puller receive events/s: The number of events that the Puller module of TiCDC nodes receives from the KV client per second
- Puller output events/s: The number of events that the Puller module of TiCDC nodes sends to the Sorter module per second
- Sink flush rows/s: The number of events that TiCDC nodes write to the downstream per second
- Puller buffer size: The number of events that TiCDC nodes cache in the Puller module
- Entry sorter buffer size: The number of events that TiCDC nodes cache in the Sorter module
- Processor/Mounter buffer size: The number of events that TiCDC nodes cache in the Processor module and the Mounter module
- Sink row buffer size: The number of events that TiCDC nodes cache in the Sink module
- Entry sorter sort duration: The histogram of the time spent by TiCDC nodes sorting events
- Entry sorter sort duration percentile: The time (P95, P99, and P999) spent by TiCDC sorting events within one second
- Entry sorter merge duration: The histogram of the time spent by TiCDC nodes merging sorted events
- Entry sorter merge duration percentile: The time (P95, P99, and P999) spent by TiCDC merging sorted events within one second
- Mounter unmarshal duration: The histogram of the time spent by TiCDC nodes unmarshaling events
- Mounter unmarshal duration percentile: The time (P95, P99, and P999) spent by TiCDC unmarshaling events within one second
- KV client dispatch events/s: The number of events that the KV client module dispatches among the TiCDC nodes
- KV client batch resolved size: The batch size of resolved timestamp messages that TiKV sends to TiCDC

14.8.7.4 TiKV

The following is an example of the **TiKV** panel:



The description of each metric in the **TiKV** panel is as follows:

- CDC endpoint CPU: The CPU usage of the CDC endpoint threads on TiKV nodes
- CDC worker CPU: The CPU usage of the CDC worker threads on TiKV nodes
- Min resolved ts: The minimum resolved timestamp on TiKV nodes
- Min resolved region: The Region ID of the minimum resolved timestamp on TiKV nodes
- Resolved ts lag duration percentile: The lag between the minimum resolved timestamp on TiKV nodes and the current time
- Initial scan duration: The histogram of the time spent on incremental scan when TiKV nodes connect to TiCDC nodes
- Initial scan duration percentile: The time (P95, P99, and P999) spent on the incremental scan of TiKV nodes within one second
- Memory without block cache: The memory usage of TiKV nodes excluding the RocksDB block cache
- CDC pending bytes in memory: The memory usage of CDC module on TiKV nodes
- Captured region count: The number of event-capturing Regions on TiKV nodes

14.9 Privileges

14.9.1 Security Compatibility with MySQL

TiDB supports similar security functionality to MySQL 5.7, with the following exceptions:

- Column level permissions are not supported
- Password expiry, as well as password last-changed tracking and password lifetime are not supported #9709
- The permission attributes `max_questions`, `max_updated`, `max_connections`, `max_user_connections` are not supported
- Password validation is not currently supported #9741

14.9.1.1 Authentication plugin status

TiDB supports multiple authentication methods. These methods can be specified on a per user basis using `CREATE USER` and `ALTER USER`. These methods are compatible with the authentication methods of MySQL with the same names.

You can use one of the following supported authentication methods in the table. To specify a default method that the server advertises when the client-server connection is being established, set the `default_authentication_plugin` variable.

The support for TLS authentication is configured differently. For detailed information, see Enable TLS between TiDB Clients and Servers.

Authentication Method	Supported
`mysql_native_password`	Yes
`sha256_password`	No
`caching_sha2_password`	Yes, since 5.2.0
`auth_socket`	Yes, since 5.3.0
[TLS Certificates]	Yes
LDAP	No
PAM	No
ed25519 (MariaDB)	No
GSSAPI (MariaDB)	No
FIDO	No

14.9.2 Privilege Management

TiDB supports MySQL 5.7's privilege management system, including the syntax and privilege types. The following features from MySQL 8.0 are also supported:

- SQL Roles, starting with TiDB 3.0.

- Dynamic privileges, starting with TiDB 5.1.

This document introduces privilege-related TiDB operations, privileges required for TiDB operations and implementation of the privilege system.

14.9.2.1 Privilege-related operations

14.9.2.1.1 Grant privileges

The `GRANT` statement grants privileges to the user accounts.

For example, use the following statement to grant the `xxx` user the privilege to read the `test` database.

```
GRANT SELECT ON test.* TO 'xxx'@'%';
```

Use the following statement to grant the `xxx` user all privileges on all databases:

```
GRANT ALL PRIVILEGES ON *.* TO 'xxx'@'%';
```

By default, `GRANT` statements will return an error if the user specified does not exist. This behavior depends on if the SQL Mode `NO_AUTO_CREATE_USER` is specified:

```
mysql> SET sql_mode=DEFAULT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+--
   ↪ ------------------------------------------------------------------------
   ↪
| @@sql_mode
   ↪
   ↪ |
+--
   ↪ ------------------------------------------------------------------------
   ↪
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
   ↪ ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
   ↪  |
+--
   ↪ ------------------------------------------------------------------------
   ↪
1 row in set (0.00 sec)

mysql> SELECT * FROM mysql.user WHERE user='idontexist';
Empty set (0.00 sec)
```

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'idontexist';
ERROR 1105 (HY000): You are not allowed to create a user with GRANT

mysql> SELECT user,host,authentication_string FROM mysql.user WHERE user='
    ↪ idontexist';
Empty set (0.00 sec)
```

In the following example, the user `idontexist` is automatically created with an empty password because the SQL Mode `NO_AUTO_CREATE_USER` was not set. This is **not recommended** since it presents a security risk: miss-spelling a username will result in a new user created with an empty password:

```
mysql> SET @@sql_mode='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
    ↪ NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,
    ↪ NO_ENGINE_SUBSTITUTION';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+--
    ↪ ------------------------------------------------------------------------
    ↪
| @@sql_mode
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------
    ↪
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
    ↪ ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION |
+--
    ↪ ------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> SELECT * FROM mysql.user WHERE user='idontexist';
Empty set (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON test.* TO 'idontexist';
Query OK, 1 row affected (0.05 sec)

mysql> SELECT user,host,authentication_string FROM mysql.user WHERE user='
    ↪ idontexist';
+------------+------+-----------------------+
| user       | host | authentication_string |
+------------+------+-----------------------+
```

```
| idontexist | % |                      |
+------------+------+----------------------+
1 row in set (0.01 sec)
```

You can use fuzzy matching in `GRANT` to grant privileges to databases.

```
mysql> GRANT ALL PRIVILEGES ON `te%`.* TO genius;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT user,host,db FROM mysql.db WHERE user='genius';
+--------/------/-----+
| user   | host | db |
+--------/------/-----+
| genius | %    | te% |
+--------/------/-----+
1 row in set (0.00 sec)
```

In this example, because of the `%` in `te%`, all the databases starting with `te` are granted the privilege.

14.9.2.1.2 Revoke privileges

The `REVOKE` statement enables system administrators to revoke privileges from the user accounts.

The `REVOKE` statement corresponds with the `REVOKE` statement:

```
REVOKE ALL PRIVILEGES ON `test`.* FROM 'genius'@'localhost';
```

> **Note:**
>
> To revoke privileges, you need the exact match. If the matching result cannot be found, an error will be displayed:

```
mysql> REVOKE ALL PRIVILEGES ON `te%`.* FROM 'genius'@'%';
ERROR 1141 (42000): There is no such grant defined for user 'genius' on
    ↪ host '%'
```

About fuzzy matching, escape, string and identifier:

```
mysql> GRANT ALL PRIVILEGES ON `te\%`.* TO 'genius'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

This example uses exact match to find the database named `te%`. Note that the `%` uses the `\` escape character so that `%` is not considered as a wildcard.

A string is enclosed in single quotation marks(”), while an identifier is enclosed in backticks (“). See the differences below:

```
mysql> GRANT ALL PRIVILEGES ON 'test'.* TO 'genius'@'localhost';
ERROR 1064 (42000): You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right
syntax to use near ''test'.* to 'genius'@'localhost'' at line 1

mysql> GRANT ALL PRIVILEGES ON `test`.* TO 'genius'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

If you want to use special keywords as table names, enclose them in backticks (“). For example:

```
mysql> CREATE TABLE `select` (id int);
Query OK, 0 rows affected (0.27 sec)
```

14.9.2.1.3 Check privileges granted to users

You can use the `SHOW GRANTS` statement to see what privileges are granted to a user. For example:

```
SHOW GRANTS; -- show grants for the current user

+-------------------------------------------------------------+
| Grants for User                                             |
+-------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION |
+-------------------------------------------------------------+
SHOW GRANTS FOR 'root'@'%'; -- show grants for a specific user
```

For example, create a user `rw_user@192.168.%` and grant the user with write privilege on the `test.write_table` table and global read privilege.

```
CREATE USER `rw_user`@`192.168.%`;
GRANT SELECT ON *.* TO `rw_user`@`192.168.%`;
GRANT INSERT, UPDATE ON `test`.`write_table` TO `rw_user`@`192.168.%`;
```

Show granted privileges of the `rw_user@192.168.%` user:

```
SHOW GRANTS FOR `rw_user`@`192.168.%`;

+-----------------------------------------------------------------+
| Grants for rw_user@192.168.%                                    |
+-----------------------------------------------------------------+
```

```
| GRANT Select ON *.* TO 'rw_user'@'192.168.%'                |
| GRANT Insert,Update ON test.write_table TO 'rw_user'@'192.168.%' |
+-------------------------------------------------------------------+
```

14.9.2.1.4 Dynamic privileges

Since v5.1, TiDB features support dynamic privileges, a feature borrowed from MySQL 8.0. Dynamic privileges are intended to replace the `SUPER` privilege by implementing more fine-grained access to certain operations. For example, using dynamic privileges, system administrators can create a user account that can only perform `BACKUP` and `RESTORE` operations.

Dynamic privileges include:

- `BACKUP_ADMIN`
- `RESTORE_ADMIN`
- `SYSTEM_USER`
- `SYSTEM_VARIABLES_ADMIN`
- `ROLE_ADMIN`
- `CONNECTION_ADMIN`
- `PLACEMENT_ADMIN` allows privilege owners to create, modify, and remove placement policies.
- `DASHBOARD_CLIENT` allows privilege owners to log in to TiDB Dashboard.
- `RESTRICTED_TABLES_ADMIN` allows privilege owners to view system tables when SEM is enabled.
- `RESTRICTED_STATUS_ADMIN` allows privilege owners to view all status variables in `SHOW [GLOBAL|SESSION] STATUS` when SEM is enabled.
- `RESTRICTED_VARIABLES_ADMIN` allows privilege owners to view all system variables when SEM is enabled.
- `RESTRICTED_USER_ADMIN` prohibits privilege owners to have their access revoked by SUPER users when SEM is enabled.
- `RESTRICTED_CONNECTION_ADMIN` allows privilege owners to kill connections of `RESTRICTED_USER_ADMIN` users. This privilege affects `KILL` and `KILL TIDB` statements.
- `RESTRICTED_REPLICA_WRITER_ADMIN` allows privilege owners to perform write or update operations without being affected when the read-only mode is enabled in the TiDB cluster. For details, see `tidb_restricted_read_only`.

To see the full set of dynamic privileges, execute the `SHOW PRIVILEGES` statement. Because plugins are permitted to add new privileges, the list of privileges that are assignable might differ based on your TiDB installation.

14.9.2.2 `SUPER` privilege

- The SUPER privilege allows the user to perform almost any operation. By default, only the root user is granted with this privilege. Be careful when granting this privilege to other users.
- The SUPER privilege is considered deprecated in MySQL 8.0 and can be replaced by dynamic privileges to provide more fine-grained access control.

14.9.2.3 Privileges required for TiDB operations

You can check privileges of TiDB users in the INFORMATION_SCHEMA.USER_PRIVILEGES table. For example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.USER_PRIVILEGES WHERE grantee = "'
    ↪ root'@'%'";
+------------+---------------+-------------------------+--------------+
| GRANTEE    | TABLE_CATALOG | PRIVILEGE_TYPE          | IS_GRANTABLE |
+------------+---------------+-------------------------+--------------+
| 'root'@'%' | def           | Select                  | YES          |
| 'root'@'%' | def           | Insert                  | YES          |
| 'root'@'%' | def           | Update                  | YES          |
| 'root'@'%' | def           | Delete                  | YES          |
| 'root'@'%' | def           | Create                  | YES          |
| 'root'@'%' | def           | Drop                    | YES          |
| 'root'@'%' | def           | Process                 | YES          |
| 'root'@'%' | def           | References              | YES          |
| 'root'@'%' | def           | Alter                   | YES          |
| 'root'@'%' | def           | Show Databases          | YES          |
| 'root'@'%' | def           | Super                   | YES          |
| 'root'@'%' | def           | Execute                 | YES          |
| 'root'@'%' | def           | Index                   | YES          |
| 'root'@'%' | def           | Create User             | YES          |
| 'root'@'%' | def           | Create Tablespace       | YES          |
| 'root'@'%' | def           | Trigger                 | YES          |
| 'root'@'%' | def           | Create View             | YES          |
| 'root'@'%' | def           | Show View               | YES          |
| 'root'@'%' | def           | Create Role             | YES          |
| 'root'@'%' | def           | Drop Role               | YES          |
| 'root'@'%' | def           | CREATE TEMPORARY TABLES | YES          |
| 'root'@'%' | def           | LOCK TABLES             | YES          |
| 'root'@'%' | def           | CREATE ROUTINE          | YES          |
| 'root'@'%' | def           | ALTER ROUTINE           | YES          |
| 'root'@'%' | def           | EVENT                   | YES          |
| 'root'@'%' | def           | SHUTDOWN                | YES          |
| 'root'@'%' | def           | RELOAD                  | YES          |
| 'root'@'%' | def           | FILE                    | YES          |
| 'root'@'%' | def           | CONFIG                  | YES          |
| 'root'@'%' | def           | REPLICATION CLIENT      | YES          |
```

```
| 'root'@'%' | def          | REPLICATION SLAVE    | YES          |
+------------+--------------+----------------------+--------------+
31 rows in set (0.00 sec)
```

14.9.2.3.1 ALTER

- For all `ALTER` statements, users must have the `ALTER` privilege for the corresponding table.
- For statements except `ALTER...DROP` and `ALTER...RENAME TO`, users must have the `INSERT` and `CREATE` privileges for the corresponding table.
- For the `ALTER...DROP` statement, users must have the `DROP` privilege for the corresponding table.
- For the `ALTER...RENAME TO` statement, users must have the `DROP` privilege for the table before renaming, and the `CREATE` and `INSERT` privileges for the table after renaming.

> **Note:**
>
> In MySQL 5.7 documentation, users need `INSERT` and `CREATE` privileges to perform the `ALTER` operation on a table. But in reality for MySQL 5.7.25, only the `ALTER` privilege is required in this case. Currently, the `ALTER` privilege in TiDB is consistent with the actual behavior in MySQL.

14.9.2.3.2 BACKUP

Requires the `SUPER` or `BACKUP_ADMIN` privilege.

14.9.2.3.3 CREATE DATABASE

Requires the `CREATE` privilege for the database.

14.9.2.3.4 CREATE INDEX

Requires the `INDEX` privilege for the table.

14.9.2.3.5 CREATE TABLE

Requires the `CREATE` privilege for the table.

To execute the `CREATE TABLE...LIKE...` statement, the `SELECT` privilege for the table is required.

14.9.2.3.6 CREATE VIEW

Requires the `CREATE VIEW` privilege.

> **Note:**
>
> If the current user is not the user that creates the View, both the `CREATE VIEW` and `SUPER` privileges are required.

14.9.2.3.7 DROP DATABASE

Requires the `DROP` privilege for the table.

14.9.2.3.8 DROP INDEX

Requires the `INDEX` privilege for the table.

14.9.2.3.9 DROP TABLES

Requires the `DROP` privilege for the table.

14.9.2.3.10 LOAD DATA

Requires the `INSERT` privilege for the table. When you use `REPLACE INTO`, the `DELETE` privilege is also required.

14.9.2.3.11 TRUNCATE TABLE

Requires the `DROP` privilege for the table.

14.9.2.3.12 RENAME TABLE

Requires the `ALTER` and `DROP` privileges for the table before renaming and the `CREATE` and `INSERT` privileges for the table after renaming.

14.9.2.3.13 ANALYZE TABLE

Requires the `INSERT` and `SELECT` privileges for the table.

14.9.2.3.14 SHOW

`SHOW CREATE TABLE` requires any single privilege to the table.

`SHOW CREATE VIEW` requires the `SHOW VIEW` privilege.

`SHOW GRANTS` requires the `SELECT` privilege to the `mysql` database. If the target user is current user, `SHOW GRANTS` does not require any privilege.

`SHOW PROCESSLIST` requires `SUPER` to show connections belonging to other users.

14.9.2.3.15 CREATE ROLE/USER

`CREATE ROLE` requires the `CREATE ROLE` privilege.

`CREATE USER` requires the `CREATE USER` privilege.

14.9.2.3.16 DROP ROLE/USER

`DROP ROLE` requires the `DROP ROLE` privilege.

`DROP USER` requires the `CREATE USER` privilege.

14.9.2.3.17 ALTER USER

Requires the `CREATE USER` privilege.

14.9.2.3.18 GRANT

Requires the `GRANT` privilege with the privileges granted by `GRANT`.

Requires additional `CREATE USER` privilege to create a user implicitly.

`GRANT ROLE` requires `SUPER` or `ROLE_ADMIN` privilege.

14.9.2.3.19 REVOKE

Requires the `GRANT` privilege and those privileges targeted by the `REVOKE` statement.

`REVOKE ROLE` requires `SUPER` or `ROLE_ADMIN` privilege.

14.9.2.3.20 SET GLOBAL

Requires `SUPER` or `SYSTEM_VARIABLES_ADMIN` privilege to set global variables.

14.9.2.3.21 ADMIN

Requires `SUPER` privilege.

14.9.2.3.22 SET DEFAULT ROLE

Requires `SUPER` privilege.

14.9.2.3.23 KILL

Requires `SUPER` or `CONNECTION_ADMIN` privilege to kill other user sessions.

14.9.2.4 Implementation of the privilege system

14.9.2.4.1 Privilege table

The following system tables are special because all the privilege-related data is stored in them:

- `mysql.user` (user account, global privilege)
- `mysql.db` (database-level privilege)
- `mysql.tables_priv` (table-level privilege)
- `mysql.columns_priv` (column-level privilege; not currently supported)

These tables contain the effective range and privilege information of the data. For example, in the `mysql.user` table:

```
mysql> SELECT User,Host,Select_priv,Insert_priv FROM mysql.user LIMIT 1;
+------/------/-------------/-------------+
| User | Host | Select_priv | Insert_priv |
+------/------/-------------/-------------+
| root | %  | Y           | Y           |
+------/------/-------------/-------------+
1 row in set (0.00 sec)
```

In this record, `Host` and `User` determine that the connection request sent by the `root` user from any host (`%`) can be accepted. `Select_priv` and `Insert_priv` mean that the user has global `Select` and `Insert` privilege. The effective range in the `mysql.user` table is global.

`Host` and `User` in `mysql.db` determine which databases users can access. The effective range is the database.

> **Note:**
>
> It is recommended to only update the privilege tables via the supplied syntax such as `GRANT`, `CREATE USER` and `DROP USER`. Making direct edits to the underlying privilege tables will not automatically update the privilege cache, leading to unpredictable behavior until `FLUSH PRIVILEGES` is executed.

14.9.2.4.2 Connection verification

When the client sends a connection request, TiDB server will verify the login operation. TiDB server first checks the `mysql.user` table. If a record of `User` and `Host` matches the connection request, TiDB server then verifies the `authentication_string`.

User identity is based on two pieces of information: `Host`, the host that initiates the connection, and `User`, the user name. If the user name is not empty, the exact match of user named is a must.

`User`+`Host` may match several rows in `user` table. To deal with this scenario, the rows in the `user` table are sorted. The table rows will be checked one by one when the client connects; the first matched row will be used to verify. When sorting, Host is ranked before User.

14.9.2.4.3 Request verification

When the connection is successful, the request verification process checks whether the operation has the privilege.

For database-related requests (`INSERT`, `UPDATE`), the request verification process first checks the user's global privileges in the `mysql.user` table. If the privilege is granted, you can access directly. If not, check the `mysql.db` table.

The `user` table has global privileges regardless of the default database. For example, the `DELETE` privilege in `user` can apply to any row, table, or database.

In the `db` table, an empty user is to match the anonymous user name. Wildcards are not allowed in the `User` column. The value for the `Host` and `Db` columns can use `%` and `_`, which can use pattern matching.

Data in the `user` and `db` tables is also sorted when loaded into memory.

The use of `%` in `tables_priv` and `columns_priv` is similar, but column value in `Db`, `Table_name` and `Column_name` cannot contain `%`. The sorting is also similar when loaded.

14.9.2.4.4 Time of effect

When TiDB starts, some privilege-check tables are loaded into memory, and then the cached data is used to verify the privileges. Executing privilege management statements such as `GRANT`, `REVOKE`, `CREATE USER`, `DROP USER` will take effect immediately.

Manually editing tables such as `mysql.user` with statements such as `INSERT`, `DELETE`, `UPDATE` will not take effect immediately. This behavior is compatible with MySQL, and privilege cache can be updated with the following statement:

```
FLUSH PRIVILEGES;
```

14.9.3 TiDB User Account Management

This document describes how to manage a TiDB user account.

14.9.3.1 User names and passwords

TiDB stores the user accounts in the table of the `mysql.user` system database. Each account is identified by a user name and the client host. Each account may have a password.

You can connect to the TiDB server using the MySQL client, and use the specified account and password to login. For each user name, make sure that it contains no more than 32 characters.

```
mysql --port 4000 --user xxx --password
```

Or use the abbreviation of command line parameters:

```
mysql -P 4000 -u xxx -p
```

14.9.3.2 Add user accounts

You can create TiDB accounts in two ways:

- By using the standard account-management SQL statements intended for creating accounts and establishing their privileges, such as `CREATE USER` and `GRANT`.
- By manipulating the privilege tables directly with statements such as `INSERT`, `UPDATE`, or `DELETE`. It is not recommended to use this method to create accounts, because it might lead to incomplete updates.

You can also create accounts by using third party GUI tools.

```
CREATE USER [IF NOT EXISTS] user [IDENTIFIED BY 'auth_string'];
```

After you assign the password, TiDB encrypts and stores the `auth_string` in the `mysql.user` table.

```
CREATE USER 'test'@'127.0.0.1' IDENTIFIED BY 'xxx';
```

The name of a TiDB account consists of a user name and a hostname. The syntax of the account name is 'user_name'@'host_name'.

- `user_name` is case sensitive.
- `host_name` is a hostname or IP address, which supports the wild card `%` or `_`. For example, the hostname `'%'` matches all hosts, and the hostname `'192.168.1.%'` matches all hosts in the subnet.

The host supports fuzzy matching:

```
CREATE USER 'test'@'192.168.10.%';
```

The `test` user is allowed to log in from any hosts on the `192.168.10` subnet.

If the host is not specified, the user is allowed to log in from any IP. If no password is specified, the default is empty password:

```
CREATE USER 'test';
```

Equivalent to:

```
CREATE USER 'test'@'%' IDENTIFIED BY '';
```

If the specified user does not exist, the behavior of automatically creating users depends on `sql_mode`. If the `sql_mode` includes `NO_AUTO_CREATE_USER`, the `GRANT` statement will not create users with an error returned.

For example, assume that the `sql_mode` does not include `NO_AUTO_CREATE_USER`, and you use the following `CREATE USER` and `GRANT` statements to create four accounts:

```
CREATE USER 'finley'@'localhost' IDENTIFIED BY 'some_pass';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'finley'@'localhost' WITH GRANT OPTION;
```

```
CREATE USER 'finley'@'%' IDENTIFIED BY 'some_pass';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'finley'@'%' WITH GRANT OPTION;
```

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin_pass';
```

```
GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
```

```
CREATE USER 'dummy'@'localhost';
```

To see the privileges granted for an account, use the `SHOW GRANTS` statement:

```
SHOW GRANTS FOR 'admin'@'localhost';
```

```
+-----------------------------------------------------+
| Grants for admin@localhost                          |
+-----------------------------------------------------+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------------------+
```

14.9.3.3 Remove user accounts

To remove a user account, use the `DROP USER` statement:

```
DROP USER 'test'@'localhost';
```

This operation clears the user's records in the `mysql.user` table and the related records in the privilege table.

14.9.3.4 Reserved user accounts

TiDB creates the `'root'@'%'` default account during the database initialization.

14.9.3.5 Set account resource limits

Currently, TiDB does not support setting account resource limits.

14.9.3.6 Assign account passwords

TiDB stores passwords in the `mysql.user` system database. Operations that assign or update passwords are permitted only to users with the `CREATE USER` privilege, or, alternatively, privileges for the `mysql` database (`INSERT` privilege to create new accounts, `UPDATE` privilege to update existing accounts).

- To assign a password when you create a new account, use `CREATE USER` and include an `IDENTIFIED BY` clause:

```
CREATE USER 'test'@'localhost' IDENTIFIED BY 'mypass';
```

- To assign or change a password for an existing account, use `SET PASSWORD FOR` or `ALTER USER`:

```
SET PASSWORD FOR 'root'@'%' = 'xxx';
```

Or:

```
ALTER USER 'test'@'localhost' IDENTIFIED BY 'mypass';
```

14.9.3.7 Forget the `root` password

1. Modify the configuration file:
    1. Log in to the machine where one of the tidb-server instances is located.
    2. Enter the `conf` directory under the TiDB node deployment directory, and find the `tidb.toml` configuration file.
    3. Add the configuration item `skip-grant-table` in the `security` section of the configuration file. If there is no `security` section, add the following two lines to the end of the tidb.toml configuration file:

```
[security]
skip-grant-table = true
```

2. Stop the tidb-server process:

1. View the tidb-server process:

```
ps aux | grep tidb-server
```

2. Find the process ID (PID) corresponding to tidb-server and use the `kill` command to stop the process:

```
kill -9 <pid>
```

3. Start TiDB using the modified configuration:

> **Note:**
> If you set `skip-grant-table` before starting the TiDB process, a check on the operating system user will be initiated. Only the `root` user of the operating system can start the TiDB process.

1. Enter the `scripts` directory under the TiDB node deployment directory.
2. Switch to the `root` account of the operating system.
3. Run the `run_tidb.sh` script in the directory in the foreground.
4. Log in as `root` in a new terminal window and change the password.

```
mysql -h 127.0.0.1 -P 4000 -u root
```

4. Stop running the `run_tidb.sh` script, remove the content added in the TiDB configuration file in step 1, and wait for tidb-server to start automatically.

14.9.3.8 `FLUSH PRIVILEGES`

Information related to users and privileges is stored in the TiKV server, and TiDB caches this information inside the process. Generally, modification of the related information through `CREATE USER`, `GRANT`, and other statements takes effect quickly within the entire cluster. If the operation is affected by some factors such as temporarily unavailable network, the modification will take effect in about 15 minutes because TiDB will periodically reload the cache information.

If you modified the privilege tables directly, run the following command to apply changes immediately:

```
FLUSH PRIVILEGES;
```

For details, see Privilege Management.

14.9.4 Role-Based Access Control

The implementation of TiDB's role-based access control (RBAC) system is similar to that of MySQL 8.0. TiDB is compatible with most RBAC syntax of MySQL.

This document introduces TiDB RBAC-related operations and implementation.

14.9.4.1 RBAC operations

A role is a collection of a series of privileges. You can do the following operations:

- Create a role.
- Delete a role.
- Grant a privilege to a role.
- Grant a role to another user. That user can obtain the privileges involved in the role, after enabling the role.

14.9.4.1.1 Create a role

For example, you can use the following statement to create the roles `app_developer`, `app_read`, and `app_write`:

```
CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

For the role naming format and rule, see TiDB User Account Management.

Roles are stored in the `mysql.user` table and the host name part of the role name (if omitted) defaults to `'%'`. The name of the role you are trying to create must be unique; otherwise, an error is reported.

To create a role, you need the `CREATE ROLE` or `CREATE USER` privilege.

14.9.4.1.2 Grant a privilege to a role

The operation of granting a privilege to a role is the same with that of granting a privilege to a user. For details, see TiDB Privilege Management.

For example, you can use the following statement to grant the `app_read` role the privilege to read the `app_db` database:

```
GRANT SELECT ON app_db.* TO 'app_read'@'%';
```

You can use the following statement to grant the `app_write` role the privilege to write data to the `app_db` database:

```
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write'@'%';
```

You can use the following statement to grant the `app_developer` role all privileges on the `app_db` database:

```
GRANT ALL ON app_db.* TO 'app_developer';
```

14.9.4.1.3 Grant a role to a user

Assume that a user `dev1` has the developer role with all the privileges on `app_db`; two users `read_user1` and `read_user2` have the read-only privilege on `app_db`; and a user `rw_user1` has read and write privileges on `app_db`.

Use `CREATE USER` to create the users:

```
CREATE USER 'dev1'@'localhost' IDENTIFIED BY 'dev1pass';
CREATE USER 'read_user1'@'localhost' IDENTIFIED BY 'read_user1pass';
CREATE USER 'read_user2'@'localhost' IDENTIFIED BY 'read_user2pass';
CREATE USER 'rw_user1'@'localhost' IDENTIFIED BY 'rw_user1pass';
```

Then use `GRANT` to grant roles to users

```
GRANT 'app_developer' TO 'dev1'@'localhost';
GRANT 'app_read' TO 'read_user1'@'localhost', 'read_user2'@'localhost';
GRANT 'app_read', 'app_write' TO 'rw_user1'@'localhost';
```

To grant a role to another user or revoke a role, you need the `SUPER` privilege.

Granting a role to a user does not mean enabling the role immediately. Enabling a role is another operation.

The following operations might form a "relation loop:"

```
CREATE USER 'u1', 'u2';
CREATE ROLE 'r1', 'r2';

GRANT 'u1' TO 'u1';
GRANT 'r1' TO 'r1';

GRANT 'r2' TO 'u2';
GRANT 'u2' TO 'r2';
```

TiDB supports this multi-level authorization relationship. You can use it to implement privilege inheritance.

14.9.4.1.4 Check a role's privileges

You can use the `SHOW GRANTS` statement to check what privileges have been granted to the user.

To check privilege-related information of another user, you need the `SELECT` privilege on the `mysql` database.

```
SHOW GRANTS FOR 'dev1'@'localhost';
```

```
+-----------------------------------------------+
| Grants for dev1@localhost                     |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO `dev1`@`localhost`      |
| GRANT `app_developer`@`%` TO `dev1`@`localhost` |
+-----------------------------------------------+
```

You can use the `USING` option in `SHOW GRANTS` to check a role's privileges:

```
SHOW GRANTS FOR 'dev1'@'localhost' USING 'app_developer';
```

```
+-----------------------------------------------------------+
| Grants for dev1@localhost                                 |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO `dev1`@`localhost`                  |
| GRANT ALL PRIVILEGES ON `app_db`.* TO `dev1`@`localhost`  |
| GRANT `app_developer`@`%` TO `dev1`@`localhost`           |
+-----------------------------------------------------------+
```

```
SHOW GRANTS FOR 'rw_user1'@'localhost' USING 'app_read', 'app_write';
```

```
+-------------------------------------------------------------------------------+
   ↪
| Grants for rw_user1@localhost                                                 |
+-------------------------------------------------------------------------------+
   ↪
| GRANT USAGE ON *.* TO `rw_user1`@`localhost`                                  |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `app_db`.* TO `rw_user1`@`
   ↪ localhost` |
| GRANT `app_read`@`%`,`app_write`@`%` TO `rw_user1`@`localhost`                |
+-------------------------------------------------------------------------------+
   ↪
```

```
SHOW GRANTS FOR 'read_user1'@'localhost' USING 'app_read';
```

```
+-------------------------------------------------------+
| Grants for read_user1@localhost                       |
+-------------------------------------------------------+
| GRANT USAGE ON *.* TO `read_user1`@`localhost`        |
| GRANT SELECT ON `app_db`.* TO `read_user1`@`localhost` |
| GRANT `app_read`@`%` TO `read_user1`@`localhost`      |
+-------------------------------------------------------+
```

You can use `SHOW GRANTS` or `SHOW GRANTS FOR CURRENT_USER()` to check the current user's privileges. `SHOW GRANTS` and `SHOW GRANTS FOR CURRENT_USER()` are different in the following aspects:

- `SHOW GRANTS` shows the privilege of the enabled role for the current user.
- `SHOW GRANTS FOR CURRENT_USER()` does not show the enabled role's privilege.

14.9.4.1.5 Set the default role

After a role is granted to a user, it does not take effect immediately. Only after the user enables this role, he can use the privilege the role owns.

You can set default roles for a user. When the user logs in, the default roles are automatically enabled.

```
SET DEFAULT ROLE
    {NONE | ALL | role [, role ] ...}
    TO user [, user ]
```

For example, you can use the following statement to set default roles of `rw_user1@localhost` to `app_read` and `app_write`:

```
SET DEFAULT ROLE app_read, app_write TO 'rw_user1'@'localhost';
```

You can use the following statement to set default roles of `dev1@localhost` to all roles:

```
SET DEFAULT ROLE ALL TO 'dev1'@'localhost';
```

You can use the following statement to disable all default roles of `dev1@localhost`:

```
SET DEFAULT ROLE NONE TO 'dev1'@'localhost';
```

> **Note:**
>
> You need to grant the role to the user before you set the default role to this role.

14.9.4.1.6 Enable a role in the current session

You can enable some role(s) in the current session.

```
SET ROLE {
    DEFAULT
  | NONE
  | ALL
```

```
  | ALL EXCEPT role [, role ] ...
  | role [, role ] ...
}
```

For example, after `rw_user1` logs in, you can use the following statement to enable roles `app_read` and `app_write` that are valid only in the current session:

```
SET ROLE 'app_read', 'app_write';
```

You can use the following statement to enable the default role of the current user:

```
SET ROLE DEFAULT
```

You can use the following statement to enable all roles granted to the current user:

```
SET ROLE ALL
```

You can use the following statement to disable all roles:

```
SET ROLE NONE
```

You can use the following statement to enable roles except `app_read`:

```
SET ROLE ALL EXCEPT 'app_read'
```

Note:

If you use `SET ROLE` to enable a role, this role is valid only in the current session.

14.9.4.1.7 Check the current enabled role

The current user can use the `CURRENT_ROLE()` function to check which role has been enabled by the current user.

For example, you can grant default roles to `rw_user1'@'localhost`:

```
SET DEFAULT ROLE ALL TO 'rw_user1'@'localhost';
```

After `rw_user1@localhost` logs in, you can execute the following statement:

```
SELECT CURRENT_ROLE();
```

```
+--------------------------------+
| CURRENT_ROLE()                 |
+--------------------------------+
| `app_read`@`%`,`app_write`@`%` |
+--------------------------------+
```

```
SET ROLE 'app_read'; SELECT CURRENT_ROLE();
```

```
+----------------+
| CURRENT_ROLE() |
+----------------+
| `app_read`@`%` |
+----------------+
```

14.9.4.1.8 Revoke a role

You can use the following statement to revoke the `app_read` role granted to the users `read_user1@localhost` and `read_user2@localhost`:

```
REVOKE 'app_read' FROM 'read_user1'@'localhost', 'read_user2'@'localhost';
```

You can use the following statement to revoke the roles `app_read` and `app_write` granted to the `rw_user1@localhost` user:

```
REVOKE 'app_read', 'app_write' FROM 'rw_user1'@'localhost';
```

The operation of revoking a role from a user is atomic. If you fail to revoke a role, this operation rolls back.

14.9.4.1.9 Revoke a privilege

The `REVOKE` statement is reverse to `GRANT`. You can use `REVOKE` to revoke the privileges of `app_write`.

```
REVOKE INSERT, UPDATE, DELETE ON app_db.* FROM 'app_write';
```

For details, see TiDB Privilege Management.

14.9.4.1.10 Delete a role

You can use the following statement to delete roles `app_read` and `app_write`:

```
DROP ROLE 'app_read', 'app_write';
```

This operation deletes the role records of `app_read` and `app_write` in the `mysql.user` table and related records in the authorization table, and terminates the authorization related to the two roles.

To delete a role, you need the `DROP ROLE` or `DROP USER` privilege.

14.9.4.1.11 Authorization table

In addition to four system privilege tables, the RBAC system introduces two new system privilege tables:

- `mysql.role_edges`: records the authorization relationship of the role and user.
- `mysql.default_roles`: records default roles of each user.

`mysql.role_edges`

`mysql.role_edges` contains the following data:

```
SELECT * FROM mysql.role_edges;
```

```
+-----------+-----------+---------+---------+-------------------+
| FROM_HOST | FROM_USER | TO_HOST | TO_USER | WITH_ADMIN_OPTION |
+-----------+-----------+---------+---------+-------------------+
| %         | r_1       | %       | u_1     | N                 |
+-----------+-----------+---------+---------+-------------------+
1 row in set (0.00 sec)
```

- `FROM_HOST` and `FROM_USER` indicate the role's host name and user name respectively.
- `TO_HOST` and `TO_USER` indicate the host name and user name of the user to which a role is granted.

`mysql.default_roles`

`mysql.default_roles` shows which roles have been enabled by default for each user.

```
SELECT * FROM mysql.default_roles;
```

```
+------+------+-------------------+-------------------+
| HOST | USER | DEFAULT_ROLE_HOST | DEFAULT_ROLE_USER |
+------+------+-------------------+-------------------+
| %    | u_1  | %                 | r_1               |
| %    | u_1  | %                 | r_2               |
+------+------+-------------------+-------------------+
2 rows in set (0.00 sec)
```

- `HOST` and `USER` indicate the user's host name and user name respectively.
- `DEFAULT_ROLE_HOST` and `DEFAULT_ROLE_USER` indicate the host name and user name of the default role respectively.

14.9.4.1.12 References

Because RBAC, user management, and privilege management are closely related, you can refer to operation details in the following resources:

- TiDB Privilege Management
- TiDB User Account Management

14.9.5 Certificate-Based Authentication for Login

TiDB supports a certificate-based authentication method for users to log into TiDB. With this method, TiDB issues certificates to different users, uses encrypted connections to transfer data, and verifies certificates when users log in. This approach is more secure than the traditional password-based authentication method commonly used by MySQL users and is thus adopted by an increasing number of users.

To use certificate-based authentication, you might need to perform the following operations:

- Create security keys and certificates
- Configure certificates for TiDB and the client
- Configure the user certificate information to be verified when the user logs in
- Update and replace certificates

The rest of the document introduces in detail how to perform these operations.

14.9.5.1 Create security keys and certificates

It is recommended that you use OpenSSL to create keys and certificates. The certificate generation process is similar to the process described in Enable TLS Between TiDB Clients and Servers. The following paragraphs demonstrate how to configure more attribute fields that need to be verified in the certificate.

14.9.5.1.1 Generate CA key and certificate

1. Execute the following command to generate the CA key:

```
sudo openssl genrsa 2048 > ca-key.pem
```

The output of the above command:

```
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................+++++
.............................................+++++
e is 65537 (0x010001)
```

2. Execute the following command to generate the certificate corresponding to the CA key:

```
sudo openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca
    ↪ -cert.pem
```

3. Enter detailed certificate information. For example:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (e.g. city) []:San Francisco
Organization Name (e.g. company) [Internet Widgits Pty Ltd]:PingCAP Inc
    ↪ .
Organizational Unit Name (e.g. section) []:TiDB
Common Name (e.g. server FQDN or YOUR name) []:TiDB admin
Email Address []:s@pingcap.com
```

> **Note:**
> In the above certificate details, texts after : are the entered information.

14.9.5.1.2 Generate server key and certificate

1. Execute the following command to generate the server key:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout server-
    ↪ key.pem -out server-req.pem
```

2. Enter detailed certificate information. For example:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (e.g. city) []:San Francisco
Organization Name (e.g. company) [Internet Widgits Pty Ltd]:PingCAP Inc
    ↪ .
Organizational Unit Name (e.g. section) []:TiKV
Common Name (e.g. server FQDN or YOUR name) []:TiKV Test Server
Email Address []:k@pingcap.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. Execute the following command to generate the RSA key of the server:

```
sudo openssl rsa -in server-key.pem -out server-key.pem
```

The output of the above command:

```
writing RSA key
```

4. Use the CA certificate signature to generate the signed server certificate:

```
sudo openssl x509 -req -in server-req.pem -days 365000 -CA ca-cert.pem
    ↪ -CAkey ca-key.pem -set_serial 01 -out server-cert.pem
```

The output of the above command (for example):

```
Signature ok
subject=C = US, ST = California, L = San Francisco, O = PingCAP Inc.,
    ↪ OU = TiKV, CN = TiKV Test Server, emailAddress = k@pingcap.com
Getting CA Private Key
```

> **Note:**
> When you log in, TiDB checks whether the information in the `subject` section of the above output is consistent or not.

14.9.5.1.3 Generate client key and certificate

After generating the server key and certificate, you need to generate the key and certificate for the client. It is often necessary to generate different keys and certificates for different users.

1. Execute the following command to generate the client key:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout client-
    ↪ key.pem -out client-req.pem
```

2. Enter detailed certificate information. For example:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (e.g. city) []:San Francisco
Organization Name (e.g. company) [Internet Widgits Pty Ltd]:PingCAP Inc
    ↪ .
Organizational Unit Name (e.g. section) []:TiDB
Common Name (e.g. server FQDN or YOUR name) []:tpch-user1
Email Address []:zz@pingcap.com
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. Execute the following command to generate the RSA key of the client:

```
sudo openssl rsa -in client-key.pem -out client-key.pem
```

The output of the above command:

```
writing RSA key
```

4. Use the CA certificate signature to generate the client certificate:

```
sudo openssl x509 -req -in client-req.pem -days 365000 -CA ca-cert.pem
    ↪ -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

The output of the above command (for example):

```
Signature ok
subject=C = US, ST = California, L = San Francisco, O = PingCAP Inc.,
    ↪ OU = TiDB, CN = tpch-user1, emailAddress = zz@pingcap.com
Getting CA Private Key
```

> **Note:**
> The information of the `subject` section in the above output is used for certificate configuration for login verification in the `require` section.

14.9.5.1.4 Verify certificate

Execute the following command to verify certificate:

```
openssl verify -CAfile ca-cert.pem server-cert.pem client-cert.pem
```

If the certificate is verified, you will see the following result:

```
server-cert.pem: OK
client-cert.pem: OK
```

14.9.5.2 Configure TiDB and the client to use certificates

After generating the certificates, you need to configure the TiDB server and the client to use the corresponding server certificate or client certificate.

14.9.5.2.1 Configure TiDB to use server certificate

Modify the `[security]` section in the TiDB configuration file. This step specifies the directory in which the CA certificate, the server key, and the server certificate are stored. You can replace `path/to/server-cert.pem`, `path/to/server-key.pem`, `path/to/ca-cert.pem` with your own directory.

```
[security]
ssl-cert ="path/to/server-cert.pem"
ssl-key ="path/to/server-key.pem"
ssl-ca="path/to/ca-cert.pem"
```

Start TiDB and check logs. If the following information is displayed in the log, the configuration is successful:

```
[INFO] [server.go:264] ["secure connection is enabled"] ["client
    ↪ verification enabled"=true]
```

14.9.5.2.2 Configure the client to use client certificate

Configure the client so that the client uses the client key and certificate for login.

Taking the MySQL client as an example, you can use the newly created client certificate, client key and CA by specifying `ssl-cert`, `ssl-key`, and `ssl-ca`:

```
mysql -utest -h0.0.0.0 -P4000 --ssl-cert /path/to/client-cert.new.pem --ssl-
    ↪ key /path/to/client-key.new.pem --ssl-ca /path/to/ca-cert.pem
```

> **Note:**
>
> `/path/to/client-cert.new.pem`, `/path/to/client-key.new.pem`, and `/
> ↪ path/to/ca-cert.pem` are the directory of the CA certificate, client key, and client certificate. You can replace them with your own directory.

14.9.5.3 Configure the user certificate information for login verification

First, connect TiDB using the client to configure the login verification. Then, you can get and configure the user certificate information to be verified.

14.9.5.3.1 Get user certificate information

The user certificate information can be specified by `require subject`, `require issuer`, `require san`, and `require cipher`, which are used to check the X509 certificate attributes.

- `require subject`: Specifies the `subject` information of the client certificate when you log in. With this option specified, you do not need to configure `require ssl` or x509. The information to be specified is consistent with the entered `subject` information in Generate client keys and certificates.

  To get this option, execute the following command:

```
openssl x509 -noout -subject -in client-cert.pem | sed 's/.\{8\}//' |
    ↪ sed 's/, /\//g' | sed 's/ = /=/g' | sed 's/^/\//'
```

- `require issuer`: Specifies the `subject` information of the CA certificate that issues the user certificate. The information to be specified is consistent with the entered `subject` information in Generate CA key and certificate.

  To get this option, execute the following command:

```
openssl x509 -noout -subject -in ca-cert.pem | sed 's/.\{8\}//' | sed '
    ↪ s/, /\//g' | sed 's/ = /=/g' | sed 's/^/\//'
```

- `require san`: Specifies the `Subject Alternative Name` information of the CA certificate that issues the user certificate. The information to be specified is consistent with the `alt_names` of the `openssl.cnf` configuration file used to generate the client certificate.

  - Execute the following command to get the information of the `require san` item in the generated certificate:

```
openssl x509 -noout -extensions subjectAltName -in client.crt
```

  - `require san` currently supports the following `Subject Alternative Name` check items:

    - URI
    - IP
    - DNS

  - Multiple check items can be configured after they are connected by commas. For example, configure `require san` as follows for the `u1` user:

```
create user 'u1'@'%' require san 'DNS:d1,URI:spiffe://example.org/
    ↪ myservice1,URI:spiffe://example.org/myservice2';
```

    The above configuration only allows the `u1` user to log in to TiDB using the certificate with the URI item `spiffe://example.org/myservice1` or `spiffe://
    ↪ example.org/myservice2` and the DNS item `d1`.

- `require cipher`: Checks the cipher method supported by the client. Use the following statement to check the list of supported cipher methods:

```
SHOW SESSION STATUS LIKE 'Ssl_cipher_list';
```

14.9.5.3.2 Configure user certificate information

After getting the user certificate information (`require subject`, `require issuer`, `require san`, `require cipher`), configure these information to be verified when creating a user, granting privileges, or altering a user. Replace `<replaceable>` with the corresponding information in the following statements.

You can configure one option or multiple options using the space or `and` as the separator.

- Configure user certificate when creating a user (`create user`):

```
create user 'u1'@'%' require issuer '<replaceable>' subject '<
    ↪ replaceable>' san '<replaceable>' cipher '<replaceable>';
```

- Configure user certificate when granting privileges:

```
grant all on *.* to 'u1'@'%' require issuer '<replaceable>' subject '<
    ↪ replaceable>' san '<replaceable>' cipher '<replaceable>';
```

- Configure user certificate when altering a user:

```
alter user 'u1'@'%' require issuer '<replaceable>' subject '<
    ↪ replaceable>' san '<replaceable>' cipher '<replaceable>';
```

After the above configuration, the following items will be verified when you log in:

- SSL is used; the CA that issues the client certificate is consistent with the CA configured in the server.
- The `issuer` information of the client certificate matches the information specified in `require issuer`.
- The `subject` information of the client certificate matches the information specified in `require cipher`.
- The `Subject Alternative Name` information of the client certificate matches the information specified in `require san`.

You can log into TiDB only after all the above items are verified. Otherwise, the `ERROR 1045 (28000): Access denied` error is returned. You can use the following command to check the TLS version, the cipher algorithm and whether the current connection uses the certificate for the login.

Connect the MySQL client and execute the following statement:

```
\s
```

The output:

```
--------------
mysql Ver 15.1 Distrib 10.4.10-MariaDB, for Linux (x86_64) using readline
    ↪ 5.1

Connection id:    1
Current database: test
Current user:     root@127.0.0.1
SSL:              Cipher in use is TLS_AES_256_GCM_SHA384
```

Then execute the following statement:

```
show variables like '%ssl%';
```

The output:

```
+---------------+---------------------------------+
| Variable_name | Value                           |
+---------------+---------------------------------+
| ssl_cert      | /path/to/server-cert.pem        |
| ssl_ca        | /path/to/ca-cert.pem            |
| have_ssl      | YES                             |
| have_openssl  | YES                             |
| ssl_key       | /path/to/server-key.pem         |
+---------------+---------------------------------+
6 rows in set (0.067 sec)
```

14.9.5.4 Update and replace certificate

The key and certificate are updated regularly. The following sections introduce how to update the key and certificate.

The CA certificate is the basis for mutual verification between the client and server. To replace the CA certificate, generate a combined certificate that supports the authentication for both old and new certificates. On the client and server, first replace the CA certificate, then replace the client/server key and certificate.

14.9.5.4.1 Update CA key and certificate

1. Back up the old CA key and certificate (suppose that `ca-key.pem` is stolen):

   ```
   mv ca-key.pem ca-key.old.pem && \
   mv ca-cert.pem ca-cert.old.pem
   ```

2. Generate the new CA key:

```
sudo openssl genrsa 2048 > ca-key.pem
```

3. Generate the new CA certificate using the newly generated CA key:

```
sudo openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca
    ↪ -cert.new.pem
```

> **Note:**
> Generating the new CA certificate is to replace the keys and certificates on the client and server, and to ensure that online users are not affected. Therefore, the appended information in the above command must be consistent with the `require issuer` information.

4. Generate the combined CA certificate:

```
cat ca-cert.new.pem ca-cert.old.pem > ca-cert.pem
```

After the above operations, restart the TiDB server with the newly created combined CA certificate. Then the server accepts both the new and old CA certificates.

Also replace the old CA certificate with the combined certificate so that the client accepts both the old and new CA certificates.

14.9.5.4.2 Update client key and certificate

> **Note:**
> Perform the following steps **only after** you have replaced the old CA certificate on the client and server with the combined CA certificate.

1. Generate the new RSA key of the client:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout client-
    ↪ key.new.pem -out client-req.new.pem && \
sudo openssl rsa -in client-key.new.pem -out client-key.new.pem
```

> **Note:**
>
> The above command is to replace the client key and certificate, and to ensure that the online users are not affected. Therefore, the appended information in the above command must be consistent with the `require subject` information.

2. Use the combined certificate and the new CA key to generate the new client certificate:

```
sudo openssl x509 -req -in client-req.new.pem -days 365000 -CA ca-cert.
    ↪ pem -CAkey ca-key.pem -set_serial 01 -out client-cert.new.pem
```

3. Make the client (for example, MySQL) connect TiDB with the new client key and certificate:

```
mysql -utest -h0.0.0.0 -P4000 --ssl-cert /path/to/client-cert.new.pem
    ↪ --ssl-key /path/to/client-key.new.pem --ssl-ca /path/to/ca-cert.
    ↪ pem
```

> **Note:**
>
> `/path/to/client-cert.new.pem`, `/path/to/client-key.new.pem`, and `/path/to/ca-cert.pem` specify the directory of the CA certificate, client key, and client certificate. You can replace them with your own directory.

14.9.5.4.3 Update the server key and certificate

1. Generate the new RSA key of the server:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout server-
    ↪ key.new.pem -out server-req.new.pem && \
sudo openssl rsa -in server-key.new.pem -out server-key.new.pem
```

2. Use the combined CA certificate and the new CA key to generate the new server certificate:

```
sudo openssl x509 -req -in server-req.new.pem -days 365000 -CA ca-cert.
    ↪ pem -CAkey ca-key.pem -set_serial 01 -out server-cert.new.pem
```

3. Configure the TiDB server to use the new server key and certificate. See Configure TiDB server for details.

14.10 SQL

14.10.1 SQL Language Structure and Syntax

14.10.1.1 Attributes

14.10.1.1.1 AUTO_INCREMENT

This document introduces the `AUTO_INCREMENT` column attribute, including its concept, implementation principles, auto-increment related features, and restrictions.

> **Note:**
>
> The `AUTO_INCREMENT` attribute might cause hotspot in production environments. See Troubleshoot HotSpot Issues for details. It is recommended to use `AUTO_RANDOM` instead.

Concept

`AUTO_INCREMENT` is a column attribute that is used to automatically fill in default column values. When the `INSERT` statement does not specify values for the `AUTO_INCREMENT` column, the system automatically assigns values to this column.

For performance reasons, `AUTO_INCREMENT` numbers are allocated in a batch of values (30 thousand by default) to each TiDB server. This means that while `AUTO_INCREMENT` numbers are guaranteed to be unique, values assigned to an `INSERT` statement will only be monotonic on a per TiDB server basis.

The following is a basic example of `AUTO_INCREMENT`:

```
CREATE TABLE t(id int PRIMARY KEY AUTO_INCREMENT, c int);
```

```
INSERT INTO t(c) VALUES (1);
INSERT INTO t(c) VALUES (2);
INSERT INTO t(c) VALUES (3), (4), (5);
```

```
mysql> SELECT * FROM t;
+----+---+
| id | c |
+----+---+
| 1  | 1 |
| 2  | 2 |
| 3  | 3 |
| 4  | 4 |
| 5  | 5 |
```

```
+----+---+
5 rows in set (0.01 sec)
```

In addition, `AUTO_INCREMENT` also supports the `INSERT` statements that explicitly specify column values. In such cases, TiDB stores the explicitly specified values:

```
INSERT INTO t(id, c) VALUES (6, 6);
```

```
mysql> SELECT * FROM t;
+----+---+
| id | c |
+----+---+
| 1  | 1 |
| 2  | 2 |
| 3  | 3 |
| 4  | 4 |
| 5  | 5 |
| 6  | 6 |
+----+---+
6 rows in set (0.01 sec)
```

The usage above is the same as that of `AUTO_INCREMENT` in MySQL. However, in terms of the specific value that is implicitly assigned, TiDB differs from MySQL significantly.

Implementation principles

TiDB implements the `AUTO_INCREMENT` implicit assignment in the following way:

For each auto-increment column, a globally visible key-value pair is used to record the maximum ID that has been assigned. In a distributed environment, communication between nodes has some overhead. Therefore, to avoid the issue of write amplification, each TiDB node applies for a batch of consecutive IDs as caches when assigning IDs, and then applies for the next batch of IDs after the first batch is assigned. Therefore, TiDB nodes do not apply to the storage node for IDs when assigning IDs each time. For example:

```
CREATE TABLE t(id int UNIQUE KEY AUTO_INCREMENT, c int);
```

Assume two TiDB instances, `A` and `B`, in the cluster. If you execute an `INSERT` statement on the `t` table on `A` and `B` respectively:

```
INSERT INTO t (c) VALUES (1)
```

Instance `A` might cache the auto-increment IDs of `[1,30000]`, and instance `B` might cache the auto-increment IDs of `[30001,60000]`. In `INSERT` statements to be executed, these cached IDs of each instance will be assigned to the `AUTO_INCREMENT` column as the default values.

Basic Features

Uniqueness

Warning:

When the cluster has multiple TiDB instances, if the table schema contains the auto-increment IDs, it is recommended not to use explicit insert and implicit assignment at the same time, which means using the default values of the auto-increment column and the custom values. Otherwise, it might break the uniqueness of implicitly assigned values.

In the example above, perform the following operations in order:

1. The client inserts a statement `INSERT INTO t VALUES (2, 1)` to instance `B`, which sets `id` to `2`. The statement is successfully executed.
2. The client sends a statement `INSERT INTO t (c)(1)` to instance `A`. This statement does not specify the value of `id`, so the ID is assigned by `A`. At present, because `A` caches the IDs of `[1, 30000]`, it might assign `2` as the value of the auto-increment ID, and increases the local counter by `1`. At this time, the data whose ID is `2` already exists in the database, so the `Duplicated Error` error is returned.

Monotonicity

TiDB guarantees that `AUTO_INCREMENT` values are monotonic (always increasing) on a per-server basis. Consider the following example where consecutive `AUTO_INCREMENT` values of 1-3 are generated:

```
CREATE TABLE t (a int PRIMARY KEY AUTO_INCREMENT, b timestamp NOT NULL
    ↪ DEFAULT NOW());
INSERT INTO t (a) VALUES (NULL), (NULL), (NULL);
SELECT * FROM t;
```

```
Query OK, 0 rows affected (0.11 sec)

Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

+---+---------------------+
| a | b                   |
+---+---------------------+
| 1 | 2020-09-09 20:38:22 |
| 2 | 2020-09-09 20:38:22 |
| 3 | 2020-09-09 20:38:22 |
+---+---------------------+
3 rows in set (0.00 sec)
```

Monotonicity is not the same guarantee as consecutive. Consider the following example:

```
CREATE TABLE t (id INT NOT NULL PRIMARY KEY auto_increment, a VARCHAR(10),
    ↪ cnt INT NOT NULL DEFAULT 1, UNIQUE KEY (a));
INSERT INTO t (a) VALUES ('A'), ('B');
SELECT * FROM t;
INSERT INTO t (a) VALUES ('A'), ('C') ON DUPLICATE KEY UPDATE cnt = cnt +
    ↪ 1;
SELECT * FROM t;
```

```
Query OK, 0 rows affected (0.00 sec)

Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

+----+------+-----+
| id | a    | cnt |
+----+------+-----+
|  1 | A    |   1 |
|  2 | B    |   1 |
+----+------+-----+
2 rows in set (0.00 sec)

Query OK, 3 rows affected (0.00 sec)
Records: 2 Duplicates: 1 Warnings: 0

+----+------+-----+
| id | a    | cnt |
+----+------+-----+
|  1 | A    |   2 |
|  2 | B    |   1 |
|  4 | C    |   1 |
+----+------+-----+
3 rows in set (0.00 sec)
```

In this example, the `AUTO_INCREMENT` value of `3` is allocated for the `INSERT` of the key `A` in `INSERT INTO t (a)VALUES ('A'), ('C')ON DUPLICATE KEY UPDATE cnt = cnt + ↪ 1;` but never used because this `INSERT` statement contains a duplicate key `A`. This leads to a gap where the sequence is non-consecutive. This behavior is considered legal, even though it differs from MySQL. MySQL will also have gaps in the sequence in other scenarios such as transactions being aborted and rolled back.

AUTO_ID_CACHE

The `AUTO_INCREMENT` sequence might appear to *jump* dramatically if an `INSERT` operation is performed against a different TiDB server. This is caused by the fact that each server has its own cache of `AUTO_INCREMENT` values:

```
CREATE TABLE t (a int PRIMARY KEY AUTO_INCREMENT, b timestamp NOT NULL
    ↪ DEFAULT NOW());
INSERT INTO t (a) VALUES (NULL), (NULL), (NULL);
INSERT INTO t (a) VALUES (NULL);
SELECT * FROM t;
```

```
Query OK, 1 row affected (0.03 sec)

+---------+---------------------+
| a       | b                   |
+---------+---------------------+
|       1 | 2020-09-09 20:38:22 |
|       2 | 2020-09-09 20:38:22 |
|       3 | 2020-09-09 20:38:22 |
| 2000001 | 2020-09-09 20:43:43 |
+---------+---------------------+
4 rows in set (0.00 sec)
```

A new `INSERT` operation against the initial TiDB server generates the `AUTO_INCREMENT` value of `4`. This is because the initial TiDB server still has space left in the `AUTO_INCREMENT` cache for allocation. In this case, the sequence of values cannot be considered globally monotonic, because the value of `4` is inserted after the value of `2000001`:

```
mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t ORDER BY b;
+---------+---------------------+
| a       | b                   |
+---------+---------------------+
|       1 | 2020-09-09 20:38:22 |
|       2 | 2020-09-09 20:38:22 |
|       3 | 2020-09-09 20:38:22 |
| 2000001 | 2020-09-09 20:43:43 |
|       4 | 2020-09-09 20:44:43 |
+---------+---------------------+
5 rows in set (0.00 sec)
```

The `AUTO_INCREMENT` cache does not persist across TiDB server restarts. The following `INSERT` statement is performed after the initial TiDB server is restarted:

```
mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t ORDER BY b;
```

```
+---------+---------------------+
| a       | b                   |
+---------+---------------------+
|       1 | 2020-09-09 20:38:22 |
|       2 | 2020-09-09 20:38:22 |
|       3 | 2020-09-09 20:38:22 |
| 2000001 | 2020-09-09 20:43:43 |
|       4 | 2020-09-09 20:44:43 |
| 2030001 | 2020-09-09 20:54:11 |
+---------+---------------------+
6 rows in set (0.00 sec)
```

A high rate of TiDB server restarts might contribute to the exhaustion of `AUTO_INCREMENT` values. In the above example, the initial TiDB server still has values `[5-30000]` free in its cache. These values are lost, and will not be reallocated.

It is not recommended to rely on`AUTO_INCREMENT` values being continuous. Consider the following example, where a TiDB server has a cache of values `[2000001-2030000]`. By manually inserting the value `2029998`, you can see the behavior as a new cache range is retrieved:

```
mysql> INSERT INTO t (a) VALUES (2029998);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t ORDER BY b;
+---------+---------------------+
| a       | b                   |
+---------+---------------------+
|       1 | 2020-09-09 20:38:22 |
|       2 | 2020-09-09 20:38:22 |
|       3 | 2020-09-09 20:38:22 |
| 2000001 | 2020-09-09 20:43:43 |
|       4 | 2020-09-09 20:44:43 |
| 2030001 | 2020-09-09 20:54:11 |
```

```
| 2029998 | 2020-09-09 21:08:11 |
| 2029999 | 2020-09-09 21:08:11 |
| 2030000 | 2020-09-09 21:08:11 |
| 2060001 | 2020-09-09 21:08:11 |
| 2060002 | 2020-09-09 21:08:11 |
+---------+---------------------+
11 rows in set (0.00 sec)
```

After the value 2030000 is inserted, the next value is 2060001. This jump in sequence is due to another TiDB server obtaining the intermediate cache range of [2030001-2060000] ↪ . When multiple TiDB servers are deployed, there will be gaps in the AUTO_INCREMENT sequence because cache requests are interleaved.

Cache size control

In earlier versions of TiDB, the cache size of the auto-increment ID was transparent to users. Starting from v3.0.14, v3.1.2, and v4.0.rc-2, TiDB has introduced the AUTO_ID_CACHE table option to allow users to set the cache size for allocating the auto-increment ID.

```
CREATE TABLE t(a int AUTO_INCREMENT key) AUTO_ID_CACHE 100;
Query OK, 0 rows affected (0.02 sec)

INSERT INTO t values();
Query OK, 1 row affected (0.00 sec)

SELECT * FROM t;
+---+
| a |
+---+
| 1 |
+---+
1 row in set (0.01 sec)

SHOW CREATE TABLE t;
+--
   ↪ -----+-----------------------------------------------------------------------
   ↪
| Table | Create Table
   ↪
   ↪ |
+--
   ↪ -----+-----------------------------------------------------------------------
   ↪
| t     | CREATE TABLE `t` (
  `a` int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`a`) /*T![clustered_index] CLUSTERED */
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin AUTO_INCREMENT
    ↪ =101 /*T![auto_id_cache] AUTO_ID_CACHE=100 */ |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

At this time, if you restart TiDB, the auto-increment ID cache will be lost, and new insert operations will allocate IDs starting from a higher value beyond the previously cached range.

```
INSERT INTO t VALUES();
Query OK, 1 row affected (0.00 sec)

SELECT * FROM t;
+-----+
| a   |
+-----+
|   1 |
| 101 |
+-----+
2 rows in set (0.01 sec)
```

The newly allocated value is `101`. This shows that the size of cache for allocating auto-increment IDs is `100`.

In addition, when the length of consecutive IDs in a batch `INSERT` statement exceeds the length of `AUTO_ID_CACHE`, TiDB increases the cache size accordingly to ensure that the statement can insert data properly.

Clear the auto-increment ID cache

In some scenarios, you might need to clear the auto-increment ID cache to ensure data consistency. For example:

- In the scenario of incremental replication using Data Migration (DM), once the replication is complete, data writing to the downstream TiDB switches from DM to your application's write operations. Meanwhile, the ID writing mode of the auto-increment column usually switches from explicit insertion to implicit allocation.
- When your application involves both explicit ID insertion and implicit ID allocation, you need to clear the auto-increment ID cache to avoid conflicts between future implicitly allocated IDs and previously explicitly inserted IDs, which could result in primary key conflict errors. For more information, see Uniqueness.

To clear the auto-increment ID cache on all TiDB nodes in the cluster, you can execute the `ALTER TABLE` statement with `AUTO_INCREMENT = 0`. For example:

```
CREATE TABLE t(a int AUTO_INCREMENT key) AUTO_ID_CACHE 100;
Query OK, 0 rows affected (0.02 sec)

INSERT INTO t VALUES();
Query OK, 1 row affected (0.02 sec)

INSERT INTO t VALUES(50);
Query OK, 1 row affected (0.00 sec)

SELECT * FROM t;
+----+
| a  |
+----+
|  1 |
| 50 |
+----+
2 rows in set (0.01 sec)
```

```
ALTER TABLE t AUTO_INCREMENT = 0;
Query OK, 0 rows affected, 1 warning (0.07 sec)

SHOW WARNINGS;
+--
   ↪ -------+------+-----------------------------------------------------------------
   ↪
| Level   | Code | Message
   ↪                                                                 |
+--
   ↪ -------+------+-----------------------------------------------------------------
   ↪
| Warning | 1105 | Can't reset AUTO_INCREMENT to 0 without FORCE option,
   ↪ using 101 instead |
+---------+------+-----------------------------------------------------------------
   ↪
1 row in set (0.01 sec)

INSERT INTO t VALUES();
Query OK, 1 row affected (0.02 sec)

SELECT * FROM t;
+-----+
| a   |
+-----+
|   1 |
```

```
|  50 |
| 101 |
+-----+
3 rows in set (0.01 sec)
```

Auto-increment step size and offset

Starting from v3.0.9 and v4.0.0-rc.1, similar to the behavior of MySQL, the value implicitly assigned to the auto-increment column is controlled by the `@@auto_increment_increment` and `@@auto_increment_offset` session variables.

The value (ID) implicitly assigned to auto-increment columns satisfies the following equation:

```
(ID - auto_increment_offset)% auto_increment_increment == 0
```

Restrictions

Currently, `AUTO_INCREMENT` has the following restrictions when used in TiDB:

- It must be defined on the first column of the primary key or the first column of an index.
- It must be defined on the column of `INTEGER`, `FLOAT`, or `DOUBLE` type.
- It cannot be specified on the same column with the `DEFAULT` column value.
- `ALTER TABLE` cannot be used to add the `AUTO_INCREMENT` attribute.
- `ALTER TABLE` can be used to remove the `AUTO_INCREMENT` attribute. However, starting from v2.1.18 and v3.0.4, TiDB uses the session variable `@@tidb_allow_remove_auto_inc` to control whether `ALTER TABLE MODIFY` or `ALTER TABLE CHANGE` can be used to remove the `AUTO_INCREMENT` attribute of a column. By default, you cannot use `ALTER TABLE MODIFY` or `ALTER TABLE CHANGE` to remove the `AUTO_INCREMENT` attribute.
- `ALTER TABLE` requires the `FORCE` option to set the `AUTO_INCREMENT` value to a smaller value.
- Setting the `AUTO_INCREMENT` to a value smaller than `MAX(<auto_increment_column>)` leads to duplicate keys because pre-existing values are not skipped.

14.10.1.1.2 AUTO_RANDOM New in v3.1.0

User scenario

Since the value of `AUTO_RANDOM` is random and unique, `AUTO_RANDOM` is often used in place of `AUTO_INCREMENT` to avoid write hotspot in a single storage node caused by TiDB assigning consecutive IDs. If the current `AUTO_INCREMENT` column is a primary key and the type is `BIGINT`, you can execute the `ALTER TABLE t MODIFY COLUMN id BIGINT AUTO_RANDOM(5);` statement to switch from `AUTO_INCREMENT` to `AUTO_RANDOM`.

For more information about how to handle highly concurrent write-heavy workloads in TiDB, see Highly concurrent write best practices.

Basic concepts

`AUTO_RANDOM` is a column attribute that is used to automatically assign values to a `BIGINT` column. Values assigned automatically are **random** and **unique**.

To create a table with an `AUTO_RANDOM` column, you can use the following statements. The `AUTO_RANDOM` column must be included in a primary key, and the primary key must only have the `AUTO_RANDOM` column.

```
CREATE TABLE t (a BIGINT AUTO_RANDOM, b VARCHAR(255), PRIMARY KEY (a));
CREATE TABLE t (a BIGINT PRIMARY KEY AUTO_RANDOM, b VARCHAR(255));
CREATE TABLE t (a BIGINT AUTO_RANDOM(6), b VARCHAR(255), PRIMARY KEY (a));
```

You can wrap the keyword `AUTO_RANDOM` in an executable comment. For more details, refer to TiDB specific comment syntax.

```
CREATE TABLE t (a bigint /*T![auto_rand] AUTO_RANDOM */, b VARCHAR(255),
    ↪ PRIMARY KEY (a));
CREATE TABLE t (a bigint PRIMARY KEY /*T![auto_rand] AUTO_RANDOM */, b
    ↪ VARCHAR(255));
CREATE TABLE t (a BIGINT /*T![auto_rand] AUTO_RANDOM(6) */, b VARCHAR(255),
    ↪  PRIMARY KEY (a));
```

When you execute an `INSERT` statement:

- If you explicitly specify the value of the `AUTO_RANDOM` column, it is inserted into the table as is.
- If you do not explicitly specify the value of the `AUTO_RANDOM` column, TiDB generates a random value and inserts it into the table.

```
tidb> CREATE TABLE t (a BIGINT PRIMARY KEY AUTO_RANDOM, b VARCHAR(255));
Query OK, 0 rows affected, 1 warning (0.01 sec)

tidb> INSERT INTO t(a, b) VALUES (1, 'string');
Query OK, 1 row affected (0.00 sec)

tidb> SELECT * FROM t;
+---+--------+
| a | b      |
+---+--------+
| 1 | string |
+---+--------+
1 row in set (0.01 sec)

tidb> INSERT INTO t(b) VALUES ('string2');
Query OK, 1 row affected (0.00 sec)

tidb> INSERT INTO t(b) VALUES ('string3');
```

```
Query OK, 1 row affected (0.00 sec)

tidb> SELECT * FROM t;
+--------------------+---------+
| a                  | b       |
+--------------------+---------+
|                  1 | string  |
| 1152921504606846978 | string2 |
| 4899916394579099651 | string3 |
+--------------------+---------+
3 rows in set (0.00 sec)
```

The `AUTO_RANDOM(S)` column value automatically assigned by TiDB has a total of 64 bits. `S` is the number of shard bits. The value ranges from `1` to `15`. The default value is `5`.

The structure of an `AUTO_RANDOM` value is as follows:

Total number of bits	Sign bit	Shard bits	Auto-increment bits
64 bits	0/1 bit	S bits	(64-1-S) bits

- The length of the sign bit is determined by the existence of an `UNSIGNED` attribute. If there is an `UNSIGNED` attribute, the length is `0`. Otherwise, the length is `1`.
- The content of the shard bits is obtained by calculating the hash value of the starting time of the current transaction. To use a different length of shard bits (such as 10), you can specify `AUTO_RANDOM(10)` when creating the table.
- The value of the auto-increment bits is stored in the storage engine and allocated sequentially. Each time a new value is allocated, the value is incremented by 1. The auto-increment bits ensure that the values of `AUTO_RANDOM` are unique globally. When the auto-increment bits are exhausted, an error `Failed to read auto-increment ↪ value from storage engine` is reported when the value is allocated again.

Note:

Selection of shard bits (`S`):

- Since there is a total of 64 available bits, the shard bits length affects the auto-increment bits length. That is, as the shard bits length increases, the length of auto-increment bits decreases, and vice versa. Therefore, you need to balance the randomness of allocated values and available space.
- The best practice is to set the shard bits as `log(2, x)`, in which `x` is the current number of storage engines. For example, if there are 16 TiKV nodes in a TiDB cluster, you can set the shard bits as `log(2, 16)`, that

> is 4. After all regions are evenly scheduled to each TiKV node, the load of bulk writes can be uniformly distributed to different TiKV nodes to maximize resource utilization.

Values allocated implicitly to the AUTO_RANDOM column affect last_insert_id(). To get the ID that TiDB last implicitly allocates, you can use the SELECT last_insert_id () statement.

To view the shard bits number of the table with an AUTO_RANDOM column, you can execute the SHOW CREATE TABLE statement. You can also see the value of the PK_AUTO_RANDOM_BITS =x mode in the TIDB_ROW_ID_SHARDING_INFO column in the information_schema.tables system table. x is the number of shard bits.

After creating a table with an AUTO_RANDOM column, you can use SHOW WARNINGS to view the maximum implicit allocation times:

```
CREATE TABLE t (a BIGINT AUTO_RANDOM, b VARCHAR(255), PRIMARY KEY (a));
SHOW WARNINGS;
```

The output is as follows:

```
+-------+------+-----------------------------------------------------------+
| Level | Code | Message                                                   |
+-------+------+-----------------------------------------------------------+
| Note | 1105 | Available implicit allocation times: 288230376151711743 |
+-------+------+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

Implicit allocation rules of IDs

TiDB implicitly allocates values to AUTO_RANDOM columns similarly to AUTO_INCREMENT columns. They are also controlled by the session-level system variables auto_increment_increment and auto_increment_offset. The auto-increment bits (ID) of implicitly allocated values conform to the equation (ID - auto_increment_offset)% auto_increment_increment == 0.

Restrictions

Pay attention to the following restrictions when you use AUTO_RANDOM:

- To insert values explicitly, you need to set the value of the @@allow_auto_random_explicit_insert system variable to 1 (0 by default). It is **not** recommended that you explicitly specify a value for the column with the AUTO_RANDOM attribute when you insert data. Otherwise, the numeral values that can be automatically allocated for this table might be used up in advance.

- Specify this attribute for the primary key column **ONLY** as the `BIGINT` type. Otherwise, an error occurs. In addition, when the attribute of the primary key is `NONCLUSTERED`, `AUTO_RANDOM` is not supported even on the integer primary key. For more details about the primary key of the `CLUSTERED` type, refer to clustered index.
- You cannot use `ALTER TABLE` to modify the `AUTO_RANDOM` attribute, including adding or removing this attribute.
- You cannot use `ALTER TABLE` to change from `AUTO_INCREMENT` to `AUTO_RANDOM` if the maximum value is close to the maximum value of the column type.
- You cannot change the column type of the primary key column that is specified with `AUTO_RANDOM` attribute.
- You cannot specify `AUTO_RANDOM` and `AUTO_INCREMENT` for the same column at the same time.
- You cannot specify `AUTO_RANDOM` and `DEFAULT` (the default value of a column) for the same column at the same time.
- When`AUTO_RANDOM` is used on a column, it is difficult to change the column attribute back to `AUTO_INCREMENT` because the auto-generated values might be very large.

14.10.1.1.3 SHARD_ROW_ID_BITS

This document introduces the `SHARD_ROW_ID_BITS` table attribute, which is used to set the number of bits of the shards after the implicit `_tidb_rowid` is sharded.

Concept

For the tables with a non-clustered primary key or no primary key, TiDB uses an implicit auto-increment row ID. When a large number of `INSERT` operations are performed, the data is written into a single Region, causing a write hot spot.

To mitigate the hot spot issue, you can configure `SHARD_ROW_ID_BITS`. The row IDs are scattered and the data are written into multiple different Regions.

- `SHARD_ROW_ID_BITS = 4` indicates 16 shards
- `SHARD_ROW_ID_BITS = 6` indicates 64 shards
- `SHARD_ROW_ID_BITS = 0` indicates the default 1 shard

For details on the usage, see the Troubleshoot Hotspot Issues guide.

Examples

```
CREATE TABLE t (
    id INT PRIMARY KEY NONCLUSTERED
) SHARD_ROW_ID_BITS = 4;
```

```
ALTER TABLE t SHARD_ROW_ID_BITS = 4;
```

14.10.1.2 Literal Values

TiDB literal values include character literals, numeric literals, time and date literals, hexadecimal, binary literals, and NULL literals. This document introduces each of these literal values.

This document describes String literals, Numeric literals, NULL values, Hexadecimal literals, Date and time literals, Boolean literals, and Bit-value literals.

14.10.1.2.1 String literals

A string is a sequence of bytes or characters, enclosed within either single quote `'` or double quote `"` characters. For example:

```
'example string'
"example string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'
'a' ' ' 'string'
"a" ' ' "string"
```

If the `ANSI_QUOTES` SQL MODE is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

The string is divided into the following two types:

- Binary string: It consists of a sequence of bytes, whose charset and collation are both `binary`, and uses **byte** as the unit when compared with each other.
- Non-binary string: It consists of a sequence of characters and has various charsets and collations other than `binary`. When compared with each other, non-binary strings use **characters** as the unit. A character might contain multiple bytes, depending on the charset.

A string literal may have an optional `character set introducer` and `COLLATE clause`, to designate it as a string that uses a specific character set and collation.

```
[_charset_name]'string' [COLLATE collation_name]
```

For example:

```
SELECT _latin1'string';
SELECT _binary'string';
SELECT _utf8'string' COLLATE utf8_bin;
```

You can use N'literal' (or n'literal') to create a string in the national character set. The following statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

To represent some special characters in a string, you can use escape characters to escape:

Escape Characters	Meaning
\0	An ASCII NUL (X'00') character
\'	A single quote ' character
\“	A double quote " character
\b	A backspace character
\n	A line break (newline) character
\r	A carriage return character
\t	A tab character
\z	ASCII 26 (Ctrl + Z)
\\	A backslash \ character
\%	A % character
_	A _ character

If you want to represent " in the string surrounded by ', or ' in the string surrounded by ", you do not need to use escape characters.

For more information, see String Literals in MySQL.

14.10.1.2.2 Numeric literals

Numeric literals include integer and DECIMAL literals and floating-point literals.

Integer may include `.` as a decimal separator. Numbers may be preceded by `-` or `+` to indicate a negative or positive value respectively.

Exact-value numeric literals can be represented as `1, .2, 3.4, -5, -6.78, +9.10`.

Numeric literals can also be represented in scientific notation, such as `1.2E3, 1.2E-3, -1.2E3, -1.2E-3`.

For more information, see Numeric Literals in MySQL.

14.10.1.2.3 Date and time literals

Date and time literal values can be represented in several formats, such as quoted strings or as numbers. When TiDB expects a date, it interprets any of `'2017-08-24'`, `'20170824'` and `20170824` as a date.

TiDB supports the following date formats:

- `'YYYY-MM-DD'` or `'YY-MM-DD'`: The `-` delimiter here is not strict. It can be any punctuation. For example, `'2017-08-24'`, `'2017&08&24'`, `'2012@12^31'` are all valid date formats. The only special punctuation is '.', which is is treated as a decimal point to separate the integer and fractional parts. Date and time can be separated by `T` or a white space. For example, `2017-8-24 10:42:00` and `2017-8-24T10:42:00` represents the same date and time.
- `'YYYYMMDDHHMMSS'` or `'YYMMDDHHMMSS'`: For example, `'20170824104520'` and `'170824104520'` are regarded as `'2017-08-24 10:45:20'`. However, if you provide a value out of range, such as `'170824304520'`, it is not treated as a valid date. Note that incorrect formats such as `YYYYMMDD HHMMSS`, `YYYYMMDD HH:MM:DD`, or `YYYY-MM-DD HHMMSS` will fail to insert.
- `YYYYMMDDHHMMSS` or `YYMMDDHHMMSS`: Note that these formats have no single or double quotes, but a number. For example, `20170824104520` is interpreted as `'2017-08-24 10:45:20'`.

DATETIME or TIMESTAMP values can be followed by a fractional part, used to represent microseconds precision (6 digits). The fractional part should always be separated from the rest of the time by a decimal point `.`.

The year value containing only two digits is ambiguous. It is recommended to use the four-digit year format. TiDB interprets the two-digit year value according to the following rules:

- If the year value is in the range of `70-99`, it is converted to `1970-1999`.
- If the year value is in the range of `00-69`, it is converted to `2000-2069`.

For month or day values less than 10, `'2017-8-4'` is the same as `'2017-08-04'`. The same is true for Time. For example, `'2017-08-24 1:2:3'` is the same as `'2017-08-24 01:02:03'`.

When the date or time value is required, TiDB selects the specified format according to the length of the value:

- 6 digits: `YYMMDD`.
- 12 digits: `YYMMDDHHMMSS`.
- 8 digits: `YYYYMMDD`.
- 14 digits: `YYYYMMDDHHMMSS`.

TiDB supports the following formats for time values:

- `'D HH:MM:SS'`, or `'HH:MM:SS'`, `'HH:MM'`, `'D HH:MM'`, `'D HH'`, `'SS'`: `D` means days and the valid value range is `0-34`.
- A number in `HHMMSS` format: For example, `231010` is interpreted as `'23:10:10'`.
- A number in any of `SS`, `MMSS`, and `HHMMSS`formats can be regarded as time.

The decimal point of the Time type is also `.`, with a precision of up to 6 digits after the decimal point.

See MySQL date and time literals for more details.

14.10.1.2.4 Boolean Literals

The constants `TRUE` and `FALSE` are equal to 1 and 0 respectively, which are not case sensitive.

```
SELECT TRUE, true, tRuE, FALSE, FaLsE, false;
```

```
+------+------+------+-------+-------+-------+
| TRUE | true | tRuE | FALSE | FaLsE | false |
+------+------+------+-------+-------+-------+
|    1 |    1 |    1 |     0 |     0 |     0 |
+------+------+------+-------+-------+-------+
1 row in set (0.00 sec)
```

14.10.1.2.5 Hexadecimal literals

Hexadecimal literal values are written using `X'val'` or `0xval` notation, where `val` contains hexadecimal digits. A leading `0x` is case sensitive and cannot be written as `0X`.

Legal hexadecimal literals:

```
X'ac12'
X'12AC'
x'ac12'
x'12AC'
0xac12
0x12AC
```

Illegal hexadecimal literals:

```
X'1z' (z is not a hexadecimal legal digit)
0X12AC (0X must be written as 0x)
```

Hexadecimal literals written using `X'val'` notation must contain an even number of digits. If the length of `val` is an odd number (for example, `X'A'` or `X'11A'`), to avoid the syntax error, pad the value with a leading zero:

```
mysql> select X'aff';
ERROR 1105 (HY000): line 0 column 13 near ""hex literal: invalid
    ↪ hexadecimal format, must even numbers, but 3 (total length 13)
mysql> select X'0aff';
+---------+
| X'0aff' |
+---------+
| 0x0aff |
+---------+
1 row in set (0.00 sec)
```

By default, a hexadecimal literal is a binary string.

To convert a string or a number to a string in hexadecimal format, use the `HEX()` function:

```
mysql> SELECT HEX('TiDB');
+-------------+
| HEX('TiDB') |
+-------------+
| 54694442    |
+-------------+
1 row in set (0.01 sec)

mysql> SELECT X'54694442';
+-------------+
| X'54694442' |
+-------------+
| TiDB        |
+-------------+
1 row in set (0.00 sec)
```

14.10.1.2.6 Bit-value literals

Bit-value literals are written using `b'val'` or `0bval` notation. The `val` is a binary value written using zeros and ones. A leading `0b` is case sensitive and cannot be written as `0B`.

Legal bit-value literals:

```
b'01'
B'01'
0b01
```

Illegal bit-value literals:

```
b'2' (2 is not a binary digit; it must be 0 or 1)
0B01 (0B must be written as 0b)
```

By default, a bit-value literal is a binary string.

Bit values are returned as binary values, which may not display well in the MySQL client. To convert a bit value to printable form, you can use a conversion function such as `BIN()` or `HEX()`.

```
CREATE TABLE t (b BIT(8));
INSERT INTO t SET b = b'00010011';
INSERT INTO t SET b = b'1110';
INSERT INTO t SET b = b'100101';

mysql> SELECT b+0, BIN(b), HEX(b) FROM t;
```

```
+------+--------+--------+
| b+0  | BIN(b) | HEX(b) |
+------+--------+--------+
|   19 | 10011  | 13     |
|   14 | 1110   | E      |
|   37 | 100101 | 25     |
+------+--------+--------+
3 rows in set (0.00 sec)
```

14.10.1.2.7 NULL Values

`NULL` means the data is empty, which is case-insensitive, and is synonymous with `\N` (case-sensitive).

> **Note:**
>
> `NULL` is not the same as `0`, nor the empty string `''`.

14.10.1.3 Schema Object Names

This document introduces schema object names in TiDB SQL statements.

Schema object names are used to name all schema objects in TiDB, including database, table, index, column, and alias. You can quote these objects using identifiers in SQL statements.

You can use backticks to enclose the identifier. For example, `SELECT * FROM t` can also be written as `SELECT * FROM `t``. But if the identifier includes one or more special characters or is a reserved keyword, it must be enclosed in backticks to quote the schema object it represents.

```
SELECT * FROM `table` WHERE `table`.id = 20;
```

If you set `ANSI_QUOTES` in SQL MODE, TiDB will recognize the string enclosed in double quotation marks `"` as an identifier.

```
CREATE TABLE "test" (a varchar(10));
```

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
    ↪ that corresponds to your TiDB version for the right syntax to use
    ↪ line 1 column 19 near ""test" (a varchar(10))"
```

```
SET SESSION sql_mode='ANSI_QUOTES';
```

```
Query OK, 0 rows affected (0.000 sec)
```

```
CREATE TABLE "test" (a varchar(10));
```

```
Query OK, 0 rows affected (0.012 sec)
```

If you want to use the backtick character in the quoted identifier, repeat the backtick twice. For example, to create a table a‘b:

```
CREATE TABLE `a``b` (a int);
```

In a SELECT statement, you can use an identifier or a string to specify an alias:

```
SELECT 1 AS `identifier`, 2 AS 'string';
```

```
+------------+--------+
| identifier | string |
+------------+--------+
|          1 |      2 |
+------------+--------+
1 row in set (0.00 sec)
```

For more information, see MySQL Schema Object Names.

14.10.1.3.1 Identifier qualifiers

Object names can be unqualified or qualified. For example, the following statement creates a table without a qualified name:

```
CREATE TABLE t (i int);
```

If you have not used the USE statement or the connection parameter to configure the database, the ERROR 1046 (3D000): No database selected error is displayed. At this time, you can specify the database qualified name:

```
CREATE TABLE test.t (i int);
```

White spaces can exist around .. table_name.col_name and table_name . col_name are equivalent.

To quote this identifier, use:

```
`table_name`.`col_name`
```

Instead of:

```
`table_name.col_name`
```

For more information, see MySQL Identifier Qualifiers.

14.10.1.4 Keywords

This article introduces the keywords in TiDB, the differences between reserved words and non-reserved words and summarizes all keywords for the query.

Keywords are words that have special meanings in SQL statements, such as `SELECT`, `UPDATE`, and `DELETE`. Some of them can be used as identifiers directly, which are called **non-reserved keywords**. Some of them require special treatment before being used as identifiers, which are called **reserved keywords**.

To use the reserved keywords as identifiers, you must enclose them in backticks `:

```
CREATE TABLE select (a INT);
```

```
ERROR 1105 (HY000): line 0 column 19 near " (a INT)" (total length 27)
```

```
CREATE TABLE `select` (a INT);
```

```
Query OK, 0 rows affected (0.09 sec)
```

The non-reserved keywords do not require backticks, such as `BEGIN` and `END`, which can be successfully used as identifiers in the following statement:

```
CREATE TABLE `select` (BEGIN int, END int);
```

```
Query OK, 0 rows affected (0.09 sec)
```

In the special case, the reserved keywords do not need backticks if they are used with the `.` delimiter:

```
CREATE TABLE test.select (BEGIN int, END int);
```

```
Query OK, 0 rows affected (0.08 sec)
```

The following list shows the keywords in TiDB. Reserved keywords are marked with `(R)`. Reserved keywords for Window Functions are marked with `(R-Window)`:

A

- ACCOUNT
- ACTION
- ADD (R)
- ADMIN (R)
- ADVISE
- AFTER
- AGAINST
- AGO
- ALGORITHM

- ALL (R)
- ALTER (R)
- ALWAYS
- ANALYZE (R)
- AND (R)
- ANY
- AS (R)
- ASC (R)
- ASCII
- AUTO_ID_CACHE
- AUTO_INCREMENT
- AUTO_RANDOM
- AUTO_RANDOM_BASE
- AVG
- AVG_ROW_LENGTH

B

- BACKEND
- BACKUP
- BACKUPS
- BEGIN
- BETWEEN (R)
- BIGINT (R)
- BINARY (R)
- BINDING
- BINDINGS
- BINLOG
- BIT
- BLOB (R)
- BLOCK
- BOOL
- BOOLEAN
- BOTH (R)
- BTREE
- BUCKETS (R)
- BUILTINS (R)
- BY (R)
- BYTE

C

- CACHE
- CANCEL (R)

- CAPTURE
- CASCADE (R)
- CASCADED
- CASE (R)
- CHAIN
- CHANGE (R)
- CHAR (R)
- CHARACTER (R)
- CHARSET
- CHECK (R)
- CHECKPOINT
- CHECKSUM
- CIPHER
- CLEANUP
- CLIENT
- CMSKETCH (R)
- COALESCE
- COLLATE (R)
- COLLATION
- COLUMN (R)
- COLUMNS
- COLUMN_FORMAT
- COMMENT
- COMMIT
- COMMITTED
- COMPACT
- COMPRESSED
- COMPRESSION
- CONCURRENCY
- CONFIG
- CONNECTION
- CONSISTENT
- CONSTRAINT (R)
- CONTEXT
- CONVERT (R)
- CPU
- CREATE (R)
- CROSS (R)
- CSV_BACKSLASH_ESCAPE
- CSV_DELIMITER
- CSV_HEADER
- CSV_NOT_NULL
- CSV_NULL
- CSV_SEPARATOR
- CSV_TRIM_LAST_SEPARATORS

- CUME_DIST (R-Window)
- CURRENT
- CURRENT_DATE (R)
- CURRENT_ROLE (R)
- CURRENT_TIME (R)
- CURRENT_TIMESTAMP (R)
- CURRENT_USER (R)
- CYCLE

D

- DATA
- DATABASE (R)
- DATABASES (R)
- DATE
- DATETIME
- DAY
- DAY_HOUR (R)
- DAY_MICROSECOND (R)
- DAY_MINUTE (R)
- DAY_SECOND (R)
- DDL (R)
- DEALLOCATE
- DECIMAL (R)
- DEFAULT (R)
- DEFINER
- DELAYED (R)
- DELAY_KEY_WRITE
- DELETE (R)
- DENSE_RANK (R-Window)
- DEPTH (R)
- DESC (R)
- DESCRIBE (R)
- DIRECTORY
- DISABLE
- DISCARD
- DISK
- DISTINCT (R)
- DISTINCTROW (R)
- DIV (R)
- DO
- DOUBLE (R)
- DRAINER (R)
- DROP (R)

- DUAL (R)
- DUPLICATE
- DYNAMIC

E

- ELSE (R)
- ENABLE
- ENCLOSED (R)
- ENCRYPTION
- END
- ENFORCED
- ENGINE
- ENGINES
- ENUM
- ERROR
- ERRORS
- ESCAPE
- ESCAPED (R)
- EVENT
- EVENTS
- EVOLVE
- EXCEPT (R)
- EXCHANGE
- EXCLUSIVE
- EXECUTE
- EXISTS (R)
- EXPANSION
- EXPIRE
- EXPLAIN (R)
- EXTENDED

F

- FALSE (R)
- FAULTS
- FIELDS
- FILE
- FIRST
- FIRST_VALUE (R-Window)
- FIXED
- FLOAT (R)
- FLUSH
- FOLLOWING

- FOR (R)
- FORCE (R)
- FOREIGN (R)
- FORMAT
- FROM (R)
- FULL
- FULLTEXT (R)
- FUNCTION

G

- GENERAL
- GENERATED (R)
- GLOBAL
- GRANT (R)
- GRANTS
- GROUP (R)
- GROUPS (R-Window)

H

- HASH
- HAVING (R)
- HIGH_PRIORITY (R)
- HISTORY
- HOSTS
- HOUR
- HOUR_MICROSECOND (R)
- HOUR_MINUTE (R)
- HOUR_SECOND (R)

I

- IDENTIFIED
- IF (R)
- IGNORE (R)
- IMPORT
- IMPORTS
- IN (R)
- INCREMENT
- INCREMENTAL
- INDEX (R)
- INDEXES

- INFILE (R)
- INNER (R)
- INSERT (R)
- INSERT_METHOD
- INSTANCE
- INT (R)
- INT1 (R)
- INT2 (R)
- INT3 (R)
- INT4 (R)
- INT8 (R)
- INTEGER (R)
- INTERVAL (R)
- INTO (R)
- INVISIBLE
- INVOKER
- IO
- IPC
- IS (R)
- ISOLATION
- ISSUER

J

- JOB (R)
- JOBS (R)
- JOIN (R)
- JSON

K

- KEY (R)
- KEYS (R)
- KEY_BLOCK_SIZE
- KILL (R)

L

- LABELS
- LAG (R-Window)
- LANGUAGE
- LAST
- LASTVAL

- LAST_BACKUP
- LAST_VALUE (R-Window)
- LEAD (R-Window)
- LEADING (R)
- LEFT (R)
- LESS
- LEVEL
- LIKE (R)
- LIMIT (R)
- LINEAR (R)
- LINES (R)
- LIST
- LOAD (R)
- LOCAL
- LOCALTIME (R)
- LOCALTIMESTAMP (R)
- LOCATION
- LOCK (R)
- LOGS
- LONG (R)
- LONGBLOB (R)
- LONGTEXT (R)
- LOW_PRIORITY (R)

M

- MASTER
- MATCH (R)
- MAXVALUE (R)
- MAX_CONNECTIONS_PER_HOUR
- MAX_IDXNUM
- MAX_MINUTES
- MAX_QUERIES_PER_HOUR
- MAX_ROWS
- MAX_UPDATES_PER_HOUR
- MAX_USER_CONNECTIONS
- MB
- MEDIUMBLOB (R)
- MEDIUMINT (R)
- MEDIUMTEXT (R)
- MEMORY
- MERGE
- MICROSECOND
- MINUTE

- MINUTE_MICROSECOND (R)
- MINUTE_SECOND (R)
- MINVALUE
- MIN_ROWS
- MOD (R)
- MODE
- MODIFY
- MONTH

N

- NAMES
- NATIONAL
- NATURAL (R)
- NCHAR
- NEVER
- NEXT
- NEXTVAL
- NO
- NOCACHE
- NOCYCLE
- NODEGROUP
- NODE_ID (R)
- NODE_STATE (R)
- NOMAXVALUE
- NOMINVALUE
- NONE
- NOT (R)
- NOWAIT
- NO_WRITE_TO_BINLOG (R)
- NTH_VALUE (R-Window)
- NTILE (R-Window)
- NULL (R)
- NULLS
- NUMERIC (R)
- NVARCHAR

O

- OFFSET
- ON (R)
- ONLINE
- ONLY
- ON_DUPLICATE

- OPEN
- OPTIMISTIC (R)
- OPTIMIZE (R)
- OPTION (R)
- OPTIONALLY (R)
- OR (R)
- ORDER (R)
- OUTER (R)
- OUTFILE (R)
- OVER (R-Window)

P

- PACK_KEYS
- PAGE
- PARSER
- PARTIAL
- PARTITION (R)
- PARTITIONING
- PARTITIONS
- PASSWORD
- PERCENT_RANK (R-Window)
- PER_DB
- PER_TABLE
- PESSIMISTIC (R)
- PLUGINS
- PRECEDING
- PRECISION (R)
- PREPARE
- PRE_SPLIT_REGIONS
- PRIMARY (R)
- PRIVILEGES
- PROCEDURE (R)
- PROCESS
- PROCESSLIST
- PROFILE
- PROFILES
- PUMP (R)

Q

- QUARTER
- QUERIES
- QUERY

- QUICK

R

- RANGE (R)
- RANK (R-Window)
- RATE_LIMIT
- READ (R)
- REAL (R)
- REBUILD
- RECOVER
- REDUNDANT
- REFERENCES (R)
- REGEXP (R)
- REGION (R)
- REGIONS (R)
- RELEASE (R)
- RELOAD
- REMOVE
- RENAME (R)
- REORGANIZE
- REPAIR
- REPEAT (R)
- REPEATABLE
- REPLACE (R)
- REPLICA
- REPLICATION
- REQUIRE (R)
- RESPECT
- RESTORE
- RESTORES
- RESTRICT (R)
- REVERSE
- REVOKE (R)
- RIGHT (R)
- RLIKE (R)
- ROLE
- ROLLBACK
- ROUTINE
- ROW (R)
- ROWS (R-Window)
- ROW_COUNT
- ROW_FORMAT
- ROW_NUMBER (R-Window)

- RTREE

S

- SAMPLES (R)
- SECOND
- SECONDARY_ENGINE
- SECONDARY_LOAD
- SECONDARY_UNLOAD
- SECOND_MICROSECOND (R)
- SECURITY
- SELECT (R)
- SEND_CREDENTIALS_TO_TIKV
- SEPARATOR
- SEQUENCE
- SERIAL
- SERIALIZABLE
- SESSION
- SET (R)
- SETVAL
- SHARD_ROW_ID_BITS
- SHARE
- SHARED
- SHOW (R)
- SHUTDOWN
- SIGNED
- SIMPLE
- SKIP_SCHEMA_FILES
- SLAVE
- SLOW
- SMALLINT (R)
- SNAPSHOT
- SOME
- SOURCE
- SPATIAL (R)
- SPLIT (R)
- SQL (R)
- SQL_BIG_RESULT (R)
- SQL_BUFFER_RESULT
- SQL_CACHE
- SQL_CALC_FOUND_ROWS (R)
- SQL_NO_CACHE
- SQL_SMALL_RESULT (R)
- SQL_TSI_DAY

- SQL_TSI_HOUR
- SQL_TSI_MINUTE
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_SECOND
- SQL_TSI_WEEK
- SQL_TSI_YEAR
- SSL (R)
- START
- STARTING (R)
- STATS (R)
- STATS_AUTO_RECALC
- STATS_BUCKETS (R)
- STATS_HEALTHY (R)
- STATS_HISTOGRAMS (R)
- STATS_META (R)
- STATS_PERSISTENT
- STATS_SAMPLE_PAGES
- STATUS
- STORAGE
- STORED (R)
- STRAIGHT_JOIN (R)
- STRICT_FORMAT
- SUBJECT
- SUBPARTITION
- SUBPARTITIONS
- SUPER
- SWAPS
- SWITCHES
- SYSTEM_TIME

T

- TABLE (R)
- TABLES
- TABLESPACE
- TABLE_CHECKSUM
- TEMPORARY
- TEMPTABLE
- TERMINATED (R)
- TEXT
- THAN
- THEN (R)
- TIDB (R)

- TIFLASH (R)
- TIKV_IMPORTER
- TIME
- TIMESTAMP
- TINYBLOB (R)
- TINYINT (R)
- TINYTEXT (R)
- TO (R)
- TOPN (R)
- TRACE
- TRADITIONAL
- TRAILING (R)
- TRANSACTION
- TRIGGER (R)
- TRIGGERS
- TRUE (R)
- TRUNCATE
- TYPE

U

- UNBOUNDED
- UNCOMMITTED
- UNDEFINED
- UNICODE
- UNION (R)
- UNIQUE (R)
- UNKNOWN
- UNLOCK (R)
- UNSIGNED (R)
- UPDATE (R)
- USAGE (R)
- USE (R)
- USER
- USING (R)
- UTC_DATE (R)
- UTC_TIME (R)
- UTC_TIMESTAMP (R)

V

- VALIDATION
- VALUE
- VALUES (R)

- VARBINARY (R)
- VARCHAR (R)
- VARCHARACTER (R)
- VARIABLES
- VARYING (R)
- VIEW
- VIRTUAL (R)
- VISIBLE

W

- WARNINGS
- WEEK
- WEIGHT_STRING
- WHEN (R)
- WHERE (R)
- WIDTH (R)
- WINDOW (R-Window)
- WITH (R)
- WITHOUT
- WRITE (R)

X

- X509
- XOR (R)

Y

- YEAR
- YEAR_MONTH (R)

Z

- ZEROFILL (R)

14.10.1.5 User-Defined Variables

This document describes the concept of user-defined variables in TiDB and the methods to set and read the user-defined variables.

> **Warning:**
>
> User-defined variables are still an experimental feature. It is **NOT** recommended that you use them in the production environment.

The format of the user-defined variables is `@var_name`. The characters that compose `var_name` can be any characters that can compose an identifier, including the numbers `0-9` ↪ , the letters `a-zA-Z`, the underscore `_`, the dollar sign `$`, and the UTF-8 characters. In addition, it also includes the English period `.`. The user-defined variables are case-insensitive.

The user-defined variables are session-specific, which means a user variable defined by one client connection cannot be seen or used by other client connections.

14.10.1.5.1 Set the user-defined variables

You can use the `SET` statement to set a user-defined variable, and the syntax is `SET` ↪ `@var_name = expr [, @var_name = expr] ...;`. For example:

```
SET @favorite_db = 'TiDB';
```

```
SET @a = 'a', @b = 'b', @c = 'c';
```

For the assignment operator, you can also use `:=`. For example:

```
SET @favorite_db := 'TiDB';
```

The content to the right of the assignment operator can be any valid expression. For example:

```
SET @c = @a + @b;
```

```
set @c = b'1000001' + b'1000001';
```

14.10.1.5.2 Read the user-defined variables

To read a user-defined variable, you can use the `SELECT` statement to query:

```
SELECT @a1, @a2, @a3
```

```
+------+------+------+
| @a1  | @a2  | @a3  |
+------+------+------+
|    1 |    2 |    4 |
+------+------+------+
```

You can also assign values in the SELECT statement:

```
SELECT @a1, @a2, @a3, @a4 := @a1+@a2+@a3;
```

```
+------+------+------+--------------------+
| @a1 | @a2 | @a3 | @a4 := @a1+@a2+@a3 |
+------+------+------+--------------------+
|    1 |    2 |    4 |                  7 |
+------+------+------+--------------------+
```

Before the variable @a4 is modified or the connection is closed, its value is always 7.

If a hexadecimal literal or binary literal is used when setting the user-defined variable, TiDB will treat it as a binary string. If you want to set it to a number, you can manually add the CAST conversion, or use the numeric operator in the expression:

```
SET @v1 = b'1000001';
SET @v2 = b'1000001'+0;
SET @v3 = CAST(b'1000001' AS UNSIGNED);
```

```
SELECT @v1, @v2, @v3;
```

```
+------+------+------+
| @v1 | @v2 | @v3 |
+------+------+------+
| A    | 65   | 65   |
+------+------+------+
```

If you refer to a user-defined variable that has not been initialized, it has a value of NULL and a type of string.

```
SELECT @not_exist;
```

```
+------------+
| @not_exist |
+------------+
| NULL       |
+------------+
```

In addition to using the SELECT statement to read the user-defined variables, another common usage is the PREPARE statement. For example:

```
SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
PREPARE stmt FROM @s;
SET @a = 6;
SET @b = 8;
EXECUTE stmt USING @a, @b;
```

```
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
```

The contents of the user-defined variables are not recognized as identifiers in the SQL statements. For example:

```
SELECT * from t;
```

```
+---+
| a |
+---+
| 1 |
+---+
```

```
SET @col = "`a`";
SELECT @col FROM t;
```

```
+------+
| @col |
+------+
| `a` |
+------+
```

14.10.1.5.3 MySQL compatibility

Except for `SELECT ... INTO <variable>`, the syntax supported in MySQL and TiDB is identical.

For more information, see User-Defined Variables in MySQL.

14.10.1.6 Expression Syntax

An expression is a combination of one or more values, operators, or functions. In TiDB, expressions are mainly used in various clauses of the `SELECT` statement, including Group by clause, Where clause, Having clause, Join condition and window function. In addition, some DDL statements also use expressions, such as the setting of the default values, columns, and partition rules when creating tables.

The expressions can be divided into the following types:

- Identifier. For reference, see Schema object names.

- Predicates, numeric values, strings, date expressions. The Literal values of these types are also expressions.
- Function calls and window functions. For reference, see Functions and operators overview and Window functions
- ParamMarker (?), system variables, user variables and CASE expressions.

The following rules are the expression syntax, which is based on the parser.y rules of TiDB parser. For the navigable version of the following syntax diagram, refer to TiDB SQL Syntax Diagram.

```
Expression ::=
    ( singleAtIdentifier assignmentEq | 'NOT' | Expression ( logOr | 'XOR' |
       ↪  logAnd ) ) Expression
|   'MATCH' '(' ColumnNameList ')' 'AGAINST' '(' BitExpr
    ↪ FulltextSearchModifierOpt ')'
|   PredicateExpr ( IsOrNotOp 'NULL' | CompareOp ( ( singleAtIdentifier
    ↪ assignmentEq )? PredicateExpr | AnyOrAll SubSelect ) )* ( IsOrNotOp (
    ↪  trueKwd | falseKwd | 'UNKNOWN' ) )?

PredicateExpr ::=
    BitExpr ( BetweenOrNotOp BitExpr 'AND' BitExpr )* ( InOrNotOp ( '('
       ↪ ExpressionList ')' | SubSelect ) | LikeOrNotOp SimpleExpr
       ↪ LikeEscapeOpt | RegexpOrNotOp SimpleExpr )?

BitExpr ::=
    BitExpr ( ( '|' | '&' | '<<' | '>>' | '*' | '/' | '%' | 'DIV' | 'MOD' |
       ↪ '^' ) BitExpr | ( '+' | '-' ) ( BitExpr | "INTERVAL" Expression
       ↪ TimeUnit ) )
|   SimpleExpr

SimpleExpr ::=
    SimpleIdent ( ( '->' | '->>' ) stringLit )?
|   FunctionCallKeyword
|   FunctionCallNonKeyword
|   FunctionCallGeneric
|   SimpleExpr ( 'COLLATE' CollationName | pipes SimpleExpr )
|   WindowFuncCall
|   Literal
|   paramMarker
|   Variable
|   SumExpr
|   ( '!' | '~' | '-' | '+' | 'NOT' | 'BINARY' ) SimpleExpr
|   'EXISTS'? SubSelect
```

```
|   ( ( '(' ( ExpressionList ',' )? | 'ROW' '(' ExpressionList ',' )
    ↪ Expression | builtinCast '(' Expression 'AS' CastType | ( 'DEFAULT' |
    ↪  'VALUES' ) '(' SimpleIdent | 'CONVERT' '(' Expression ( ',' CastType
    ↪  | 'USING' CharsetName ) ) ')'
|   'CASE' ExpressionOpt WhenClause+ ElseOpt 'END'
```

14.10.1.7 Comment Syntax

This document describes the comment syntax supported by TiDB.

TiDB supports three comment styles:

- Use # to comment a line:

```
SELECT 1+1;   # comments
```

```
+------+
| 1+1  |
+------+
|    2 |
+------+
1 row in set (0.00 sec)
```

- Use -- to comment a line:

```
SELECT 1+1;   -- comments
```

```
+------+
| 1+1  |
+------+
|    2 |
+------+
1 row in set (0.00 sec)
```

And this style requires at least one whitespace after --:

```
SELECT 1+1--1;
```

```
+--------+
| 1+1--1 |
+--------+
|      3 |
+--------+
1 row in set (0.01 sec)
```

- Use `/* */` to comment a block or multiple lines:

```
SELECT 1 /* this is an in-line comment */ + 1;
```

```
+--------+
| 1  + 1 |
+--------+
|      2 |
+--------+
1 row in set (0.01 sec)
```

```
SELECT 1+
/*
/*> this is a
/*> multiple-line comment
/*> */
    1;
```

```
+--------------------+
| 1+
       1 |
+--------------------+
|                2 |
+--------------------+
1 row in set (0.001 sec)
```

14.10.1.7.1 MySQL-compatible comment syntax

The same as MySQL, TiDB supports a variant of C comment style:

```
/*! Specific code */
```

or

```
/*!50110 Specific code */
```

In this style, TiDB runs the statements in the comment.

For example:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

In TiDB, you can also use another version:

```
SELECT STRAIGHT_JOIN col1 FROM table1,table2 WHERE ...
```

If the server version number is specified in the comment, for example, `/*!50110 KEY_BLOCK_SIZE=1024 */`, in MySQL it means that the contents in this comment are processed only when the MySQL version is or higher than 5.1.10. But in TiDB, the MySQL version number does not work and all contents in the comment are processed.

14.10.1.7.2 TiDB specific comment syntax

TiDB has its own comment syntax (that is, TiDB specific comment syntax), which can be divided into the following two types:

- `/*T! Specific code */`: This syntax can only be parsed and executed by TiDB, and be ignored in other databases.
- `/*T![feature_id] Specific code */`: This syntax is used to ensure compatibility between different versions of TiDB. TiDB can parse the SQL fragment in this comment only if it implements the corresponding feature of `feature_id` in the current version. For example, as the `AUTO_RANDOM` feature is introduced in v3.1.1, this version of TiDB can parse `/*T![auto_rand] auto_random */` into `auto_random`. Because the `AUTO_RANDOM` feature is not implemented in v3.0.0, the SQL statement fragment above is ignored. **Do not leave any space inside the `/*T![` characters**.

14.10.1.7.3 Optimizer comment syntax

Another type of comment is specially treated as an optimizer hint:

```
SELECT /*+ hint */ FROM ...;
```

For details about the optimizer hints that TiDB supports, see Optimizer hints.

> **Note:**
>
> In MySQL client, the TiDB-specific comment syntax is treated as comments and cleared by default. In MySQL client before 5.7.7, hints are also seen as comments and are cleared by default. It is recommended to use the `--comments` option when you start the client. For example, `mysql -h 127.0.0.1 -P 4000 -uroot --comments`.

For more information, see Comment Syntax.

14.10.2 SQL Statements

14.10.2.1 ADD COLUMN

The `ALTER TABLE.. ADD COLUMN` statement adds a column to an existing table. This operation is online in TiDB, which means that neither reads or writes to the table are blocked by adding a column.

14.10.2.1.1 Synopsis

```
AlterTableStmt
        ::= 'ALTER' 'IGNORE'? 'TABLE' TableName AddColumnSpec ( ','
           ↪ AddColumnSpec )*

TableName ::=
    Identifier ('.' Identifier)?

AddColumnSpec
        ::= 'ADD' 'COLUMN' 'IF NOT EXISTS'? ColumnName ColumnType
           ↪ ColumnOption+ ( 'FIRST' | 'AFTER' ColumnName )?

ColumnType
        ::= NumericType
          | StringType
          | DateAndTimeType
          | 'SERIAL'

ColumnOption
        ::= 'NOT'? 'NULL'
          | 'AUTO_INCREMENT'
          | 'PRIMARY'? 'KEY' ( 'CLUSTERED' | 'NONCLUSTERED' )?
          | 'UNIQUE' 'KEY'?
          | 'DEFAULT' ( NowSymOptionFraction | SignedLiteral |
             ↪ NextValueForSequence )
          | 'SERIAL' 'DEFAULT' 'VALUE'
          | 'ON' 'UPDATE' NowSymOptionFraction
          | 'COMMENT' stringLit
          | ( 'CONSTRAINT' Identifier? )? 'CHECK' '(' Expression ')' ( 'NOT
             ↪ '? ( 'ENFORCED' | 'NULL' ) )?
          | 'GENERATED' 'ALWAYS' 'AS' '(' Expression ')' ( 'VIRTUAL' | '
             ↪ STORED' )?
          | 'REFERENCES' TableName ( '(' IndexPartSpecificationList ')' )?
             ↪ Match? OnDeleteUpdateOpt
          | 'COLLATE' CollationName
          | 'COLUMN_FORMAT' ColumnFormat
          | 'STORAGE' StorageMedia
          | 'AUTO_RANDOM' ( '(' LengthNum ')' )?
```

14.10.2.1.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT);
Query OK, 0 rows affected (0.11 sec)
```

```
mysql> INSERT INTO t1 VALUES (NULL);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM t1;
+----+
| id |
+----+
|  1 |
+----+
1 row in set (0.00 sec)

mysql> ALTER TABLE t1 ADD COLUMN c1 INT NOT NULL;
Query OK, 0 rows affected (0.28 sec)

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  0 |
+----+----+
1 row in set (0.00 sec)

mysql> ALTER TABLE t1 ADD c2 INT NOT NULL AFTER c1;
Query OK, 0 rows affected (0.28 sec)

mysql> SELECT * FROM t1;
+----+----+----+
| id | c1 | c2 |
+----+----+----+
|  1 |  0 |  0 |
+----+----+----+
1 row in set (0.00 sec)
```

14.10.2.1.3 MySQL compatibility

- Adding multiple columns at the same time in a statement is currently not supported.
- Adding a new column and setting it to the `PRIMARY KEY` is not supported.
- Adding a new column and setting it to `AUTO_INCREMENT` is not supported.
- There are limitations on adding generated columns, refer to: generated column limitations.

14.10.2.1.4 See also

- ADD INDEX

- CREATE TABLE

14.10.2.2 ADD INDEX

The `ALTER TABLE.. ADD INDEX` statement adds an index to an existing table. This operation is online in TiDB, which means that neither reads or writes to the table are blocked by adding an index.

> **Warning:**
>
> - **DO NOT** upgrade a TiDB cluster when a DDL statement is being executed in the cluster (usually for the time-consuming DDL statements such as `ADD INDEX` and the column type changes).
> - Before the upgrade, it is recommended to use the `ADMIN SHOW DDL` command to check whether the TiDB cluster has an ongoing DDL job. If the cluster has a DDL job, to upgrade the cluster, wait until the DDL execution is finished or use the `ADMIN CANCEL DDL` command to cancel the DDL job before you upgrade the cluster.
> - In addition, during the cluster upgrade, **DO NOT** execute any DDL statement. Otherwise, the issue of undefined behavior might occur.

14.10.2.2.1 Synopsis

```
AlterTableStmt
        ::= 'ALTER' 'IGNORE'? 'TABLE' TableName AddIndexSpec ( ','
           ↪ AddIndexSpec )*

AddIndexSpec
        ::= 'ADD' ( ( 'PRIMARY' 'KEY' | ( 'KEY' | 'INDEX' ) 'IF NOT EXISTS'?
           ↪  | 'UNIQUE' ( 'KEY' | 'INDEX' )? ) ( ( Identifier? 'USING' |
           ↪ Identifier 'TYPE' ) IndexType )? | 'FULLTEXT' ( 'KEY' | '
           ↪ INDEX' )? IndexName ) '(' IndexPartSpecification ( ','
           ↪ IndexPartSpecification )* ')' IndexOption*

IndexPartSpecification
        ::= ( ColumnName ( '(' LengthNum ')' )? | '(' Expression ')' ) ( '
           ↪ ASC' | 'DESC' )

IndexOption
        ::= 'KEY_BLOCK_SIZE' '='? LengthNum
          | IndexType
          | 'WITH' 'PARSER' Identifier
```

```
          | 'COMMENT' stringLit
          | 'VISIBLE'
          | 'INVISIBLE'

IndexType
        ::= 'BTREE'
          | 'HASH'
          | 'RTREE'
```

14.10.2.2.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
    ↪ ----------------------+---------+----------+--------------+----------------
    ↪
| id                    | estRows | task     | access object | operator info
    ↪            |
+--
    ↪ ----------------------+---------+----------+--------------+----------------
    ↪
| TableReader_7         | 10.00   | root     |               | data:Selection_6
    ↪           |
| └─Selection_6         | 10.00   | cop[tikv] |              | eq(test.t1.c1,
    ↪ 3)          |
|   └─TableFullScan_5   | 10000.00 | cop[tikv] | table:t1 | keep order:false
    ↪ , stats:pseudo |
+--
    ↪ ----------------------+---------+----------+--------------+----------------
    ↪
3 rows in set (0.00 sec)

mysql> ALTER TABLE t1 ADD INDEX (c1);
Query OK, 0 rows affected (0.30 sec)

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
```

```
    ↪ ---------------------+---------+----------+-----------------------+----------
    ↪
| id                     | estRows | task     | access object         | operator
    ↪ info                              |
+--
    ↪ ---------------------+---------+----------+-----------------------+----------
    ↪
| IndexReader_6          | 0.01    | root     |                       | index:
    ↪ IndexRangeScan_5                  |
| └─IndexRangeScan_5     | 0.01    | cop[tikv] | table:t1, index:c1(c1) | range
    ↪ :[3,3], keep order:false, stats:pseudo |
+--
    ↪ ---------------------+---------+----------+-----------------------+----------
    ↪
2 rows in set (0.00 sec)
```

14.10.2.2.3 MySQL compatibility

- `FULLTEXT`, `HASH` and `SPATIAL` indexes are not supported.
- Descending indexes are not supported (similar to MySQL 5.7).
- Adding multiple indexes at the same time is currently not supported.
- Adding the primary key of the `CLUSTERED` type to a table is not supported. For more details about the primary key of the `CLUSTERED` type, refer to clustered index.

14.10.2.2.4 See also

- Index Selection
- Wrong Index Solution
- CREATE INDEX
- DROP INDEX
- RENAME INDEX
- ALTER INDEX
- ADD COLUMN
- CREATE TABLE
- EXPLAIN

14.10.2.3 ADMIN

This statement is a TiDB extension syntax, used to view the status of TiDB and check the data of tables in TiDB. This document introduces the following `ADMIN` related statements:

- `ADMIN RELOAD`
- `ADMIN PLUGINS`

- ADMIN ... BINDINGS
- ADMIN REPAIR
- ADMIN SHOW SLOW

14.10.2.3.1 DDL related statement

Statement	Description
ADMIN CANCEL DDL JOBS	Cancels a currently running DDL jobs.
ADMIN CHECKSUM TABLE	Calculates the CRC64 of all rows + indexes of a table.
[ADMIN CHECK [TABLE|INDEX]](#admin-check-[table index])	
[ADMIN SHOW DDL [JOBS|QUERIES]](#admin-show-ddl-[jobs job-queries])	
ADMIN SHOW TELEMETRY	Shows information that will be reported back to PingCAP as part of the telemetry feature.

14.10.2.3.2 `ADMIN RELOAD` statement

```
ADMIN RELOAD expr_pushdown_blacklist;
```

The above statement is used to reload the blocklist pushed down by the expression.

```
ADMIN RELOAD opt_rule_blacklist;
```

The above statement is used to reload the blocklist of logic optimization rules.

14.10.2.3.3 `ADMIN PLUGINS` related statement

```
ADMIN PLUGINS ENABLE plugin_name [, plugin_name] ...;
```

The above statement is used to enable the `plugin_name` plugin.

```
ADMIN PLUGINS DISABLE plugin_name [, plugin_name] ...;
```

The above statement is used to disable the `plugin_name` plugin.

14.10.2.3.4 `ADMIN BINDINGS` related statement

```
ADMIN FLUSH BINDINGS;
```

The above statement is used to persist SQL Plan binding information.

```
ADMIN CAPTURE BINDINGS;
```

The above statement can generate the binding of SQL Plan from the `SELECT` statement that occurs more than once.

```
ADMIN EVOLVE BINDINGS;
```

After the automatic binding feature is enabled, the evolution of SQL Plan binding information is triggered every `bind-info-leave` (the default value is `3s`). The above statement is used to proactively trigger this evolution.

```
ADMIN RELOAD BINDINGS;
```

The above statement is used to reload SQL Plan binding information.

14.10.2.3.5 `ADMIN REPAIR` statement

To overwrite the metadata of the stored table in an untrusted way in extreme cases, use `ADMIN REPAIR TABLE`:

```
ADMIN REPAIR TABLE tbl_name CREATE TABLE STATEMENT;
```

Here "untrusted" means that you need to manually ensure that the metadata of the original table can be covered by the `CREATE TABLE STATEMENT` operation. To use this `REPAIR` statement, enable the `repair-mode` configuration item, and make sure that the tables to be repaired are listed in the `repair-table-list`.

14.10.2.3.6 `ADMIN SHOW SLOW` statement

```
ADMIN SHOW SLOW RECENT N;
```

```
ADMIN SHOW SLOW TOP [INTERNAL | ALL] N;
```

For details, refer to admin show slow statement

14.10.2.3.7 Synopsis

```
AdminStmt ::=
    'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
       ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
       ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
       ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '
       ↪ RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
       ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
       ↪  'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
```

```
      ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
      ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
      ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
      ↪ ' ) 'BINDINGS' )
```

14.10.2.3.8 Examples

Run the following command to view the last 10 completed DDL jobs in the currently running DDL job queue. When `NUM` is not specified, only the last 10 completed DDL jobs is presented by default.

```
ADMIN SHOW DDL JOBS;
```

```
+--------+---------+-----------+--------------------+---------------+-----------+----
   ↪
| JOB_ID | DB_NAME | TABLE_NAME | JOB_TYPE      | SCHEMA_STATE | SCHEMA_ID |
   ↪  TABLE_ID | ROW_COUNT | START_TIME                | END_TIME
   ↪                       | STATE       |
+--------+---------+-----------+--------------------+---------------+-----------+----
   ↪
| 45     | test    | t1        | add index    | write reorganization | 32     |
   ↪ 37       | 0         | 2019-01-10 12:38:36.501 +0800 CST |
   ↪                            | running     |
| 44     | test    | t1        | add index    | none                 | 32       |
   ↪ 37       | 0         | 2019-01-10 12:36:55.18 +0800 CST | 2019-01-10
   ↪ 12:36:55.852 +0800 CST | rollback done |
| 43     | test    | t1        | add index    | public               | 32       |
   ↪ 37       | 6         | 2019-01-10 12:35:13.66 +0800 CST | 2019-01-10
   ↪ 12:35:14.925 +0800 CST | synced |
| 42     | test    | t1        | drop index   | none                 | 32       |
   ↪ 37       | 0         | 2019-01-10 12:34:35.204 +0800 CST | 2019-01-10
   ↪ 12:34:36.958 +0800 CST | synced |
| 41     | test    | t1        | add index    | public               | 32       |
   ↪ 37       | 0         | 2019-01-10 12:33:22.62 +0800 CST | 2019-01-10
   ↪ 12:33:24.625 +0800 CST | synced |
| 40     | test    | t1        | drop column | none                 | 32       |
   ↪ 37       | 0         | 2019-01-10 12:33:08.212 +0800 CST | 2019-01-10
   ↪ 12:33:09.78 +0800 CST | synced |
| 39     | test    | t1        | add column   | public               | 32       |
   ↪ 37       | 0         | 2019-01-10 12:32:55.42 +0800 CST | 2019-01-10
   ↪ 12:32:56.24 +0800 CST | synced |
| 38     | test    | t1        | create table | public               | 32       |
   ↪ 37       | 0         | 2019-01-10 12:32:41.956 +0800 CST | 2019-01-10
   ↪ 12:32:43.956 +0800 CST | synced |
```

```
| 36    | test   |           | drop table  | none                | 32       |
   ↪ 34      | 0        | 2019-01-10 11:29:59.982 +0800 CST | 2019-01-10
   ↪ 11:30:00.45 +0800 CST | synced |
| 35    | test   |           | create table | public             | 32       |
   ↪ 34      | 0        | 2019-01-10 11:29:40.741 +0800 CST | 2019-01-10
   ↪ 11:29:41.682 +0800 CST | synced |
| 33    | test   |           | create schema | public            | 32       | 0
   ↪        | 0        | 2019-01-10 11:29:22.813 +0800 CST | 2019-01-10
   ↪ 11:29:23.954 +0800 CST | synced |
+--------+--------+-----------+--------------------+---------------+----------+----
   ↪
```

Run the following command to view the last 5 completed DDL jobs in the currently running DDL job queue:

```
ADMIN SHOW DDL JOBS 5;
```

```
+--------+--------+-----------+--------------------+---------------+----------+----
   ↪
| JOB_ID | DB_NAME | TABLE_NAME | JOB_TYPE      | SCHEMA_STATE | SCHEMA_ID |
   ↪  TABLE_ID | ROW_COUNT | START_TIME                | END_TIME
   ↪                       | STATE       |
+--------+--------+-----------+--------------------+---------------+----------+----
   ↪
| 45    | test   | t1        | add index    | write reorganization | 32    |
   ↪ 37      | 0        | 2019-01-10 12:38:36.501 +0800 CST |
   ↪                            | running     |
| 44    | test   | t1        | add index    | none                | 32       |
   ↪ 37      | 0        | 2019-01-10 12:36:55.18 +0800 CST | 2019-01-10
   ↪ 12:36:55.852 +0800 CST | rollback done |
| 43    | test   | t1        | add index    | public              | 32       |
   ↪ 37      | 6        | 2019-01-10 12:35:13.66 +0800 CST | 2019-01-10
   ↪ 12:35:14.925 +0800 CST | synced |
| 42    | test   | t1        | drop index   | none                | 32       |
   ↪ 37      | 0        | 2019-01-10 12:34:35.204 +0800 CST | 2019-01-10
   ↪ 12:34:36.958 +0800 CST | synced |
| 41    | test   | t1        | add index    | public              | 32       |
   ↪ 37      | 0        | 2019-01-10 12:33:22.62 +0800 CST | 2019-01-10
   ↪ 12:33:24.625 +0800 CST | synced |
| 40    | test   | t1        | drop column | none                | 32       |
   ↪ 37      | 0        | 2019-01-10 12:33:08.212 +0800 CST | 2019-01-10
   ↪ 12:33:09.78 +0800 CST | synced |
+--------+--------+-----------+--------------------+---------------+----------+----
   ↪
```

Run the following command to view the uncompleted DDL jobs in the test database. The results include the DDL jobs that are running and the last 5 DDL jobs that are completed but failed.

```
ADMIN SHOW DDL JOBS 5 WHERE state != 'synced' AND db_name = 'test';
```

```
+--------+---------+-----------+--------------------+---------------+-----------+----
   ↪
| JOB_ID | DB_NAME | TABLE_NAME | JOB_TYPE      | SCHEMA_STATE | SCHEMA_ID |
   ↪  TABLE_ID | ROW_COUNT | START_TIME                | END_TIME
   ↪                        | STATE       |
+--------+---------+-----------+--------------------+---------------+-----------+----
   ↪
| 45     | test    | t1        | add index    | write reorganization | 32     |
   ↪ 37       | 0         | 2019-01-10 12:38:36.501 +0800 CST |
   ↪                            | running     |
| 44     | test    | t1        | add index    | none                 | 32       |
   ↪ 37       | 0         | 2019-01-10 12:36:55.18 +0800 CST | 2019-01-10
   ↪ 12:36:55.852 +0800 CST | rollback done |
+--------+---------+-----------+--------------------+---------------+-----------+----
   ↪
```

- `JOB_ID`: each DDL operation corresponds to one DDL job. `JOB_ID` is globally unique.
- `DB_NAME`: the name of the database on which the DDL operations are performed.
- `TABLE_NAME`: the name of the table on which the DDL operations are performed.
- `JOB_TYPE`: the type of the DDL operations.
- `SCHEMA_STATE`: the current state of the schema. If the `JOB_TYPE` is `add index`, it is the state of the index; if the `JOB_TYPE` is `add column`, it is the state of the column; if the `JOB_TYPE` is `create table`, it is the state of the table. The common states include:
    - `none`: it indicates not existing. When the `drop` or `create` operation fails and rolls back, it usually becomes the `none` state.
    - `delete only`, `write only`, `delete reorganization`, `write reorganization`: these four states are intermediate states. These states are not visible in common operations, because the conversion from the intermediate states is so quick. You can see the `write reorganization` state only in `add index` operations, which means that the index data is being added.
    - `public`: it indicates existing and usable. When operations like `create table` and `add index/column` are finished, it usually becomes the `public` state, which means that the created table/column/index can be normally read and written now.
- `SCHEMA_ID`: the ID of the database on which the DDL operations are performed.
- `TABLE_ID`: the ID of the table on which the DDL operations are performed.
- `ROW_COUNT`: the number of the data rows that have been added when running the `add index` operation.

- `START_TIME`: the start time of the DDL operations.
- `END_TIME`: the end time of the DDL operations.
- `STATE`: the state of the DDL operations. The common states include:
    - `none`: it indicates that the operation task has been put in the DDL job queue but has not been performed yet, because it is waiting for the previous tasks to complete. Another reason might be that it becomes the `none` state after running the drop operation, but it will soon be updated to the `synced` state, which means that all TiDB instances have been synced to this state.
    - `running`: it indicates that the operation is being performed.
    - `synced`: it indicates that the operation has been performed successfully and all TiDB instances have been synced to this state.
    - `rollback done`: it indicates that the operation has failed and has finished rolling back.
    - `rollingback`: it indicates that the operation has failed and is rolling back.
    - `cancelling`: it indicates that the operation is being cancelled. This state only occurs when you cancel DDL jobs using the `ADMIN CANCEL DDL JOBS` command.

14.10.2.3.9 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.4 ADMIN CANCEL DDL

The `ADMIN CANCEL DDL` statement allows you to cancel a running DDL job. The `job_id` can be found by running `ADMIN SHOW DDL JOBS`.

14.10.2.4.1 Synopsis

```
AdminStmt ::=
    'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
       ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
       ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
       ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '
       ↪ RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
       ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
       ↪  'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
       ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
       ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
       ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
       ↪ ' ) 'BINDINGS' )

NumList ::=
    Int64Num ( ',' Int64Num )*
```

14.10.2.4.2 Examples

To cancel the currently running DDL jobs and return whether the corresponding jobs are successfully cancelled, use `ADMIN CANCEL DDL JOBS`:

```
ADMIN CANCEL DDL JOBS job_id [, job_id] ...;
```

If the operation fails to cancel the jobs, specific reasons are displayed.

Note:

- Only this operation can cancel DDL jobs. All other operations and environment changes (such as machine restart and cluster restart) cannot cancel these jobs.
- This operation can cancel multiple DDL jobs at the same time. You can get the ID of DDL jobs using the `ADMIN SHOW DDL JOBS` statement.
- If the jobs you want to cancel are finished, the cancellation operation fails.

14.10.2.4.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.4.4 See also

- `ADMIN SHOW DDL [JOBS|JOB QUERIES]`

14.10.2.5 ADMIN CHECKSUM TABLE

The `ADMIN CHECKSUM TABLE` statement calculates a CRC64 checksum for the data and indexes of a table. This statement is used by programs such as TiDB Lightning to ensure that import operations have completed successfully.

14.10.2.5.1 Synopsis

```
AdminStmt ::=
    'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
       ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
       ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
       ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '
       ↪ RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
       ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
       ↪  'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
```

```
        ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
        ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
        ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
        ↪ ' ) 'BINDINGS' )

TableNameList ::=
    TableName ( ',' TableName )*
```

14.10.2.5.2 Examples

Create table `t1`:

```
CREATE TABLE t1(id INT PRIMARY KEY);
```

Insert some data into `t1`:

```
INSERT INTO t1 VALUES (1),(2),(3);
```

Calculate the checksum for `t1`:

```
ADMIN CHECKSUM TABLE t1;
```

The output is as follows:

```
+---------+------------+---------------------+-----------+-------------+
| Db_name | Table_name | Checksum_crc64_xor | Total_kvs | Total_bytes |
+---------+------------+---------------------+-----------+-------------+
| test    | t1         | 10909174369497628533 |     3 |         75 |
+---------+------------+---------------------+-----------+-------------+
1 row in set (0.00 sec)
```

14.10.2.5.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.6 ADMIN CHECK [TABLE|INDEX]

The `ADMIN CHECK [TABLE|INDEX]` statement checks for data consistency of tables and indexes.

It does not support the following:

- Checking the PRIMARY KEY index if a clustered primary key is used.

If `ADMIN CHECK [TABLE|INDEX]` finds any issues, you can resolve them by dropping and recreating the index. If the issue is not resolved, you can report an issue on GitHub.

14.10.2.6.1 Principles

The `ADMIN CHECK TABLE` statement takes the following steps to check the table:

1. For each index, it checks if the number of records in the index is the same as that in the table.
2. For each index, it loops over the values in each row and compares the values with that in the table.

If you use the `ADMIN CHECK INDEX` statement, it only checks the specified index.

14.10.2.6.2 Synopsis

```
AdminStmt ::=
    'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
       ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
       ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
       ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '
       ↪ RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
       ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
       ↪  'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
       ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
       ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
       ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
       ↪ ' ) 'BINDINGS' )

TableNameList ::=
    TableName ( ',' TableName )*
```

14.10.2.6.3 Examples

To check the consistency of all the data and corresponding indexes in the `tbl_name` table, use `ADMIN CHECK TABLE`:

```
ADMIN CHECK TABLE tbl_name [, tbl_name] ...;
```

If the consistency check is passed, an empty result is returned. Otherwise, an error message is returned indicating that the data is inconsistent.

```
ADMIN CHECK INDEX tbl_name idx_name;
```

The above statement is used to check the consistency of the column data and index data corresponding to the `idx_name` index in the `tbl_name` table. If the consistency check is passed, an empty result is returned; otherwise, an error message is returned indicating that the data is inconsistent.

```
ADMIN CHECK INDEX tbl_name idx_name (lower_val, upper_val) [, (lower_val,
    ↪ upper_val)] ...;
```

The above statement is used to check the consistency of the column data and index data corresponding to the `idx_name` index in the `tbl_name` table, with the data range (to be checked) specified. If the consistency check is passed, an empty result is returned. Otherwise, an error message is returned indicating that the data is inconsistent.

14.10.2.6.4 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.6.5 See also

- ADMIN REPAIR

14.10.2.7 ADMIN CLEANUP INDEX

The `ADMIN CLEANUP INDEX` statement is used to delete redundant indexes from a table when the table has inconsistent data and index. Note that this syntax does not support foreign key constraints yet.

14.10.2.7.1 Synopsis

```
AdminCleanupStmt ::=
    'ADMIN' 'CLEANUP' ( 'INDEX' TableName IndexName | 'TABLE' 'LOCK'
        ↪ TableNameList )

TableNameList ::=
    TableName ( ',' TableName )*
```

14.10.2.7.2 Examples

Assume that the `tbl` table in a database has inconsistent data and index due to some reasons (for example, some row data is lost in the cluster in a disaster recovery scenario):

```
SELECT * FROM tbl;
ERROR 1105 (HY000): inconsistent index idx handle count 3 isn't equal to
    ↪ value count 2

ADMIN CHECK INDEX tbl idx ;
ERROR 1105 (HY000): handle &kv.CommonHandle{encoded:[]uint8{0x1, 0x0, 0x0, 0
    ↪ x0, 0x0, 0x0, 0x0, 0x0, 0xf8}, colEndOffsets:[]uint16{0xa}},
    ↪ index:types.Datum{k:0x5, decimal:0x0, length:0x0, i:0, collation:"
    ↪ utf8mb4_bin", b:[]uint8{0x0}, x:interface {}(nil)} != record:<nil>
```

It can be seen from the error message of the `SELECT` query that, the `tbl` table contains two rows of data and three rows of index data, which means inconsistent row and index data. At the same time, at least one index is in dangling state. In this case, you can use the `ADMIN CLEANUP INDEX` statement to delete the dangling index:

```
ADMIN CLEANUP INDEX tbl idx;
```

The execution result is as follows:

```
ADMIN CLEANUP INDEX tbl idx;
+---------------+
| REMOVED_COUNT |
+---------------+
|             1 |
+---------------+
```

You can execute the `ADMIN CHECK INDEX` statement again to check the consistency of data and index, and verify whether the data has been restored to a normal state:

```
ADMIN CHECK INDEX tbl idx;
Query OK, 0 rows affected (0.01 sec)
```

Note:

When the data and index are inconsistent due to the loss of replicas:

- There might be a loss of both row data and index data. To restore the consistency, use the `ADMIN CLEANUP INDEX` and `ADMIN RECOVER INDEX` statements together.
- The `ADMIN CLEANUP INDEX` statement is always executed in a single thread. When the table data is large, it is recommended to recover the index data by rebuilding the index.
- When you execute the `ADMIN CLEANUP INDEX` statement, the corresponding table or index is not locked and TiDB allows other sessions to modify the table records at the same time. However, in this case, `ADMIN CLEANUP INDEX` might not be able to handle all table records correctly. Therefore, when you execute `ADMIN CLEANUP INDEX`, avoid modifying the table data at the same time.
- If you use the enterprise edition of TiDB, you can submit a request to contact the support engineer for help.

The `ADMIN CLEANUP INDEX` statement is not atomic: if the statement is interrupted during execution, it is recommended to execute it again until it succeeds.

14.10.2.7.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.7.4 See also

- ADMIN CHECK TABLE/INDEX
- ADMIN RECOVER INDEX

14.10.2.8 ADMIN RECOVER INDEX

When the row data and index data are inconsistent, you can use the `ADMIN RECOVER INDEX` statement to recover the consistency based on the redundant indexes. Note that this syntax does not support foreign key constraints yet.

14.10.2.8.1 Synopsis

```
AdminCleanupStmt ::=
    'ADMIN' 'RECOVER' 'INDEX' TableName IndexName
```

14.10.2.8.2 Examples

Assume that the `tbl` table in a database has inconsistent row data and index due to some reasons (for example, some row data is lost in the cluster in a disaster recovery scenario):

```
SELECT * FROM tbl;
ERROR 1105 (HY000): inconsistent index idx handle count 2 isn't equal to
    ↪ value count 3

ADMIN CHECK INDEX tbl idx ;
ERROR 1105 (HY000): handle &kv.CommonHandle{encoded:[]uint8{0x1, 0x0, 0x0, 0
    ↪ x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xf8}, colEndOffsets:[]uint16{0xa}},
    ↪ index:types.Datum{k:0x5, decimal:0x0, length:0x0, i:0, collation:"
    ↪ utf8mb4_bin", b:[]uint8{0x0}, x:interface {}(nil)} != record:<nil>
```

It can be seen from the error message of the `SELECT` query that, the `tbl` table contains three rows of data and two rows of index data, which means inconsistent row data and index data. At the same time, at least one row of data has no corresponding index. In this case, you can use the `ADMIN RECOVER INDEX` statement to supplement the missing index:

```
ADMIN RECOVER INDEX tbl idx;
```

The execution result is as follows:

```
ADMIN RECOVER INDEX tbl idx;
+-------------+------------+
| ADDED_COUNT | SCAN_COUNT |
+-------------+------------+
|           1 |          3 |
+-------------+------------+
1 row in set (0.00 sec)
```

You can execute the `ADMIN CHECK INDEX` statement again to check the consistency of data and index, and verify whether the data has been restored to normal state:

```
ADMIN CHECK INDEX tbl idx;
Query OK, 0 rows affected (0.01 sec)
```

Note:

When the data and index are inconsistent due to the loss of replicas:

- There might be a loss of both row data and index data. To address the issue, use the `ADMIN CLEANUP INDEX` and `ADMIN RECOVER INDEX` statements together to recover the consistency of row data and index data.
- The `ADMIN RECOVER INDEX` statement is always executed in a single thread. When the table data is large, it is recommended to recover the index data by rebuilding the index.
- When you execute the `ADMIN RECOVER INDEX` statement, the corresponding table or index is not locked and TiDB allows other sessions to modify the table records at the same time. However, in this case, `ADMIN RECOVER INDEX` might not be able to handle all table records correctly. Therefore, when you execute `ADMIN RECOVER INDEX`, avoid modifying the table data at the same time.
- If you use the enterprise edition of TiDB, you can submit a request to contact the support engineer for help.

The `ADMIN RECOVER INDEX` statement is not atomic: if the statement is interrupted during execution, it is recommended to execute it again until it succeeds.

14.10.2.8.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.8.4 See also

- ADMIN CHECK TABLE/INDEX
- ADMIN CLEANUP INDEX

14.10.2.9 ADMIN SHOW DDL [JOBS|JOB QUERIES]

The `ADMIN SHOW DDL [JOBS|JOB QUERIES]` statement shows information about running and recently completed DDL jobs.

14.10.2.9.1 Synopsis

```
AdminStmt ::=
    'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
       ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
       ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
       ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '
       ↪ RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
       ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
       ↪  'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
       ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
       ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
       ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
       ↪ ' ) 'BINDINGS' )

NumList ::=
    Int64Num ( ',' Int64Num )*

WhereClauseOptional ::=
    WhereClause?
```

14.10.2.9.2 Examples

`ADMIN SHOW DDL`

To view the currently running DDL jobs, use `ADMIN SHOW DDL`:

```
ADMIN SHOW DDL;
```

```
mysql> ADMIN SHOW DDL;
+--
   ↪ ----------+----------------------------------------+--------------+-------------
   ↪
| SCHEMA_VER | OWNER_ID                               | OWNER_ADDRESS | RUNNING_JOBS
   ↪  | SELF_ID                              | QUERY |
```

```
+--
    ↪ ----------+--------------------------------------+--------------+-----------
    ↪
|        26 | 2d1982af-fa63-43ad-a3d5-73710683cc63 | 0.0.0.0:4000 |      | 2
    ↪ d1982af-fa63-43ad-a3d5-73710683cc63 | |
+--
    ↪ ----------+--------------------------------------+--------------+-----------
    ↪
1 row in set (0.00 sec)
```

ADMIN SHOW DDL JOBS

To view all the results in the current DDL job queue (including tasks that are running and waiting to be run) and the last ten results in the completed DDL job queue, use ADMIN ↪ SHOW DDL JOBS:

```
ADMIN SHOW DDL JOBS;
```

```
mysql> ADMIN SHOW DDL JOBS;
+--
    ↪ ------+---------+-------------------+-------------+--------------------+----
    ↪
| JOB_ID | DB_NAME | TABLE_NAME     | JOB_TYPE    | SCHEMA_STATE       |
    ↪ SCHEMA_ID | TABLE_ID | ROW_COUNT | CREATE_TIME  | START_TIME         |
    ↪ END_TIME          | STATE  |
+--
    ↪ ------+---------+-------------------+-------------+--------------------+----
    ↪
|    59 | test   | t1               | add index   | write reorganization |
    ↪     1 |      55 |    88576 | 2020-08-17 07:51:58 | 2020-08-17 07:51:58
    ↪   | NULL        | running |
|    60 | test   | t2               | add index   | none                 |
    ↪         1 |      57 |        0 | 2020-08-17 07:51:59 | 2020-08-17
    ↪ 07:51:59 | NULL        | none   |
|    58 | test   | t2               | create table | public              |
    ↪         1 |      57 |        0 | 2020-08-17 07:41:28 | 2020-08-17
    ↪ 07:41:28 | 2020-08-17 07:41:28 | synced |
|    56 | test   | t1               | create table | public              |
    ↪         1 |      55 |        0 | 2020-08-17 07:41:02 | 2020-08-17
    ↪ 07:41:02 | 2020-08-17 07:41:02 | synced |
|    54 | test   | t1               | drop table  | none                 |
    ↪         1 |      50 |        0 | 2020-08-17 07:41:02 | 2020-08-17
    ↪ 07:41:02 | 2020-08-17 07:41:02 | synced |
|    53 | test   | t1               | drop index  | none                 |
    ↪         1 |      50 |        0 | 2020-08-17 07:35:44 | 2020-08-17
    ↪ 07:35:44 | 2020-08-17 07:35:44 | synced |
```

```
|    52 | test   | t1                | add index  | public             |
    ↪        1 |      50 |   451010 | 2020-08-17 07:34:43 | 2020-08-17
    ↪ 07:34:43 | 2020-08-17 07:35:16 | synced |
|    51 | test   | t1                | create table | public           |
    ↪        1 |      50 |        0 | 2020-08-17 07:34:02 | 2020-08-17
    ↪ 07:34:02 | 2020-08-17 07:34:02 | synced |
|    49 | test   | t1                | drop table | none               |
    ↪        1 |      47 |        0 | 2020-08-17 07:34:02 | 2020-08-17
    ↪ 07:34:02 | 2020-08-17 07:34:02 | synced |
|    48 | test   | t1                | create table | public           |
    ↪        1 |      47 |        0 | 2020-08-17 07:33:37 | 2020-08-17
    ↪ 07:33:37 | 2020-08-17 07:33:37 | synced |
|    46 | mysql  | stats_extended    | create table | public           |
    ↪        3 |      45 |        0 | 2020-08-17 06:42:38 | 2020-08-17
    ↪ 06:42:38 | 2020-08-17 06:42:38 | synced |
|    44 | mysql  | opt_rule_blacklist | create table | public         |
    ↪        3 |      43 |        0 | 2020-08-17 06:42:38 | 2020-08-17
    ↪ 06:42:38 | 2020-08-17 06:42:38 | synced |
+--
    ↪ ------+---------+--------------------+--------------+--------------------+----
    ↪
12 rows in set (0.00 sec)
```

From the output above:

- Job 59 is currently in progress (`STATE` of `running`). The schema state is currently in `write reorganization`, but will switch to `public` once the task is completed to note that the change can be observed publicly by user sessions. The `end_time` column is also `NULL` indicating that the completion time for the job is currently not known.
- Job 60 is an `add index` job, which is currently queued waiting for job 59 to complete. When job 59 completes, the `STATE` of job 60 will switch to `running`.
- For destructive changes such as dropping an index or dropping a table, the `SCHEMA_STATE` will change to `none` when the job is complete. For additive changes, the `SCHEMA_STATE` will change to `public`.

To limit the number of rows shown, specify a number and a where condition:

```
ADMIN SHOW DDL JOBS [NUM] [WHERE where_condition];
```

- `NUM`: to view the last `NUM` results in the completed DDL job queue. If not specified, `NUM` is by default 10.
- `WHERE`: to add filter conditions.

`ADMIN SHOW DDL JOB QUERIES`

To view the original SQL statements of the DDL job corresponding to `job_id`, use `ADMIN SHOW DDL JOB QUERIES`:

```
ADMIN SHOW DDL JOBS;
ADMIN SHOW DDL JOB QUERIES 51;
```

```
mysql> ADMIN SHOW DDL JOB QUERIES 51;
+-------------------------------------------------------------+
| QUERY                                                       |
+-------------------------------------------------------------+
| CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY auto_increment) |
+-------------------------------------------------------------+
1 row in set (0.02 sec)
```

You can only search the running DDL job corresponding to `job_id` within the last ten results in the DDL history job queue.

14.10.2.9.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.9.4 See also

- ADMIN CANCEL DDL

14.10.2.10 ADMIN SHOW TELEMETRY

The `ADMIN SHOW TELEMETRY` statement shows the information that will be reported back to PingCAP as part of the telemetry feature.

14.10.2.10.1 Synopsis

```
AdminStmt ::=
    'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
       ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
       ↪ AdminShowSlow | 'TELEMETRY' ) | 'CHECK' ( 'TABLE' TableNameList |
       ↪ 'INDEX' TableName Identifier ( HandleRange ( ',' HandleRange )* )?
       ↪  ) | 'RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
       ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
       ↪  'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
       ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
       ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
       ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
       ↪ ' ) 'BINDINGS' )
```

14.10.2.10.2 Examples

```
ADMIN SHOW TELEMETRY\G
```

```
*************************** 1. row ***************************
 TRACKING_ID: a1ba1d97-b940-4d5b-a9d5-ddb0f2ac29e7
 LAST_STATUS: {
  "check_at": "2021-08-11T08:23:38+02:00",
  "is_error": false,
  "error_msg": "",
  "is_request_sent": true
}
DATA_PREVIEW: {
  "hardware": [
    {
      "instanceType": "tidb",
      "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
      "listenPort": "4000",
      "cpu": {
        "cache": "8192",
        "cpuFrequency": "2301.00MHz",
        "cpuLogicalCores": "8",
        "cpuPhysicalCores": "4"
      },
      "memory": {
        "capacity": "16410021888"
      },
      "disk": {
        "ebbca862689fa9fef7c55c3112e375c4ce575fe4": {
          "deviceName": "ebbca862689fa9fef7c55c3112e375c4ce575fe4",
          "free": "624438726656",
          "freePercent": "0.61",
          "fstype": "btrfs",
          "opts": "bind,rw,relatime",
          "path": "fb365c1216b59e1cfc86950425867007a60f4435",
          "total": "1022488477696",
          "used": "397115568128",
          "usedPercent": "0.39"
        },
        "nvme0n1p1": {
          "deviceName": "nvme0n1p1",
          "free": "582250496",
          "freePercent": "0.93",
          "fstype": "vfat",
          "opts": "rw,relatime",
```

```
        "path": "0fc8c8d71702d81a02e216fb6ef19f4dda4973df",
        "total": "627900416",
        "used": "45649920",
        "usedPercent": "0.07"
      },
      "nvme0n1p2": {
        "deviceName": "nvme0n1p2",
        "free": "701976576",
        "freePercent": "0.74",
        "fstype": "ext4",
        "opts": "rw,relatime",
        "path": "/boot",
        "total": "1023303680",
        "used": "250863616",
        "usedPercent": "0.26"
      }
    }
  },
  {
    "instanceType": "pd",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "2379",
    "cpu": {
      "cache": "8192",
      "cpuFrequency": "2301.00MHz",
      "cpuLogicalCores": "8",
      "cpuPhysicalCores": "4"
    },
    "memory": {
      "capacity": "16410021888"
    },
    "disk": {
      "ebbca862689fa9fef7c55c3112e375c4ce575fe4": {
        "deviceName": "ebbca862689fa9fef7c55c3112e375c4ce575fe4",
        "free": "624438726656",
        "freePercent": "0.61",
        "fstype": "btrfs",
        "opts": "bind,rw,relatime",
        "path": "fb365c1216b59e1cfc86950425867007a60f4435",
        "total": "1022488477696",
        "used": "397115568128",
        "usedPercent": "0.39"
      },
      "nvme0n1p1": {
        "deviceName": "nvme0n1p1",
```

```
        "free": "582250496",
        "freePercent": "0.93",
        "fstype": "vfat",
        "opts": "rw,relatime",
        "path": "0fc8c8d71702d81a02e216fb6ef19f4dda4973df",
        "total": "627900416",
        "used": "45649920",
        "usedPercent": "0.07"
      },
      "nvme0n1p2": {
        "deviceName": "nvme0n1p2",
        "free": "701976576",
        "freePercent": "0.74",
        "fstype": "ext4",
        "opts": "rw,relatime",
        "path": "/boot",
        "total": "1023303680",
        "used": "250863616",
        "usedPercent": "0.26"
      }
    }
  },
  {
    "instanceType": "tikv",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "20160",
    "cpu": {
      "cpuFrequency": "3730MHz",
      "cpuLogicalCores": "8",
      "cpuPhysicalCores": "4",
      "cpuVendorId": "GenuineIntel",
      "l1CacheLineSize": "64",
      "l1CacheSize": "32768",
      "l2CacheLineSize": "64",
      "l2CacheSize": "262144",
      "l3CacheLineSize": "64",
      "l3CacheSize": "8388608"
    },
    "memory": {
      "capacity": "16803861504"
    },
    "disk": {
      "36e7dfacbb83843f83075d78aeb4cf850a4882a1": {
        "deviceName": "36e7dfacbb83843f83075d78aeb4cf850a4882a1",
        "free": "624438726656",
```

```
        "freePercent": "0.61",
        "fstype": "btrfs",
        "path": "fb365c1216b59e1cfc86950425867007a60f4435",
        "total": "1022488477696",
        "used": "398049751040",
        "usedPercent": "0.39"
      },
      "nvme0n1p1": {
        "deviceName": "nvme0n1p1",
        "free": "582250496",
        "freePercent": "0.93",
        "fstype": "vfat",
        "path": "0fc8c8d71702d81a02e216fb6ef19f4dda4973df",
        "total": "627900416",
        "used": "45649920",
        "usedPercent": "0.07"
      },
      "nvme0n1p2": {
        "deviceName": "nvme0n1p2",
        "free": "701976576",
        "freePercent": "0.69",
        "fstype": "ext4",
        "path": "/boot",
        "total": "1023303680",
        "used": "321327104",
        "usedPercent": "0.31"
      }
    }
  },
  {
    "instanceType": "tiflash",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "3930",
    "cpu": {
      "cpuFrequency": "3400MHz",
      "cpuLogicalCores": "8",
      "cpuPhysicalCores": "4",
      "l1CacheLineSize": "64",
      "l1CacheSize": "32768",
      "l2CacheLineSize": "64",
      "l2CacheSize": "262144",
      "l3CacheLineSize": "64",
      "l3CacheSize": "8388608"
    },
    "memory": {
```

```
      "capacity": "16410021888"
    },
    "disk": {
      "36e7dfacbb83843f83075d78aeb4cf850a4882a1": {
        "deviceName": "36e7dfacbb83843f83075d78aeb4cf850a4882a1",
        "free": "624438726656",
        "freePercent": "0.61",
        "fstype": "btrfs",
        "path": "fb365c1216b59e1cfc86950425867007a60f4435",
        "total": "1022488477696",
        "used": "398049751040",
        "usedPercent": "0.39"
      }
    }
  }
],
"instances": [
  {
    "instanceType": "tidb",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "4000",
    "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "statusPort": "10080",
    "version": "5.1.1",
    "gitHash": "797bddd25310ed42f0791c8eccb78be8cce2f502",
    "startTime": "2021-08-11T08:23:38+02:00",
    "upTime": "22.210217487s"
  },
  {
    "instanceType": "pd",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "2379",
    "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "statusPort": "2379",
    "version": "5.1.1",
    "gitHash": "7cba1912b317a533e18b16ea2ba9a14ed2891129",
    "startTime": "2021-08-11T08:23:32+02:00",
    "upTime": "28.210220368s"
  },
  {
    "instanceType": "tikv",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "20160",
    "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "statusPort": "20180",
```

```
      "version": "5.1.1",
      "gitHash": "4705d7c6e9c42d129d3309e05911ec6b08a25a38",
      "startTime": "2021-08-11T08:23:33+02:00",
      "upTime": "27.210221447s"
    },
    {
      "instanceType": "tiflash",
      "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
      "listenPort": "3930",
      "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
      "statusPort": "20292",
      "version": "v5.1.1",
      "gitHash": "c8fabfb50fe28db17cc5118133a69be255c40efd",
      "startTime": "2021-08-11T08:23:40+02:00",
      "upTime": "20.210222452s"
    }
  ],
  "hostExtra": {
    "cpuFlags": [
      "fpu",
      "vme",
      "de",
      "pse",
      "tsc",
      "msr",
      "pae",
      "mce",
      "cx8",
      "apic",
      "sep",
      "mtrr",
      "pge",
      "mca",
      "cmov",
      "pat",
      "pse36",
      "clflush",
      "dts",
      "acpi",
      "mmx",
      "fxsr",
      "sse",
      "sse2",
      "ss",
      "ht",
```

```
        "tm",
        "pbe",
        "syscall",
        "nx",
        "pdpe1gb",
        "rdtscp",
        "lm",
        "constant_tsc",
        "art",
        "arch_perfmon",
        "pebs",
        "bts",
        "rep_good",
        "nopl",
        "xtopology",
        "nonstop_tsc",
        "cpuid",
        "aperfmperf",
        "pni",
        "pclmulqdq",
        "dtes64",
        "monitor",
        "ds_cpl",
        "vmx",
        "est",
        "tm2",
        "ssse3",
        "sdbg",
        "fma",
        "cx16",
        "xtpr",
        "pdcm",
        "pcid",
        "sse4_1",
        "sse4_2",
        "x2apic",
        "movbe",
        "popcnt",
        "tsc_deadline_timer",
        "aes",
        "xsave",
        "avx",
        "f16c",
        "rdrand",
        "lahf_lm",
```

```
        "abm",
        "3dnowprefetch",
        "cpuid_fault",
        "epb",
        "invpcid_single",
        "ssbd",
        "ibrs",
        "ibpb",
        "stibp",
        "ibrs_enhanced",
        "tpr_shadow",
        "vnmi",
        "flexpriority",
        "ept",
        "vpid",
        "ept_ad",
        "fsgsbase",
        "tsc_adjust",
        "sgx",
        "bmi1",
        "avx2",
        "smep",
        "bmi2",
        "erms",
        "invpcid",
        "mpx",
        "rdseed",
        "adx",
        "smap",
        "clflushopt",
        "intel_pt",
        "xsaveopt",
        "xsavec",
        "xgetbv1",
        "xsaves",
        "dtherm",
        "ida",
        "arat",
        "pln",
        "pts",
        "hwp",
        "hwp_notify",
        "hwp_act_window",
        "hwp_epp",
        "md_clear",
```

```
      "flush_l1d",
      "arch_capabilities"
    ],
    "cpuModelName": "Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz",
    "os": "linux",
    "platform": "fedora",
    "platformFamily": "fedora",
    "platformVersion": "34",
    "kernelVersion": "5.13.5-200.fc34.x86_64",
    "kernelArch": "x86_64",
    "virtualizationSystem": "kvm",
    "virtualizationRole": "host"
  },
  "reportTimestamp": 1628663040,
  "trackingId": "a1ba1d97-b940-4d5b-a9d5-ddb0f2ac29e7",
  "featureUsage": {
    "txn": {
      "asyncCommitUsed": true,
      "onePCUsed": true,
      "txnCommitCounter": {
        "twoPC": 9,
        "asyncCommit": 0,
        "onePC": 0
      }
    },
    "clusterIndex": {},
    "temporaryTable": false,
    "cte": {
      "nonRecursiveCTEUsed": 0,
      "recursiveUsed": 0,
      "nonCTEUsed": 13
    }
  },
  "windowedStats": [],
  "slowQueryStats": {
    "slowQueryBucket": {}
  }
}
1 row in set (0.0259 sec)
```

14.10.2.10.3 MySQL compatibility

The `ADMIN` statement is a TiDB extension to MySQL syntax.

14.10.2.10.4 See also

- Telemetry
- `tidb_enable_telemetry` System Variable

14.10.2.11 ALTER DATABASE

`ALTER DATABASE` is used to specify or modify the default character set and collation of the current database. `ALTER SCHEMA` has the same effect as `ALTER DATABASE`.

14.10.2.11.1 Synopsis

```
AlterDatabaseStmt ::=
    'ALTER' 'DATABASE' DBName? DatabaseOptionList

DatabaseOption ::=
    DefaultKwdOpt ( CharsetKw '='? CharsetName | 'COLLATE' '='?
        ↪ CollationName | 'ENCRYPTION' '='? EncryptionOpt )
```

14.10.2.11.2 Examples

Modify the test database schema to use the utf8mb4 character set:

```
ALTER DATABASE test DEFAULT CHARACTER SET = utf8mb4;
```

```
Query OK, 0 rows affected (0.00 sec)
```

Currently, TiDB only supports some character sets and collations. See Character Set and Collation Support for details.

14.10.2.11.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.11.4 See also

- CREATE DATABASE
- SHOW DATABASES

14.10.2.12 ALTER INDEX

The `ALTER INDEX` statement is used to modify the visibility of the index to `Visible` or `Invisible`. Invisible indexes are maintained by DML statements, but will not be used by the query optimizer. This is useful in scenarios where you want to double-check before removing an index permanently.

14.10.2.12.1 Synopsis

```
AlterTableStmt
        ::= 'ALTER' 'IGNORE'? 'TABLE' TableName AlterIndexSpec ( ','
           ↪ AlterIndexSpec )*

AlterIndexSpec
        ::= 'ALTER' 'INDEX' Identifier ( 'VISIBLE' | 'INVISIBLE' )
```

14.10.2.12.2 Examples

You can modify the visibility of an index using the `ALTER TABLE ... ALTER INDEX ...` statement.

```
CREATE TABLE t1 (c1 INT, UNIQUE(c1));
ALTER TABLE t1 ALTER INDEX c1 INVISIBLE;
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
SHOW CREATE TABLE t1;
```

```
+--
   ↪ -----+-----------------------------------------------------------------
   ↪
| Table | Create Table
                                |
+--
   ↪ -----+-----------------------------------------------------------------
   ↪
| t1    | CREATE TABLE `t1` (
  `c1` int(11) DEFAULT NULL,
  UNIQUE KEY `c1` (`c1`) /*!80000 INVISIBLE */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin |
+--
   ↪ -----+-----------------------------------------------------------------
   ↪
1 row in set (0.00 sec)
```

The optimizer cannot use the **invisible index** of `c1`.

```
EXPLAIN SELECT c1 FROM t1 ORDER BY c1;
```

```
+--
   ↪ ----------------------+----------+-----------+--------------+-----------
   ↪
```

```
| id                    | estRows | task     | access object | operator info
    ↪             |
+--
    ↪ ----------------------+---------+----------+---------------+----------------
    ↪
| Sort_4                | 10000.00 | root    |               | test.t1.c1:asc
    ↪              |
|  ─TableReader_8       | 10000.00 | root    |               | data:
    ↪ TableFullScan_7        |
|    ─TableFullScan_7   | 10000.00 | cop[tikv] | table:t1 | keep order:false
    ↪ , stats:pseudo |
+--
    ↪ ----------------------+---------+----------+---------------+----------------
    ↪
3 rows in set (0.00 sec)
```

By comparison, c2 is a **visible index** and can be used by the optimizer.

```
EXPLAIN SELECT c2 FROM t1 ORDER BY c2;
```

```
+--
    ↪ ----------------------+---------+----------+-----------------------+--------
    ↪
| id                    | estRows | task     | access object         | operator
    ↪ info              |
+--
    ↪ ----------------------+---------+----------+-----------------------+--------
    ↪
| IndexReader_13        | 10000.00 | root    |                       | index:
    ↪ IndexFullScan_12     |
|  ─IndexFullScan_12    | 10000.00 | cop[tikv] | table:t1, index:c2(c2) |
    ↪ keep order:true, stats:pseudo |
+--
    ↪ ----------------------+---------+----------+-----------------------+--------
    ↪
2 rows in set (0.00 sec)
```

Even if you use the USE INDEX SQL hint to forcibly use indexes, the optimizer still cannot use invisible indexes; otherwise, an error is returned.

```
SELECT * FROM t1 USE INDEX(c1);
```

```
ERROR 1176 (42000): Key 'c1' doesn't exist in table 't1'
```

> **Note:**
>
> "Invisible" here means invisible only to the optimizer. You can still modify or delete invisible indexes.

```
ALTER TABLE t1 DROP INDEX c1;
```

```
Query OK, 0 rows affected (0.02 sec)
```

14.10.2.12.3 MySQL compatibility

- Invisible indexes in TiDB are modeled on the equivalent feature from MySQL 8.0.
- Similiar to MySQL, TiDB does not permit `PRIMARY KEY` indexes to be made invisible.
- MySQL provides an optimizer switch `use_invisible_indexes=on` to make all invisible indexes *visible* again. This functionality is not available in TiDB.

14.10.2.12.4 See also

- CREATE TABLE
- CREATE INDEX
- ADD INDEX
- DROP INDEX
- RENAME INDEX

14.10.2.13 ALTER INSTANCE

The `ALTER INSTANCE` statement is used to make changes to a single TiDB instance. Currently, TiDB only supports the `RELOAD TLS` clause.

14.10.2.13.1 RELOAD TLS

You can execute the `ALTER INSTANCE RELOAD TLS` statement to reload the certificate (`ssl-cert`), the key (`ssl-key`), and the CA (`ssl-ca`) from the original configuration path.

The newly loaded certificate, key, and CA take effect on the connection that is established after the statement is successfully executed. The connection established before this statement execution is not affected.

When an error occurs during reloading, by default, this error message is returned and the previous key and certificate continue to be used. However, if you have added the optional `NO ROLLBACK ON ERROR`, when an error occurs during reloading, the error is not returned, and the subsequent requests are handled with the TLS security connection disabled.

14.10.2.13.2 Syntax diagram

AlterInstanceStmt:

```
AlterInstanceStmt ::=
    'ALTER' 'INSTANCE' InstanceOption

InstanceOption ::=
    'RELOAD' 'TLS' ('NO' 'ROLLBACK' 'ON' 'ERROR')?
```

14.10.2.13.3 Example

```
ALTER INSTANCE RELOAD TLS;
```

14.10.2.13.4 MySQL compatibility

The `ALTER INSTANCE RELOAD TLS` statement only supports reloading from the original configuration path. It does not support dynamically modifying the loading path or dynamically enabling the TLS encrypted connection feature when TiDB is started. This feature is disabled by default when you restart TiDB.

14.10.2.13.5 See also

Enable TLS Between TiDB Clients and Servers.

14.10.2.14 ALTER PLACEMENT POLICY

`ALTER PLACEMENT POLICY` is used to modify existing placement policies that have previously been created. All the tables and partitions which use the placement policy will automatically be updated.

`ALTER PLACEMENT POLICY` *replaces* the previous policy with the new definition. It does not *merge* the old policy with the new one. In the following example, `FOLLOWERS=4` is lost when the `ALTER PLACEMENT POLICY` is executed:

```
CREATE PLACEMENT POLICY p1 FOLLOWERS=4;
ALTER PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-
    ↪ west-1";
```

14.10.2.14.1 Synopsis

```
AlterPolicyStmt ::=
    "ALTER" "PLACEMENT" "POLICY" IfExists PolicyName PlacementOptionList

PolicyName ::=
    Identifier
```

```
PlacementOptionList ::=
    PlacementOption
|   PlacementOptionList PlacementOption
|   PlacementOptionList ',' PlacementOption

PlacementOption ::=
    CommonPlacementOption
|   SugarPlacementOption
|   AdvancedPlacementOption

CommonPlacementOption ::=
    "FOLLOWERS" EqOpt LengthNum

SugarPlacementOption ::=
    "PRIMARY_REGION" EqOpt stringLit
|   "REGIONS" EqOpt stringLit
|   "SCHEDULE" EqOpt stringLit

AdvancedPlacementOption ::=
    "LEARNERS" EqOpt LengthNum
|   "CONSTRAINTS" EqOpt stringLit
|   "LEADER_CONSTRAINTS" EqOpt stringLit
|   "FOLLOWER_CONSTRAINTS" EqOpt stringLit
|   "LEARNER_CONSTRAINTS" EqOpt stringLit
```

14.10.2.14.2 Examples

> **Note:**
>
> To know which regions are available in your cluster, see SHOW PLACEMENT LABELS.
>
> If you do not see any available regions, your TiKV installation might not have labels set correctly.

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us
    -west-1";
CREATE TABLE t1 (i INT) PLACEMENT POLICY=p1; -- Assign policy p1 to table
    t1
ALTER PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-
    west-1,us-west-2" FOLLOWERS=4; -- The rules of t1 will be updated
```

```
    ↪ automatically.
SHOW CREATE PLACEMENT POLICY p1\G;
```

```
Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.10 sec)

***************************[ 1. row ]***************************
Policy      | p1
Create Policy | CREATE PLACEMENT POLICY `p1` PRIMARY_REGION="us-east-1"
    ↪ REGIONS="us-east-1,us-west-1,us-west-2" FOLLOWERS=4
1 row in set (0.00 sec)
```

14.10.2.14.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.14.4 See also

- Placement Rules in SQL
- SHOW PLACEMENT
- CREATE PLACEMENT POLICY
- DROP PLACEMENT POLICY

14.10.2.15 ALTER TABLE

This statement modifies an existing table to conform to a new table structure. The statement `ALTER TABLE` can be used to:

- `ADD`, `DROP`, or `RENAME` indexes
- `ADD`, `DROP`, `MODIFY` or `CHANGE` columns
- `COMPACT` table data

14.10.2.15.1 Synopsis

```
AlterTableStmt ::=
    'ALTER' IgnoreOptional 'TABLE' TableName (
        AlterTableSpecListOpt AlterTablePartitionOpt |
        'ANALYZE' 'PARTITION' PartitionNameList ( 'INDEX' IndexNameList )?
            ↪ AnalyzeOptionListOpt |
        'COMPACT' 'TIFLASH' 'REPLICA'
    )

TableName ::=
```

```
    Identifier ('.' Identifier)?

AlterTableSpec ::=
    TableOptionList
|   'SET' 'TIFLASH' 'REPLICA' LengthNum LocationLabelList
|   'CONVERT' 'TO' CharsetKw ( CharsetName | 'DEFAULT' ) OptCollate
|   'ADD' ( ColumnKeywordOpt IfNotExists ( ColumnDef ColumnPosition | '('
    ↪ TableElementList ')' ) | Constraint | 'PARTITION' IfNotExists
    ↪ NoWriteToBinLogAliasOpt ( PartitionDefinitionListOpt | 'PARTITIONS'
    ↪ NUM ) )
|   ( ( 'CHECK' | 'TRUNCATE' ) 'PARTITION' | ( 'OPTIMIZE' | 'REPAIR' | '
    ↪ REBUILD' ) 'PARTITION' NoWriteToBinLogAliasOpt )
    ↪ AllOrPartitionNameList
|   'COALESCE' 'PARTITION' NoWriteToBinLogAliasOpt NUM
|   'DROP' ( ColumnKeywordOpt IfExists ColumnName RestrictOrCascadeOpt | '
    ↪ PRIMARY' 'KEY' | 'PARTITION' IfExists PartitionNameList | (
    ↪ KeyOrIndex IfExists | 'CHECK' ) Identifier | 'FOREIGN' 'KEY' IfExists
    ↪  Symbol )
|   'EXCHANGE' 'PARTITION' Identifier 'WITH' 'TABLE' TableName
    ↪ WithValidationOpt
|   ( 'IMPORT' | 'DISCARD' ) ( 'PARTITION' AllOrPartitionNameList )? '
    ↪ TABLESPACE'
|   'REORGANIZE' 'PARTITION' NoWriteToBinLogAliasOpt
    ↪ ReorganizePartitionRuleOpt
|   'ORDER' 'BY' AlterOrderItem ( ',' AlterOrderItem )*
|   ( 'DISABLE' | 'ENABLE' ) 'KEYS'
|   ( 'MODIFY' ColumnKeywordOpt IfExists | 'CHANGE' ColumnKeywordOpt
    ↪ IfExists ColumnName ) ColumnDef ColumnPosition
|   'ALTER' ( ColumnKeywordOpt ColumnName ( 'SET' 'DEFAULT' ( SignedLiteral
    ↪ | '(' Expression ')' ) | 'DROP' 'DEFAULT' ) | 'CHECK' Identifier
    ↪ EnforcedOrNot | 'INDEX' Identifier IndexInvisible )
|   'RENAME' ( ( 'COLUMN' | KeyOrIndex ) Identifier 'TO' Identifier | ( 'TO'
    ↪  | '='? | 'AS' ) TableName )
|   LockClause
|   AlgorithmClause
|   'FORCE'
|   ( 'WITH' | 'WITHOUT' ) 'VALIDATION'
|   'SECONDARY_LOAD'
|   'SECONDARY_UNLOAD'
|   ( 'AUTO_INCREMENT' | 'AUTO_ID_CACHE' | 'AUTO_RANDOM_BASE' | '
    ↪ SHARD_ROW_ID_BITS' ) EqOpt LengthNum
|   ( 'CACHE' | 'NOCACHE' )
|   PlacementPolicyOption

PlacementPolicyOption ::=
```

```
    "PLACEMENT" "POLICY" EqOpt PolicyName
|   "PLACEMENT" "POLICY" (EqOpt | "SET") "DEFAULT"
```

14.10.2.15.2 Examples

Create a table with some initial data:

```
CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT NOT
    ↪ NULL);
INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
```

```
Query OK, 0 rows affected (0.11 sec)

Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

The following query requires a full table scan because the column c1 is not indexed:

```
EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
```

```
+--
    ↪ ----------------------+----------+-----------+---------------+----------------
    ↪
| id                    | estRows | task      | access object | operator info
    ↪             |
+--
    ↪ ----------------------+----------+-----------+---------------+----------------
    ↪
| TableReader_7         | 10.00   | root      |               | data:Selection_6
    ↪           |
| └─Selection_6         | 10.00   | cop[tikv] |               | eq(test.t1.c1,
    ↪ 3)          |
|   └─TableFullScan_5   | 10000.00 | cop[tikv] | table:t1 | keep order:false
    ↪ , stats:pseudo |
+--
    ↪ ----------------------+----------+-----------+---------------+----------------
    ↪
3 rows in set (0.00 sec)
```

The statement ALTER TABLE .. ADD INDEX can be used to add an index on the table t1. EXPLAIN confirms that the original query now uses an index range scan, which is more efficient:

```
ALTER TABLE t1 ADD INDEX (c1);
EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
```

```
Query OK, 0 rows affected (0.30 sec)

+--
    ↪ ---------------------+---------+-----------+-----------------------+---------
    ↪
| id                    | estRows | task    | access object       | operator
    ↪ info                          |
+--
    ↪ ---------------------+---------+-----------+-----------------------+---------
    ↪
| IndexReader_6         | 10.00   | root      |                       | index:
    ↪ IndexRangeScan_5                |
|  ─IndexRangeScan_5    | 10.00   | cop[tikv] | table:t1, index:c1(c1) | range
    ↪ :[3,3], keep order:false, stats:pseudo |
+--
    ↪ ---------------------+---------+-----------+-----------------------+---------
    ↪
2 rows in set (0.00 sec)
```

TiDB supports the ability to assert that DDL changes will use a particular `ALTER` algorithm. This is only an assertion, and does not change the actual algorithm which will be used to modify the table. It can be useful if you only want to permit instant DDL changes during the peak hours of your cluster:

```
ALTER TABLE t1 DROP INDEX c1, ALGORITHM=INSTANT;
```

```
Query OK, 0 rows affected (0.24 sec)
```

Using the `ALGORITHM=INSTANT` assertion on an operation that requires the `INPLACE` algorithm results in a statement error:

```
ALTER TABLE t1 ADD INDEX (c1), ALGORITHM=INSTANT;
```

```
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot
    ↪ alter table by INSTANT. Try ALGORITHM=INPLACE.
```

However, using the `ALGORITHM=COPY` assertion for an `INPLACE` operation generates a warning instead of an error. This is because TiDB interprets the assertion as *this algorithm or better*. This behavior difference is useful for MySQL compatibility because the algorithm TiDB uses might differ from MySQL:

```
ALTER TABLE t1 ADD INDEX (c1), ALGORITHM=COPY;
SHOW WARNINGS;
```

```
Query OK, 0 rows affected, 1 warning (0.25 sec)

+--
    ↪ -----+------+-------------------------------------------------------------------
    ↪
| Level | Code | Message
    ↪
    ↪ |
+--
    ↪ -----+------+-------------------------------------------------------------------
    ↪
| Error | 1846 | ALGORITHM=COPY is not supported. Reason: Cannot alter
    ↪ table by COPY. Try ALGORITHM=INPLACE. |
+--
    ↪ -----+------+-------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.2.15.3 MySQL compatibility

The following major restrictions apply to `ALTER TABLE` in TiDB:

- Making multiple changes in a single `ALTER TABLE` statement is currently not supported.
- Changes of the Reorg-Data types on primary key columns are not supported.
- Changes of column types on partitioned tables are not supported.
- Changes of column types on generated columns are not supported.
- Changes of some data types (for example, some TIME, Bit, Set, Enum, and JSON types) are not supported due to the compatibility issues of the `CAST` function's behavior between TiDB and MySQL.
- Spatial data types are not supported.
- `ALTER TABLE t CACHE | NOCACHE` is a TiDB extension to MySQL syntax. For details, see Cached Tables.

For further restrictions, see MySQL Compatibility.

14.10.2.15.4 See also

- MySQL Compatibility
- ADD COLUMN
- DROP COLUMN
- ADD INDEX
- DROP INDEX
- RENAME INDEX
- ALTER INDEX
- CREATE TABLE
- DROP TABLE
- SHOW CREATE TABLE

14.10.2.16 ALTER TABLE ... COMPACT

> **Warning:**
>
> This statement is still an experimental feature. It is NOT recommended that you use it in the production environment.

To enhance read performance and reduce disk usage, TiDB automatically schedules data compaction on storage nodes in the background. During the compaction, storage nodes rewrite physical data, including cleaning up deleted rows and merging multiple versions of data caused by updates. The `ALTER TABLE ... COMPACT` statement allows you to initiate compaction for a specific table immediately, without waiting until compaction is triggered in the background.

The execution of this statement does not block existing SQL statements or affect any TiDB features, such as transactions, DDL, and GC. Data that can be selected via SQL statements will not be changed either. However, executing this statement consumes some IO and CPU resources, which might result in higher SQL execution latency.

The compaction statement will be finished and returned when all replicas of a table are compacted. During the execution process, you can safely interrupt the compaction by executing the `KILL` statement. Interrupting a compaction does not break data consistency or lead to data loss, nor does it affect subsequent manual or background compactions.

This data compaction statement is currently supported only for TiFlash replicas, not for TiKV replicas.

14.10.2.16.1 Synopsis

```
AlterTableCompactStmt ::=
    'ALTER' 'TABLE' TableName 'COMPACT' 'TIFLASH' 'REPLICA'
```

14.10.2.16.2 Examples

Compact TiFlash replicas in a table

The following takes an `employees` table as an example, which has 4 partitions with 2 TiFlash replicas:

```
CREATE TABLE employees (
    id INT NOT NULL,
    hired DATE NOT NULL DEFAULT '1970-01-01',
    store_id INT
)
PARTITION BY LIST (store_id) (
    PARTITION pNorth VALUES IN (1, 2, 3, 4, 5),
    PARTITION pEast VALUES IN (6, 7, 8, 9, 10),
    PARTITION pWest VALUES IN (11, 12, 13, 14, 15),
    PARTITION pCentral VALUES IN (16, 17, 18, 19, 20)
);
ALTER TABLE employees SET TIFLASH REPLICA 2;
```

You can execute the following statement to immediately initiate the compaction for the 2 TiFlash replicas for all partitions in the `employees` table:

```
ALTER TABLE employees COMPACT TIFLASH REPLICA;
```

14.10.2.16.3 Concurrency

The `ALTER TABLE ... COMPACT` statement compacts all replicas in a table simultaneously.

To avoid a significant impact on online business, each TiFlash instance only compacts data in one table at a time by default (except for the compaction triggered in the background). This means that if you execute the `ALTER TABLE ... COMPACT` statement on multiple tables at the same time, their executions will be queued on the same TiFlash instance, rather than being executed simultaneously.

To obtain greater table-level concurrency with higher resource usage, you can modify the TiFlash configuration `manual_compact_pool_size`. For example, when `manual_compact_pool_size` is set to 2, compaction for 2 tables can be processed simultaneously.

14.10.2.16.4 Observe data compaction progress

You can observe the progress of data compaction or determine whether to initiate compaction for a table by checking the `TOTAL_DELTA_ROWS` column in the `INFORMATION_SCHEMA.TIFLASH_TABLES` table. The larger the value of `TOTAL_DELTA_ROWS`, the more data that can be compacted. If `TOTAL_DELTA_ROWS` is `0`, all data in the table is in the best state and does not need to be compacted.

Example: Check the compaction state of a non-partitioned table

```
USE test;

CREATE TABLE foo(id INT);

ALTER TABLE foo SET TIFLASH REPLICA 1;

SELECT TOTAL_DELTA_ROWS, TOTAL_STABLE_ROWS FROM INFORMATION_SCHEMA.
    ↪ TIFLASH_TABLES
    WHERE IS_TOMBSTONE = 0 AND
    `TIDB_DATABASE` = "test" AND `TIDB_TABLE` = "foo";
+------------------+-------------------+
| TOTAL_DELTA_ROWS | TOTAL_STABLE_ROWS |
+------------------+-------------------+
|                0 |                 0 |
+------------------+-------------------+

INSERT INTO foo VALUES (1), (3), (7);

SELECT TOTAL_DELTA_ROWS, TOTAL_STABLE_ROWS FROM INFORMATION_SCHEMA.
    ↪ TIFLASH_TABLES
    WHERE IS_TOMBSTONE = 0 AND
    `TIDB_DATABASE` = "test" AND `TIDB_TABLE` = "foo";
+------------------+-------------------+
| TOTAL_DELTA_ROWS | TOTAL_STABLE_ROWS |
+------------------+-------------------+
|                3 |                 0 |
+------------------+-------------------+
-- Newly written data can be compacted

ALTER TABLE foo COMPACT TIFLASH REPLICA;

SELECT TOTAL_DELTA_ROWS, TOTAL_STABLE_ROWS FROM INFORMATION_SCHEMA.
    ↪ TIFLASH_TABLES
    WHERE IS_TOMBSTONE = 0 AND
    `TIDB_DATABASE` = "test" AND `TIDB_TABLE` = "foo";
+------------------+-------------------+
| TOTAL_DELTA_ROWS | TOTAL_STABLE_ROWS |
+------------------+-------------------+
|                0 |                 3 |
+------------------+-------------------+
-- All data is in the best state and no compaction is needed
```

Example: Check the compaction state of a partitioned table

```
USE test;

CREATE TABLE employees
    (id INT NOT NULL, store_id INT)
    PARTITION BY LIST (store_id) (
        PARTITION pNorth VALUES IN (1, 2, 3, 4, 5),
        PARTITION pEast VALUES IN (6, 7, 8, 9, 10),
        PARTITION pWest VALUES IN (11, 12, 13, 14, 15),
        PARTITION pCentral VALUES IN (16, 17, 18, 19, 20)
    );

ALTER TABLE employees SET TIFLASH REPLICA 1;

INSERT INTO employees VALUES (1, 1), (6, 6), (10, 10);

SELECT PARTITION_NAME, TOTAL_DELTA_ROWS, TOTAL_STABLE_ROWS
    FROM INFORMATION_SCHEMA.TIFLASH_TABLES t, INFORMATION_SCHEMA.PARTITIONS
        ↪ p
    WHERE t.IS_TOMBSTONE = 0 AND t.TABLE_ID = p.TIDB_PARTITION_ID AND
    p.TABLE_SCHEMA = "test" AND p.TABLE_NAME = "employees";
+----------------+------------------+--------------------+
| PARTITION_NAME | TOTAL_DELTA_ROWS | TOTAL_STABLE_ROWS |
+----------------+------------------+--------------------+
| pNorth         |                1 |                  0 |
| pEast          |                2 |                  0 |
| pWest          |                0 |                  0 |
| pCentral       |                0 |                  0 |
+----------------+------------------+--------------------+
-- Some partitions can be compacted

ALTER TABLE employees COMPACT TIFLASH REPLICA;

SELECT PARTITION_NAME, TOTAL_DELTA_ROWS, TOTAL_STABLE_ROWS
    FROM INFORMATION_SCHEMA.TIFLASH_TABLES t, INFORMATION_SCHEMA.PARTITIONS
        ↪ p
    WHERE t.IS_TOMBSTONE = 0 AND t.TABLE_ID = p.TIDB_PARTITION_ID AND
    p.TABLE_SCHEMA = "test" AND p.TABLE_NAME = "employees";
+----------------+------------------+--------------------+
| PARTITION_NAME | TOTAL_DELTA_ROWS | TOTAL_STABLE_ROWS |
+----------------+------------------+--------------------+
| pNorth         |                0 |                  1 |
| pEast          |                0 |                  2 |
| pWest          |                0 |                  0 |
| pCentral       |                0 |                  0 |
+----------------+------------------+--------------------+
```

```
-- Data in all partitions is in the best state and no compaction is needed
```

Note:

- If data is updated during compaction, `TOTAL_DELTA_ROWS` might still be a non-zero value after compaction is done. This is normal and indicates that these updates have not been compacted. To compact these updates, execute the `ALTER TABLE ... COMPACT` statement again.
- `TOTAL_DELTA_ROWS` indicates the data version, not the number of rows. For example, if you insert a row and then delete it, `TOTAL_DELTA_ROWS` will increase by 2.

14.10.2.16.5 Compatibility

MySQL compatibility

The `ALTER TABLE ... COMPACT` syntax is TiDB specific, which is an extension to the standard SQL syntax. Although there is no equivalent MySQL syntax, you can still execute this statement by using MySQL clients or various database drivers that comply with the MySQL protocol.

TiDB Binlog and TiCDC compatibility

The `ALTER TABLE ... COMPACT` statement does not result in logical data changes and are therefore not replicated to downstream by TiDB Binlog or TiCDC.

14.10.2.16.6 See also

- ALTER TABLE
- KILL TIDB

14.10.2.17 ALTER USER

This statement changes an existing user inside the TiDB privilege system. In the MySQL privilege system, a user is the combination of a username and the host from which they are connecting from. Thus, it is possible to create a user `'newuser2'@'192.168.1.1'` who is only able to connect from the IP address `192.168.1.1`. It is also possible to have two users have the same user-portion, and different permissions as they login from different hosts.

14.10.2.17.1 Synopsis

```
AlterUserStmt ::=
    'ALTER' 'USER' IfExists (UserSpecList RequireClauseOpt ConnectionOptions
       ↪  PasswordOrLockOptions | 'USER' '(' ')' 'IDENTIFIED' 'BY'
       ↪ AuthString)

UserSpecList ::=
    UserSpec ( ',' UserSpec )*

UserSpec ::=
    Username AuthOption

Username ::=
    StringName ('@' StringName | singleAtIdentifier)? | 'CURRENT_USER'
       ↪ OptionalBraces

AuthOption ::=
    ( 'IDENTIFIED' ( 'BY' ( AuthString | 'PASSWORD' HashString ) | 'WITH'
       ↪ StringName ( 'BY' AuthString | 'AS' HashString )? ) )?
```

14.10.2.17.2 Examples

```
mysql> CREATE USER 'newuser' IDENTIFIED BY 'newuserpassword';
Query OK, 1 row affected (0.01 sec)

mysql> SHOW CREATE USER 'newuser';
+--
    ↪ ------------------------------------------------------------------------
    ↪
| CREATE USER for newuser@%
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------
    ↪
| CREATE USER 'newuser'@'%' IDENTIFIED WITH 'mysql_native_password' AS '
    ↪ *5806E04BBEE79E1899964C6A04D68BCA69B1A879' REQUIRE NONE PASSWORD
    ↪ EXPIRE DEFAULT ACCOUNT UNLOCK |
+--
    ↪ ------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> ALTER USER 'newuser' IDENTIFIED BY 'newnewpassword';
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> SHOW CREATE USER 'newuser';
+--
    ↪ ------------------------------------------------------------------------------
    ↪
| CREATE USER for newuser@%
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------------
    ↪
| CREATE USER 'newuser'@'%' IDENTIFIED WITH 'mysql_native_password' AS '*
    ↪ FB8A1EA1353E8775CA836233E367FBDFCB37BE73' REQUIRE NONE PASSWORD
    ↪ EXPIRE DEFAULT ACCOUNT UNLOCK |
+--
    ↪ ------------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.2.17.3 MySQL compatibility

- In MySQL this statement is used to change attributes such as to expire a password. This functionality is not yet supported by TiDB.

14.10.2.17.4 See also

- Security Compatibility with MySQL

- CREATE USER
- DROP USER
- SHOW CREATE USER

14.10.2.18 ANALYZE

This statement updates the statistics that TiDB builds on tables and indexes. It is recommended to run `ANALYZE` after performing a large batch update or import of records, or when you notice that query execution plans are sub-optimal.

TiDB will also automatically update its statistics over time as it discovers that they are inconsistent with its own estimates.

Currently, TiDB collects statistical information in two ways: full collection (implemented using the `ANALYZE TABLE` statement) and incremental collection (implemented using the `ANALYZE INCREMENTAL TABLE` statement). For detailed usage of these two statements, refer to introduction to statistics

14.10.2.18.1 Synopsis

```
AnalyzeTableStmt ::=
    'ANALYZE' ( 'TABLE' ( TableNameList ( 'ALL COLUMNS' | 'PREDICATE COLUMNS
       ↪ ' ) | TableName ( 'INDEX' IndexNameList? | AnalyzeColumnOption | '
       ↪ PARTITION' PartitionNameList ( 'INDEX' IndexNameList? |
       ↪ AnalyzeColumnOption )? )? ) | 'INCREMENTAL' 'TABLE' TableName ( '
       ↪ PARTITION' PartitionNameList )? 'INDEX' IndexNameList? )
       ↪ AnalyzeOptionListOpt

AnalyzeOptionListOpt ::=
( WITH AnalyzeOptionList )?

AnalyzeOptionList ::=
AnalyzeOption ( ',' AnalyzeOption )*

AnalyzeOption ::=
( NUM ( 'BUCKETS' | 'TOPN' | ( 'CMSKETCH' ( 'DEPTH' | 'WIDTH' ) ) | 'SAMPLES
   ↪ ' ) ) | ( FLOATNUM 'SAMPLERATE' )

AnalyzeColumnOption ::=
( 'ALL COLUMNS' | 'PREDICATE COLUMNS' | 'COLUMNS' ColumnNameList )

TableNameList ::=
    TableName (',' TableName)*

TableName ::=
    Identifier ( '.' Identifier )?

ColumnNameList ::=
    Identifier ( ',' Identifier )*

IndexNameList ::=
    Identifier ( ',' Identifier )*

PartitionNameList ::=
    Identifier ( ',' Identifier )*
```

14.10.2.18.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
   ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
```

```
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE t1 ADD INDEX (c1);
Query OK, 0 rows affected (0.30 sec)

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
   ↪ ---------------------+---------+----------+-----------------------+----------
   ↪
| id                    | estRows | task     | access object         | operator
   ↪ info                            |
+--
   ↪ ---------------------+---------+----------+-----------------------+----------
   ↪
| IndexReader_6         | 10.00   | root     |                       | index:
   ↪ IndexRangeScan_5                  |
|  ─IndexRangeScan_5    | 10.00   | cop[tikv] | table:t1, index:c1(c1) | range
   ↪ :[3,3], keep order:false, stats:pseudo |
+--
   ↪ ---------------------+---------+----------+-----------------------+----------
   ↪
2 rows in set (0.00 sec)

mysql> analyze table t1;
Query OK, 0 rows affected (0.13 sec)

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
   ↪ ---------------------+---------+----------+-----------------------+----------
   ↪
| id                    | estRows | task     | access object         | operator
   ↪ info                |
+--
   ↪ ---------------------+---------+----------+-----------------------+----------
   ↪
| IndexReader_6         | 1.00    | root     |                       | index:
   ↪ IndexRangeScan_5    |
|  ─IndexRangeScan_5    | 1.00    | cop[tikv] | table:t1, index:c1(c1) | range
   ↪ :[3,3], keep order:false |
+--
   ↪ ---------------------+---------+----------+-----------------------+----------
   ↪
2 rows in set (0.00 sec)
```

14.10.2.18.3 MySQL compatibility

TiDB differs from MySQL in **both** the statistics it collects and how it makes use of statistics during query execution. While this statement is syntactically similar to MySQL, the following differences apply:

1. TiDB might not include very recently committed changes when running `ANALYZE TABLE`. After a batch-update of rows, you might need to `sleep(1)` before executing `ANALYZE TABLE` in order for the statistics update to reflect these changes. #16570.
2. `ANALYZE TABLE` takes significantly longer to execute in TiDB than MySQL. This performance difference can be partially mitigated by enabling fast analyze with `SET GLOBAL tidb_enable_fast_analyze=1`. Fast analyze makes use of sampling, leading to less accurate statistics. Its usage is still considered experimental.

MySQL does not support the `ANALYZE INCREMENTAL TABLE` statement. TiDB supports incremental collection of statistics. For detailed usage, refer to incremental collection.

14.10.2.18.4 See also

- EXPLAIN
- EXPLAIN ANALYZE

14.10.2.19 BACKUP

This statement is used to perform a distributed backup of the TiDB cluster.

> **Warning:**
>
> This feature is experimental. It is not recommended that you use it in the production environment. This feature might be changed or removed without prior notice. If you find a bug, you can report an issue on GitHub.

The `BACKUP` statement uses the same engine as the BR tool does, except that the backup process is driven by TiDB itself rather than a separate BR tool. All benefits and warnings of BR also apply in this statement.

Executing `BACKUP` requires either the `BACKUP_ADMIN` or `SUPER` privilege. Additionally, both the TiDB node executing the backup and all TiKV nodes in the cluster must have read or write permission to the destination. Local storage (storage paths starting with `local://`) is not permitted when Security Enhanced Mode is enabled.

The `BACKUP` statement is blocked until the entire backup task is finished, failed, or canceled. A long-lasting connection should be prepared for executing `BACKUP`. The task can be canceled using the `KILL TIDB QUERY` statement.

Only one BACKUP and RESTORE task can be executed at a time. If a BACKUP or RESTORE statement is already being executed on the same TiDB server, the new BACKUP execution will wait until all previous tasks are finished.

BACKUP can only be used with "tikv" storage engine. Using BACKUP with the "unistore" engine will fail.

14.10.2.19.1 Synopsis

```
BackupStmt ::=
    "BACKUP" BRIETables "TO" stringLit BackupOption*

BRIETables ::=
    "DATABASE" ( '*' | DBName (',' DBName)* )
|   "TABLE" TableNameList

BackupOption ::=
    "RATE_LIMIT" '='? LengthNum "MB" '/' "SECOND"
|   "CONCURRENCY" '='? LengthNum
|   "CHECKSUM" '='? Boolean
|   "SEND_CREDENTIALS_TO_TIKV" '='? Boolean
|   "LAST_BACKUP" '='? BackupTSO
|   "SNAPSHOT" '='? ( BackupTSO | LengthNum TimestampUnit "AGO" )

Boolean ::=
    NUM | "TRUE" | "FALSE"

BackupTSO ::=
    LengthNum | stringLit
```

14.10.2.19.2 Examples

Back up databases

```
BACKUP DATABASE `test` TO 'local:///mnt/backup/2020/04/';
```

```
+--
    ↪ ----------------------------+----------+----------------+--------------------
    ↪
| Destination                | Size     | BackupTS       | Queue Time         |
    ↪ Execution Time    |
+--
    ↪ ----------------------------+----------+----------------+--------------------
    ↪
| local:///mnt/backup/2020/04/ | 248665063 | 416099531454472 | 2020-04-12
    ↪ 23:09:48 | 2020-04-12 23:09:48 |
```

```
+--
    ↪ ---------------------------+----------+----------------+---------------------
    ↪
1 row in set (58.453 sec)
```

In the example above, the `test` database is backed up into the local filesystem. The data is saved as SST files in the `/mnt/backup/2020/04/` directories distributed among all TiDB and TiKV nodes.

The first row of the result above is described as follows:

Column	Description
`Destination` ↪	The destination URL
`Size`	The total size of the backup archive, in bytes
`BackupTS`	The TSO of the snapshot when the backup is created (useful for incremental backup)
`Queue` ↪ `Time`	The timestamp (in current time zone) when the `BACKUP` task is queued.
`Execution` ↪ `Time`	The timestamp (in current time zone) when the `BACKUP` task starts to run.

Back up tables

```
BACKUP TABLE `test`.`sbtest01` TO 'local:///mnt/backup/sbtest01/';
```

```
BACKUP TABLE sbtest02, sbtest03, sbtest04 TO 'local:///mnt/backup/sbtest/';
```

Back up the entire cluster

```
BACKUP DATABASE * TO 'local:///mnt/backup/full/';
```

Note that the system tables (`mysql.*`, `INFORMATION_SCHEMA.*`, `PERFORMANCE_SCHEMA.*`, …) will not be included into the backup.

External storages

BR supports backing up data to S3 or GCS:

```
BACKUP DATABASE `test` TO 's3://example-bucket-2020/backup-05/?region=us-west-2&access-key={YOUR_ACCESS_KEY}&secret-access-key={YOUR_SECRET_KEY}';
```

The URL syntax is further explained in External Storages.

When running on cloud environment where credentials should not be distributed, set the `SEND_CREDENTIALS_TO_TIKV` option to `FALSE`:

```
BACKUP DATABASE `test` TO 's3://example-bucket-2020/backup-05/?region=us-west-2'
    SEND_CREDENTIALS_TO_TIKV = FALSE;
```

Performance fine-tuning

Use `RATE_LIMIT` to limit the average upload speed per TiKV node to reduce network bandwidth.

By default, every TiKV node would run 4 backup threads. This value can be adjusted with the `CONCURRENCY` option.

Before backup is completed, `BACKUP` would perform a checksum against the data on the cluster to verify correctness. This step can be disabled with the `CHECKSUM` option if you are confident that this is unnecessary.

```
BACKUP DATABASE `test` TO 's3://example-bucket-2020/backup-06/'
    RATE_LIMIT = 120 MB/SECOND
    CONCURRENCY = 8
    CHECKSUM = FALSE;
```

Snapshot

Specify a timestamp, TSO or relative time to backup historical data.

```
-- relative time
BACKUP DATABASE `test` TO 'local:///mnt/backup/hist01'
    SNAPSHOT = 36 HOUR AGO;

-- timestamp (in current time zone)
BACKUP DATABASE `test` TO 'local:///mnt/backup/hist02'
    SNAPSHOT = '2020-04-01 12:00:00';

-- timestamp oracle
BACKUP DATABASE `test` TO 'local:///mnt/backup/hist03'
    SNAPSHOT = 415685305958400;
```

The supported units for relative time are:

- MICROSECOND
- SECOND
- MINUTE
- HOUR
- DAY
- WEEK

Note that, following SQL standard, the units are always singular.

Incremental backup

Supply the `LAST_BACKUP` option to only backup the changes between the last backup to the current snapshot.

```
-- timestamp (in current time zone)
BACKUP DATABASE `test` TO 'local:///mnt/backup/hist02'
    LAST_BACKUP = '2020-04-01 12:00:00';

-- timestamp oracle
BACKUP DATABASE `test` TO 'local:///mnt/backup/hist03'
    LAST_BACKUP = 415685305958400;
```

14.10.2.19.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.19.4 See also

- RESTORE
- SHOW BACKUPS

14.10.2.20 BATCH

The `BATCH` syntax splits a DML statement into multiple statements in TiDB for execution. This means that there are **no guarantees** of transactional atomicity and isolation. Therefore, it is a "non-transactional" statement.

Currently, only `DELETE` is supported in `BATCH`.

Based on a column, the `BATCH` syntax divides a DML statement into multiple ranges of scope for execution. In each range, a single SQL statement is executed.

For details about the usage and restrictions, see Non-transactional DML statements.

14.10.2.20.1 Synopsis

```
NonTransactionalDeleteStmt ::=
    'BATCH' ( 'ON' ColumnName )? 'LIMIT' NUM DryRunOptions? DeleteFromStmt

DryRunOptions ::=
    'DRY' 'RUN' 'QUERY'?
```

14.10.2.20.2 MySQL compatibility

The `BATCH` syntax is TiDB-specific and not compatible with MySQL.

14.10.2.20.3 See also

- Non-transactional DML statements

14.10.2.21 BEGIN

This statement starts a new transaction inside of TiDB. It is similar to the statements `START TRANSACTION` and `SET autocommit=0`.

In the absence of a `BEGIN` statement, every statement will by default autocommit in its own transaction. This behavior ensures MySQL compatibility.

14.10.2.21.1 Synopsis

```
BeginTransactionStmt ::=
    'BEGIN' ( 'PESSIMISTIC' | 'OPTIMISTIC' )?
|   'START' 'TRANSACTION' ( 'READ' ( 'WRITE' | 'ONLY' ( 'WITH' 'TIMESTAMP' '
    ↪ BOUND' TimestampBound )? ) | 'WITH' 'CONSISTENT' 'SNAPSHOT' )?
```

14.10.2.21.2 Examples

```
mysql> CREATE TABLE t1 (a int NOT NULL PRIMARY KEY);
Query OK, 0 rows affected (0.12 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

14.10.2.21.3 MySQL compatibility

TiDB supports the syntax extension of `BEGIN PESSIMISTIC` or `BEGIN OPTIMISTIC`. This enables you to override the default transactional model for your transaction.

14.10.2.21.4 See also

- COMMIT
- ROLLBACK
- START TRANSACTION
- TiDB optimistic transaction model
- TiDB pessimistic transaction mode

14.10.2.22 CHANGE COLUMN

The `ALTER TABLE.. CHANGE COLUMN` statement changes a column on an existing table. The change can include both renaming the column, and changing the data type to a compatible type.

Since v5.1.0, TiDB has supported changing the Reorg data type, including but not limited to:

- Changing `VARCHAR` to `BIGINT`
- Modifying the `DECIMAL` precision
- Compressing the length of `VARCHAR(10)` to `VARCHAR(5)`

14.10.2.22.1 Synopsis

```
AlterTableStmt
        ::= 'ALTER' 'IGNORE'? 'TABLE' TableName ChangeColumnSpec ( ','
           ↪ ChangeColumnSpec )*
```

```
ChangeColumnSpec
        ::= 'CHANGE' ColumnKeywordOpt 'IF EXISTS' ColumnName ColumnName
           ↪ ColumnType ColumnOption* ( 'FIRST' | 'AFTER' ColumnName )?

ColumnType
        ::= NumericType
          | StringType
          | DateAndTimeType
          | 'SERIAL'

ColumnOption
        ::= 'NOT'? 'NULL'
          | 'AUTO_INCREMENT'
          | 'PRIMARY'? 'KEY' ( 'CLUSTERED' | 'NONCLUSTERED' )?
          | 'UNIQUE' 'KEY'?
          | 'DEFAULT' ( NowSymOptionFraction | SignedLiteral |
             ↪ NextValueForSequence )
          | 'SERIAL' 'DEFAULT' 'VALUE'
          | 'ON' 'UPDATE' NowSymOptionFraction
          | 'COMMENT' stringLit
          | ( 'CONSTRAINT' Identifier? )? 'CHECK' '(' Expression ')' ( 'NOT
             ↪ '? ( 'ENFORCED' | 'NULL' ) )?
          | 'GENERATED' 'ALWAYS' 'AS' '(' Expression ')' ( 'VIRTUAL' | '
             ↪ STORED' )?
          | 'REFERENCES' TableName ( '(' IndexPartSpecificationList ')' )?
             ↪ Match? OnDeleteUpdateOpt
          | 'COLLATE' CollationName
          | 'COLUMN_FORMAT' ColumnFormat
          | 'STORAGE' StorageMedia
          | 'AUTO_RANDOM' ( '(' LengthNum ')' )?

ColumnName ::=
    Identifier ( '.' Identifier ( '.' Identifier )? )?
```

14.10.2.22.2 Examples

```
CREATE TABLE t1 (id int not null primary key AUTO_INCREMENT, col1 INT);
```

```
Query OK, 0 rows affected (0.11 sec)
```

```
INSERT INTO t1 (col1) VALUES (1),(2),(3),(4),(5);
```

```
Query OK, 5 rows affected (0.02 sec)
```

```
Records: 5 Duplicates: 0 Warnings: 0
```

```
ALTER TABLE t1 CHANGE col1 col2 INT;
```

```
Query OK, 0 rows affected (0.09 sec)
```

```
ALTER TABLE t1 CHANGE col2 col3 BIGINT, ALGORITHM=INSTANT;
```

```
Query OK, 0 rows affected (0.08 sec)
```

```
ALTER TABLE t1 CHANGE col3 col4 BIGINT, CHANGE id id2 INT NOT NULL;
```

```
ERROR 1105 (HY000): can't run multi schema change
```

```
CREATE TABLE t (a int primary key);
ALTER TABLE t CHANGE COLUMN a a VARCHAR(10);
```

```
ERROR 8200 (HY000): Unsupported modify column: column has primary key flag
```

```
CREATE TABLE t (c1 INT, c2 INT, c3 INT) partition by range columns(c1) (
    ↪ partition p0 values less than (10), partition p1 values less than (
    ↪ maxvalue));
ALTER TABLE t CHANGE COLUMN c1 c1 DATETIME;
```

```
ERROR 8200 (HY000): Unsupported modify column: table is partition table
```

```
CREATE TABLE t (a INT, b INT as (a+1));
ALTER TABLE t CHANGE COLUMN b b VARCHAR(10);
```

```
ERROR 8200 (HY000): Unsupported modify column: column is generated
```

```
CREATE TABLE t (a DECIMAL(13, 7));
ALTER TABLE t CHANGE COLUMN a a DATETIME;
```

```
ERROR 8200 (HY000): Unsupported modify column: change from original type
    ↪ decimal(13,7) to datetime is currently unsupported yet
```

14.10.2.22.3 MySQL compatibility

- Making multiple changes in a single `ALTER TABLE` statement is currently not supported.
- Changes of Reorg-Data types on primary key columns are not supported.
- Changes of column types on partitioned tables are not supported.
- Changes of column types on generated columns are not supported.
- Changes of some data types (for example, some TIME, Bit, Set, Enum, and JSON types) are not supported due to the compatibility issues of the `CAST` function's behavior between TiDB and MySQL.

14.10.2.22.4 See also

- CREATE TABLE
- SHOW CREATE TABLE
- ADD COLUMN
- DROP COLUMN
- MODIFY COLUMN

14.10.2.23 COMMIT

This statement commits a transaction inside of the TIDB server.

In the absence of a `BEGIN` or `START TRANSACTION` statement, the default behavior of TiDB is that every statement will be its own transaction and autocommit. This behavior ensures MySQL compatibility.

14.10.2.23.1 Synopsis

```
CommitStmt ::=
    'COMMIT' CompletionTypeWithinTransaction?

CompletionTypeWithinTransaction ::=
    'AND' ( 'CHAIN' ( 'NO' 'RELEASE' )? | 'NO' 'CHAIN' ( 'NO'? 'RELEASE' )?
       ↪ )
|   'NO'? 'RELEASE'
```

14.10.2.23.2 Examples

```
mysql> CREATE TABLE t1 (a int NOT NULL PRIMARY KEY);
Query OK, 0 rows affected (0.12 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

14.10.2.23.3 MySQL compatibility

- Currently, TiDB does not use Metadata Locking (MDL) to prevent DDL statements from modifying tables used by transactions. If the definition of a table has changed,

committing a transaction will result in an `Information schema is changed` error. In this event, the transaction is rolled back automatically.

- By default, TiDB 3.0.8 and later versions use Pessimistic Locking. When using Optimistic Locking, it is important to consider that a `COMMIT` statement might fail because rows have been modified by another transaction.
- When Optimistic Locking is enabled, `UNIQUE` and `PRIMARY KEY` constraint checks are deferred until statement commit. This results in additional situations where a a `COMMIT` statement might fail. This behavior can be changed by setting `tidb_constraint_check_in_place=TRUE`.
- TiDB parses but ignores the syntax `ROLLBACK AND [NO] RELEASE`. This functionality is used in MySQL to disconnect the client session immediately after committing the transaction. In TiDB, it is recommended to instead use the `mysql_close()` functionality of your client driver.
- TiDB parses but ignores the syntax `ROLLBACK AND [NO] CHAIN`. This functionality is used in MySQL to immediately start a new transaction with the same isolation level while the current transaction is being committed. In TiDB, it is recommended to instead start a new transaction.

14.10.2.23.4 See also

- START TRANSACTION
- ROLLBACK
- BEGIN
- Lazy checking of constraints

14.10.2.24 CHANGE DRAINER

The `CHANGE DRAINER` statement modifies the status information for Drainer in the cluster.

> **Tip:**
>
> Drainer's state is automatically reported to PD while running. Only when Drainer is under abnormal circumstances and its state is inconsistent with the state information stored in PD, you can use the `CHANGE DRAINER` statement to modify the state information stored in PD.

14.10.2.24.1 Examples

```
SHOW DRAINER STATUS;
```

```
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
|  NodeID |     Address    | State |   Max_Commit_Ts   |    Update_Time     |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
| drainer1 | 127.0.0.3:8249 | Online | 408553768673342532 | 2019-04-30
    ↪ 00:00:03 |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
| drainer2 | 127.0.0.4:8249 | Online | 408553768673345531 | 2019-05-01
    ↪ 00:00:04 |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
2 rows in set (0.00 sec)
```

It can be seen that drainer1's state has not been updated for more than a day, the Drainer is in an abnormal state, but the `State` remains `Online`. After using `CHANGE DRAINER`, the Drainer's `State` is changed to 'paused':

```
CHANGE DRAINER TO NODE_STATE ='paused' FOR NODE_ID 'drainer1';
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
SHOW DRAINER STATUS;
```

```
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
|  NodeID |     Address    | State |   Max_Commit_Ts   |    Update_Time     |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
| drainer1 | 127.0.0.3:8249 | Paused | 408553768673342532 | 2019-04-30
    ↪ 00:00:03 |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
| drainer2 | 127.0.0.4:8249 | Online | 408553768673345531 | 2019-05-01
    ↪ 00:00:04 |
```

```
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
2 rows in set (0.00 sec)
```

14.10.2.24.2 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.24.3 See also

- SHOW PUMP STATUS
- SHOW DRAINER STATUS
- CHANGE PUMP STATUS

14.10.2.25 CHANGE PUMP

The `CHANGE PUMP` statement modifies the status information for Pump in the cluster.

> **Tip:**
>
> Pump's state is automatically reported to PD while running. Only when Pump is under abnormal circumstances and its state is inconsistent with the state information stored in PD, you can use the `CHANGE PUMP` statement to modify the state information stored in PD.

14.10.2.25.1 Examples

```
SHOW PUMP STATUS;
```

```
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
| NodeID |    Address    | State |   Max_Commit_Ts  |    Update_Time     |
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
| pump1 | 127.0.0.1:8250 | Online | 408553768673342237 | 2019-04-30 00:00:01
    ↪  |
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
```

```
| pump2 | 127.0.0.2:8250 | Online | 408553768673342335 | 2019-05-01 00:00:02
    ↪  |
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
2 rows in set (0.00 sec)
```

It can be seen that pump1's state has not been updated for more than a day, the Pump is in an abnormal state, but the `State` remains `Online`. After using `CHANGE PUMP`, the Pump's `State` is changed to 'paused' :

```
CHANGE PUMP TO NODE_STATE ='paused' FOR NODE_ID 'pump1';
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
SHOW PUMP STATUS;
```

```
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
| NodeID |    Address    | State |   Max_Commit_Ts  |    Update_Time     |
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
| pump1 | 127.0.0.1:8250 | Paused | 408553768673342237 | 2019-04-30 00:00:01
    ↪  |
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
| pump2 | 127.0.0.2:8250 | Online | 408553768673342335 | 2019-05-01 00:00:02
    ↪  |
+--
    ↪ ------/----------------/--------/--------------------/--------------------/
    ↪
2 rows in set (0.00 sec)
```

14.10.2.25.2 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.25.3 See also

- SHOW PUMP STATUS
- SHOW DRAINER STATUS
- CHANGE DRAINER STATUS

14.10.2.26 CREATE [GLOBAL|SESSION] BINDING

This statement creates a new execution plan binding in TiDB. Binding can be used to inject a hint into a statement without requiring changes to the underlying query.

A `BINDING` can be on either a `GLOBAL` or `SESSION` basis. The default is `SESSION`.

The bound SQL statement is parameterized and stored in the system table. When a SQL query is processed, as long as the parameterized SQL statement and a bound one in the system table are consistent and the system variable `tidb_use_plan_baselines` is set to `ON` (default), the corresponding optimizer hint is available. If multiple execution plans are available, the optimizer chooses to bind the plan with the least cost.

14.10.2.26.1 Synopsis

```
CreateBindingStmt ::=
    'CREATE' GlobalScope 'BINDING' 'FOR' BindableStmt 'USING' BindableStmt

GlobalScope ::=
    ( 'GLOBAL' | 'SESSION' )?

BindableStmt ::=
    ( SelectStmt | UpdateStmt | InsertIntoStmt | ReplaceIntoStmt |
       ↪ DeleteStmt )
```

14.10.2.26.2 Examples

```
mysql> CREATE TABLE t1 (
    -> id INT NOT NULL PRIMARY KEY auto_increment,
    -> b INT NOT NULL,
    -> pad VARBINARY(255),
    -> INDEX(b)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM dual;
Query OK, 1 row affected (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 1 row affected (0.00 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 8 rows affected (0.00 sec)
Records: 8 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 1000 rows affected (0.04 sec)
Records: 1000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (1.74 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (2.15 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (2.64 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> SELECT SLEEP(1);
+----------+
| SLEEP(1) |
+----------+
|        0 |
+----------+
1 row in set (1.00 sec)

mysql> ANALYZE TABLE t1;
Query OK, 0 rows affected (1.33 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
    ↪ ----------------------------+---------+---------+-----------+-----------------
    ↪
| id                          | estRows | actRows | task      | access object
    ↪      | execution info
    ↪                                                          | operator info
    ↪                  | memory       | disk |
```

```
+--
   ↪ ---------------------------+---------+---------+----------+-----------------
   ↪
| IndexLookUp_10             | 583.00 | 297    | root     |
   ↪                    | time:10.545072ms, loops:2, rpc num: 1, rpc time
   ↪ :398.359µs, proc keys:297 |                       | 109.1484375 KB | N/
   ↪ A |
|  ─IndexRangeScan_8(Build)  | 583.00 | 297    | cop[tikv] | table:t1, index
   ↪ :b(b) | time:0s, loops:4                                              |
   ↪  range:[123,123], keep order:false | N/A      | N/A |
|  ─TableRowIDScan_9(Probe)  | 583.00 | 297    | cop[tikv] | table:t1
   ↪         | time:12ms, loops:4
   ↪                                                | keep order:false
   ↪              | N/A          | N/A |
+--
   ↪ ---------------------------+---------+---------+----------+-----------------
   ↪
3 rows in set (0.02 sec)

mysql> CREATE SESSION BINDING FOR
    ->  SELECT * FROM t1 WHERE b = 123
    -> USING
    ->  SELECT * FROM t1 IGNORE INDEX (b) WHERE b = 123;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
   ↪ ----------------------+----------+---------+----------+--------------+------
   ↪
| id                   | estRows  | actRows | task    | access object |
   ↪ execution info                                                    |
   ↪ operator info    | memory      | disk |
+--
   ↪ ----------------------+----------+---------+----------+--------------+------
   ↪
| TableReader_7        | 583.00   | 297    | root     |              | time
   ↪ :222.32506ms, loops:2, rpc num: 1, rpc time:222.078952ms, proc keys
   ↪ :301010 | data:Selection_6 | 88.6640625 KB | N/A |
|  ─Selection_6        | 583.00   | 297    | cop[tikv] |              | time
   ↪ :224ms, loops:298                                                   | eq(
   ↪ test.t1.b, 123) | N/A       | N/A |
|    ─TableFullScan_5   | 301010.00 | 301010 | cop[tikv] | table:t1 | time
   ↪ :220ms, loops:298                                                    |
   ↪ keep order:false | N/A        | N/A |
+--
```

```
    ↪ ----------------------+----------+--------+----------+--------------+------
    ↪
3 rows in set (0.22 sec)

mysql> SHOW SESSION BINDINGS\G
*************************** 1. row ***************************
Original_sql: select * from t1 where b = ?
    Bind_sql: SELECT * FROM t1 IGNORE INDEX (b) WHERE b = 123
  Default_db: test
      Status: using
 Create_time: 2020-05-22 14:38:03.456
 Update_time: 2020-05-22 14:38:03.456
     Charset: utf8mb4
   Collation: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)

mysql> DROP SESSION BINDING FOR SELECT * FROM t1 WHERE b = 123;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
    ↪ ----------------------------+---------+---------+----------+----------------
    ↪
| id                          | estRows | actRows | task   | access object
    ↪      | execution info
    ↪ | operator info                  | memory       | disk |
+--
    ↪ ----------------------------+---------+---------+----------+----------------
    ↪
| IndexLookUp_10              | 583.00 | 297    | root     |
    ↪                     | time:5.31206ms, loops:2, rpc num: 1, rpc time
    ↪ :665.927µs, proc keys:297 |                       | 109.1484375 KB | N
    ↪ /A |
|  ─IndexRangeScan_8(Build)  | 583.00 | 297    | cop[tikv] | table:t1, index
    ↪ :b(b) | time:0s, loops:4                                              |
    ↪ range:[123,123], keep order:false | N/A      | N/A |
|  ─TableRowIDScan_9(Probe)  | 583.00 | 297    | cop[tikv] | table:t1
    ↪         | time:0s, loops:4
    ↪                                              | keep order:false
    ↪              | N/A          | N/A |
+--
    ↪ ----------------------------+---------+---------+----------+----------------
    ↪
3 rows in set (0.01 sec)
```

14.10.2.26.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.26.4 See also

- DROP [GLOBAL|SESSION] BINDING
- SHOW [GLOBAL|SESSION] BINDINGS
- ANALYZE TABLE
- Optimizer Hints
- SQL Plan Management

14.10.2.27 CREATE DATABASE

This statement creates a new database in TiDB. The MySQL terminology for 'database' most closely maps to a schema in the SQL standard.

14.10.2.27.1 Synopsis

```
CreateDatabaseStmt ::=
    'CREATE' 'DATABASE' IfNotExists DBName DatabaseOptionListOpt

IfNotExists ::=
    ( 'IF' 'NOT' 'EXISTS' )?

DBName ::=
    Identifier

DatabaseOptionListOpt ::=
    DatabaseOptionList?

DatabaseOptionList ::=
    DatabaseOption ( ','? DatabaseOption )*

DatabaseOption ::=
    DefaultKwdOpt ( CharsetKw '='? CharsetName | 'COLLATE' '='?
        ↪ CollationName | 'ENCRYPTION' '='? EncryptionOpt )
|   DefaultKwdOpt PlacementPolicyOption

PlacementPolicyOption ::=
    "PLACEMENT" "POLICY" EqOpt PolicyName
|   "PLACEMENT" "POLICY" (EqOpt | "SET") "DEFAULT"
```

14.10.2.27.2 Syntax

The `CREATE DATABASE` statement is used to create a database, and to specify the default properties of the database, such as the default character set and collation. `CREATE SCHEMA` is a synonym for `CREATE DATABASE`.

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

If you create an existing database and does not specify `IF NOT EXISTS`, an error is displayed.

The `create_specification` option is used to specify the specific `CHARACTER SET` and `COLLATE` in the database. Currently, TiDB only supports some of the character sets and collations. For details, see Character Set and Collation Support.

14.10.2.27.3 Examples

```
mysql> CREATE DATABASE mynewdatabase;
Query OK, 0 rows affected (0.09 sec)

mysql> USE mynewdatabase;
Database changed
mysql> CREATE TABLE t1 (a int);
Query OK, 0 rows affected (0.11 sec)

mysql> SHOW TABLES;
+------------------------+
| Tables_in_mynewdatabase |
+------------------------+
| t1                     |
+------------------------+
1 row in set (0.00 sec)
```

14.10.2.27.4 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.27.5 See also

- USE

- ALTER DATABASE
- DROP DATABASE
- SHOW DATABASES

14.10.2.28 CREATE INDEX

This statement adds a new index to an existing table. It is an alternative syntax to `ALTER TABLE .. ADD INDEX`, and included for MySQL compatibility.

14.10.2.28.1 Synopsis

```
CreateIndexStmt ::=
    'CREATE' IndexKeyTypeOpt 'INDEX' IfNotExists Identifier IndexTypeOpt 'ON
       ↪ ' TableName '(' IndexPartSpecificationList ')' IndexOptionList
       ↪ IndexLockAndAlgorithmOpt

IndexKeyTypeOpt ::=
    ( 'UNIQUE' | 'SPATIAL' | 'FULLTEXT' )?

IfNotExists ::=
    ( 'IF' 'NOT' 'EXISTS' )?

IndexTypeOpt ::=
    IndexType?

IndexPartSpecificationList ::=
    IndexPartSpecification ( ',' IndexPartSpecification )*

IndexOptionList ::=
    IndexOption*

IndexLockAndAlgorithmOpt ::=
    ( LockClause AlgorithmClause? | AlgorithmClause LockClause? )?

IndexType ::=
    ( 'USING' | 'TYPE' ) IndexTypeName

IndexPartSpecification ::=
    ( ColumnName OptFieldLen | '(' Expression ')' ) Order

IndexOption ::=
    'KEY_BLOCK_SIZE' '='? LengthNum
|   IndexType
|   'WITH' 'PARSER' Identifier
|   'COMMENT' stringLit
```

```
|   IndexInvisible

IndexTypeName ::=
    'BTREE'
|   'HASH'
|   'RTREE'

ColumnName ::=
    Identifier ( '.' Identifier ( '.' Identifier )? )?

OptFieldLen ::=
    FieldLen?

IndexNameList ::=
    ( Identifier | 'PRIMARY' )? ( ',' ( Identifier | 'PRIMARY' ) )*

KeyOrIndex ::=
    'Key' | 'Index'
```

14.10.2.28.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
    ↪ ----------------------+----------+-----------+---------------+----------------
    ↪
| id                    | estRows | task      | access object | operator info
    ↪            |
+--
    ↪ ----------------------+----------+-----------+---------------+----------------
    ↪
| TableReader_7         | 10.00   | root      |               | data:Selection_6
    ↪           |
|  ─Selection_6         | 10.00   | cop[tikv] |               | eq(test.t1.c1,
    ↪ 3)          |
|    ─TableFullScan_5   | 10000.00 | cop[tikv] | table:t1 | keep order:false
    ↪ , stats:pseudo |
```

```
+--
    ↪ ---------------------+---------+----------+--------------+-----------------
    ↪
3 rows in set (0.00 sec)

mysql> CREATE INDEX c1 ON t1 (c1);
Query OK, 0 rows affected (0.30 sec)

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
    ↪ ---------------------+---------+----------+-----------------------+---------
    ↪
| id                    | estRows | task     | access object         | operator
    ↪ info                            |
+--
    ↪ ---------------------+---------+----------+-----------------------+---------
    ↪
| IndexReader_6         | 10.00   | root     |                       | index:
    ↪ IndexRangeScan_5                |
|  ─IndexRangeScan_5    | 10.00   | cop[tikv] | table:t1, index:c1(c1) | range
    ↪ :[3,3], keep order:false, stats:pseudo |
+--
    ↪ ---------------------+---------+----------+-----------------------+---------
    ↪
2 rows in set (0.00 sec)

mysql> ALTER TABLE t1 DROP INDEX c1;
Query OK, 0 rows affected (0.30 sec)

mysql> CREATE UNIQUE INDEX c1 ON t1 (c1);
Query OK, 0 rows affected (0.31 sec)
```

14.10.2.28.3 Expression index

In some scenarios, the filtering condition of a query is based on a certain expression. In these scenarios, the query performance is relatively poor because ordinary indexes cannot take effect, the query can only be executed by scanning the entire table. The expression index is a type of special index that can be created on an expression. Once an expression index is created, TiDB can use the index for the expression-based query, which significantly improves the query performance.

For example, if you want to create an index based on `lower(col1)`, execute the following SQL statement:

```
CREATE INDEX idx1 ON t1 ((lower(col1)));
```

Or you can execute the following equivalent statement:

```
ALTER TABLE t1 ADD INDEX idx1((lower(col1)));
```

You can also specify the expression index when you create the table:

```
CREATE TABLE t1(col1 char(10), col2 char(10), index((lower(col1))));
```

> **Note:**
>
> The expression in an expression index must be surrounded by `(` and `)`. Otherwise, a syntax error is reported.

You can drop an expression index in the same way as dropping an ordinary index:

```
DROP INDEX idx1 ON t1;
```

Expression index involves various kinds of expressions. To ensure correctness, only some fully tested functions are allowed for creating an expression index. This means that only these functions are allowed in expressions in a production environment. You can get these functions by querying `tidb_allow_function_for_expression_index` variable.

```
mysql> select @@tidb_allow_function_for_expression_index;
+-------------------------------------------+
| @@tidb_allow_function_for_expression_index |
+-------------------------------------------+
| lower, md5, reverse, upper, vitess_hash |
+-------------------------------------------+
1 row in set (0.00 sec)
```

For the functions that are not included in the returned result above, those functions are not fully tested and not recommended for a production environment, which can be seen as experimental. Other expressions such as operators, `cast`, and `case when` are also seen as experimental and not recommended for production.

If you still want to use those expressions, you can make the following configuration in the TiDB configuration file:

```
allow-expression-index = true
```

> **Note:**
>
> An expression index cannot be created on a primary key.
>
> The expression in an expression index cannot contain the following content:

- Volatile functions, such as `rand()` and `now()`.
- System variables and user variables.
- Subqueries.
- `AUTO_INCREMENT` column. You can remove this restriction by setting the value of `tidb_enable_auto_increment_in_generated` (system variable) to `true`.
- Window functions.
- ROW functions, such as `create table t (j json, key k (((j,j))) ↪ );`.
- Aggregate functions.

An expression index implicitly takes up a name (for example, `_V$_{ ↪ index_name}_{index_offset}`). If you try to create a new expression index with the name that a column has already had, an error occurs. In addition, if you add a new column with the same name, an error also occurs.

Make sure that the number of function parameters in the expression of an expression index is correct.

When the expression of an index contains a string-related function, affected by the returned type and the length, creating the expression index might fail. In this situation, you can use the `cast()` function to explicitly specify the returned type and the length. For example, to create an expression index based on the `repeat(a, 3)` expression, you need to modify this expression to `cast(repeat(a, 3)as char(20))`.

When the expression in a query statement matches the expression in an expression index, the optimizer can choose the expression index for the query. In some cases, the optimizer might not choose an expression index depending on statistics. In this situation, you can force the optimizer to select an expression index by using optimizer hints.

In the following examples, suppose that you create the expression index `idx` on the expression `lower(col1)`:

If the results of the query statement are the same expressions, the expression index applies. Take the following statement as an example:

```
SELECT lower(col1) FROM t;
```

If the same expression is included in the filtering conditions, the expression index applies. Take the following statements as an example:

```
SELECT * FROM t WHERE lower(col1) = "a";
SELECT * FROM t WHERE lower(col1) > "a";
SELECT * FROM t WHERE lower(col1) BETWEEN "a" AND "b";
SELECT * FROM t WHERE lower(col1) in ("a", "b");
```

```
SELECT * FROM t WHERE lower(col1) > "a" AND lower(col1) < "b";
SELECT * FROM t WHERE lower(col1) > "b" OR lower(col1) < "a";
```

When the queries are sorted by the same expression, the expression index applies. Take the following statement as an example:

```
SELECT * FROM t ORDER BY lower(col1);
```

If the same expression is included in the aggregate (`GROUP BY`) functions, the expression index applies. Take the following statements as an example:

```
SELECT max(lower(col1)) FROM t;
SELECT min(col1) FROM t GROUP BY lower(col1);
```

To see the expression corresponding to the expression index, execute `show index` ↪ , or check the system tables `information_schema.tidb_indexes` and the table `information_schema.STATISTICS`. The `Expression` column in the output indicates the corresponded expression. For the non-expression indexes, the column shows `NULL`.

The cost of maintaining an expression index is higher than that of maintaining other indexes, because the value of the expression needs to be calculated whenever a row is inserted or updated. The value of the expression is already stored in the index, so this value does not require recalculation when the optimizer selects the expression index.

Therefore, when the query performance outweighs the insert and update performance, you can consider indexing the expressions.

Expression indexes have the same syntax and limitations as in MySQL. They are implemented by creating indexes on generated virtual columns that are invisible, so the supported expressions inherit all limitations of virtual generated columns.

14.10.2.28.4 Invisible index

Invisible indexes are indexes that are ignored by the query optimizer:

```
CREATE TABLE t1 (c1 INT, c2 INT, UNIQUE(c2));
CREATE UNIQUE INDEX c1 ON t1 (c1) INVISIBLE;
```

For details, see `ALTER INDEX`.

14.10.2.28.5 Associated system variables

The system variables associated with the `CREATE INDEX` statement are `tidb_ddl_reorg_worker_cnt` ↪ , `tidb_ddl_reorg_batch_size`, `tidb_enable_auto_increment_in_generated`, and `tidb_ddl_reorg_priority`. Refer to system variables for details.

14.10.2.28.6 MySQL compatibility

- TiDB supports parsing the `FULLTEXT` and `SPATIAL` syntax but does not support using the `FULLTEXT`, `HASH`, and `SPATIAL` indexes.
- Descending indexes are not supported (similar to MySQL 5.7).
- Adding the primary key of the `CLUSTERED` type to a table is not supported. For more details about the primary key of the `CLUSTERED` type, refer to clustered index.
- Expression indexes are incompatible with views. When a query is executed using a view, the expression index cannot be used at the same time.
- Expression indexes have compatibility issues with bindings. When the expression of an expression index has a constant, the binding created for the corresponding query expands its scope. For example, suppose that the expression in the expression index is `a+1`, and the corresponding query condition is `a+1 > 2`. In this case, the created binding is `a+? > ?`, which means that the query with the condition such as `a+2 > 2` is also forced to use the expression index and results in a poor execution plan. In addition, this also affects the baseline capturing and baseline evolution in SQL Plan Management (SPM).

14.10.2.28.7 See also

- Index Selection
- Wrong Index Solution
- ADD INDEX
- DROP INDEX
- RENAME INDEX
- ALTER INDEX
- ADD COLUMN
- CREATE TABLE
- EXPLAIN

14.10.2.29 CREATE PLACEMENT POLICY

`CREATE PLACEMENT POLICY` is used to create a named placement policy that can later be assigned to tables, partitions, or database schemas.

14.10.2.29.1 Synopsis

```
CreatePolicyStmt ::=
    "CREATE" "PLACEMENT" "POLICY" IfNotExists PolicyName PlacementOptionList

PolicyName ::=
    Identifier

PlacementOptionList ::=
```

```
    PlacementOption
|   PlacementOptionList PlacementOption
|   PlacementOptionList ',' PlacementOption

PlacementOption ::=
    CommonPlacementOption
|   SugarPlacementOption
|   AdvancedPlacementOption

CommonPlacementOption ::=
    "FOLLOWERS" EqOpt LengthNum

SugarPlacementOption ::=
    "PRIMARY_REGION" EqOpt stringLit
|   "REGIONS" EqOpt stringLit
|   "SCHEDULE" EqOpt stringLit

AdvancedPlacementOption ::=
    "LEARNERS" EqOpt LengthNum
|   "CONSTRAINTS" EqOpt stringLit
|   "LEADER_CONSTRAINTS" EqOpt stringLit
|   "FOLLOWER_CONSTRAINTS" EqOpt stringLit
|   "LEARNER_CONSTRAINTS" EqOpt stringLit
```

14.10.2.29.2 Examples

> **Note:**
>
> To know which regions are available in your cluster, see SHOW PLACEMENT LABELS.
>
> If you do not see any available regions, your TiKV installation might not have labels set correctly.

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us
    -west-1" FOLLOWERS=4;
CREATE TABLE t1 (a INT) PLACEMENT POLICY=p1;
SHOW CREATE PLACEMENT POLICY p1;
```

```
Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.10 sec)
```

```
+--------+-------------------------------------------------------------------------------
    ↪
| Policy | Create Policy
    ↪
    ↪ |
+--------+-------------------------------------------------------------------------------
    ↪
| p1     | CREATE PLACEMENT POLICY `p1` PRIMARY_REGION="us-east-1" REGIONS="
    ↪ us-east-1,us-west-1" FOLLOWERS=4 |
+--------+-------------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.2.29.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.29.4 See also

- Placement Rules in SQL
- SHOW PLACEMENT
- ALTER PLACEMENT POLICY
- DROP PLACEMENT POLICY

14.10.2.30 CREATE ROLE

This statement creates a new role, which can be assigned to users as part of role-based access control.

14.10.2.30.1 Synopsis

```
CreateRoleStmt ::=
    'CREATE' 'ROLE' IfNotExists RoleSpec (',' RoleSpec)*

IfNotExists ::=
    ('IF' 'NOT' 'EXISTS')?

RoleSpec ::=
    Rolename
```

14.10.2.30.2 Examples

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

Create a new role `analyticsteam` and a new user `jennifer`:

```
CREATE ROLE analyticsteam;
Query OK, 0 rows affected (0.02 sec)

GRANT SELECT ON test.* TO analyticsteam;
Query OK, 0 rows affected (0.02 sec)

CREATE USER jennifer;
Query OK, 0 rows affected (0.01 sec)

GRANT analyticsteam TO jennifer;
Query OK, 0 rows affected (0.01 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

Note that by default `jennifer` needs to execute `SET ROLE analyticsteam` in order to be able to use the privileges associated with the `analyticsteam` role:

```
SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-------------------------------------------+
2 rows in set (0.00 sec)

SHOW TABLES in test;
ERROR 1044 (42000): Access denied for user 'jennifer'@'%' to database 'test'
SET ROLE analyticsteam;
Query OK, 0 rows affected (0.00 sec)

SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT SELECT ON test.* TO 'jennifer'@'%'  |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
```

```
+-----------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

The statement `SET DEFAULT ROLE` can be used to associate the role `analyticsteam` to `jennifer`:

```
SET DEFAULT ROLE analyticsteam TO jennifer;
Query OK, 0 rows affected (0.02 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

After this, the user `jennifer` has the privileges associated with the role `analyticsteam` and `jennifer` does not have to execute the statement `SET ROLE`:

```
SHOW GRANTS;
+-----------------------------------------+
| Grants for User                         |
+-----------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'    |
| GRANT SELECT ON test.* TO 'jennifer'@'%' |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-----------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

14.10.2.30.3 MySQL compatibility

This statement is understood to be fully compatible with roles, which are a feature of MySQL 8.0. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.30.4 See also

- DROP ROLE
- GRANT <role>
- REVOKE <role>
- SET ROLE
- SET DEFAULT ROLE

- Role-Based Access Control

14.10.2.31 CREATE SEQUENCE

The `CREATE SEQUENCE` statement creates sequence objects in TiDB. The sequence is a database object that is on a par with the table and the `View` object. The sequence is used to generate serialized IDs in a customized way.

14.10.2.31.1 Synopsis

```
CreateSequenceStmt ::=
    'CREATE' 'SEQUENCE' IfNotExists TableName CreateSequenceOptionListOpt
       ↪ CreateTableOptionListOpt

IfNotExists ::=
    ('IF' 'NOT' 'EXISTS')?

TableName ::=
    Identifier ('.' Identifier)?

CreateSequenceOptionListOpt ::=
    SequenceOption*

SequenceOptionList ::=
    SequenceOption

SequenceOption ::=
    ( 'INCREMENT' ( '='? | 'BY' ) | 'START' ( '='? | 'WITH' ) | ( 'MINVALUE'
       ↪  | 'MAXVALUE' | 'CACHE' ) '='? ) SignedNum
|   'NOMINVALUE'
|   'NO' ( 'MINVALUE' | 'MAXVALUE' | 'CACHE' | 'CYCLE' )
|   'NOMAXVALUE'
```

```
|   'NOCACHE'
|   'CYCLE'
|   'NOCYCLE'
```

14.10.2.31.2 Syntax

```
CREATE [TEMPORARY] SEQUENCE [IF NOT EXISTS] sequence_name
    [ INCREMENT [ BY | = ] increment ]
    [ MINVALUE [=] minvalue | NO MINVALUE | NOMINVALUE ]
    [ MAXVALUE [=] maxvalue | NO MAXVALUE | NOMAXVALUE ]
    [ START [ WITH | = ] start ]
    [ CACHE [=] cache | NOCACHE | NO CACHE]
    [ CYCLE | NOCYCLE | NO CYCLE]
    [table_options]
```

14.10.2.31.3 Parameters

Parameters	Default value	Description
`TEMPORARY`	`false`	TiDB currently does not support the `TEMPORARY` option and provides only syntax compatibility for it.

Parameters	Default values	Description
`INCREMENT`	`1`	Specifies the increment of a sequence. Its positive or negative values can control the growth direction of the sequence.

Parameters	Default value	Description
`MINVALUE` or `-9223372036854775807`	1	Specifies the minimum value of a sequence. When `INCREMENT` > 0, the default value is 1. When `INCREMENT` < 0, the default value is `-9223372036854775807` .

Parameters	Default value	Description
`MAXVALUE`	`9223372036854775806` or `-1`	Specifies the maximum value of a sequence. When `INCREMENT` > `0`, the default value is `9223372036854775806`. When `INCREMENT` < `0`, the default value is `-1`.

Parameters	Default value	Description
`START`	`MINVALUE` or `MAXVALUE`	Specifies the initial value of a sequence. When `INCREMENT` > `0`, the default value is `MINVALUE`. When `INCREMENT` < `0`, the default value is `MAXVALUE`.
`CACHE`	`1000`	Specifies the local cache size of a sequence in TiDB.

Parameters	Default value	Description
`CYCLE` ↪	`NO` ↪ `CYCLE`	Specifies whether a sequence restarts from the minimum value (or maximum for the descending sequence). When `INCREMENT` ↪ `> 0`, the default value is `MINVALUE` ↪ . When `INCREMENT` ↪ `< 0`, the default value is `MAXVALUE` ↪ .

Parameters	Default values	Description

14.10.2.31.4 `SEQUENCE` function

You can control a sequence through the following expression functions:

- `NEXTVAL` or `NEXT VALUE FOR`

  Essentially, both are the `nextval()` function that gets the next valid value of a sequence object. The parameter of the `nextval()` function is the `identifier` of the sequence.

- `LASTVAL`

  This function gets the last used value of this session. If the value does not exist, `NULL` is used. The parameter of this function is the `identifier` of the sequence.

- `SETVAL`

  This function sets the progression of the current value for a sequence. The first parameter of this function is the `identifier` of the sequence; the second parameter is `num`.

> **Note:**
>
> In the implementation of a sequence in TiDB, the `SETVAL` function cannot change the initial progression or cycle progression of this sequence. This function only returns the next valid value based on this progression.

14.10.2.31.5 Examples

- Create a sequence object with the default parameter:

```
CREATE SEQUENCE seq;
```

```
Query OK, 0 rows affected (0.06 sec)
```

- Use the `nextval()` function to get the next value of the sequence object:

```
SELECT nextval(seq);
```

```
+--------------+
| nextval(seq) |
+--------------+
|            1 |
+--------------+
1 row in set (0.02 sec)
```

- Use the `lastval()` function to get the value generated by the last call to a sequence object in this session:

```
SELECT lastval(seq);
```

```
+--------------+
| lastval(seq) |
+--------------+
|            1 |
+--------------+
1 row in set (0.02 sec)
```

- Use the `setval()` function to set the current value (or the current position) of the sequence object:

```
SELECT setval(seq, 10);
```

```
+-----------------+
| setval(seq, 10) |
+-----------------+
|              10 |
+-----------------+
1 row in set (0.01 sec)
```

- You can also use the `next value for` syntax to get the next value of the sequence:

```
SELECT next value for seq;
```

```
+--------------------+
| next value for seq |
+--------------------+
|                 11 |
+--------------------+
1 row in set (0.00 sec)
```

- Create a sequence object with default custom parameters:

```
CREATE SEQUENCE seq2 start 3 increment 2 minvalue 1 maxvalue 10 cache
    ↪ 3;
```

```
Query OK, 0 rows affected (0.01 sec)
```

- When the sequence object has not been used in this session, the `lastval()` function returns a `NULL` value.

```
SELECT lastval(seq2);
```

```
+---------------+
| lastval(seq2) |
+---------------+
|          NULL |
+---------------+
1 row in set (0.01 sec)
```

- The first valid value of the `nextval()` function for the sequence object is the value of `START` parameter.

```
SELECT nextval(seq2);
```

```
+---------------+
| nextval(seq2) |
+---------------+
|             3 |
+---------------+
1 row in set (0.00 sec)
```

- Although the `setval()` function can change the current value of the sequence object, it cannot change the arithmetic progression rule for the next value.

```
SELECT setval(seq2, 6);
```

```
+-----------------+
| setval(seq2, 6) |
+-----------------+
|               6 |
+-----------------+
1 row in set (0.00 sec)
```

- When you use `nextval()` to get the next value, the next value will follow the arithmetic progression rule defined by the sequence.

```
SELECT next value for seq2;
```

```
+---------------------+
| next value for seq2 |
+---------------------+
|                   7 |
+---------------------+
1 row in set (0.00 sec)
```

- You can use the next value of the sequence as the default value for the column, as in the following example.

```
CREATE table t(a int default next value for seq2);
```

```
Query OK, 0 rows affected (0.02 sec)
```

- In the following example, the value is not specified, so the default value of `seq2` is used.

```
INSERT into t values();
```

```
Query OK, 1 row affected (0.00 sec)
```

```
SELECT * from t;
```

```
+------+
| a    |
+------+
|    9 |
+------+
1 row in set (0.00 sec)
```

- In the following example, the value is not specified, so the default value of `seq2` is used. But the next value of `seq2` is not within the range defined in the above example (`CREATE SEQUENCE seq2 start 3 increment 2 minvalue 1 maxvalue 10 cache 3;`), so an error is returned.

```
INSERT into t values();
```

```
ERROR 4135 (HY000): Sequence 'test.seq2' has run out
```

14.10.2.31.6 MySQL compatibility

This statement is a TiDB extension. The implementation is modeled on sequences available in MariaDB.

Except for the `SETVAL` function, all other functions have the same *progressions* as MariaDB. Here "progression" means that the numbers in a sequence follow a certain arithmetic

progression rule defined by the sequence. Although you can use `SETVAL` to set the current value of a sequence, the subsequent values of the sequence still follow the original progression rule.

For example:

```
1, 3, 5, ...          // The sequence starts from 1 and increments by 2.
select setval(seq, 6) // Sets the current value of a sequence to 6.
7, 9, 11, ...         // Subsequent values still follow the progression rule.
```

In the `CYCLE` mode, the initial value of a sequence in the first round is the value of the `START` parameter, and the initial value in the subsequent rounds is the value of `MinValue` (`INCREMENT` > 0) or `MaxValue` (`INCREMENT` < 0).

14.10.2.31.7 See also

- DROP SEQUENCE
- SHOW CREATE SEQUENCE

14.10.2.32 CREATE TABLE LIKE

This statement copies the definition of an existing table, without copying any data.

14.10.2.32.1 Synopsis

```
CreateTableLikeStmt ::=
    'CREATE' OptTemporary 'TABLE' IfNotExists TableName
       ↪ LikeTableWithOrWithoutParen OnCommitOpt

OptTemporary ::=
    ( 'TEMPORARY' | ('GLOBAL' 'TEMPORARY') )?

LikeTableWithOrWithoutParen ::=
    'LIKE' TableName
|   '(' 'LIKE' TableName ')'

OnCommitOpt ::=
    ('ON' 'COMMIT' 'DELETE' 'ROWS')?
```

14.10.2.32.2 Examples

```
mysql> CREATE TABLE t1 (a INT NOT NULL);
Query OK, 0 rows affected (0.13 sec)

mysql> INSERT INTO t1 VALUES (1),(2),(3),(4),(5);
```

```
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+---+
5 rows in set (0.00 sec)

mysql> CREATE TABLE t2 LIKE t1;
Query OK, 0 rows affected (0.10 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

14.10.2.32.3 Pre-split region

If the table to be copied is defined with the `PRE_SPLIT_REGIONS` attribute, the table created using the `CREATE TABLE LIKE` statement inherits this attribute, and the Region on the new table will be split. For details of `PRE_SPLIT_REGIONS`, see `CREATE TABLE` Statement.

14.10.2.32.4 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.32.5 See also

- CREATE TABLE
- SHOW CREATE TABLE

14.10.2.33 CREATE TABLE

This statement creates a new table in the currently selected database. It behaves similarly to the `CREATE TABLE` statement in MySQL.

14.10.2.33.1 Synopsis

```
CreateTableStmt ::=
    'CREATE' OptTemporary 'TABLE' IfNotExists TableName (
       ↪ TableElementListOpt CreateTableOptionListOpt PartitionOpt
       ↪ DuplicateOpt AsOpt CreateTableSelectOpt |
       ↪ LikeTableWithOrWithoutParen ) OnCommitOpt

OptTemporary ::=
    ( 'TEMPORARY' | ('GLOBAL' 'TEMPORARY') )?

IfNotExists ::=
    ('IF' 'NOT' 'EXISTS')?

TableName ::=
    Identifier ('.' Identifier)?

TableElementListOpt ::=
    ( '(' TableElementList ')' )?

TableElementList ::=
    TableElement ( ',' TableElement )*

TableElement ::=
    ColumnDef
|   Constraint

ColumnDef ::=
    ColumnName ( Type | 'SERIAL' ) ColumnOptionListOpt

ColumnOptionListOpt ::=
    ColumnOption*

ColumnOptionList ::=
    ColumnOption*

ColumnOption ::=
    'NOT'? 'NULL'
|   'AUTO_INCREMENT'
|   PrimaryOpt 'KEY'
|   'UNIQUE' 'KEY'?
|   'DEFAULT' DefaultValueExpr
|   'SERIAL' 'DEFAULT' 'VALUE'
|   'ON' 'UPDATE' NowSymOptionFraction
|   'COMMENT' stringLit
```

```
|   ConstraintKeywordOpt 'CHECK' '(' Expression ')'
    ↪ EnforcedOrNotOrNotNullOpt
|   GeneratedAlways 'AS' '(' Expression ')' VirtualOrStored
|   ReferDef
|   'COLLATE' CollationName
|   'COLUMN_FORMAT' ColumnFormat
|   'STORAGE' StorageMedia
|   'AUTO_RANDOM' OptFieldLen

CreateTableOptionListOpt ::=
    TableOptionList?

PartitionOpt ::=
    ( 'PARTITION' 'BY' PartitionMethod PartitionNumOpt SubPartitionOpt
       ↪ PartitionDefinitionListOpt )?

DuplicateOpt ::=
    ( 'IGNORE' | 'REPLACE' )?

TableOptionList ::=
    TableOption ( ','? TableOption )*

TableOption ::=
    PartDefOption
|   DefaultKwdOpt ( CharsetKw EqOpt CharsetName | 'COLLATE' EqOpt
    ↪ CollationName )
|   ( 'AUTO_INCREMENT' | 'AUTO_ID_CACHE' | 'AUTO_RANDOM_BASE' | '
    ↪ AVG_ROW_LENGTH' | 'CHECKSUM' | 'TABLE_CHECKSUM' | 'KEY_BLOCK_SIZE' |
    ↪ 'DELAY_KEY_WRITE' | 'SHARD_ROW_ID_BITS' | 'PRE_SPLIT_REGIONS' ) EqOpt
    ↪  LengthNum
|   ( 'CONNECTION' | 'PASSWORD' | 'COMPRESSION' ) EqOpt stringLit
|   RowFormat
|   ( 'STATS_PERSISTENT' | 'PACK_KEYS' ) EqOpt StatsPersistentVal
|   ( 'STATS_AUTO_RECALC' | 'STATS_SAMPLE_PAGES' ) EqOpt ( LengthNum | '
    ↪ DEFAULT' )
|   'STORAGE' ( 'MEMORY' | 'DISK' )
|   'SECONDARY_ENGINE' EqOpt ( 'NULL' | StringName )
|   'UNION' EqOpt '(' TableNameListOpt ')'
|   'ENCRYPTION' EqOpt EncryptionOpt
|   PlacementPolicyOption

OnCommitOpt ::=
    ('ON' 'COMMIT' 'DELETE' 'ROWS')?

PlacementPolicyOption ::=
```

```
    "PLACEMENT" "POLICY" EqOpt PolicyName
|   "PLACEMENT" "POLICY" (EqOpt | "SET") "DEFAULT"
```

The following *table_options* are supported. Other options such as `AVG_ROW_LENGTH` ↪ , `CHECKSUM`, `COMPRESSION`, `CONNECTION`, `DELAY_KEY_WRITE`, `ENGINE`, `KEY_BLOCK_SIZE`, `MAX_ROWS`, `MIN_ROWS`, `ROW_FORMAT` and `STATS_PERSISTENT` are parsed but ignored.

Options	Description	Example
`AUTO_INCREMENT` ↪	The initial value of the increment field	`AUTO_INCREMENT` ↪ = 5
`SHARD_ROW_ID_BITS` ↪	To set the number of bits for the implicit `_tidb_rowid` ↪ shards	`SHARD_ROW_ID_BITS` ↪ = 4
`PRE_SPLIT_REGIONS` ↪	To pre-split 2^(↪ `PRE_SPLIT_REGIONS` ↪) Regions when creating a table	`PRE_SPLIT_REGIONS` ↪ = 4
`AUTO_ID_CACHE` ↪	To set the auto ID cache size in a TiDB instance. By default, TiDB automatically changes this size according to allocation speed of auto ID	`AUTO_ID_CACHE` ↪ = 200

Options	Description	Example
AUTO_RANDOM_BASE	To set the initial incremental part value of auto_random. This option can be considered as a part of the internal interface. Users can ignore this parameter	AUTO_RANDOM_BASE = 0
CHARACTER SET	To specify the character set for the table	CHARACTER SET = 'utf8mb4'
COMMENT	The comment information	COMMENT = 'comment info'

Note:

The `split-table` configuration option is enabled by default. When it is enabled, a separate Region is created for each newly created table. For details, see TiDB configuration file.

14.10.2.33.2 Examples

Creating a simple table and inserting one row:

```
CREATE TABLE t1 (a int);
DESC t1;
SHOW CREATE TABLE t1\G
INSERT INTO t1 (a) VALUES (1);
SELECT * FROM t1;
```

```
mysql> drop table if exists t1;
Query OK, 0 rows affected (0.23 sec)

mysql> CREATE TABLE t1 (a int);
Query OK, 0 rows affected (0.09 sec)

mysql> DESC t1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
1 row in set (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM t1;
+------+
| a    |
+------+
|    1 |
+------+
1 row in set (0.00 sec)
```

Dropping a table if it exists, and conditionally creating a table if it does not exist:

```
DROP TABLE IF EXISTS t1;
CREATE TABLE IF NOT EXISTS t1 (
 id BIGINT NOT NULL PRIMARY KEY auto_increment,
 b VARCHAR(200) NOT NULL
);
DESC t1;
```

```
mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.22 sec)
```

```
mysql> CREATE TABLE IF NOT EXISTS t1 (
    -> id BIGINT NOT NULL PRIMARY KEY auto_increment,
    -> b VARCHAR(200) NOT NULL
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> DESC t1;
+-------+--------------+------+------+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+------+---------+----------------+
| id    | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| b     | varchar(200) | NO |     | NULL    |                |
+-------+--------------+------+------+---------+----------------+
2 rows in set (0.00 sec)
```

14.10.2.33.3 MySQL compatibility

- All of the data types except spatial types are supported.
- `FULLTEXT`, `HASH` and `SPATIAL` indexes are not supported.

- For compatibility, the `index_col_name` attribute supports the length option with a maximum length limit of 3072 bytes by default. The length limit can be changed through the `max-index-length` configuration option. For details, see TiDB configuration file.

- The `[ASC | DESC]` in `index_col_name` is currently parsed but ignored (MySQL 5.7 compatible behavior).
- The `COMMENT` attribute does not support the `WITH PARSER` option.
- TiDB supports 1017 columns in a single table by default and 4096 columns at most. The corresponding number limit in InnoDB is 1017 columns, and the hard limit in MySQL is 4096 columns. For details, see TiDB Limitations.
- For partitioned tables, only Range, Hash and Range Columns (single column) are supported. For details, see partitioned table.
- `CHECK` constraints are parsed but ignored (MySQL 5.7 compatible behavior). For details, see Constraints.
- `FOREIGN KEY` constraints are parsed and stored, but not enforced by DML statements. For details, see Constraints.

14.10.2.33.4 See also

- Data Types
- DROP TABLE
- CREATE TABLE LIKE
- SHOW CREATE TABLE

14.10.2.34 CREATE USER

This statement creates a new user, specified with a password. In the MySQL privilege system, a user is the combination of a username and the host from which they are connecting from. Thus, it is possible to create a user `'newuser2'@'192.168.1.1'` who is only able to connect from the IP address `192.168.1.1`. It is also possible to have two users have the same user-portion, and different permissions as they login from different hosts.

14.10.2.34.1 Synopsis

```
CreateUserStmt ::=
    'CREATE' 'USER' IfNotExists UserSpecList RequireClauseOpt
        ↪ ConnectionOptions PasswordOrLockOptions

IfNotExists ::=
    ('IF' 'NOT' 'EXISTS')?

UserSpecList ::=
    UserSpec ( ',' UserSpec )*

UserSpec ::=
    Username AuthOption

AuthOption ::=
    ( 'IDENTIFIED' ( 'BY' ( AuthString | 'PASSWORD' HashString ) | 'WITH'
        ↪ StringName ( 'BY' AuthString | 'AS' HashString )? ) )?

StringName ::=
    stringLit
|   Identifier
```

14.10.2.34.2 Examples

Create a user with the `newuserpassword` password.

```
mysql> CREATE USER 'newuser' IDENTIFIED BY 'newuserpassword';
Query OK, 1 row affected (0.04 sec)
```

Create a user who can only log in to `192.168.1.1`.

```
mysql> CREATE USER 'newuser2'@'192.168.1.1' IDENTIFIED BY 'newuserpassword'
    ↪ ;
Query OK, 1 row affected (0.02 sec)
```

Create a user who is enforced to log in using TLS connection.

```
CREATE USER 'newuser3'@'%' IDENTIFIED BY 'newuserpassword' REQUIRE SSL;
Query OK, 1 row affected (0.02 sec)
```

Create a user who is required to use X.509 certificate at login.

```
CREATE USER 'newuser4'@'%' IDENTIFIED BY 'newuserpassword' REQUIRE ISSUER '
    ↪ /C=US/ST=California/L=San Francisco/O=PingCAP';
Query OK, 1 row affected (0.02 sec)
```

14.10.2.34.3 MySQL compatibility

The following `CREATE USER` options are not yet supported by TiDB, and will be parsed but ignored:

- TiDB does not support `WITH MAX_QUERIES_PER_HOUR`, `WITH MAX_UPDATES_PER_HOUR`, and `WITH MAX_USER_CONNECTIONS` options.
- TiDB does not support the `DEFAULT ROLE` option.
- TiDB does not support `PASSWORD EXPIRE`, `PASSWORD HISTORY` or other options related to password.
- TiDB does not support the `ACCOUNT LOCK` and `ACCOUNT UNLOCK` options.

14.10.2.34.4 See also

- Security Compatibility with MySQL
- Privilege Management

- DROP USER
- SHOW CREATE USER
- ALTER USER

14.10.2.35 CREATE VIEW

The `CREATE VIEW` statement saves a `SELECT` statement as a queryable object, similar to a table. Views in TiDB are non-materialized. This means that as a view is queried, TiDB will internally rewrite the query to combine the view definition with the SQL query.

14.10.2.35.1 Synopsis

```
CreateViewStmt ::=
    'CREATE' OrReplace ViewAlgorithm ViewDefiner ViewSQLSecurity 'VIEW'
        ↪ ViewName ViewFieldList 'AS' CreateViewSelectOpt ViewCheckOption

OrReplace ::=
```

```
    ( 'OR' 'REPLACE' )?

ViewAlgorithm ::=
    ( 'ALGORITHM' '=' ( 'UNDEFINED' | 'MERGE' | 'TEMPTABLE' ) )?

ViewDefiner ::=
    ( 'DEFINER' '=' Username )?

ViewSQLSecurity ::=
    ( 'SQL' 'SECURITY' ( 'DEFINER' | 'INVOKER' ) )?

ViewName ::= TableName

ViewFieldList ::=
    ( '(' Identifier ( ',' Identifier )* ')' )?

ViewCheckOption ::=
    ( 'WITH' ( 'CASCADED' | 'LOCAL' ) 'CHECK' 'OPTION' )?
```

14.10.2.35.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE c1 > 2;
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
+----+----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM v1;
+----+----+
| id | c1 |
+----+----+
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
+----+----+
3 rows in set (0.00 sec)

mysql> INSERT INTO t1 (c1) VALUES (6);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM v1;
+----+----+
| id | c1 |
+----+----+
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
|  6 |  6 |
+----+----+
4 rows in set (0.00 sec)

mysql> INSERT INTO v1 (c1) VALUES (7);
ERROR 1105 (HY000): insert into view v1 is not supported now.
```

14.10.2.35.3 MySQL compatibility

- Currently, any view in TiDB cannot be inserted or updated (that is, `INSERT VIEW` and `UPDATE VIEW` are not supported). `WITH CHECK OPTION` is only syntactically compatible but does not take effect.
- Currently, the view in TiDB does not support `ALTER VIEW`, but you can use `CREATE OR REPLACE` instead.
- Currently, the `ALGORITHM` field is only syntactically compatible in TiDB but does not take effect. TiDB currently only supports the MERGE algorithm.

14.10.2.35.4 See also

- DROP VIEW
- CREATE TABLE
- SHOW CREATE TABLE
- DROP TABLE

14.10.2.36 DEALLOCATE

The `DEALLOCATE` statement provides an SQL interface to server-side prepared statements.

14.10.2.36.1 Synopsis

```
DeallocateStmt ::=
    DeallocateSym 'PREPARE' Identifier

DeallocateSym ::=
    'DEALLOCATE'
|   'DROP'

Identifier ::=
    identifier
|   UnReservedKeyword
|   NotKeywordToken
|   TiDBKeyword
```

14.10.2.36.2 Examples

```
mysql> PREPARE mystmt FROM 'SELECT ? as num FROM DUAL';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @number = 5;
Query OK, 0 rows affected (0.00 sec)

mysql> EXECUTE mystmt USING @number;
+------+
| num |
+------+
| 5   |
+------+
1 row in set (0.00 sec)

mysql> DEALLOCATE PREPARE mystmt;
Query OK, 0 rows affected (0.00 sec)
```

14.10.2.36.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.36.4 See also

- PREPARE
- EXECUTE

14.10.2.37 DELETE

The DELETE statement removes rows from a specified table.

14.10.2.37.1 Synopsis

```
DeleteFromStmt ::=
    'DELETE' TableOptimizerHints PriorityOpt QuickOptional IgnoreOptional (
        ↪ 'FROM' ( TableName TableAsNameOpt IndexHintListOpt
        ↪ WhereClauseOptional OrderByOptional LimitClause |
        ↪ TableAliasRefList 'USING' TableRefs WhereClauseOptional ) |
        ↪ TableAliasRefList 'FROM' TableRefs WhereClauseOptional )
```

14.10.2.37.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
+----+----+
5 rows in set (0.00 sec)

mysql> DELETE FROM t1 WHERE id = 4;
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM t1;
+----+----+
```

```
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
|  5 |  5 |
+----+----+
4 rows in set (0.00 sec)
```

14.10.2.37.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.37.4 See also

- INSERT
- SELECT
- UPDATE
- REPLACE

14.10.2.38 DESC

This statement is an alias to `EXPLAIN`. It is included for compatibility with MySQL.

14.10.2.39 DESCRIBE

This statement is an alias to `EXPLAIN`. It is included for compatibility with MySQL.

14.10.2.40 DO

`DO` executes the expressions but does not return any results. In most cases, `DO` is equivalent to `SELECT expr, ...` that does not return a result.

> **Note:**
>
> `DO` only executes expressions. It cannot be used in all cases where `SELECT` can be used. For example, `DO id FROM t1` is invalid because it references a table.

In MySQL, a common use case is to execute stored procedure or trigger. Since TiDB does not provide stored procedure or trigger, this function has a limited use.

14.10.2.40.1 Synopsis

```
DoStmt  ::= 'DO' ExpressionList

ExpressionList ::=
    Expression ( ',' Expression )*

Expression ::=
    ( singleAtIdentifier assignmentEq | 'NOT' | Expression ( logOr | 'XOR' |
        ↪ logAnd ) ) Expression
|   'MATCH' '(' ColumnNameList ')' 'AGAINST' '(' BitExpr
    ↪ FulltextSearchModifierOpt ')'
|   PredicateExpr ( IsOrNotOp 'NULL' | CompareOp ( ( singleAtIdentifier
    ↪ assignmentEq )? PredicateExpr | AnyOrAll SubSelect ) )* ( IsOrNotOp (
    ↪  trueKwd | falseKwd | 'UNKNOWN' ) )?
```

14.10.2.40.2 Examples

This SELECT statement pauses, but also produces a result set.

```
mysql> SELECT SLEEP(5);
+----------+
| SLEEP(5) |
+----------+
|        0 |
+----------+
1 row in set (5.00 sec)
```

DO, on the other hand, pauses without producing a result set.

```
mysql> DO SLEEP(5);
Query OK, 0 rows affected (5.00 sec)

mysql> DO SLEEP(1), SLEEP(1.5);
Query OK, 0 rows affected (2.50 sec)
```

14.10.2.40.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.40.4 See also

- SELECT

14.10.2.41 DROP [GLOBAL|SESSION] BINDING

This statement removes a binding from a specific SQL statement. Bindings can be used to inject a hint into a statement without requiring changes to the underlying query.

A `BINDING` can be on either a `GLOBAL` or `SESSION` basis. The default is `SESSION`.

14.10.2.41.1 Synopsis

```
DropBindingStmt ::=
    'DROP' GlobalScope 'BINDING' 'FOR' BindableStmt ( 'USING' BindableStmt )
       ↪ ?

GlobalScope ::=
    ( 'GLOBAL' | 'SESSION' )?

BindableStmt ::=
    ( SelectStmt | UpdateStmt | InsertIntoStmt | ReplaceIntoStmt |
       ↪ DeleteStmt )
```

14.10.2.41.2 Syntax description

```
mysql> CREATE TABLE t1 (
    ->  id INT NOT NULL PRIMARY KEY auto_increment,
    ->  b INT NOT NULL,
    ->  pad VARBINARY(255),
    ->  INDEX(b)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM dual;
Query OK, 1 row affected (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 1 row affected (0.00 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 8 rows affected (0.00 sec)
Records: 8 Duplicates: 0 Warnings: 0
```

```
mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 1000 rows affected (0.04 sec)
Records: 1000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (1.74 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (2.15 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (2.64 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> SELECT SLEEP(1);
+----------+
| SLEEP(1) |
+----------+
|        0 |
+----------+
1 row in set (1.00 sec)

mysql> ANALYZE TABLE t1;
Query OK, 0 rows affected (1.33 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
    ↪ ----------------------------+---------+---------+-----------+-----------------
    ↪
| id                          | estRows | actRows | task  | access object
    ↪     | execution info
    ↪                                                          | operator info
    ↪                 | memory       | disk |
+--
    ↪ ----------------------------+---------+---------+-----------+-----------------
    ↪
| IndexLookUp_10              | 583.00 | 297    | root     |
    ↪                    | time:10.545072ms, loops:2, rpc num: 1, rpc time
    ↪ :398.359µs, proc keys:297 |                       | 109.1484375 KB | N/
```

```
    ↪ A |
|  ─IndexRangeScan_8(Build)  | 583.00 | 297    | cop[tikv] | table:t1, index
    ↪ :b(b) | time:0s, loops:4                                                |
    ↪  range:[123,123], keep order:false | N/A      | N/A |
|  ─TableRowIDScan_9(Probe)  | 583.00 | 297    | cop[tikv] | table:t1
    ↪         | time:12ms, loops:4
    ↪                                                | keep order:false
    ↪              | N/A          | N/A |
+--
    ↪ ---------------------------+--------+--------+----------+-----------------
    ↪
3 rows in set (0.02 sec)

mysql> CREATE SESSION BINDING FOR
    ->  SELECT * FROM t1 WHERE b = 123
    -> USING
    ->  SELECT * FROM t1 IGNORE INDEX (b) WHERE b = 123;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
    ↪ ----------------------+----------+--------+----------+--------------+------
    ↪
| id                   | estRows  | actRows | task    | access object |
    ↪ execution info                                                    |
    ↪ operator info    | memory       | disk |
+--
    ↪ ----------------------+----------+--------+----------+--------------+------
    ↪
| TableReader_7        | 583.00   | 297    | root     |               | time
    ↪ :222.32506ms, loops:2, rpc num: 1, rpc time:222.078952ms, proc keys
    ↪ :301010 | data:Selection_6 | 88.6640625 KB | N/A |
|  ─Selection_6        | 583.00   | 297    | cop[tikv] |             | time
    ↪ :224ms, loops:298                                                  | eq(
    ↪ test.t1.b, 123) | N/A       | N/A |
|    ─TableFullScan_5  | 301010.00 | 301010 | cop[tikv] | table:t1 | time
    ↪ :220ms, loops:298                                                   |
    ↪ keep order:false | N/A         | N/A |
+--
    ↪ ----------------------+----------+--------+----------+--------------+------
    ↪
3 rows in set (0.22 sec)

mysql> SHOW SESSION BINDINGS\G
*************************** 1. row ***************************
```

```
Original_sql: select * from t1 where b = ?
    Bind_sql: SELECT * FROM t1 IGNORE INDEX (b) WHERE b = 123
  Default_db: test
      Status: using
 Create_time: 2020-05-22 14:38:03.456
 Update_time: 2020-05-22 14:38:03.456
     Charset: utf8mb4
   Collation: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)

mysql> DROP SESSION BINDING FOR SELECT * FROM t1 WHERE b = 123;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
   ↪ ----------------------------+---------+---------+-----------+-----------------
   ↪
| id                          | estRows | actRows | task  | access object
   ↪     | execution info
   ↪ | operator info                 | memory       | disk |
+--
   ↪ ----------------------------+---------+---------+-----------+-----------------
   ↪
| IndexLookUp_10              | 583.00 | 297    | root     |
   ↪                    | time:5.31206ms, loops:2, rpc num: 1, rpc time
   ↪ :665.927µs, proc keys:297 |                       | 109.1484375 KB | N
   ↪ /A |
|  ─IndexRangeScan_8(Build)  | 583.00 | 297    | cop[tikv] | table:t1, index
   ↪ :b(b) | time:0s, loops:4                                              |
   ↪ range:[123,123], keep order:false | N/A     | N/A |
|  ─TableRowIDScan_9(Probe)  | 583.00 | 297    | cop[tikv] | table:t1
   ↪         | time:0s, loops:4
   ↪                                                | keep order:false
   ↪              | N/A          | N/A |
+--
   ↪ ----------------------------+---------+---------+-----------+-----------------
   ↪
3 rows in set (0.01 sec)

mysql> SHOW SESSION BINDINGS\G
Empty set (0.00 sec)
```

14.10.2.41.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.41.4 See also

- CREATE [GLOBAL|SESSION] BINDING
- SHOW [GLOBAL|SESSION] BINDINGS
- ANALYZE TABLE
- Optimizer Hints
- SQL Plan Management

14.10.2.42 DROP COLUMN

This statement drops a column from a specified table. `DROP COLUMN` is online in TiDB, which means that it does not block read or write operations.

14.10.2.42.1 Synopsis

```
AlterTableStmt
        ::= 'ALTER' 'IGNORE'? 'TABLE' TableName DropColumnSpec ( ','
           ↪ DropColumnSpec )*

DropColumnSpec
        ::= 'DROP' 'COLUMN'? 'IF EXISTS'? ColumnName ( 'RESTRICT' | 'CASCADE
           ↪ ' )?

ColumnName
        ::= Identifier ( '.' Identifier ( '.' Identifier )? )?
```

14.10.2.42.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, col1
   ↪ INT NOT NULL, col2 INT NOT NULL);
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO t1 (col1,col2) VALUES (1,1),(2,2),(3,3),(4,4),(5,5);
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+----+------+------+
| id | col1 | col2 |
+----+------+------+
|  1 |    1 |    1 |
|  2 |    2 |    2 |
```

```
|  3 |    3 |    3 |
|  4 |    4 |    4 |
|  5 |    5 |    5 |
+----+------+------+
5 rows in set (0.01 sec)

mysql> ALTER TABLE t1 DROP COLUMN col1, DROP COLUMN col2;
ERROR 1105 (HY000): can't run multi schema change
mysql> SELECT * FROM t1;
+----+------+------+
| id | col1 | col2 |
+----+------+------+
|  1 |    1 |    1 |
|  2 |    2 |    2 |
|  3 |    3 |    3 |
|  4 |    4 |    4 |
|  5 |    5 |    5 |
+----+------+------+
5 rows in set (0.00 sec)

mysql> ALTER TABLE t1 DROP COLUMN col1;
Query OK, 0 rows affected (0.27 sec)

mysql> SELECT * FROM t1;
+----+------+
| id | col2 |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
|  5 |    5 |
+----+------+
5 rows in set (0.00 sec)
```

14.10.2.42.3 MySQL compatibility

- Dropping multiple columns in the same statement is not supported.
- Dropping primary key columns or columns covered by the composite index is not supported.

14.10.2.42.4 See also

- ADD COLUMN

- SHOW CREATE TABLE
- CREATE TABLE

14.10.2.43 DROP DATABASE

The `DROP DATABASE` statement permanently removes a specified database schema, and all of the tables and views that were created inside. User privileges that are associated with the dropped database remain unaffected.

14.10.2.43.1 Synopsis

```
DropDatabaseStmt ::=
    'DROP' 'DATABASE' IfExists DBName

IfExists ::= ( 'IF' 'EXISTS' )?
```

14.10.2.43.2 Examples

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql> DROP DATABASE test;
Query OK, 0 rows affected (0.25 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| mysql              |
+--------------------+
3 rows in set (0.00 sec)
```

14.10.2.43.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.43.4 See also

- CREATE DATABASE
- ALTER DATABASE

14.10.2.44 DROP INDEX

This statement removes an index from a specified table, marking space as free in TiKV.

14.10.2.44.1 Synopsis

```
DropIndexStmt ::=
    "DROP" "INDEX" IfExists Identifier "ON" TableName
        ↪ IndexLockAndAlgorithmOpt

IfExists ::=
    ( 'IF' 'EXISTS' )?

IndexLockAndAlgorithmOpt ::=
    ( LockClause AlgorithmClause? | AlgorithmClause LockClause? )?
```

14.10.2.44.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
    ↪ ----------------------+---------+----------+--------------+----------------
    ↪
| id                    | estRows | task     | access object | operator info
    ↪            |
+--
    ↪ ----------------------+---------+----------+--------------+----------------
    ↪
```

```
| TableReader_7        | 10.00   | root     |              | data:Selection_6
    ↪           |
|  ─Selection_6        | 10.00   | cop[tikv] |             | eq(test.t1.c1,
    ↪ 3)          |
|    ─TableFullScan_5  | 10000.00 | cop[tikv] | table:t1 | keep order:false
    ↪ , stats:pseudo |
+--
    ↪ ---------------------+---------+----------+--------------+-----------------
    ↪
3 rows in set (0.00 sec)

mysql> CREATE INDEX c1 ON t1 (c1);
Query OK, 0 rows affected (0.30 sec)

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--
    ↪ ---------------------+---------+----------+-----------------------+---------
    ↪
| id                  | estRows | task    | access object       | operator
    ↪ info                          |
+--
    ↪ ---------------------+---------+----------+-----------------------+---------
    ↪
| IndexReader_6       | 0.01    | root     |                     | index:
    ↪ IndexRangeScan_5                |
|  ─IndexRangeScan_5  | 0.01    | cop[tikv] | table:t1, index:c1(c1) | range
    ↪ :[3,3], keep order:false, stats:pseudo |
+--
    ↪ ---------------------+---------+----------+-----------------------+---------
    ↪
2 rows in set (0.00 sec)

mysql> DROP INDEX c1 ON t1;
Query OK, 0 rows affected (0.30 sec)
```

14.10.2.44.3 MySQL compatibility

- Dropping the primary key of the `CLUSTERED` type is not supported. For more details about the primary key of the `CLUSTERED` type, refer to clustered index.

14.10.2.44.4 See also

- SHOW INDEX

- CREATE INDEX
- ADD INDEX
- RENAME INDEX
- ALTER INDEX

14.10.2.45 DROP PLACEMENT POLICY

`DROP PLACEMENT POLICY` is used to drop a previously created placement policy.

14.10.2.45.1 Synopsis

```
DropPolicyStmt ::=
    "DROP" "PLACEMENT" "POLICY" IfExists PolicyName

PolicyName ::=
    Identifier
```

14.10.2.45.2 Examples

Placement policies can only be dropped when they are not referenced by any tables or partitions.

```
CREATE PLACEMENT POLICY p1 FOLLOWERS=4;
CREATE TABLE t1 (a INT PRIMARY KEY) PLACEMENT POLICY=p1;
DROP PLACEMENT POLICY p1; -- This statement fails because the placement
    ↪ policy p1 is referenced.

-- Finds which tables and partitions reference the placement policy.
SELECT table_schema, table_name FROM information_schema.tables WHERE
    ↪ tidb_placement_policy_name='p1';
SELECT table_schema, table_name FROM information_schema.partitions WHERE
    ↪ tidb_placement_policy_name='p1';

ALTER TABLE t1 PLACEMENT POLICY=default; -- Removes the placement policy
    ↪ from t1.
DROP PLACEMENT POLICY p1; -- Succeeds.
```

```
Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.11 sec)

ERROR 8241 (HY000): Placement policy 'p1' is still in use

+--------------+------------+
| table_schema | table_name |
```

```
+--------------+------------+
| test         | t1         |
+--------------+------------+
1 row in set (0.00 sec)

Empty set (0.01 sec)

Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.21 sec)
```

14.10.2.45.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.45.4 See also

- Placement Rules in SQL
- SHOW PLACEMENT
- CREATE PLACEMENT POLICY
- ALTER PLACEMENT POLICY

14.10.2.46 DROP ROLE

This statement removes a role, that was previously created with `CREATE ROLE`.

14.10.2.46.1 Synopsis

```
DropRoleStmt ::=
    'DROP' 'ROLE' ( 'IF' 'EXISTS' )? RolenameList

RolenameList ::=
    Rolename ( ',' Rolename )*
```

14.10.2.46.2 Examples

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

Create a new role `analyticsteam` and a new user `jennifer`:

```
CREATE ROLE analyticsteam;
Query OK, 0 rows affected (0.02 sec)
```

```
GRANT SELECT ON test.* TO analyticsteam;
Query OK, 0 rows affected (0.02 sec)

CREATE USER jennifer;
Query OK, 0 rows affected (0.01 sec)

GRANT analyticsteam TO jennifer;
Query OK, 0 rows affected (0.01 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

Note that by default `jennifer` needs to execute `SET ROLE analyticsteam` in order to be able to use the privileges associated with the `analyticsteam` role:

```
SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-------------------------------------------+
2 rows in set (0.00 sec)

SHOW TABLES in test;
ERROR 1044 (42000): Access denied for user 'jennifer'@'%' to database 'test'
SET ROLE analyticsteam;
Query OK, 0 rows affected (0.00 sec)

SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT SELECT ON test.* TO 'jennifer'@'%' |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-------------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
```

```
1 row in set (0.00 sec)
```

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

The statement `SET DEFAULT ROLE` can be used to associate the role `analyticsteam` to `jennifer`:

```
SET DEFAULT ROLE analyticsteam TO jennifer;
Query OK, 0 rows affected (0.02 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

After this, the user `jennifer` has the privileges associated with the role `analyticsteam` and `jennifer` does not have to execute the statement `SET ROLE`:

```
SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT SELECT ON test.* TO 'jennifer'@'%'  |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-------------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

Drop the role for the `analyticsteam`:

```
DROP ROLE analyticsteam;
Query OK, 0 rows affected (0.02 sec)
```

`jennifer` no longer has the default role of `analyticsteam` associated, nor can set the role to `analyticsteam`.

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

Show the privileges of `jennifer`:

```
SHOW GRANTS;
+-------------------------------------+
| Grants for User                     |
+-------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%' |
+-------------------------------------+
1 row in set (0.00 sec)

SET ROLE analyticsteam;
ERROR 3530 (HY000): `analyticsteam`@`%` is is not granted to jennifer@%
```

14.10.2.46.3 MySQL compatibility

This statement is understood to be fully compatible with roles, which are a feature of MySQL 8.0. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.46.4 See also

- CREATE ROLE
- GRANT <role>
- REVOKE <role>
- SET ROLE
- SET DEFAULT ROLE

- Role-Based Access Control

14.10.2.47 DROP SEQUENCE

The `DROP SEQUENCE` statement drops the sequence object in TiDB.

14.10.2.47.1 Synopsis

```
DropSequenceStmt ::=
    'DROP' 'SEQUENCE' IfExists TableNameList

IfExists ::= ( 'IF' 'EXISTS' )?

TableNameList ::=
    TableName ( ',' TableName )*
```

```
TableName ::=
    Identifier ('.' Identifier)?
```

14.10.2.47.2 Examples

```
DROP SEQUENCE seq;
```

```
Query OK, 0 rows affected (0.10 sec)
```

```
DROP SEQUENCE seq, seq2;
```

```
Query OK, 0 rows affected (0.03 sec)
```

14.10.2.47.3 MySQL compatibility

This statement is a TiDB extension. The implementation is modeled on sequences available in MariaDB.

14.10.2.47.4 See also

- CREATE SEQUENCE
- SHOW CREATE SEQUENCE

14.10.2.48 DROP STATS

The `DROP STATS` statement is used to delete the statistics of the selected table from the selected database.

14.10.2.48.1 Synopsis

```
DropStatsStmt ::=
    'DROP' 'STATS' TableName

TableName ::=
    Identifier ('.' Identifier)?
```

14.10.2.48.2 Examples

```
CREATE TABLE t(a INT);
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
SHOW STATS_META WHERE db_name='test' and table_name='t';
```

```
+--
    ↪ -------+-----------+---------------+---------------------+--------------+-----
    ↪
| Db_name | Table_name | Partition_name | Update_time | Modify_count |
    ↪ Row_count |
+--
    ↪ -------+-----------+---------------+---------------------+--------------+-----
    ↪
| test   | t          |                | 2020-05-25 20:34:33 |        0 |
    ↪         0 |
+--
    ↪ -------+-----------+---------------+---------------------+--------------+-----
    ↪
1 row in set (0.00 sec)
```

```
DROP STATS t;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
SHOW STATS_META WHERE db_name='test' and table_name='t';
```

```
Empty set (0.00 sec)
```

14.10.2.48.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.48.4 See also

- Introduction to Statistics

14.10.2.49 DROP TABLE

This statement drops a table from the currently selected database. An error is returned if the table does not exist, unless the `IF EXISTS` modifier is used.

14.10.2.49.1 Synopsis

```
DropTableStmt ::=
    'DROP' OptTemporary TableOrTables IfExists TableNameList
        ↪ RestrictOrCascadeOpt
```

```
OptTemporary ::=
    ( 'TEMPORARY' | ('GLOBAL' 'TEMPORARY') )?

TableOrTables ::=
    'TABLE'
|   'TABLES'

TableNameList ::=
    TableName ( ',' TableName )*
```

14.10.2.49.2 Drop temporary tables

You can use the following syntax to drop ordinary tables and temporary tables:

- Use `DROP TEMPORARY TABLE` to drop local temporary tables.
- Use `DROP GLOBAL TEMPORARY TABLE` to drop global temporary tables.
- Use `DROP TABLE` to drop ordinary tables or temporary tables.

14.10.2.49.3 Examples

```
mysql> CREATE TABLE t1 (a INT);
Query OK, 0 rows affected (0.11 sec)

mysql> DROP TABLE t1;
Query OK, 0 rows affected (0.22 sec)

mysql> DROP TABLE table_not_exists;
ERROR 1051 (42S02): Unknown table 'test.table_not_exists'

mysql> DROP TABLE IF EXISTS table_not_exists;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+-------+------+----------------------------------------+
| Level | Code | Message                                |
+-------+------+----------------------------------------+
| Note | 1051 | Unknown table 'test.table_not_exists' |
+-------+------+----------------------------------------+
1 row in set (0.01 sec)

mysql> CREATE VIEW v1 AS SELECT 1;
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> DROP TABLE v1;
Query OK, 0 rows affected (0.23 sec)
```

14.10.2.49.4 MySQL compatibility

Currently, `RESTRICT` and `CASCADE` are only supported syntactically.

14.10.2.49.5 See also

- CREATE TABLE
- SHOW CREATE TABLE
- SHOW TABLES

14.10.2.50 DROP USER

This statement removes a user from the TiDB system database. The optional keyword `IF EXISTS` can be used to silence an error if the user does not exist. This statement requires the `CREATE USER` privilege.

14.10.2.50.1 Synopsis

```
DropUserStmt ::=
    'DROP' 'USER' ( 'IF' 'EXISTS' )? UsernameList

Username ::=
    StringName ('@' StringName | singleAtIdentifier)? | 'CURRENT_USER'
        ↪ OptionalBraces
```

14.10.2.50.2 Examples

```
mysql> DROP USER idontexist;
ERROR 1396 (HY000): Operation DROP USER failed for idontexist@%

mysql> DROP USER IF EXISTS 'idontexist';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER 'newuser' IDENTIFIED BY 'mypassword';
Query OK, 1 row affected (0.02 sec)

mysql> GRANT ALL ON test.* TO 'newuser';
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW GRANTS FOR 'newuser';
+-------------------------------------------------+
```

```
| Grants for newuser@%                          |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO 'newuser'@'%'           |
| GRANT ALL PRIVILEGES ON test.* TO 'newuser'@'%' |
+-----------------------------------------------+
2 rows in set (0.00 sec)

mysql> REVOKE ALL ON test.* FROM 'newuser';
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW GRANTS FOR 'newuser';
+-----------------------------------+
| Grants for newuser@%              |
+-----------------------------------+
| GRANT USAGE ON *.* TO 'newuser'@'%' |
+-----------------------------------+
1 row in set (0.00 sec)

mysql> DROP USER 'newuser';
Query OK, 0 rows affected (0.14 sec)

mysql> SHOW GRANTS FOR 'newuser';
ERROR 1141 (42000): There is no such grant defined for user 'newuser' on
    ↪ host '%'
```

14.10.2.50.3 MySQL compatibility

- Dropping a user that does not exist with `IF EXISTS` will not create a warning in TiDB. Issue #10196.

14.10.2.50.4 See also

- CREATE USER
- ALTER USER
- SHOW CREATE USER

- Privilege Management

14.10.2.51 DROP VIEW

This statement drops an view object from the currently selected database. It does not effect any base tables that a view references.

14.10.2.51.1 Synopsis

```
DropViewStmt ::=
    'DROP' 'VIEW' ( 'IF' 'EXISTS' )? TableNameList RestrictOrCascadeOpt

TableNameList ::=
    TableName ( ',' TableName )*

TableName ::=
    Identifier ('.' Identifier)?
```

14.10.2.51.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE c1 > 2;
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
+----+----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM v1;
+----+----+
| id | c1 |
+----+----+
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
+----+----+
3 rows in set (0.00 sec)
```

```
mysql> DROP VIEW v1;
Query OK, 0 rows affected (0.23 sec)

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
+----+----+
5 rows in set (0.00 sec)
```

14.10.2.51.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.51.4 See also

- CREATE VIEW
- DROP TABLE

14.10.2.52 EXECUTE

The `EXECUTE` statement provides an SQL interface to server-side prepared statements.

14.10.2.52.1 Synopsis

```
ExecuteStmt ::=
    'EXECUTE' Identifier ( 'USING' UserVariable ( ',' UserVariable )* )?
```

14.10.2.52.2 Examples

```
mysql> PREPARE mystmt FROM 'SELECT ? as num FROM DUAL';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @number = 5;
Query OK, 0 rows affected (0.00 sec)

mysql> EXECUTE mystmt USING @number;
```

```
+------+
| num |
+------+
| 5    |
+------+
1 row in set (0.00 sec)

mysql> DEALLOCATE PREPARE mystmt;
Query OK, 0 rows affected (0.00 sec)
```

14.10.2.52.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.52.4 See also

- PREPARE
- DEALLOCATE

14.10.2.53 EXPLAIN ANALYZE

The `EXPLAIN ANALYZE` statement works similar to `EXPLAIN`, with the major difference being that it will actually execute the statement. This allows you to compare the estimates used as part of query planning to actual values encountered during execution. If the estimates differ significantly from the actual values, you should consider running `ANALYZE TABLE` on the affected tables.

> **Note:**
>
> When you use `EXPLAIN ANALYZE` to execute DML statements, modification to data is normally executed. Currently, the execution plan for DML statements **cannot** be shown yet.

14.10.2.53.1 Synopsis

```
ExplainSym ::=
    'EXPLAIN'
|   'DESCRIBE'
|    'DESC'
```

```
ExplainStmt ::=
    ExplainSym ( TableName ColumnName? | 'ANALYZE'? ExplainableStmt | 'FOR'
        ↪ 'CONNECTION' NUM | 'FORMAT' '=' ( stringLit | ExplainFormatType )
        ↪ ( 'FOR' 'CONNECTION' NUM | ExplainableStmt ) )

ExplainableStmt ::=
    SelectStmt
|   DeleteFromStmt
|   UpdateStmt
|   InsertIntoStmt
|   ReplaceIntoStmt
|   UnionStmt
```

14.10.2.53.2 EXPLAIN ANALYZE output format

Different from `EXPLAIN`, `EXPLAIN ANALYZE` executes the corresponding SQL statement, records its runtime information, and returns the information together with the execution plan. Therefore, you can regard `EXPLAIN ANALYZE` as an extension of the `EXPLAIN` statement. Compared to `EXPLAIN` (for debugging query exeuction), the return results of `EXPLAIN ANALYZE` also include columns of information such as `actRows`, `execution info`, `memory`, and `disk`. The details of these columns are shown as follows:

attribute name	description
actRows	Number of rows output by the operator.
execution info	Execution information of the operator. `time` represents the total `wall time` from entering the operator to leaving the operator, including the total execution time of all sub-operators. If the operator is called many times by the parent operator (in loops), then the time refers to the accumulated time. `loops` is the number of times the current operator is called by the parent operator.
memory	Memory space occupied by the operator.
disk	Disk space occupied by the operator.

14.10.2.53.3 Examples

```
CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT NOT
    ↪ NULL);
```

```
Query OK, 0 rows affected (0.12 sec)
```

```
INSERT INTO t1 (c1) VALUES (1), (2), (3);
```

```
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
EXPLAIN ANALYZE SELECT * FROM t1 WHERE id = 1;
```

```
+--
    ↪ ----------+---------+---------+------+--------------+----------------------
    ↪
| id         | estRows | actRows | task | access object | execution info
    ↪                                    | operator info | memory | disk |
+--
    ↪ ----------+---------+---------+------+--------------+----------------------
    ↪
| Point_Get_1 | 1.00 | 1       | root | table:t1    | time:757.205µs, loops:2,
    ↪  Get:{num_rpc:1, total_time:697.051µs} | handle:1 | N/A | N/A |
+--
    ↪ ----------+---------+---------+------+--------------+----------------------
    ↪
1 row in set (0.01 sec)
```

```
EXPLAIN ANALYZE SELECT * FROM t1;
```

```
+--
    ↪ ----------------+---------+---------+----------+--------------+-----------
    ↪
| id                | estRows | actRows | task    | access object | execution
    ↪ info
    ↪
    ↪ | operator info               | memory   | disk |
+--
    ↪ ----------------+---------+---------+----------+--------------+-----------
    ↪
| TableReader_5   | 10000.00 | 3     | root     |               | time:278.2µs,
    ↪  loops:2, cop_task: {num: 1, max: 437.6µs, proc_keys: 3, rpc_num: 1,
    ↪ rpc_time: 423.9µs, copr_cache_hit_ratio: 0.00}
    ↪
    ↪ | data:TableFullScan_4        | 251 Bytes | N/A |
```

```
|  └─TableFullScan_4 | 10000.00 | 3   | cop[tikv] | table:t1   | tikv_task:{
    ↪ time:0s, loops:1}, scan_detail: {total_process_keys: 3,
    ↪ total_process_keys_size: 111, total_keys: 4, rocksdb: {
    ↪ delete_skipped_count: 0, key_skipped_count: 3, block: {
    ↪ cache_hit_count: 0, read_count: 0, read_byte: 0 Bytes}}} | keep order
    ↪ :false, stats:pseudo | N/A | N/A |
+--
    ↪ ----------------+----------+---------+-----------+--------------+------------
    ↪
2 rows in set (0.00 sec)
```

14.10.2.53.4 Execution information of operators

In addition to the basic `time` and `loop` execution information, `execution info` also contains operator-specific execution information, which mainly includes the time consumed for the operator to send RPC requests and the duration of other steps.

Point_Get

The execution information from a `Point_Get` operator will typically contain the following information:

- `Get:{num_rpc:1, total_time:697.051µs}`: The number of the `Get` RPC requests (`num_rpc`) sent to TiKV and the total duration (`total_time`) of all RPC requests.
- `ResolveLock:{num_rpc:1, total_time:12.117495ms}`: If TiDB encounters a lock when reading data, it has to resolve the lock first, which generally occurs in the scenario of read-write conflict. This information indicates the duration of resolving locks.
- `regionMiss_backoff:{num:11, total_time:2010 ms},tikvRPC_backoff:{num ↪ :11, total_time:10691 ms}`: When an RPC request fails, TiDB will wait the backoff time before retrying the request. Backoff statistics include the type of backoff (such as `regionMiss` and `tikvRPC`), the total waiting time (`total_time`), and the total number of backoffs (`num`).

Batch_Point_Get

The execution information of the `Batch_Point_Get` operator is similar to that of the `Point_Get` operator, but `Batch_Point_Get` generally sends `BatchGet` RPC requests to TiKV to read data.

`BatchGet:{num_rpc:2, total_time:83.13µs}`: The number of RPC requests (`num_rpc`) of the `BatchGet` type sent to TiKV and the total time consumed (`total_time`) for all RPC requests.

TableReader

The execution information of a `TableReader` operator is typically as follows:

```
cop_task: {num: 6, max: 1.07587ms, min: 844.312µs, avg: 919.601µs, p95:
    ↪ 1.07587ms, max_proc_keys: 16, p95_proc_keys: 16, tot_proc: 1ms,
    ↪ tot_wait: 1ms, rpc_num: 6, rpc_time: 5.313996 ms,
    ↪ copr_cache_hit_ratio: 0.00}
```

- `cop_task`: Contains the execution information of `cop` tasks. For example:
  - `num`: The number of cop tasks.
  - `max`, `min`, `avg`, `p95`: The maximum, minimum, average, and P95 values of the execution time consumed for executing cop tasks.
  - `max_proc_keys` and `p95_proc_keys`: The maximum and P95 key-values scanned by TiKV in all cop tasks. If the difference between the maximum value and the P95 value is large, the data distribution might be imbalanced.
  - `rpc_num`, `rpc_time`: The total number and total time consumed for `Cop` RPC requests sent to TiKV.
  - `copr_cache_hit_ratio`: The hit rate of Coprocessor Cache for `cop` task requests.
- `backoff`: Contains different types of backoff and the total waiting time of backoff.

Insert

The execution information of an `Insert` operator is typically as follows:

```
prepare:109.616µs, check_insert:{total_time:1.431678ms, mem_insert_time
    ↪ :667.878µs, prefetch:763.8µs, rpc:{BatchGet:{num_rpc:1, total_time
    ↪ :699.166µs},Get:{num_rpc:1, total_time:378.276µs }}}
```

- `prepare`: The time consumed for preparing to write, including expression, default value and auto-increment value calculations.
- `check_insert`: This information generally appears in `insert ignore` and `insert ↪ on duplicate` statements, including conflict checking and the time consumed for writing data to TiDB transaction cache. Note that this time consumption does not include the time consumed for transaction commit. It contains the following information:
  - `total_time`: The total time spent on the `check_insert` step.
  - `mem_insert_time`: The time consumed for writing data to the TiDB transaction cache.
  - `prefetch`: The duration of retrieving the data that needs to be checked for conflicts from TiKV. This step sends a `Batch_Get` RPC request to TiKV to retrieve data.
  - `rpc`: The total time consumed for sending RPC requests to TiKV, which generally includes two types of RPC time, `BatchGet` and `Get`, among which:
    - `BatchGet` RPC request is sent in the `prefetch` step.

    - `Get` RPC request is sent when the `insert on duplicate` statement executes `duplicate update`.

- `backoff`: Contains different types of backoff and the total waiting time of backoff.

IndexJoin

The `IndexJoin` operator has 1 outer worker and N inner workers for concurrent execution. The join result preserves the order of the outer table. The detailed execution process is as follows:

1. The outer worker reads N outer rows, then wraps it into a task, and sends it to the result channel and the inner worker channel.
2. The inner worker receives the task, build key ranges from the task, and fetches inner rows according to the key ranges. It then builds the inner row hash table.
3. The main `IndexJoin` thread receives the task from the result channel and waits for the inner worker to finish handling the task.
4. The main `IndexJoin` thread joins each outer row by looking up to the inner rows' hash table.

The `IndexJoin` operator contains the following execution information:

```
inner:{total:4.297515932s, concurrency:5, task:17, construct:97.96291ms,
    ↪ fetch:4.164310088s, build:35.219574ms}, probe:53.574945ms
```

- `Inner`: The execution information of inner worker:
    - `total`: The total time consumed by the inner worker.
    - `concurrency`: The number of concurrent inner workers.
    - `task`: The total number of tasks processed by the inner worker.
    - `construct`: The preparation time before the inner worker reads the inner table rows corresponding to the task.
    - `fetch`: The total time consumed for it takes for the inner worker to read inner table rows.
    - `Build`: The total time consumed for it takes for the inner worker to construct the hash table of the corresponding inner table rows.
- `probe`: The total time consumed by the main `IndexJoin` thread to perform join operations with the hash table of the outer table rows and the inner table rows.

IndexHashJoin

The execution process of the `IndexHashJoin` operator is similar to that of the `IndexJoin` operator. `IndexHashJoin` operator also has 1 outer worker and N inner workers to execute in parallel, but the output order is not guaranteed to be consistent with that of the outer table. The detailed execution process is as follows:

1. The outer worker reads N outer rows, builds a task, and sends it to the inner worker channel.
2. The inner worker receives the tasks from the inner worker channel and performs the following three operations in order for every task:

a. Build a hash table from the outer rows
b. Build key ranges from outer rows and fetches inner rows
c. Probe the hash table and sends the join result to the result channel. Note: step a and step b are running concurrently.

3. The main thread of `IndexHashJoin` receives the join results from the result channel.

The `IndexHashJoin` operator contains the following execution information:

```
inner:{total:4.429220003s, concurrency:5, task:17, construct:96.207725ms,
    ↪ fetch:4.239324006s, build:24.567801ms, join:93.607362ms}
```

- `Inner`: the execution information of inner worker:
  - `total`: the total time consumed by the inner worker.
  - `concurrency`: the number of inner workers.
  - `task`: The total number of tasks processed by the inner worker.
  - `construct`: The preparation time before the inner worker reads the inner table rows.
  - `fetch`: The total time consumed for inner worker to read inner table rows.
  - `Build`: The total time consumed for inner worker to construct the hash table of the outer table rows.
  - `join`: The total time consumed for inner worker to do join with the inner table rows and the hash table of outer table rows.

HashJoin

The `HashJoin` operator has an inner worker, an outer worker, and N join workers. The detailed execution process is as follows:

1. The inner worker reads inner table rows and constructs a hash table.
2. The outer worker reads the outer table rows, then wraps it into a task and sends it to the join worker.
3. The join worker waits for the hash table construction in step 1 to finish.
4. The join worker uses the outer table rows and hash table in the task to perform join operations, and then sends the join result to the result channel.
5. The main thread of `HashJoin` receives the join result from the result channel.

The `HashJoin` operator contains the following execution information:

```
build_hash_table:{total:146.071334ms, fetch:110.338509ms, build:35.732825ms
    ↪ }, probe:{concurrency:5, total:857.162518ms, max:171.48271ms, probe
    ↪ :125.341665ms, fetch:731.820853ms}
```

- `build_hash_table`: Reads the data of the inner table and constructs the execution information of the hash table:
    - `total`: The total time consumption.
    - `fetch`: The total time spent reading inner table data.
    - `build`: The total time spent constructing a hash table.
- `probe`: The execution information of join workers:
    - `concurrency`: The number of join workers.
    - `total`: The total time consumed by all join workers.
    - `max`: The longest time for a single join worker to execute.
    - `probe`: The total time consumed for joining with outer table rows and the hash table.
    - `fetch`: The total time that the join worker waits to read the outer table rows data.

lock_keys execution information

When a DML statement is executed in a pessimistic transaction, the execution information of the operator might also include the execution information of `lock_keys`. For example:

```
lock_keys: {time:94.096168ms, region:6, keys:8, lock_rpc:274.503214ms,
    ↪ rpc_count:6}
```

- `time`: The total duration of executing the `lock_keys` operation.
- `region`: The number of Regions involved in executing the `lock_keys` operation.
- `keys`: The number of `Keys` that need `Lock`.
- `lock_rpc`: The total time spent sending an RPC request of the `Lock` type to TiKV. Because multiple RPC requests can be sent in parallel, the total RPC time consumption might be greater than the total time consumption of the `lock_keys` operation.
- `rpc_count`: The total number of RPC requests of the `Lock` type sent to TiKV.

commit_txn execution information

When a write-type DML statement is executed in a transaction with `autocommit=1`, the execution information of the write operator will also include the duration information of the transaction commit. For example:

```
commit_txn: {prewrite:48.564544ms, wait_prewrite_binlog:47.821579,
    ↪ get_commit_ts:4.277455ms, commit:50.431774ms, region_num:7,
    ↪ write_keys:16, write_byte:536}
```

- `prewrite`: The time consumed for the `prewrite` phase of the 2PC commit of the transaction.
- `wait_prewrite_binlog`:: The time consumed for waiting to write the prewrite Binlog.
- `get_commit_ts`: The time consumed for getting the transaction commit timestamp.
- `commit`: The time consumed for the `commit` phase during the 2PC commit of the transaction.
- `write_keys`: The total `keys` written in the transaction.
- `write_byte`: The total bytes of `key-value` written in the transaction, and the unit is byte.

Other common execution information

The Coprocessor operators usually contain two parts of execution time information: `cop_task` and `tikv_task`. `cop_task` is the time recorded by TiDB, and it is from the moment that the request is sent to the server to the moment that the response is received. `tikv_task` is the time recorded by TiKV Coprocessor itself. If there is much difference between the two, it might indicate that the time spent waiting for the response is too long, or the time spent on gRPC or network is too long.

14.10.2.53.5 MySQL compatibility

`EXPLAIN ANALYZE` is a feature of MySQL 8.0, but both the output format and the potential execution plans in TiDB differ substantially from MySQL.

14.10.2.53.6 See also

- Understanding the Query Execution Plan
- EXPLAIN
- ANALYZE TABLE
- TRACE

14.10.2.54 `EXPLAIN`

The `EXPLAIN` statement shows the execution plan for a query without executing it. It is complimented by `EXPLAIN ANALYZE` which will execute the query. If the output of `EXPLAIN` does not match the expected result, consider executing `ANALYZE TABLE` on each table in the query.

The statements `DESC` and `DESCRIBE` are aliases of this statement. The alternative usage of `EXPLAIN <tableName>` is documented under `SHOW [FULL] COLUMNS FROM`.

TiDB supports the `EXPLAIN [options] FOR CONNECTION connection_id` statement. However, this statement is different from the `EXPLAIN FOR` statement in MySQL. For more details, see `EXPLAIN FOR CONNECTION`.

14.10.2.54.1 Synopsis

```
ExplainSym ::=
    'EXPLAIN'
|   'DESCRIBE'
|   'DESC'

ExplainStmt ::=
    ExplainSym ( TableName ColumnName? | 'ANALYZE'? ExplainableStmt | 'FOR'
        ↪ 'CONNECTION' NUM | 'FORMAT' '=' ( stringLit | ExplainFormatType )
        ↪ ( 'FOR' 'CONNECTION' NUM | ExplainableStmt ) )

ExplainableStmt ::=
    SelectStmt
|   DeleteFromStmt
|   UpdateStmt
|   InsertIntoStmt
|   ReplaceIntoStmt
|   UnionStmt
```

14.10.2.54.2 `EXPLAIN` output format

> **Note:**
>
> When you use the MySQL client to connect to TiDB, to read the output result in a clearer way without line wrapping, you can use the `pager less -S` command. Then, after the `EXPLAIN` result is output, you can press the right arrow → button on your keyboard to horizontally scroll through the output.

Currently, `EXPLAIN` in TiDB outputs 5 columns: `id`, `estRows`, `task`, `access object`, `operator info`. Each operator in the execution plan is described by these attributes, with each row in the `EXPLAIN` output describing an operator. The description of each attribute is as follows:

Attribute name	Description
id	The operator ID is the unique identifier of the operator in the entire execution plan. In TiDB 2.1, the ID is formatted to display the tree structure of the operator. Data flows from the child node to the parent node. One and only one parent node for each operator.
estRows	The number of rows that the operator is expected to output. This number is estimated according to the statistics and the operator's logic. `estRows` is called `count` in the earlier versions of TiDB 4.0.
task	The type of task the operator belongs to. Currently, the execution plans are divided into two tasks: **root** task, which is executed on tidb-server, and **cop** task, which is performed in parallel on TiKV or TiFlash. The topology of the execution plan at the task level is that a root task followed by many cop tasks. The root task uses the output of cop tasks as input. The cop tasks refer to tasks that TiDB pushes down to TiKV or TiFlash. Each cop task is distributed in the TiKV cluster or the TiFlash cluster, and is executed by multiple processes.
access object	Data item information accessed by the operator. The information includes `table`, `partition`, and `index` (if any). Only operators that directly access the data have such information.
operator info	Other information about the operator. `operator info` of each operator is different. You can refer to the following examples.

14.10.2.54.3 Examples

```
EXPLAIN SELECT 1;
```

```
+------------------+---------+------+---------------+---------------+
| id               | estRows | task | access object | operator info |
+------------------+---------+------+---------------+---------------+
| Projection_3     | 1.00    | root |               | 1->Column#1 |
| └─TableDual_4    | 1.00    | root |               | rows:1      |
+------------------+---------+------+---------------+---------------+
2 rows in set (0.00 sec)
```

```
CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT NOT
    ↪ NULL);
```

```
Query OK, 0 rows affected (0.10 sec)
```

```
INSERT INTO t1 (c1) VALUES (1), (2), (3);
```

```
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
EXPLAIN SELECT * FROM t1 WHERE id = 1;
```

```
+-------------+---------+------+--------------+--------------+
| id          | estRows | task | access object | operator info |
+-------------+---------+------+--------------+--------------+
| Point_Get_1 | 1.00 | root | table:t1    | handle:1     |
+-------------+---------+------+--------------+--------------+
1 row in set (0.00 sec)
```

```
DESC SELECT * FROM t1 WHERE id = 1;
```

```
+-------------+---------+------+--------------+--------------+
| id          | estRows | task | access object | operator info |
+-------------+---------+------+--------------+--------------+
| Point_Get_1 | 1.00 | root | table:t1    | handle:1     |
+-------------+---------+------+--------------+--------------+
1 row in set (0.00 sec)
```

```
DESCRIBE SELECT * FROM t1 WHERE id = 1;
```

```
+-------------+---------+------+--------------+--------------+
| id          | estRows | task | access object | operator info |
+-------------+---------+------+--------------+--------------+
| Point_Get_1 | 1.00 | root | table:t1    | handle:1     |
+-------------+---------+------+--------------+--------------+
1 row in set (0.00 sec)
```

```
EXPLAIN INSERT INTO t1 (c1) VALUES (4);
```

```
+---------+---------+------+--------------+--------------+
| id      | estRows | task | access object | operator info |
+---------+---------+------+--------------+--------------+
| Insert_1 | N/A   | root |              | N/A          |
+---------+---------+------+--------------+--------------+
1 row in set (0.00 sec)
```

```
EXPLAIN UPDATE t1 SET c1=5 WHERE c1=3;
```

```
+--
   ↪ ----------------------+---------+----------+--------------+----------------
   ↪
| id                     | estRows | task    | access object | operator info
   ↪            |
```

```
+--
    ↪ ----------------------+---------+----------+--------------+---------------
    ↪
| Update_4                | N/A     | root     |              | N/A
    ↪                         |
|  ─TableReader_8         | 0.00    | root     |              | data:
    ↪ Selection_7             |
|    ─Selection_7         | 0.00    | cop[tikv] |              | eq(test.t1.c1,
    ↪ 3)           |
|      ─TableFullScan_6   | 3.00    | cop[tikv] | table:t1   | keep order:
    ↪ false, stats:pseudo |
+--
    ↪ ----------------------+---------+----------+--------------+---------------
    ↪
4 rows in set (0.00 sec)
```

```
EXPLAIN DELETE FROM t1 WHERE c1=3;
```

```
+--
    ↪ ----------------------+---------+----------+--------------+---------------
    ↪
| id                      | estRows | task     | access object | operator info
    ↪             |
+--
    ↪ ----------------------+---------+----------+--------------+---------------
    ↪
| Delete_4                | N/A     | root     |              | N/A
    ↪                         |
|  ─TableReader_8         | 0.00    | root     |              | data:
    ↪ Selection_7             |
|    ─Selection_7         | 0.00    | cop[tikv] |              | eq(test.t1.c1,
    ↪ 3)           |
|      ─TableFullScan_6   | 3.00    | cop[tikv] | table:t1   | keep order:
    ↪ false, stats:pseudo |
+--
    ↪ ----------------------+---------+----------+--------------+---------------
    ↪
4 rows in set (0.01 sec)
```

To specify the content and format of the output, you can use the `FORMAT = xxx` syntax in the `EXPLAIN` statement.

FORMAT	Description
Empty	Same as `row`

FORMAT	Description
`row`	The `EXPLAIN` statement outputs results in a tabular format. See Understand the Query Execution Plan for more information.

FORMAT	Description
`brief`	The operator IDs in the output of the `EXPLAIN` statement are simplified, compared with those when `FORMAT` is left unspecified.

FORMAT	Description
`dot`	The `EXPLAIN` statement outputs DOT execution plans, which can be used to generate PNG files through a `dot` program (in the `graphviz` package).

The following is an example when `FORMAT` is `"brief"` in `EXPLAIN`:

```
EXPLAIN FORMAT = "brief" DELETE FROM t1 WHERE c1 = 3;
```

```
+--
   ↪ ----------------------+---------+-----------+--------------+------------------
   ↪
| id                    | estRows | task    | access object | operator info
   ↪            |
+--
   ↪ ----------------------+---------+-----------+--------------+------------------
   ↪
| Delete                | N/A     | root      |               | N/A
   ↪                         |
|  ─TableReader         | 0.00    | root      |               | data:Selection
   ↪              |
```

```
|   └─Selection         | 0.00   | cop[tikv] |           | eq(test.t1.c1, 3)
    ↪           |
|     └─TableFullScan   | 3.00   | cop[tikv] | table:t1  | keep order:false,
    ↪  stats:pseudo |
+--
    ↪ ----------------------+--------+----------+-------------+----------------
    ↪
4 rows in set (0.001 sec)
```

In addition to the MySQL standard result format, TiDB also supports DotGraph and you need to specify `FORMAT = "dot"` as in the following example:

```
CREATE TABLE t(a bigint, b bigint);
EXPLAIN format = "dot" SELECT A.a, B.b FROM t A JOIN t B ON A.a > B.b WHERE
    ↪  A.a < 10;
```

```
+--
    ↪ ----------------------------------------------------------------------------
    ↪
| dot contents
    ↪
    ↪ |
+--
    ↪ ----------------------------------------------------------------------------
    ↪
|
digraph Projection_8 {
subgraph cluster8{
node [style=filled, color=lightgrey]
color=black
label = "root"
"Projection_8" -> "HashJoin_9"
"HashJoin_9" -> "TableReader_13"
"HashJoin_9" -> "Selection_14"
"Selection_14" -> "TableReader_17"
}
subgraph cluster12{
node [style=filled, color=lightgrey]
color=black
label = "cop"
"Selection_12" -> "TableFullScan_11"
}
subgraph cluster16{
node [style=filled, color=lightgrey]
color=black
```

```
label = "cop"
"Selection_16" -> "TableFullScan_15"
}
"TableReader_13" -> "Selection_12"
"TableReader_17" -> "Selection_16"
}
 |
+--
    ↪ ------------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

If your computer has a `dot` program, you can generate a PNG file using the following method:

```
dot xx.dot -T png -O

The xx.dot is the result returned by the above statement.
```

If your computer has no `dot` program, copy the result to this website to get a tree diagram:

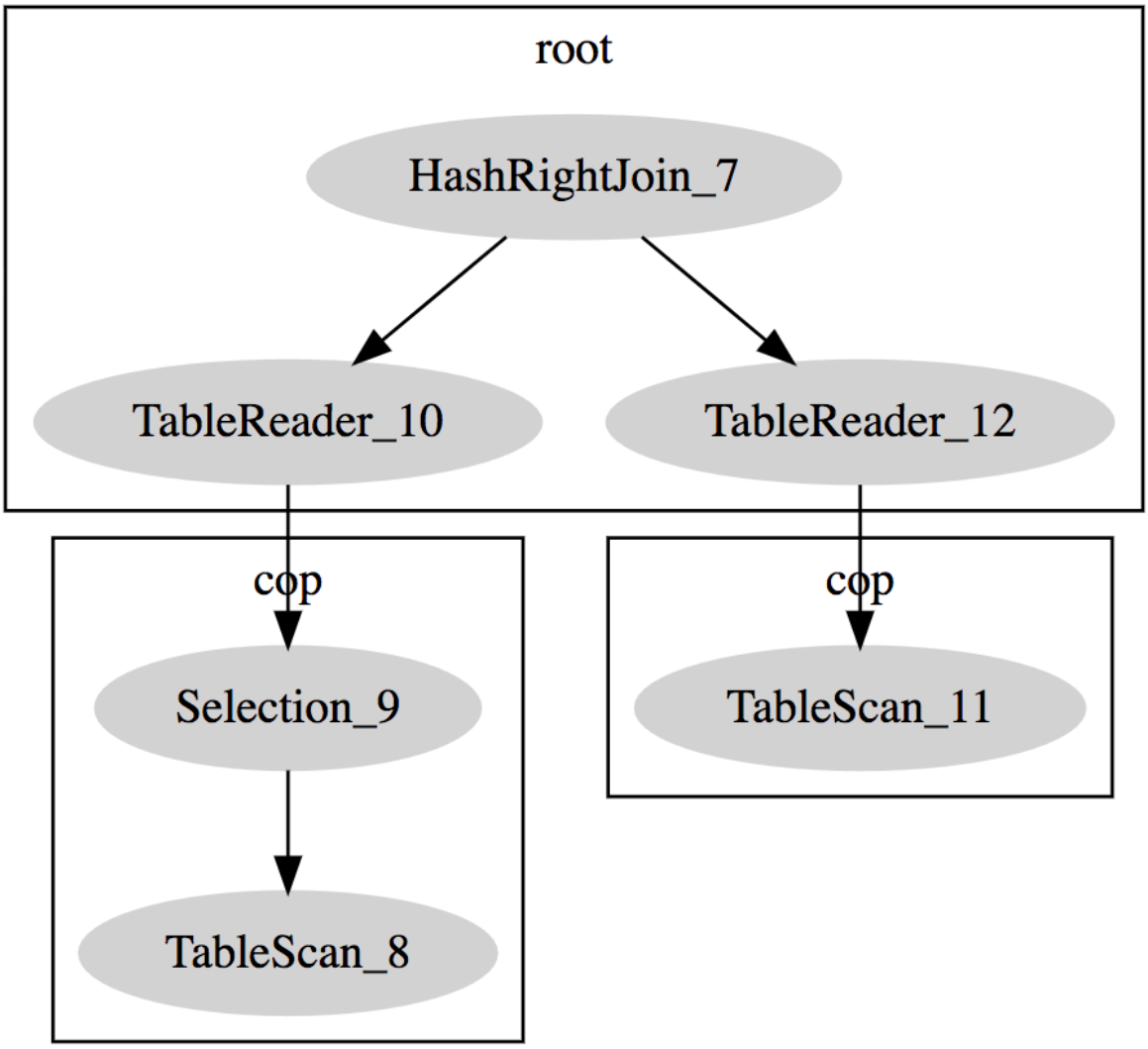


Figure 357: Explain Dot

14.10.2.54.4 MySQL compatibility

- Both the format of `EXPLAIN` and the potential execution plans in TiDB differ substantially from MySQL.
- TiDB does not support the `FORMAT=JSON` or `FORMAT=TREE` options.

`EXPLAIN FOR CONNECTION`

`EXPLAIN FOR CONNECTION` is used to get the execution plan of the currently executed SQL query or the last executed SQL query in a connection. The output format is the same as that of `EXPLAIN`. However, the implementation of `EXPLAIN FOR CONNECTION` in TiDB is

different from that in MySQL. Their differences (apart from the output format) are listed as follows:

- If the connection is sleeping, MySQL returns an empty result, while TiDB returns the last executed query plan.
- If you try to get the execution plan of the current session, MySQL returns an error, while TiDB returns the result normally.
- MySQL requires the login user to be the same as the connection being queried, or the login user has the **PROCESS** privilege; while TiDB requires the login user to be the same as the connection being queried, or the login user has the **SUPER** privilege.

14.10.2.54.5 See also

- Understanding the Query Execution Plan
- EXPLAIN ANALYZE
- ANALYZE TABLE
- TRACE

14.10.2.55 FLASHBACK TABLE

The `FLASHBACK TABLE` syntax is introduced since TiDB 4.0. You can use the `FLASHBACK TABLE` statement to restore the tables and data dropped by the `DROP` or `TRUNCATE` operation within the Garbage Collection (GC) lifetime.

The system variable `tidb_gc_life_time` (default: `10m0s`) defines the retention time of earlier versions of rows. The current `safePoint` of where garbage collection has been performed up to can be obtained with the following query:

```
SELECT * FROM mysql.tidb WHERE variable_name = 'tikv_gc_safe_point';
```

As long as the table is dropped by `DROP` or `TRUNCATE` statements after the `tikv_gc_safe_point` time, you can restore the table using the `FLASHBACK TABLE` statement.

14.10.2.55.1 Syntax

```
FLASHBACK TABLE table_name [TO other_table_name]
```

14.10.2.55.2 Synopsis

```
FlashbackTableStmt ::=
    'FLASHBACK' 'TABLE' TableName FlashbackToNewName

TableName ::=
    Identifier ( '.' Identifier )?
```

```
FlashbackToNewName ::=
    ( 'TO' Identifier )?
```

14.10.2.55.3 Notes

If a table is dropped and the GC lifetime has passed, you can no longer use the `FLASHBACK TABLE` statement to recover the dropped data. Otherwise, an error like `Can't find dropped / truncated table 't' in GC safe point 2020-03-16 16:34:52 +0800 CST` will be returned.

Pay attention to the following conditions and requirements when you enable TiDB Binlog and use the `FLASHBACK TABLE` statement:

- The downstream secondary cluster must also support `FLASHBACK TABLE`.
- The GC lifetime of the secondary cluster must be longer than that of the primary cluster.
- The delay of replication between the upstream and downstream might also cause the failure to recover data to the downstream.
- If an error occurs when TiDB Binlog is replicating a table, you need to filter that table in TiDB Binlog and manually import all data of that table.

14.10.2.55.4 Example

- Recover the table data dropped by the `DROP` operation:

```
DROP TABLE t;
```

```
FLASHBACK TABLE t;
```

- Recover the table data dropped by the `TRUNCATE` operation. Because the truncated table `t` still exists, you need to rename the table `t` to be recovered. Otherwise, an error will be returned because the table `t` already exists.

```
TRUNCATE TABLE t;
```

```
FLASHBACK TABLE t TO t1;
```

14.10.2.55.5 Implementation principle

When deleting a table, TiDB only deletes the table metadata, and writes the table data (row data and index data) to be deleted to the `mysql.gc_delete_range` table. The GC Worker in the TiDB background periodically removes from the `mysql.gc_delete_range` table the keys that exceed the GC lifetime.

Therefore, to recover a table, you only need to recover the table metadata and delete the corresponding row record in the `mysql.gc_delete_range` table before the GC Worker deletes the table data. You can use a snapshot read of TiDB to recover the table metadata. For details of snapshot read, refer to Read Historical Data.

The following is the working process of `FLASHBACK TABLE t TO t1`:

1. TiDB searches the recent DDL history jobs and locates the first DDL operation of the `DROP TABLE` or the `truncate table` type on table `t`. If TiDB fails to locate one, an error is returned.
2. TiDB checks whether the starting time of the DDL job is before `tikv_gc_safe_point`. If it is before `tikv_gc_safe_point`, it means that the table dropped by the `DROP` or `TRUNCATE` operation has been cleaned up by the GC and an error is returned.
3. TiDB uses the starting time of the DDL job as the snapshot to read historical data and read table metadata.
4. TiDB deletes GC tasks related to table `t` in `mysql.gc_delete_range`.
5. TiDB changes `name` in the table's metadata to `t1`, and uses this metadata to create a new table. Note that only the table name is changed but not the table ID. The table ID is the same as that of the previously dropped table `t`.

From the above process, you can see that TiDB always operates on the metadata of the table, and the user data of the table has never been modified. The restored table `t1` has the same ID as the previously dropped table `t`, so `t1` can read the user data of `t`.

> **Note:**
>
> You cannot use `FLASHBACK` statements to restore the same deleted table multiple times, because the ID of the restored table is the same ID of the dropped table, and TiDB requires that all existing tables must have a globally unique table ID.

The `FLASHBACK TABLE` operation is done by TiDB obtaining the table metadata through snapshot read, and then going through the process of table creation similar to `CREATE TABLE`. Therefore, `FLASHBACK TABLE` is, in essence, a kind of DDL operation.

14.10.2.55.6 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.56 FLUSH PRIVILEGES

This statement triggers TiDB to reload the in-memory copy of privileges from the privilege tables. You should execute `FLUSH PRIVILEGES` after making manual edits to tables such as `mysql.user`. Executing this statement is not required after using privilege statements such as `GRANT` or `REVOKE`. Executing this statement requires the `RELOAD` privilege.

14.10.2.56.1 Synopsis

```
FlushStmt ::=
    'FLUSH' NoWriteToBinLogAliasOpt FlushOption

NoWriteToBinLogAliasOpt ::=
    ( 'NO_WRITE_TO_BINLOG' | 'LOCAL' )?

FlushOption ::=
    'PRIVILEGES'
|   'STATUS'
|    'TIDB' 'PLUGINS' PluginNameList
|    'HOSTS'
|   LogTypeOpt 'LOGS'
|   TableOrTables TableNameListOpt WithReadLockOpt
```

14.10.2.56.2 Examples

```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)
```

14.10.2.56.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.56.4 See also

- SHOW GRANTS
- Privilege Management

14.10.2.57 FLUSH STATUS

This statement is included for compatibility with MySQL. It has no effect on TiDB, which uses Prometheus and Grafana for centralized metrics collection instead of `SHOW STATUS`.

14.10.2.57.1 Synopsis

```
FlushStmt ::=
    'FLUSH' NoWriteToBinLogAliasOpt FlushOption

NoWriteToBinLogAliasOpt ::=
    ( 'NO_WRITE_TO_BINLOG' | 'LOCAL' )?
```

```
FlushOption ::=
    'PRIVILEGES'
|   'STATUS'
|    'TIDB' 'PLUGINS' PluginNameList
|    'HOSTS'
|   LogTypeOpt 'LOGS'
|   TableOrTables TableNameListOpt WithReadLockOpt
```

14.10.2.57.2 Examples

```
mysql> show status;
+--------------------+--------------------------------------+
| Variable_name    | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher_list  |                                      |
| server_id        | 93e2e07d-6bb4-4a1b-90b7-e035fae154fe |
| ddl_schema_version | 141                                |
| Ssl_verify_mode  | 0                                    |
| Ssl_version      |                                      |
| Ssl_cipher       |                                      |
+--------------------+--------------------------------------+
6 rows in set (0.01 sec)

mysql> show global status;
+--------------------+--------------------------------------+
| Variable_name    | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher       |                                      |
| Ssl_cipher_list  |                                      |
| Ssl_verify_mode  | 0                                    |
| Ssl_version      |                                      |
| server_id        | 93e2e07d-6bb4-4a1b-90b7-e035fae154fe |
| ddl_schema_version | 141                                |
+--------------------+--------------------------------------+
6 rows in set (0.00 sec)

mysql> flush status;
Query OK, 0 rows affected (0.00 sec)

mysql> show status;
+--------------------+--------------------------------------+
| Variable_name    | Value                                |
+--------------------+--------------------------------------+
```

```
| Ssl_cipher       |                                      |
| Ssl_cipher_list  |                                      |
| Ssl_verify_mode  | 0                                    |
| Ssl_version      |                                      |
| ddl_schema_version | 141                                |
| server_id        | 93e2e07d-6bb4-4a1b-90b7-e035fae154fe |
+------------------+--------------------------------------+
6 rows in set (0.00 sec)
```

14.10.2.57.3 MySQL compatibility

- This statement is included only for compatibility with MySQL.

14.10.2.57.4 See also

- SHOW [GLOBAL|SESSION] STATUS

14.10.2.58 FLUSH TABLES

This statement is included for compatibility with MySQL. It has no effective usage in TiDB.

14.10.2.58.1 Synopsis

```
FlushStmt ::=
    'FLUSH' NoWriteToBinLogAliasOpt FlushOption

NoWriteToBinLogAliasOpt ::=
    ( 'NO_WRITE_TO_BINLOG' | 'LOCAL' )?

FlushOption ::=
    'PRIVILEGES'
|   'STATUS'
|    'TIDB' 'PLUGINS' PluginNameList
|    'HOSTS'
|    LogTypeOpt 'LOGS'
|    TableOrTables TableNameListOpt WithReadLockOpt

LogTypeOpt ::=
    ( 'BINARY' | 'ENGINE' | 'ERROR' | 'GENERAL' | 'SLOW' )?

TableOrTables ::=
    'TABLE'
```

```
|   'TABLES'

TableNameListOpt ::=
    TableNameList?

WithReadLockOpt ::=
    ( 'WITH' 'READ' 'LOCK' )?
```

14.10.2.58.2 Examples

```
mysql> FLUSH TABLES;
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH TABLES WITH READ LOCK;
ERROR 1105 (HY000): FLUSH TABLES WITH READ LOCK is not supported. Please
    ↪ use @@tidb_snapshot
```

14.10.2.58.3 MySQL compatibility

- TiDB does not have a concept of table cache as in MySQL. Thus, `FLUSH TABLES` is parsed but ignored in TiDB for compatibility.
- The statement `FLUSH TABLES WITH READ LOCK` produces an error, as TiDB does not currently support locking tables. It is recommended to use Historical reads for this purpose instead.

14.10.2.58.4 See also

- Read historical data

14.10.2.59 `GRANT <privileges>`

This statement allocates privileges to a pre-existing user in TiDB. The privilege system in TiDB follows MySQL, where credentials are assigned based on a database/table pattern. Executing this statement requires the `GRANT OPTION` privilege and all privileges you allocate.

14.10.2.59.1 Synopsis

```
GrantStmt ::=
    'GRANT' PrivElemList 'ON' ObjectType PrivLevel 'TO' UserSpecList
       ↪ RequireClauseOpt WithGrantOptionOpt

PrivElemList ::=
    PrivElem ( ',' PrivElem )*
```

```
PrivElem ::=
    PrivType ( '(' ColumnNameList ')' )?

PrivType ::=
    'ALL' 'PRIVILEGES'?
|    'ALTER' 'ROUTINE'?
|   'CREATE' ( 'USER' | 'TEMPORARY' 'TABLES' | 'VIEW' | 'ROLE' | 'ROUTINE' )
    ↪ ?
|   'TRIGGER'
|   'DELETE'
|   'DROP' 'ROLE'?
|    'PROCESS'
|   'EXECUTE'
|   'INDEX'
|   'INSERT'
|   'SELECT'
|   'SUPER'
|   'SHOW' ( 'DATABASES' | 'VIEW' )
|   'UPDATE'
|    'GRANT' 'OPTION'
|    'REFERENCES'
|    'REPLICATION' ( 'SLAVE' | 'CLIENT' )
|    'USAGE'
|   'RELOAD'
|   'FILE'
|   'CONFIG'
|   'LOCK' 'TABLES'
|   'EVENT'
|   'SHUTDOWN'

ObjectType ::=
    'TABLE'?

PrivLevel ::=
    '*' ( '.' '*' )?
|    Identifier ( '.' ( '*' | Identifier ) )?

UserSpecList ::=
    UserSpec ( ',' UserSpec )*
```

14.10.2.59.2 Examples

```
mysql> CREATE USER 'newuser' IDENTIFIED BY 'mypassword';
```

```
Query OK, 1 row affected (0.02 sec)

mysql> GRANT ALL ON test.* TO 'newuser';
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW GRANTS FOR 'newuser';
+-------------------------------------------------+
| Grants for newuser@%                            |
+-------------------------------------------------+
| GRANT USAGE ON *.* TO 'newuser'@'%'             |
| GRANT ALL PRIVILEGES ON test.* TO 'newuser'@'%' |
+-------------------------------------------------+
2 rows in set (0.00 sec)
```

14.10.2.59.3 MySQL compatibility

- Similar to MySQL, the `USAGE` privilege denotes the ability to log into a TiDB server.
- Column level privileges are not currently supported.
- Similar to MySQL, when the `NO_AUTO_CREATE_USER` sql mode is not present, the `GRANT` statement will automatically create a new user with an empty password when a user does not exist. Removing this sql-mode (it is enabled by default) presents a security risk.

14.10.2.59.4 See also

- GRANT <role>
- REVOKE <privileges>
- SHOW GRANTS

- Privilege Management

14.10.2.60 GRANT <role>

Assigns a previously created role to an existing user. The user can use then use the statement `SET ROLE <rolename>` to assume the privileges of the role, or `SET ROLE ALL` to assume all roles that have been assigned.

14.10.2.60.1 Synopsis

```
GrantRoleStmt ::=
    'GRANT' RolenameList 'TO' UsernameList

RolenameList ::=
```

```
    Rolename ( ',' Rolename )*

UsernameList ::=
    Username ( ',' Username )*
```

14.10.2.60.2 Examples

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

Create a new role `analyticsteam` and a new user `jennifer`:

```
CREATE ROLE analyticsteam;
Query OK, 0 rows affected (0.02 sec)

GRANT SELECT ON test.* TO analyticsteam;
Query OK, 0 rows affected (0.02 sec)

CREATE USER jennifer;
Query OK, 0 rows affected (0.01 sec)

GRANT analyticsteam TO jennifer;
Query OK, 0 rows affected (0.01 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

Note that by default `jennifer` needs to execute `SET ROLE analyticsteam` in order to be able to use the privileges associated with the `analyticsteam` role:

```
SHOW GRANTS;
+-----------------------------------------------+
| Grants for User                               |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'          |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-----------------------------------------------+
2 rows in set (0.00 sec)

SHOW TABLES in test;
ERROR 1044 (42000): Access denied for user 'jennifer'@'%' to database 'test'
SET ROLE analyticsteam;
Query OK, 0 rows affected (0.00 sec)
```

```
SHOW GRANTS;
+-----------------------------------------+
| Grants for User                         |
+-----------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'    |
| GRANT SELECT ON test.* TO 'jennifer'@'%' |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-----------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

The statement `SET DEFAULT ROLE` can be used to associate the role `analyticsteam` to `jennifer`:

```
SET DEFAULT ROLE analyticsteam TO jennifer;
Query OK, 0 rows affected (0.02 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

After this, the user `jennifer` has the privileges associated with the role `analyticsteam` and `jennifer` does not have to execute the statement `SET ROLE`:

```
SHOW GRANTS;
+-----------------------------------------+
| Grants for User                         |
+-----------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'    |
| GRANT SELECT ON test.* TO 'jennifer'@'%' |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-----------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
```

```
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

14.10.2.60.3 MySQL compatibility

This statement is understood to be fully compatible with roles, which are a feature of MySQL 8.0. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.60.4 See also

- GRANT <privileges>
- CREATE ROLE
- DROP ROLE
- REVOKE <role>
- SET ROLE
- SET DEFAULT ROLE

- Role-Based Access Control

14.10.2.61 INSERT

This statement inserts new rows into a table.

14.10.2.61.1 Synopsis

```
InsertIntoStmt ::=
    'INSERT' TableOptimizerHints PriorityOpt IgnoreOptional IntoOpt
       ↪ TableName PartitionNameListOpt InsertValues OnDuplicateKeyUpdate

TableOptimizerHints ::=
    hintComment?

PriorityOpt ::=
    ( 'LOW_PRIORITY' | 'HIGH_PRIORITY' | 'DELAYED' )?

IgnoreOptional ::=
    'IGNORE'?

IntoOpt ::= 'INTO'?

TableName ::=
```

```
    Identifier ( '.' Identifier )?

PartitionNameListOpt ::=
    ( 'PARTITION' '(' Identifier ( ',' Identifier )* ')' )?

InsertValues ::=
    '(' ( ColumnNameListOpt ')' ( ValueSym ValuesList | SelectStmt | '('
       ↪ SelectStmt ')' | UnionStmt ) | SelectStmt ')' )
|   ValueSym ValuesList
|   SelectStmt
|   UnionStmt
|   'SET' ColumnSetValue? ( ',' ColumnSetValue )*

OnDuplicateKeyUpdate ::=
    ( 'ON' 'DUPLICATE' 'KEY' 'UPDATE' AssignmentList )?
```

14.10.2.61.2 Examples

```
mysql> CREATE TABLE t1 (a INT);
Query OK, 0 rows affected (0.11 sec)

mysql> CREATE TABLE t2 LIKE t1;
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 VALUES (1);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t2 SELECT * FROM t1;
Query OK, 2 rows affected (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+------+
| a    |
+------+
|    1 |
|    1 |
+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
```

```
+------+
| a    |
+------+
|    1 |
|    1 |
+------+
2 rows in set (0.00 sec)

mysql> INSERT INTO t2 VALUES (2),(3),(4);
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t2;
+------+
| a    |
+------+
|    1 |
|    1 |
|    2 |
|    3 |
|    4 |
+------+
5 rows in set (0.00 sec)
```

14.10.2.61.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.61.4 See also

- DELETE
- SELECT
- UPDATE
- REPLACE

14.10.2.62 KILL

The `KILL` statement is used to terminate a connection in any TiDB instance in the current TiDB cluster.

14.10.2.62.1 Synopsis

```
KillStmt ::= 'KILL' 'TIDB'? ( 'CONNECTION' | 'QUERY' )? CONNECTION_ID
```

14.10.2.62.2 Examples

The following example shows how to get all active queries in the current cluster and terminate any one of them.

```
SELECT ID, USER, INSTANCE, INFO FROM INFORMATION_SCHEMA.CLUSTER_PROCESSLIST
    ↪ ;
```

```
+--------------------+------+----------------+-----------------------------------------
    ↪
| ID                 | USER | INSTANCE       | INFO
    ↪                                                                              |
+--------------------+------+----------------+-----------------------------------------
    ↪
| 8306449708033769879 | root | 127.0.0.1:10082 | select sleep(30), 'foo'
    ↪                                                 |
| 5857102839209263511 | root | 127.0.0.1:10080 | select sleep(50)
    ↪                                                  |
| 5857102839209263513 | root | 127.0.0.1:10080 | SELECT ID, USER, INSTANCE,
    ↪ INFO FROM INFORMATION_SCHEMA.CLUSTER_PROCESSLIST |
+--------------------+------+----------------+-----------------------------------------
    ↪
```

```
KILL 5857102839209263511;
```

```
Query OK, 0 rows affected (0.00 sec)
```

14.10.2.62.3 MySQL compatibility

- The `KILL` statement of MySQL can only terminate a connection in the currently connected MySQL instance, while the `KILL` statement of TiDB can terminate a connection in any TiDB instance in the entire cluster.
- Currently, using the MySQL command line ctrl+c to terminate a query or connection in TiDB is not supported.

14.10.2.62.4 Behavior change descriptions

Starting from v6.1.0, TiDB supports the Global Kill feature, which is enabled by default and controlled by the `enable-global-kill` configuration.

When the Global Kill feature is enabled, both `KILL` and `KILL TIDB` statements can terminate queries or connections across instances so you do not need to worry about erroneously terminating queries or connections. When you use a client to connect to any TiDB instance and execute the `KILL` or `KILL TIDB` statement, the statement will be forwarded to the target TiDB instance. If there is a proxy between the client and the TiDB cluster, the `KILL` and `KILL TIDB` statements will also be forwarded to the target TiDB instance for execution.

If the Global Kill feature is not enabled or you are using a TiDB version earlier than v6.1.0, note the following:

- By default, `KILL` is not compatible with MySQL. This helps prevent against a case of a connection being terminated by a wrong TiDB server, because it is common to place multiple TiDB servers behind a load balancer. To terminate other connections on the currently connected TiDB instance, you need to add the `TIDB` suffix explicitly by executing the `KILL TIDB` statement.

- It is **STRONGLY NOT RECOMMENDED** to set `compatible-kill-query = true` in your configuration file UNLESS you are certain that clients will be always connected to the same TiDB instance. This is because pressing ctrl+c in the default MySQL client opens a new connection in which `KILL` is executed. If there is a proxy between the client and the TiDB cluster, the new connection might be routed to a different TiDB instance, which possibly kills a different session by mistake.

- The `KILL TIDB` statement is a TiDB extension. The feature of this statement is similar to the MySQL `KILL [CONNECTION|QUERY]` command and the MySQL command line ctrl+c. It is safe to use `KILL TIDB` on the same TiDB instance.

14.10.2.62.5 See also

- SHOW [FULL] PROCESSLIST
- CLUSTER_PROCESSLIST

14.10.2.63 LOAD DATA

The `LOAD DATA` statement batch loads data into a TiDB table.

14.10.2.63.1 Synopsis

```
LoadDataStmt ::=
    'LOAD' 'DATA' LocalOpt 'INFILE' stringLit DuplicateOpt 'INTO' 'TABLE'
        ↪ TableName CharsetOpt Fields Lines IgnoreLines
        ↪ ColumnNameOrUserVarListOptWithBrackets LoadDataSetSpecOpt
```

14.10.2.63.2 Parameters

`LocalOpt`

You can specify that the imported data file is located on the client or on the server by configuring the `LocalOpt` parameter. Currently, TiDB only supports data import from the client. Therefore, when importing data, set the value of `LocalOpt` to `Local`.

`Fields` and `Lines`

You can specify how to process the data format by configuring the `Fields` and `Lines` parameters.

- `FIELDS TERMINATED BY`: Specifies the separating character of each data.
- `FIELDS ENCLOSED BY`: Specifies the enclosing character of each data.
- `LINES TERMINATED BY`: Specifies the line terminator, if you want to end a line with a certain character.

Take the following data format as an example:

```
"bob","20","street 1"\r\n
"alice","33","street 1"\r\n
```

If you want to extract `bob`, 20, and `street 1`, specify the separating character as `','`, and the enclosing character as `'\"'`:

```
FIELDS TERMINATED BY ',' ENCLOSED BY '\"' LINES TERMINATED BY '\r\n'
```

If you do not specify the parameters above, the imported data is processed in the following way by default:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY ''
LINES TERMINATED BY '\n'
```

`IGNORE number LINES`

You can ignore the first `number` lines of a file by configuring the `IGNORE number LINES` parameter. For example, if you configure `IGNORE 1 LINES`, the first line of a file is ignored.

14.10.2.63.3 Examples

```
CREATE TABLE trips (
    trip_id bigint NOT NULL PRIMARY KEY AUTO_INCREMENT,
    duration integer not null,
    start_date datetime,
    end_date datetime,
    start_station_number integer,
    start_station varchar(255),
    end_station_number integer,
    end_station varchar(255),
    bike_number varchar(255),
    member_type varchar(255)
    );
```

```
Query OK, 0 rows affected (0.14 sec)
```

The following example imports data using `LOAD DATA`. Comma is specified as the separating character. The double quotation marks that enclose the data is ignored. The first line of the file is ignored.

If you see the error message `ERROR 1148 (42000): the used command is not allowed with this TiDB version`, refer to ERROR 1148 (42000): the used command is not allowed with this TiDB version.

```
LOAD DATA LOCAL INFILE '/mnt/evo970/data-sets/bikeshare-data/2017Q4-
    ↪ capitalbikeshare-tripdata.csv' INTO TABLE trips FIELDS TERMINATED BY
    ↪ ',' ENCLOSED BY '\"' LINES TERMINATED BY '\r\n' IGNORE 1 LINES (
    ↪ duration, start_date, end_date, start_station_number, start_station,
    ↪ end_station_number, end_station, bike_number, member_type);
```

```
Query OK, 815264 rows affected (39.63 sec)
Records: 815264 Deleted: 0 Skipped: 0 Warnings: 0
```

`LOAD DATA` also supports using hexadecimal ASCII character expressions or binary ASCII character expressions as the parameters for `FIELDS ENCLOSED BY` and `FIELDS TERMINATED BY`. See the following example:

```
LOAD DATA LOCAL INFILE '/mnt/evo970/data-sets/bikeshare-data/2017Q4-
    ↪ capitalbikeshare-tripdata.csv' INTO TABLE trips FIELDS TERMINATED BY
    ↪ x'2c' ENCLOSED BY b'100010' LINES TERMINATED BY '\r\n' IGNORE 1 LINES
    ↪  (duration, start_date, end_date, start_station_number, start_station
    ↪ , end_station_number, end_station, bike_number, member_type);
```

In the above example, `x'2c'` is the hexadecimal representation of the `,` character and `b'100010'` is the binary representation of the `"` character.

14.10.2.63.4 MySQL compatibility

The syntax of the `LOAD DATA` statement is compatible with that of MySQL, except for character set options which are parsed but ignored. If you find any syntax compatibility difference, you can report it via an issue on GitHub.

Note:

- For versions earlier than TiDB v4.0.0, `LOAD DATA` committed every 20000 rows. By default, TiDB now commits all rows in one transaction. This can result in the error `ERROR 8004 (HY000)at line 1: Transaction is too large, size: 100000058` after upgrading from TiDB 4.0 or earlier versions. The recommended way to resolve this error is to increase the `txn-total-size-limit` value in your `tidb.toml` file. If you are unable to increase this limit, you can also restore the previous behavior by setting `tidb_dml_batch_size` to 20000.

- No matter how many rows are committed in a transaction, `LOAD DATA` is not rolled back by the `ROLLBACK` statement in an explicit transaction.
- The `LOAD DATA` statement is always executed in optimistic transaction mode, regardless of the TiDB transaction mode configuration.

14.10.2.63.5 See also

- INSERT
- Import Example Database
- TiDB Optimistic Transaction Model
- TiDB Pessimistic Transaction Mode
- TiDB Lightning

14.10.2.64 LOAD STATS

The `LOAD STATS` statement is used to load the statistics into TiDB.

14.10.2.64.1 Synopsis

```
LoadStatsStmt ::=
    'LOAD' 'STATS' stringLit
```

14.10.2.64.2 Examples

You can access the address `http://${tidb-server-ip}:${tidb-server-status-port}/stats/dump/${db_name}/${table_name}` to download the TiDB instance's statistics.

You can also use `LOAD STATS ${stats_path}` to load the specific statistics file.

The `${stats_path}` can be an absolute path or a relative path. If you use a relative path, the corresponding file is found starting from the path where `tidb-server` is started. Here is an example:

```
LOAD STATS '/tmp/stats.json';
```

```
Query OK, 0 rows affected (0.00 sec)
```

14.10.2.64.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.64.4 See also

- Statistics

14.10.2.65 LOCK TABLES and UNLOCK TABLES

> **Warning:**
>
> `LOCK TABLES` and `UNLOCK TABLES` are experimental features for the current version. It is not recommended to use it in the production environment.

TiDB enables client sessions to acquire table locks for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

`LOCK TABLES` acquires table locks for the current client session. If you have the `LOCK TABLES` and `SELECT` privileges for each object to be locked, you can acquire table locks for common tables.

`UNLOCK TABLES` explicitly releases any table locks held by the current session. `LOCK TABLES` implicitly releases all table locks held by the current session before acquiring new locks.

A table lock protects against reads or writes by other sessions. A session that holds a `WRITE` lock can perform table-level operations such as `DROP TABLE` or `TRUNCATE TABLE`.

> **Warning:**
>
> To enable table locks, you need to set `enable-table-lock` to `true` in the configuration files of all TiDB instances.

14.10.2.65.1 Synopsis

```
LockTablesDef
        ::= 'LOCK' ( 'TABLES' | 'TABLE' ) TableName LockType ( ',' TableName
           ↪  LockType)*


UnlockTablesDef
        ::= 'UNLOCK' 'TABLES'

LockType
        ::= 'READ' ('LOCAL')?
          | 'WRITE' ('LOCAL')?
```

14.10.2.65.2 Acquire table locks

You can acquire table locks within the current session by using the `LOCK TABLES` statement. The following lock types are available:

`READ` lock:

- The session that holds this lock can read the table, but cannot write it.
- Multiple sessions can acquire a `READ` lock from the same table at the same time.
- Other sessions can read the table without explicitly acquiring a `READ` lock.

The `READ LOCAL` lock is only for syntax compatibility with MySQL and is not supported.

`WRITE` lock:

- The session that holds this lock can read and write the table.
- Only the session that holds this lock can access the table. No other sessions can access it until the lock is released.

`WRITE LOCAL` lock:

- The session that holds this lock can read and write the table.
- Only the session that holds this lock can access the table. Other sessions can read the table, but cannot write it.

If the lock that the `LOCK TABLES` statement needs is held by another session, the `LOCK TABLES` statement must wait, and an error is returned upon the execution of this statement, for example:

```
> LOCK TABLES t1 READ;
ERROR 8020 (HY000): Table 't1' was locked in WRITE by server: f4799bcb-cad7
    ↪ -4285-8a6d-23d3555173f1_session: 2199023255959
```

The preceding error message indicates that the session with ID `2199023255959` in TiDB `f4799bcb-cad7-4285-8a6d-23d3555173f1` already holds a `WRITE` lock on table `t1`. Therefore, the current session cannot acquire a `READ` lock on table `t1`.

You cannot acquire the same table lock multiple times in a single `LOCK TABLES` statement.

```
> LOCK TABLES t WRITE, t READ;
ERROR 1066 (42000): Not unique table/alias: 't'
```

14.10.2.65.3 Release table locks

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly or implicitly.

- A session can release its locks explicitly with `UNLOCK TABLES`.
- If a session issues a `LOCK TABLES` statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are acquired.

If the connection for a client session terminates, whether normally or abnormally, TiDB implicitly releases all table locks held by the session. If the client reconnects, the locks are no longer in effect. For this reason, it is not recommended to enable auto-reconnection on the client. If you enable auto-reconnection, the client is not notified when reconnection occurs, and all table locks or current transactions are lost. By contrast, with auto-reconnection disabled, if the connection drops, an error occurs when the next statement is issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction.

14.10.2.65.4 Table-locking restrictions and conditions

You can safely use `KILL` to terminate a session that holds a table lock.

You cannot acquire table locks on tables in the following databases:

- `INFORMATION_SCHEMA`
- `PERFORMANCE_SCHEMA`
- `METRICS_SCHEMA`
- `mysql`

14.10.2.65.5 MySQL compatibility

Table lock acquisition

- In TiDB, if session A has already held a table lock, an error is returned if session B attempts to write to the table. In MySQL, the write request of session B is blocked until session A releases the table lock, and requests for locking the table from other sessions are blocked until the current session releases the `WRITE` lock.
- In TiDB, if the lock that the `LOCK TABLES` statement needs is held by another session, the `LOCK TABLES` statement must wait, and an error is returned upon the execution of this statement. In MySQL, this statement is blocked until the lock is acquired.
- In TiDB, the `LOCK TABLES` statement is effective in the whole cluster. In MySQL, this statement is effective only in the current MySQL server, and is not compatible with the NDB cluster.

Table lock release

When a transaction is explicitly started in a TiDB session (for example, with the `BEGIN` statement), TiDB does not implicitly release the table locks held by the session; but MySQL does.

14.10.2.66 MODIFY COLUMN

The `ALTER TABLE.. MODIFY COLUMN` statement modifies a column on an existing table. The modification can include changing the data type and attributes. To rename at the same time, use the `CHANGE COLUMN` statement instead.

Since v5.1.0, TiDB has supported changes of data types for Reorg data, including but not limited to:

- Changing `VARCHAR` to `BIGINT`
- Modifying the `DECIMAL` precision
- Compressing the length of `VARCHAR(10)` to `VARCHAR(5)`

14.10.2.66.1 Synopsis

```
AlterTableStmt
        ::= 'ALTER' 'IGNORE'? 'TABLE' TableName ModifyColumnSpec ( ','
           ↪ ModifyColumnSpec )*

ModifyColumnSpec
        ::= 'MODIFY' ColumnKeywordOpt 'IF EXISTS' ColumnName ColumnType
           ↪ ColumnOption* ( 'FIRST' | 'AFTER' ColumnName )?

ColumnType
        ::= NumericType
          | StringType
          | DateAndTimeType
          | 'SERIAL'

ColumnOption
        ::= 'NOT'? 'NULL'
          | 'AUTO_INCREMENT'
          | 'PRIMARY'? 'KEY' ( 'CLUSTERED' | 'NONCLUSTERED' )?
          | 'UNIQUE' 'KEY'?
          | 'DEFAULT' ( NowSymOptionFraction | SignedLiteral |
             ↪ NextValueForSequence )
          | 'SERIAL' 'DEFAULT' 'VALUE'
          | 'ON' 'UPDATE' NowSymOptionFraction
          | 'COMMENT' stringLit
          | ( 'CONSTRAINT' Identifier? )? 'CHECK' '(' Expression ')' ( 'NOT
             ↪ '? ( 'ENFORCED' | 'NULL' ) )?
          | 'GENERATED' 'ALWAYS' 'AS' '(' Expression ')' ( 'VIRTUAL' | '
             ↪ STORED' )?
          | 'REFERENCES' TableName ( '(' IndexPartSpecificationList ')' )?
             ↪ Match? OnDeleteUpdateOpt
          | 'COLLATE' CollationName
```

```
          | 'COLUMN_FORMAT' ColumnFormat
          | 'STORAGE' StorageMedia
          | 'AUTO_RANDOM' ( '(' LengthNum ')' )?

ColumnName ::=
    Identifier ( '.' Identifier ( '.' Identifier )? )?
```

14.10.2.66.2 Examples

Meta-Only Change

```
CREATE TABLE t1 (id int not null primary key AUTO_INCREMENT, col1 INT);
```

```
Query OK, 0 rows affected (0.11 sec)
```

```
INSERT INTO t1 (col1) VALUES (1),(2),(3),(4),(5);
```

```
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

```
Query OK, 0 rows affected (0.09 sec)
```

```
SHOW CREATE TABLE t1\G;
```

```
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `col1` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin AUTO_INCREMENT
    ↪ =30001
1 row in set (0.00 sec)
```

Reorg-Data Change

```
CREATE TABLE t1 (id int not null primary key AUTO_INCREMENT, col1 INT);
```

```
Query OK, 0 rows affected (0.11 sec)
```

```
INSERT INTO t1 (col1) VALUES (12345),(67890);
```

```
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
ALTER TABLE t1 MODIFY col1 VARCHAR(5);
```

```
Query OK, 0 rows affected (2.52 sec)
```

```
SHOW CREATE TABLE t1\G;
```

```
*************************** 1. row ***************************
       Table: t1
CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `col1` varchar(5) DEFAULT NULL,
  PRIMARY KEY (`id`) /*T![clustered_index] CLUSTERED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin AUTO_INCREMENT
    ↪ =30001
1 row in set (0.00 sec)
```

Note:

- TiDB returns an error when the changed data type conflicts with an existing data row. In the above example, TiDB returns the following error:

```
alter table t1 modify column col1 varchar(4); ERROR 1406
↪ (22001): Data Too Long, field len 4, data len 5
```

- Due to the compatibility with the Async Commit feature, the DDL statement waits for a period of time (about 2.5s) before starting to process into Reorg Data.

```
Query OK, 0 rows affected (2.52 sec)
```

14.10.2.66.3 MySQL compatibility

- Does not support modifying multiple columns using a single `ALTER TABLE` statement. For example:

```
ALTER TABLE t1 MODIFY col1 BIGINT, MODIFY id BIGINT NOT NULL;
ERROR 1105 (HY000): Unsupported multi schema change
```

- Does not support modifying the Reorg-Data types on the primary key columns but supports modifying the Meta-Only types. For example:

```
CREATE TABLE t (a int primary key);
ALTER TABLE t MODIFY COLUMN a VARCHAR(10);
ERROR 8200 (HY000): Unsupported modify column: column has primary key
    ↪ flag
```

```
CREATE TABLE t (a int primary key);
ALTER TABLE t MODIFY COLUMN a INT(10) UNSIGNED;
ERROR 8200 (HY000): Unsupported modify column: column has primary key
    ↪ flag
```

```
CREATE TABLE t (a int primary key);
ALTER TABLE t MODIFY COLUMN a bigint;
Query OK, 0 rows affected (0.01 sec)
```

- Does not support modifying the column types on generated columns. For example:

```
CREATE TABLE t (a INT, b INT as (a+1));
ALTER TABLE t MODIFY COLUMN b VARCHAR(10);
ERROR 8200 (HY000): Unsupported modify column: column is generated
```

- Does not support modifying the column types on the partitioned tables. For example:

```
CREATE TABLE t (c1 INT, c2 INT, c3 INT) partition by range columns(c1)
    ↪  ( partition p0 values less than (10), partition p1 values less
    ↪ than (maxvalue));
ALTER TABLE t MODIFY COLUMN c1 DATETIME;
ERROR 8200 (HY000): Unsupported modify column: table is partition
    ↪ table
```

- Does not support modifying some data types (for example, some TIME types, Bit, Set, Enum, JSON) are not supported due to some compatibility issues of the `cast` function's behavior between TiDB and MySQL.

```
CREATE TABLE t (a DECIMAL(13, 7));
ALTER TABLE t MODIFY COLUMN a DATETIME;
ERROR 8200 (HY000): Unsupported modify column: change from original
    ↪ type decimal(13,7) to datetime is currently unsupported yet
```

14.10.2.66.4 See also

- CREATE TABLE
- SHOW CREATE TABLE
- ADD COLUMN
- DROP COLUMN
- CHANGE COLUMN

14.10.2.67 PREPARE

The `PREPARE` statement provides an SQL interface to server-side prepared statements.

14.10.2.67.1 Synopsis

```
PreparedStmt ::=
    'PREPARE' Identifier 'FROM' PrepareSQL

PrepareSQL ::=
    stringLit
|   UserVariable
```

14.10.2.67.2 Examples

```
mysql> PREPARE mystmt FROM 'SELECT ? as num FROM DUAL';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @number = 5;
Query OK, 0 rows affected (0.00 sec)

mysql> EXECUTE mystmt USING @number;
+------+
| num |
+------+
| 5   |
+------+
1 row in set (0.00 sec)

mysql> DEALLOCATE PREPARE mystmt;
Query OK, 0 rows affected (0.00 sec)
```

14.10.2.67.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.67.4 See also

- EXECUTE
- DEALLOCATE

14.10.2.68 RECOVER TABLE

`RECOVER TABLE` is used to recover a deleted table and the data on it within the GC (Garbage Collection) life time after the `DROP TABLE` statement is executed.

14.10.2.68.1 Syntax

```
RECOVER TABLE table_name;
```

```
RECOVER TABLE BY JOB JOB_ID;
```

14.10.2.68.2 Synopsis

```
RecoverTableStmt ::=
    'RECOVER' 'TABLE' ( 'BY' 'JOB' Int64Num | TableName Int64Num? )

TableName ::=
    Identifier ( '.' Identifier )?

Int64Num ::= NUM

NUM ::= intLit
```

Note:

- If a table is deleted and the GC lifetime is out, the table cannot be recovered with `RECOVER TABLE`. Execution of `RECOVER TABLE` in this scenario returns an error like: `snapshot is older than GC safe point` ↪ `2019-07-10 13:45:57 +0800 CST`.
- If the TiDB version is 3.0.0 or later, it is not recommended for you to use `RECOVER TABLE` when TiDB Binlog is used.
- `RECOVER TABLE` is supported in the Binlog version 3.0.1, so you can use `RECOVER TABLE` in the following three situations:
  - Binlog version is 3.0.1 or later.
  - TiDB 3.0 is used both in the upstream cluster and the downstream cluster.

- The GC life time of the secondary cluster must be longer than that of the primary cluster. However, as latency occurs during data replication between upstream and downstream databases, data recovery might fail in the downstream.

Troubleshoot errors during TiDB Binlog replication

When you use `RECOVER TABLE` in the upstream TiDB during TiDB Binlog replication, TiDB Binlog might be interrupted in the following three situations:

- The downstream database does not support the `RECOVER TABLE` statement. An error instance: `check the manual that corresponds to your MySQL server version for the right syntax to use near 'RECOVER TABLE table_name'`.
- The GC life time is not consistent between the upstream database and the downstream database. An error instance: `snapshot is older than GC safe point 2019-07-10 13:45:57 +0800 CST`.
- Latency occurs during replication between upstream and downstream databases. An error instance: `snapshot is older than GC safe point 2019-07-10 13:45:57 +0800 CST`.

For the above three situations, you can resume data replication from TiDB Binlog with a full import of the deleted table.

14.10.2.68.3 Examples

- Recover the deleted table according to the table name.

```
DROP TABLE t;
```

```
RECOVER TABLE t;
```

This method searches the recent DDL job history and locates the first DDL operation of the `DROP TABLE` type, and then recovers the deleted table with the name identical to the one table name specified in the `RECOVER TABLE` statement.

- Recover the deleted table according to the table's `DDL JOB ID` used.

Suppose that you had deleted the table `t` and created another `t`, and again you deleted the newly created `t`. Then, if you want to recover the `t` deleted in the first place, you must use the method that specifies the `DDL JOB ID`.

```
DROP TABLE t;
```

```
ADMIN SHOW DDL JOBS 1;
```

The second statement above is used to search for the table's `DDL JOB ID` to delete `t`. In the following example, the ID is `53`.

```
+--------+---------+------------+------------+--------------+-----------+----------
   ↪
| JOB_ID | DB_NAME | TABLE_NAME | JOB_TYPE | SCHEMA_STATE | SCHEMA_ID |
   ↪  TABLE_ID | ROW_COUNT | START_TIME          | STATE |
+--------+---------+------------+------------+--------------+-----------+----------
   ↪
| 53     | test    |            | drop table | none         | 1         | 41
   ↪       | 0         | 2019-07-10 13:23:18.277 +0800 CST | synced |
+--------+---------+------------+------------+--------------+-----------+----------
   ↪
```

```
RECOVER TABLE BY JOB 53;
```

This method recovers the deleted table via the `DDL JOB ID`. If the corresponding DDL job is not of the `DROP TABLE` type, an error occurs.

14.10.2.68.4 Implementation principle

When deleting a table, TiDB only deletes the table metadata, and writes the table data (row data and index data) to be deleted to the `mysql.gc_delete_range` table. The GC Worker in the TiDB background periodically removes from the `mysql.gc_delete_range` table the keys that exceed the GC life time.

Therefore, to recover a table, you only need to recover the table metadata and delete the corresponding row record in the `mysql.gc_delete_range` table before the GC Worker deletes the table data. You can use a snapshot read of TiDB to recover the table metadata. Refer to Read Historical Data for details.

Table recovery is done by TiDB obtaining the table metadata through snapshot read, and then going through the process of table creation similar to `CREATE TABLE`. Therefore, `RECOVER TABLE` itself is, in essence, a kind of DDL operation.

14.10.2.68.5 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.69 RENAME USER

`RANAME USER` is used to rename an existing user.

14.10.2.69.1 Synopsis

```
RenameUserStmt ::=
    'RENAME' 'USER' UserToUser ( ',' UserToUser )*
UserToUser ::=
    Username 'TO' Username
Username ::=
    StringName ('@' StringName | singleAtIdentifier)? | 'CURRENT_USER'
       ↪ OptionalBraces
```

14.10.2.69.2 Examples

```
CREATE USER 'newuser' IDENTIFIED BY 'mypassword';
```

```
Query OK, 1 row affected (0.02 sec)
```

```
SHOW GRANTS FOR 'newuser';
```

```
+-----------------------------------+
| Grants for newuser@%              |
+-----------------------------------+
| GRANT USAGE ON *.* TO 'newuser'@'%' |
+-----------------------------------+
1 row in set (0.00 sec)
```

```
RENAME USER 'newuser' TO 'testuser';
```

```
Query OK, 0 rows affected (0.08 sec)
```

```
SHOW GRANTS FOR 'testuser';
```

```
+-----------------------------------+
| Grants for testuser@%             |
+-----------------------------------+
| GRANT USAGE ON *.* TO 'testuser'@'%' |
+-----------------------------------+
1 row in set (0.00 sec)
```

```
SHOW GRANTS FOR 'newuser';
```

```
ERROR 1141 (42000): There is no such grant defined for user 'newuser' on
   ↪ host '%'
```

14.10.2.69.3 MySQL compatibility

`RENAME USER` is expected to be fully compatible with MySQL. If you find any compatibility difference, submit a GitHub issue.

14.10.2.69.4 See also

- CREATE USER
- SHOW GRANTS
- DROP USER

14.10.2.70 RENAME INDEX

The statement `ALTER TABLE .. RENAME INDEX` renames an existing index to a new name. This operation is instant in TiDB, and requires only a meta data change.

14.10.2.70.1 Synopsis

```
AlterTableStmt
        ::= 'ALTER' 'IGNORE'? 'TABLE' TableName RenameIndexSpec ( ','
           ↪ RenameIndexSpec )*

RenameIndexSpec
        ::= 'RENAME' ( 'KEY' | 'INDEX' ) Identifier 'TO' Identifier
```

14.10.2.70.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
   ↪ NOT NULL, INDEX col1 (c1));
Query OK, 0 rows affected (0.11 sec)

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `c1` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `col1` (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
1 row in set (0.00 sec)

mysql> ALTER TABLE t1 RENAME INDEX col1 TO c1;
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `c1` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `c1` (`c1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
1 row in set (0.00 sec)
```

14.10.2.70.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.70.4 See also

- SHOW CREATE TABLE
- CREATE INDEX
- DROP INDEX
- SHOW INDEX
- ALTER INDEX

14.10.2.71 RENAME TABLE

This statement renames an existing table to a new name.

14.10.2.71.1 Synopsis

```
RenameTableStmt ::=
    'RENAME' 'TABLE' TableToTable ( ',' TableToTable )*

TableToTable ::=
    TableName 'TO' TableName
```

14.10.2.71.2 Examples

```
mysql> CREATE TABLE t1 (a int);
Query OK, 0 rows affected (0.12 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
```

```
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)

mysql> RENAME TABLE t1 TO t2;
Query OK, 0 rows affected (0.08 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| t2             |
+----------------+
1 row in set (0.00 sec)
```

14.10.2.71.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.71.4 See also

- CREATE TABLE
- SHOW TABLES
- ALTER TABLE

14.10.2.72 REPLACE

The `REPLACE` statement is semantically a combined `DELETE`+`INSERT` statement. It can be used to simplify application code.

14.10.2.72.1 Synopsis

```
ReplaceIntoStmt ::=
    'REPLACE' PriorityOpt IntoOpt TableName PartitionNameListOpt
       ↪ InsertValues

PriorityOpt ::=
    ( 'LOW_PRIORITY' | 'HIGH_PRIORITY' | 'DELAYED' )?

IntoOpt ::= 'INTO'?

TableName ::=
```

```
    Identifier ( '.' Identifier )?

PartitionNameListOpt ::=
    ( 'PARTITION' '(' Identifier ( ',' Identifier )* ')' )?

InsertValues ::=
    '(' ( ColumnNameListOpt ')' ( ValueSym ValuesList | SelectStmt | '('
       ↪ SelectStmt ')' | UnionStmt ) | SelectStmt ')' )
|   ValueSym ValuesList
|   SelectStmt
|   UnionStmt
|   'SET' ColumnSetValue? ( ',' ColumnSetValue )*
```

14.10.2.72.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO t1 (c1) VALUES (1), (2), (3);
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
+----+----+
3 rows in set (0.00 sec)

mysql> REPLACE INTO t1 (id, c1) VALUES(3, 99);
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 | 99 |
+----+----+
```

```
3 rows in set (0.00 sec)
```

14.10.2.72.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.72.4 See also

- DELETE
- INSERT
- SELECT
- UPDATE

14.10.2.73 RESTORE

This statement performs a distributed restore from a backup archive previously produced by a `BACKUP` statement.

> **Warning:**
>
> This feature is experimental. It is not recommended that you use it in the production environment. This feature might be changed or removed without prior notice. If you find a bug, you can report an issue on GitHub.

The `RESTORE` statement uses the same engine as the BR tool, except that the restore process is driven by TiDB itself rather than a separate BR tool. All benefits and caveats of BR also apply here. In particular, **`RESTORE` is currently not ACID-compliant**. Before running `RESTORE`, ensure that the following requirements are met:

- The cluster is "offline", and the current TiDB session is the only active SQL connection to access all tables being restored.
- When a full restore is being performed, the tables being restored should not already exist, because existing data might be overridden and causes inconsistency between the data and indices.
- When an incremental restore is being performed, the tables should be at the exact same state as the `LAST_BACKUP` timestamp when the backup is created.

Running `RESTORE` requires either the `RESTORE_ADMIN` or `SUPER` privilege. Additionally, both the TiDB node executing the restore and all TiKV nodes in the cluster must have read permission from the destination.

The RESTORE statement is blocking, and will finish only after the entire restore task is finished, failed, or canceled. A long-lasting connection should be prepared for running RESTORE. The task can be canceled using the KILL TIDB QUERY statement.

Only one BACKUP and RESTORE task can be executed at a time. If a BACKUP or RESTORE task is already running on the same TiDB server, the new RESTORE execution will wait until all previous tasks are done.

RESTORE can only be used with "tikv" storage engine. Using RESTORE with the "unistore" engine will fail.

14.10.2.73.1 Synopsis

```
RestoreStmt ::=
    "RESTORE" BRIETables "FROM" stringLit RestoreOption*

BRIETables ::=
    "DATABASE" ( '*' | DBName (',' DBName)* )
|   "TABLE" TableNameList

RestoreOption ::=
    "RATE_LIMIT" '='? LengthNum "MB" '/' "SECOND"
|   "CONCURRENCY" '='? LengthNum
|   "CHECKSUM" '='? Boolean
|   "SEND_CREDENTIALS_TO_TIKV" '='? Boolean

Boolean ::=
    NUM | "TRUE" | "FALSE"
```

14.10.2.73.2 Examples

Restore from backup archive

```
RESTORE DATABASE * FROM 'local:///mnt/backup/2020/04/';
```

```
+--
    ↪ ----------------------------+----------+---------+--------------------+------
    ↪
| Destination                | Size     | BackupTS | Queue Time         |
    ↪ Execution Time    |
+--
    ↪ ----------------------------+----------+---------+--------------------+------
    ↪
| local:///mnt/backup/2020/04/ | 248665063 | 0 | 2020-04-21 17:16:55 |
    ↪ 2020-04-21 17:16:55 |
```

```
+--
    ↪ ---------------------------+----------+---------+--------------------+------
    ↪
1 row in set (28.961 sec)
```

In the example above, all data is restored from a backup archive at the local filesystem. The data is read as SST files from the `/mnt/backup/2020/04/` directories distributed among all TiDB and TiKV nodes.

The first row of the result above is described as follows:

Column	Description
`Destination` ↪	The destination URL to read from
`Size`	The total size of the backup archive, in bytes
`BackupTS`	(not used)
`Queue` ↪ `Time`	The timestamp (in current time zone) when the `RESTORE` task was queued.
`Execution` ↪ `Time`	The timestamp (in current time zone) when the `RESTORE` task starts to run.

Partial restore

You can specify which databases or tables to restore. If some databases or tables are missing from the backup archive, they will be ignored, and thus `RESTORE` would complete without doing anything.

```
RESTORE DATABASE `test` FROM 'local:///mnt/backup/2020/04/';
```

```
RESTORE TABLE `test`.`sbtest01`, `test`.`sbtest02` FROM 'local:///mnt/
    ↪ backup/2020/04/';
```

External storages

BR supports restoring data from S3 or GCS:

```
RESTORE DATABASE * FROM 's3://example-bucket-2020/backup-05/?region=us-west
    ↪ -2';
```

The URL syntax is further explained in External Storages.

When running on cloud environment where credentials should not be distributed, set the `SEND_CREDENTIALS_TO_TIKV` option to `FALSE`:

```
RESTORE DATABASE * FROM 's3://example-bucket-2020/backup-05/?region=us-west
    ↪ -2'
    SEND_CREDENTIALS_TO_TIKV = FALSE;
```

Performance fine-tuning

Use `RATE_LIMIT` to limit the average download speed per TiKV node to reduce network bandwidth.

By default, TiDB node would run 128 restore threads. This value can be adjusted with the `CONCURRENCY` option.

Before restore is completed, `RESTORE` would perform a checksum against the data from the archive to verify correctness. This step can be disabled with the `CHECKSUM` option if you are confident that this is unnecessary.

```
RESTORE DATABASE * FROM 's3://example-bucket-2020/backup-06/'
    RATE_LIMIT = 120 MB/SECOND
    CONCURRENCY = 64
    CHECKSUM = FALSE;
```

Incremental restore

There is no special syntax to perform incremental restore. TiDB will recognize whether the backup archive is full or incremental and take appropriate action. You only need to apply each incremental restore in correct order.

For instance, if a backup task is created as follows:

```
BACKUP DATABASE `test` TO 's3://example-bucket/full-backup' SNAPSHOT =
    ↪ 413612900352000;
BACKUP DATABASE `test` TO 's3://example-bucket/inc-backup-1' SNAPSHOT =
    ↪ 414971854848000 LAST_BACKUP = 413612900352000;
BACKUP DATABASE `test` TO 's3://example-bucket/inc-backup-2' SNAPSHOT =
    ↪ 416353458585600 LAST_BACKUP = 414971854848000;
```

then the same order should be applied in the restore:

```
RESTORE DATABASE * FROM 's3://example-bucket/full-backup';
RESTORE DATABASE * FROM 's3://example-bucket/inc-backup-1';
RESTORE DATABASE * FROM 's3://example-bucket/inc-backup-2';
```

14.10.2.73.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.73.4 See also

- BACKUP
- SHOW RESTORES

14.10.2.74 `REVOKE <privileges>`

This statement removes privileges from an existing user. Executing this statement requires the `GRANT OPTION` privilege and all privileges you revoke.

14.10.2.74.1 Synopsis

```
GrantStmt ::=
    'GRANT' PrivElemList 'ON' ObjectType PrivLevel 'TO' UserSpecList
       ↪ RequireClauseOpt WithGrantOptionOpt

PrivElemList ::=
    PrivElem ( ',' PrivElem )*

PrivElem ::=
    PrivType ( '(' ColumnNameList ')' )?

PrivType ::=
    'ALL' 'PRIVILEGES'?
|   'ALTER' 'ROUTINE'?
|   'CREATE' ( 'USER' | 'TEMPORARY' 'TABLES' | 'VIEW' | 'ROLE' | 'ROUTINE' )
    ↪ ?
|    'TRIGGER'
|   'DELETE'
|    'DROP' 'ROLE'?
|    'PROCESS'
|    'EXECUTE'
|   'INDEX'
|   'INSERT'
|   'SELECT'
```

```
|   'SUPER'
|    'SHOW' ( 'DATABASES' | 'VIEW' )
|   'UPDATE'
|   'GRANT' 'OPTION'
|   'REFERENCES'
|   'REPLICATION' ( 'SLAVE' | 'CLIENT' )
|   'USAGE'
|    'RELOAD'
|   'FILE'
|   'CONFIG'
|   'LOCK' 'TABLES'
|    'EVENT'
|   'SHUTDOWN'

ObjectType ::=
    'TABLE'?

PrivLevel ::=
    '*' ( '.' '*' )?
|    Identifier ( '.' ( '*' | Identifier ) )?

UserSpecList ::=
    UserSpec ( ',' UserSpec )*
```

14.10.2.74.2 Examples

```
mysql> CREATE USER 'newuser' IDENTIFIED BY 'mypassword';
Query OK, 1 row affected (0.02 sec)

mysql> GRANT ALL ON test.* TO 'newuser';
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW GRANTS FOR 'newuser';
+-------------------------------------------------+
| Grants for newuser@%                            |
+-------------------------------------------------+
| GRANT USAGE ON *.* TO 'newuser'@'%'             |
| GRANT ALL PRIVILEGES ON test.* TO 'newuser'@'%' |
+-------------------------------------------------+
2 rows in set (0.00 sec)

mysql> REVOKE ALL ON test.* FROM 'newuser';
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SHOW GRANTS FOR 'newuser';
+-----------------------------------+
| Grants for newuser@%              |
+-----------------------------------+
| GRANT USAGE ON *.* TO 'newuser'@'%' |
+-----------------------------------+
1 row in set (0.00 sec)

mysql> DROP USER 'newuser';
Query OK, 0 rows affected (0.14 sec)

mysql> SHOW GRANTS FOR 'newuser';
ERROR 1141 (42000): There is no such grant defined for user 'newuser' on
    ↪ host '%'
```

14.10.2.74.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.74.4 See also

- GRANT <privileges>
- SHOW GRANTS

- Privilege Management

14.10.2.75 REVOKE <role>

This statement removes a previously assigned role from a specified user (or list of users).

14.10.2.75.1 Synopsis

```
RevokeRoleStmt ::=
    'REVOKE' RolenameList 'FROM' UsernameList

RolenameList ::=
    Rolename ( ',' Rolename )*

UsernameList ::=
    Username ( ',' Username )*
```

14.10.2.75.2 Examples

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

Create a new role `analyticsteam` and a new user `jennifer`:

```
CREATE ROLE analyticsteam;
Query OK, 0 rows affected (0.02 sec)

GRANT SELECT ON test.* TO analyticsteam;
Query OK, 0 rows affected (0.02 sec)

CREATE USER jennifer;
Query OK, 0 rows affected (0.01 sec)

GRANT analyticsteam TO jennifer;
Query OK, 0 rows affected (0.01 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

Note that by default `jennifer` needs to execute `SET ROLE analyticsteam` in order to be able to use the privileges associated with the `analyticsteam` role:

```
SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-------------------------------------------+
2 rows in set (0.00 sec)

SHOW TABLES in test;
ERROR 1044 (42000): Access denied for user 'jennifer'@'%' to database 'test'
SET ROLE analyticsteam;
Query OK, 0 rows affected (0.00 sec)

SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT SELECT ON test.* TO 'jennifer'@'%'  |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
```

```
+--------------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

The statement `SET DEFAULT ROLE` can be used to associate the role `analyticsteam` to `jennifer`:

```
SET DEFAULT ROLE analyticsteam TO jennifer;
Query OK, 0 rows affected (0.02 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

After this, the user `jennifer` has the privileges associated with the role `analyticsteam` and `jennifer` does not have to execute the statement `SET ROLE`:

```
SHOW GRANTS;
+--------------------------------------------+
| Grants for User                            |
+--------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'       |
| GRANT SELECT ON test.* TO 'jennifer'@'%'   |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+--------------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

Revoke the role of analyticsteam from `jennifer`:

```
REVOKE analyticsteam FROM jennifer;
Query OK, 0 rows affected (0.01 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

Show the privileges of `jennifer`:

```
SHOW GRANTS;
+--------------------------------------+
| Grants for User                      |
+--------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%' |
+--------------------------------------+
1 row in set (0.00 sec)
```

14.10.2.75.3 MySQL compatibility

This statement is understood to be fully compatible with roles, which are a feature of MySQL 8.0. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.75.4 See also

- CREATE ROLE
- DROP ROLE
- GRANT <role>
- SET ROLE
- SET DEFAULT ROLE

- Role-Based Access Control

14.10.2.76 ROLLBACK

This statement reverts all changes in the current transaction inside of TIDB. It is the opposite of a `COMMIT` statement.

14.10.2.76.1 Synopsis

```
RollbackStmt ::=
    'ROLLBACK' CompletionTypeWithinTransaction?

CompletionTypeWithinTransaction ::=
    'AND' ( 'CHAIN' ( 'NO' 'RELEASE' )? | 'NO' 'CHAIN' ( 'NO'? 'RELEASE' )?
       ↪ )
|   'NO'? 'RELEASE'
```

14.10.2.76.2 Examples

```
mysql> CREATE TABLE t1 (a INT NOT NULL PRIMARY KEY);
Query OK, 0 rows affected (0.12 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM t1;
Empty set (0.01 sec)
```

14.10.2.76.3 MySQL compatibility

- TiDB does not support savepoints or the syntax `ROLLBACK TO SAVEPOINT`.
- TiDB parses but ignores the syntax `ROLLBACK AND [NO] RELEASE`. This functionality is used in MySQL to disconnect the client session immediately after rolling back the transaction. In TiDB, it is recommended to instead use the `mysql_close()` functionality of your client driver.
- TiDB parses but ignores the syntax `ROLLBACK AND [NO] CHAIN`. This functionality is used in MySQL to immediately start a new transaction with the same isolation level while the current transaction is being rolled back. In TiDB, it is recommended to instead start a new transaction.

14.10.2.76.4 See also

- COMMIT
- BEGIN
- START TRANSACTION

14.10.2.77 SELECT

The `SELECT` statement is used to read data from TiDB.

14.10.2.77.1 Synopsis

SelectStmt:

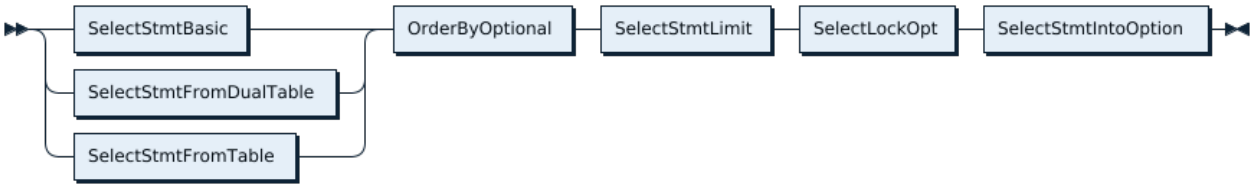


Figure 358: SelectStmt

FromDual:

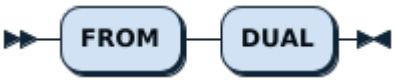


Figure 359: FromDual

SelectStmtOpts:

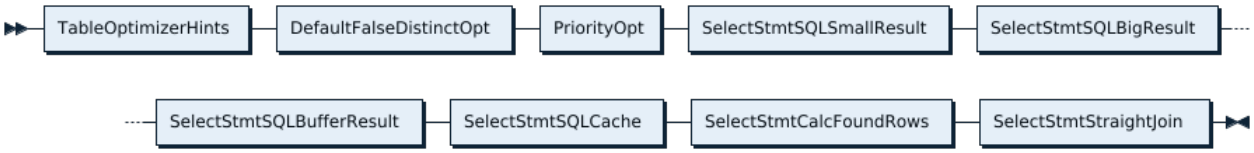


Figure 360: SelectStmtOpts

SelectStmtFieldList:

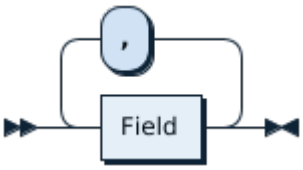


Figure 361: SelectStmtFieldList

TableRefsClause:

```
TableRefsClause ::=
    TableRef AsOfClause? ( ',' TableRef AsOfClause? )*

AsOfClause ::=
    'AS' 'OF' 'TIMESTAMP' Expression
```

WhereClauseOptional:

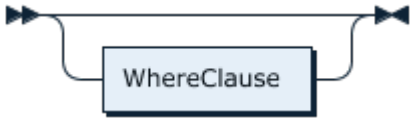


Figure 362: WhereClauseOptional

SelectStmtGroup:



Figure 363: SelectStmtGroup

HavingClause:

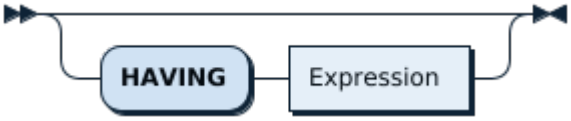


Figure 364: HavingClause

OrderByOptional:



Figure 365: OrderByOptional

SelectStmtLimit:

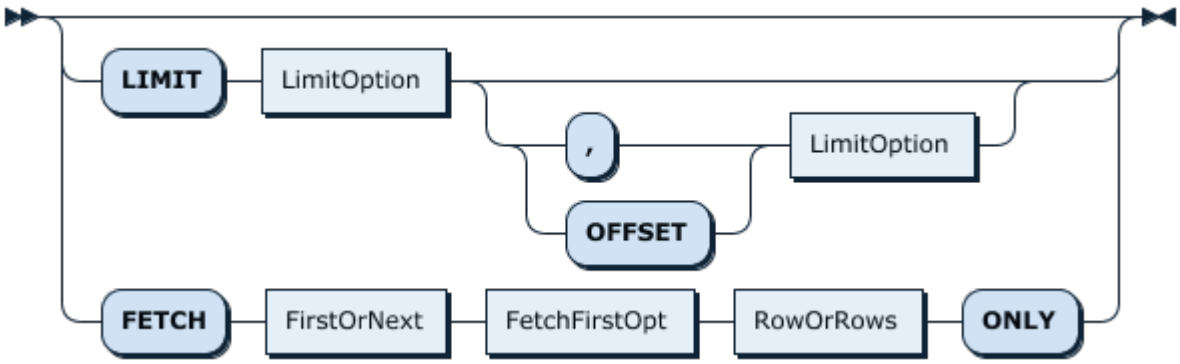


Figure 366: SelectStmtLimit

FirstOrNext:

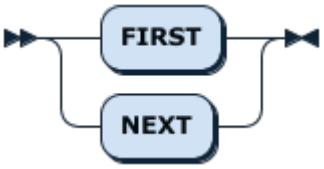


Figure 367: FirstOrNext

FetchFirstOpt:



Figure 368: FetchFirstOpt

RowOrRows:

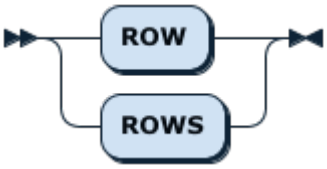


Figure 369: RowOrRows

SelectLockOpt:

```
SelectLockOpt ::=
    ( ( 'FOR' 'UPDATE' ( 'OF' TableList )? 'NOWAIT'? )
|   ( 'LOCK' 'IN' 'SHARE' 'MODE' ) )?
```

```
TableList ::=
    TableName ( ',' TableName )*
```

WindowClauseOptional

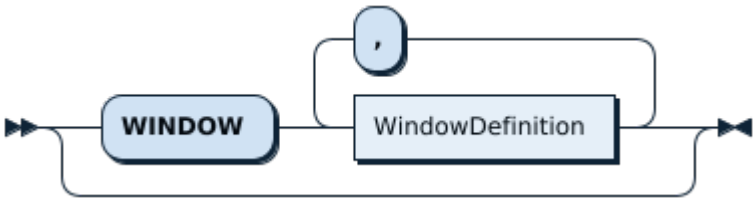


Figure 370: WindowClauseOptional

TableSampleOpt

```
TableSampleOpt ::=
    'TABLESAMPLE' 'REGIONS()'
```

14.10.2.77.2 Description of the syntax elements

Syntax Element	Description
`TableOptimizerHints`	This is the hint to control the behavior of TiDB's optimizer. For more information, refer to Optimizer Hints.
`ALL`, `DISTINCT`, `DISTINCTROW`	The `ALL`, `DISTINCT`/`DISTINCTROW` modifiers specify whether duplicate rows should be returned. ALL (the default) specifies that all matching rows should be returned.
`HIGH_PRIORITY`	`HIGH_PRIORITY` gives the current statement higher priority than other statements.
`SQL_CALC_FOUND_ROWS`	TiDB does not support this feature, and will return an error unless `tidb_enable_noop_functions=1` is set.
`SQL_CACHE`, `SQL_NO_CACHE`	`SQL_CACHE` and `SQL_NO_CACHE` are used to control whether to cache the request results to the `BlockCache` of TiKV (RocksDB). For a one-time query on a large amount of data, such as the `count(*)` query, it is recommended to fill in `SQL_NO_CACHE` to avoid flushing the hot user data in `BlockCache`.
`STRAIGHT_JOIN`	`STRAIGHT_JOIN` forces the optimizer to do a union query in the order of the tables used in the `FROM` clause. When the optimizer chooses a join order that is not good, you can use this syntax to speed up the execution of the query.

Syntax Element	Description
`select_expr`	Each `select_expr` indicates a column to retrieve. including the column names and expressions. `\*` represents all the columns.
`FROM` ↪ `table_references`	The `FROM table_references` clause indicates the table (such as `select * from t;`), or tables (such as `select * from t1 join t2;`) or even 0 tables (such as `select 1+1 from dual;` which is equivalent to `select 1+1;`) from which to retrieve rows.
`WHERE` ↪ `where_condition`	The `WHERE` clause, if given, indicates the condition or conditions that rows must satisfy to be selected. The result contains only the data that meets the condition(s).
`GROUP BY`	The `GROUP BY` statement is used to group the result-set.
`HAVING` ↪ `where_condition`	The `HAVING` clause and the `WHERE` clause are both used to filter the results. The `HAVING` clause filters the results of `GROUP BY`, while the `WHERE` clause filter the results before aggregation.
`ORDER BY`	The `ORDER BY` clause is used to sort the data in ascending or descending order, based on columns, expressions or items in the `select_expr` list.
`LIMIT`	The `LIMIT` clause can be used to constrain the number of rows. `LIMIT` takes one or two numeric arguments. With one argument, the argument specifies the maximum number of rows to return, the first row to return is the first row of the table by default; with two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. TiDB also supports the `FETCH FIRST/NEXT n ROW/ROWS ONLY` syntax, which has the same effect as `LIMIT n`. You can omit `n` in this syntax and its effect is the same as `LIMIT 1`.
`Window` ↪ `window_definition`	This is the syntax for window function, which is usually used to do some analytical computation. For more information, refer to Window Function.

Syntax Element	Description
`FOR UPDATE`	The `SELECT FOR UPDATE` clause locks all the data in the result sets to detect concurrent updates from other transactions. Data that match the query conditions but do not exist in the result sets are not read-locked, such as the row data written by other transactions after the current transaction is started. TiDB uses the Optimistic Transaction Model. The transaction conflicts are not detected in the statement execution phase. Therefore, the current transaction does not block other transactions from executing `UPDATE`, `DELETE` or `SELECT FOR UPDATE` like other databases such as PostgreSQL. In the committing phase, the rows read by `SELECT FOR UPDATE` are committed in two phases, which means they can also join the conflict detection. If write conflicts occur, the commit fails for all transactions that include the `SELECT FOR UPDATE` clause. If no conflict is detected, the commit succeeds. And a new version is generated for the locked rows, so that write conflicts can be detected when other uncommitted transactions are being committed later. When using pessimistic transaction mode, the behavior is basically the same as other databases. Refer to Difference with MySQL InnoDB to see the details. TiDB supports the `NOWAIT` modifier for `FOR UPDATE`. See TiDB Pessimistic Transaction Mode for details.
`LOCK IN SHARE MODE`	To guarantee compatibility, TiDB parses these three modifiers, but will ignore them.
`TABLESAMPLE`	To get a sample of rows from the table.

14.10.2.77.3 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
```

```
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
|  4 |  4 |
|  5 |  5 |
+----+----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT AVG(s_quantity), COUNT(s_quantity) FROM stock TABLESAMPLE
    ↪ REGIONS();
+----------------+------------------+
| AVG(s_quantity) | COUNT(s_quantity) |
+----------------+------------------+
|        59.5000 |                4 |
+----------------+------------------+
1 row in set (0.00 sec)

mysql> SELECT AVG(s_quantity), COUNT(s_quantity) FROM stock;
+----------------+------------------+
| AVG(s_quantity) | COUNT(s_quantity) |
+----------------+------------------+
|        54.9729 |          1000000 |
+----------------+------------------+
1 row in set (0.52 sec)
```

The above example uses data generated with `tiup bench tpcc prepare`. The first query shows the use of `TABLESAMPLE`.

14.10.2.77.4 MySQL compatibility

- The syntax `SELECT ... INTO @variable` is not supported.
- The syntax `SELECT ... GROUP BY ... WITH ROLLUP` is not supported.
- The syntax `SELECT .. GROUP BY expr` does not imply `GROUP BY expr ORDER BY ↪ expr` as it does in MySQL 5.7. TiDB instead matches the behavior of MySQL 8.0 and does not imply a default order.
- The syntax `SELECT ... TABLESAMPLE ...` is a TiDB extension and not supported by MySQL.

14.10.2.77.5 See also

- INSERT
- DELETE
- UPDATE
- REPLACE

14.10.2.78 `SET DEFAULT ROLE`

This statement sets a specific role to be applied to a user by default. Thus, they will automatically have the permissions associated with a role without having to execute `SET ROLE <rolename>` or `SET ROLE ALL`.

14.10.2.78.1 Synopsis

SetDefaultRoleStmt:



Figure 371: SetDefaultRoleStmt

SetDefaultRoleOpt:

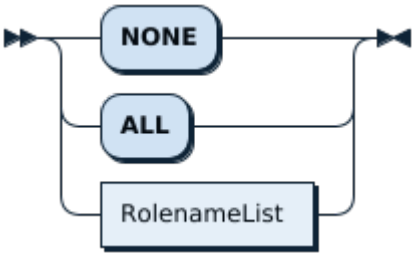


Figure 372: SetDefaultRoleOpt

RolenameList:

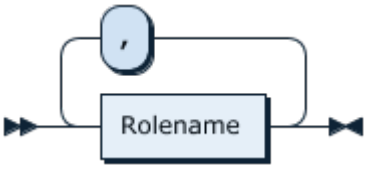


Figure 373: RolenameList

UsernameList:

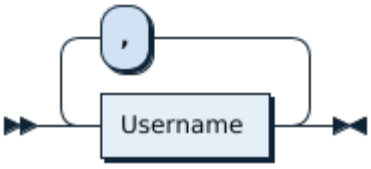


Figure 374: UsernameList

14.10.2.78.2 Examples

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

Create a new role `analyticsteam` and a new user `jennifer`:

```
CREATE ROLE analyticsteam;
Query OK, 0 rows affected (0.02 sec)

GRANT SELECT ON test.* TO analyticsteam;
Query OK, 0 rows affected (0.02 sec)

CREATE USER jennifer;
Query OK, 0 rows affected (0.01 sec)

GRANT analyticsteam TO jennifer;
Query OK, 0 rows affected (0.01 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

Note that by default `jennifer` needs to execute `SET ROLE analyticsteam` in order to be able to use the privileges associated with the `analyticsteam` role:

```
SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-------------------------------------------+
2 rows in set (0.00 sec)

SHOW TABLES in test;
ERROR 1044 (42000): Access denied for user 'jennifer'@'%' to database 'test'
SET ROLE analyticsteam;
Query OK, 0 rows affected (0.00 sec)

SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT Select ON test.* TO 'jennifer'@'%'  |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
```

```
+-------------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

Connect to TiDB as the `root` user:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

The statement `SET DEFAULT ROLE` can be used to associate the role `analyticsteam` to `jennifer`:

```
SET DEFAULT ROLE analyticsteam TO jennifer;
Query OK, 0 rows affected (0.02 sec)
```

Connect to TiDB as the `jennifer` user:

```
mysql -h 127.0.0.1 -P 4000 -u jennifer
```

After this, the user `jennifer` has the privileges associated with the role `analyticsteam` and `jennifer` does not have to execute the statement `SET ROLE`:

```
SHOW GRANTS;
+-------------------------------------------+
| Grants for User                           |
+-------------------------------------------+
| GRANT USAGE ON *.* TO 'jennifer'@'%'      |
| GRANT Select ON test.* TO 'jennifer'@'%'  |
| GRANT 'analyticsteam'@'%' TO 'jennifer'@'%' |
+-------------------------------------------+
3 rows in set (0.00 sec)

SHOW TABLES IN test;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
1 row in set (0.00 sec)
```

`SET DEFAULT ROLE` will not automatically `GRANT` the associated role to the user. Attempting to `SET DEFAULT ROLE` for a role that `jennifer` does not have granted results in the following error:

```
SET DEFAULT ROLE analyticsteam TO jennifer;
ERROR 3530 (HY000): `analyticsteam`@`%` is is not granted to jennifer@%
```

14.10.2.78.3 MySQL compatibility

This statement is understood to be fully compatible with roles, which are a feature of MySQL 8.0. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.78.4 See also

- `CREATE ROLE`
- `DROP ROLE`
- `GRANT <role>`
- `REVOKE <role>`
- `SET ROLE`

- Role-Based Access Control

14.10.2.79 SET [NAMES|CHARACTER SET]

The statements `SET NAMES`, `SET CHARACTER SET` and `SET CHARSET` modify the variables `character_set_client`, `character_set_results` and `character_set_connection` for the current connection.

14.10.2.79.1 Synopsis

SetNamesStmt:



Figure 375: SetNamesStmt

VariableAssignmentList:

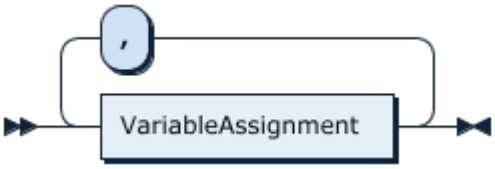


Figure 376: VariableAssignmentList

VariableAssignment:

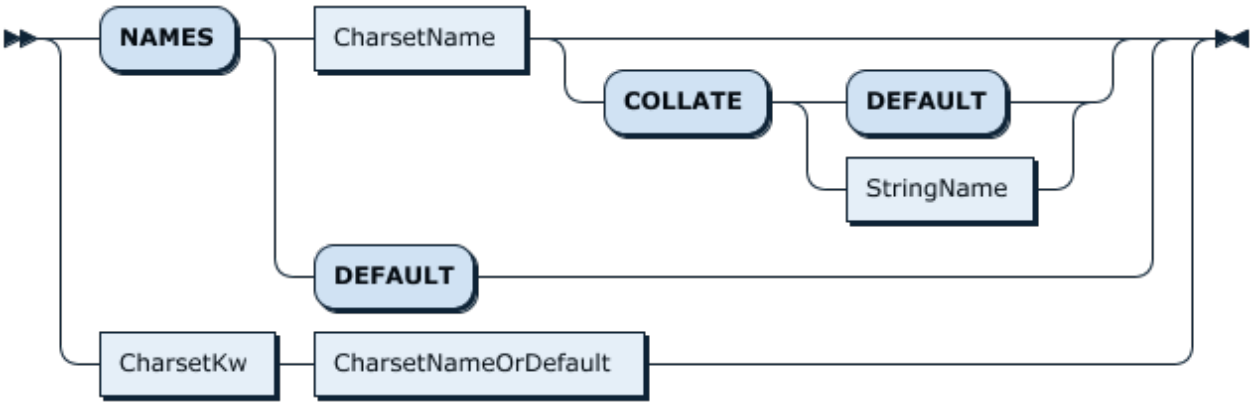


Figure 377: VariableAssignment

CharsetName:



Figure 378: CharsetName

StringName:



Figure 379: StringName

CharsetKw:

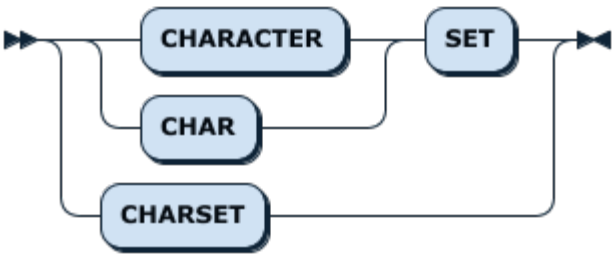


Figure 380: CharsetKw

CharsetNameOrDefault:

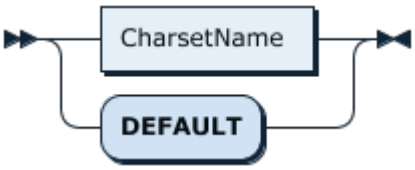


Figure 381: CharsetNameOrDefault

14.10.2.79.2 Examples

```
mysql> SHOW VARIABLES LIKE 'character_set%';
+--
   ↪ ----------------------+----------------------------------------------------------
   ↪
| Variable_name         | Value                                                    |
+--
   ↪ ----------------------+----------------------------------------------------------
   ↪
| character_sets_dir    | /usr/local/mysql-5.6.25-osx10.8-x86_64/share/
   ↪ charsets/ |
| character_set_connection | utf8mb4                                                  |
| character_set_system  | utf8                                                     |
| character_set_results | utf8mb4                                                  |
| character_set_client  | utf8mb4                                                  |
| character_set_database | utf8mb4                                                 |
| character_set_filesystem | binary                                                  |
| character_set_server  | utf8mb4                                                  |
+--
   ↪ ----------------------+----------------------------------------------------------
   ↪
8 rows in set (0.01 sec)

mysql> SET NAMES utf8;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'character_set%';
+--
   ↪ ----------------------+----------------------------------------------------------
   ↪
| Variable_name         | Value                                                    |
+--
   ↪ ----------------------+----------------------------------------------------------
   ↪
| character_sets_dir    | /usr/local/mysql-5.6.25-osx10.8-x86_64/share/
   ↪ charsets/ |
| character_set_connection | utf8                                                     |
```

```
| character_set_system  | utf8                                                    |
| character_set_results | utf8                                                    |
| character_set_client  | utf8                                                    |
| character_set_server  | utf8mb4                                                 |
| character_set_database | utf8mb4                                                |
| character_set_filesystem | binary                                                |
+--
    ↪ ----------------------+-------------------------------------------------------
    ↪
8 rows in set (0.00 sec)

mysql> SET CHARACTER SET utf8mb4;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'character_set%';
+--
    ↪ ----------------------+-------------------------------------------------------
    ↪
| Variable_name         | Value                                                   |
+--
    ↪ ----------------------+-------------------------------------------------------
    ↪
| character_set_connection | utf8mb4                                              |
| character_set_system  | utf8                                                    |
| character_set_results | utf8mb4                                                 |
| character_set_client  | utf8mb4                                                 |
| character_sets_dir    | /usr/local/mysql-5.6.25-osx10.8-x86_64/share/
    ↪ charsets/ |
| character_set_database | utf8mb4                                                |
| character_set_filesystem | binary                                                |
| character_set_server  | utf8mb4                                                 |
+--
    ↪ ----------------------+-------------------------------------------------------
    ↪
8 rows in set (0.00 sec)
```

14.10.2.79.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.79.4 See also

- SHOW [GLOBAL|SESSION] VARIABLES

- SET <variable>
- Character Set and Collation Support

14.10.2.80 SET PASSWORD

This statement changes the user password for a user account in the TiDB system database.

14.10.2.80.1 Synopsis

SetStmt:

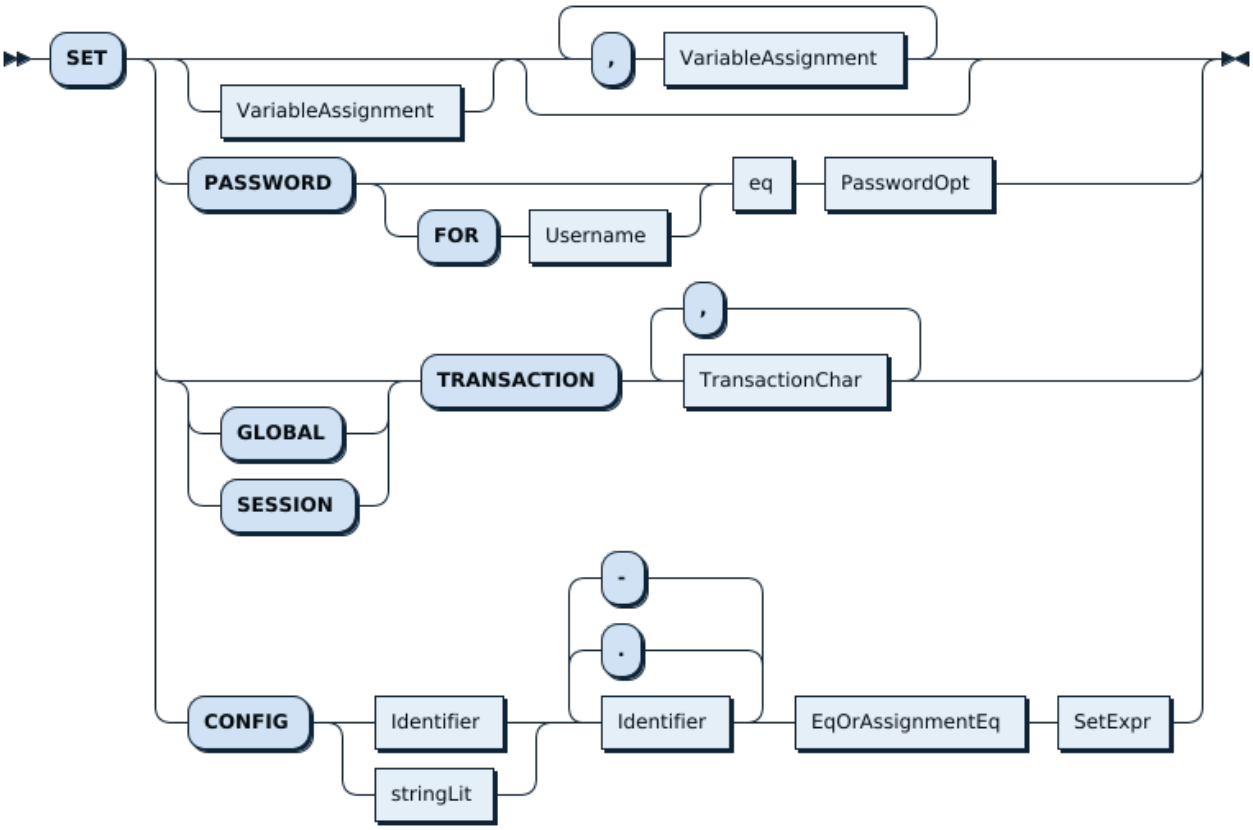


Figure 382: SetStmt

14.10.2.80.2 Examples

```
mysql> SET PASSWORD='test'; -- change my password
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER 'newuser' IDENTIFIED BY 'test';
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SHOW CREATE USER 'newuser';
+--
    ↪ ------------------------------------------------------------------------
    ↪
| CREATE USER for newuser@%
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------
    ↪
| CREATE USER 'newuser'@'%' IDENTIFIED WITH 'mysql_native_password' AS '*94
    ↪ BDCEBE19083CE2A1F959FD02F964C7AF4CFC29' REQUIRE NONE PASSWORD EXPIRE
    ↪ DEFAULT ACCOUNT UNLOCK |
+--
    ↪ ------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> SET PASSWORD FOR newuser = 'test';
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW CREATE USER 'newuser';
+--
    ↪ ------------------------------------------------------------------------
    ↪
| CREATE USER for newuser@%
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------
    ↪
| CREATE USER 'newuser'@'%' IDENTIFIED WITH 'mysql_native_password' AS '*94
    ↪ BDCEBE19083CE2A1F959FD02F964C7AF4CFC29' REQUIRE NONE PASSWORD EXPIRE
    ↪ DEFAULT ACCOUNT UNLOCK |
+--
    ↪ ------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> SET PASSWORD FOR newuser = PASSWORD('test'); -- deprecated syntax
    ↪ from earlier MySQL releases
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW CREATE USER 'newuser';
+--
```

```
    ↪ ------------------------------------------------------------------------------
    ↪
| CREATE USER for newuser@%
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------------
    ↪
| CREATE USER 'newuser'@'%' IDENTIFIED WITH 'mysql_native_password' AS '*94
    ↪ BDCEBE19083CE2A1F959FD02F964C7AF4CFC29' REQUIRE NONE PASSWORD EXPIRE
    ↪ DEFAULT ACCOUNT UNLOCK |
+--
    ↪ ------------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.2.80.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.80.4 See also

- CREATE USER
- Privilege Management

14.10.2.81 SET ROLE

The `SET ROLE` statement is used to enable roles in the current session. After enabling roles, users can use the privileges of the role(s).

14.10.2.81.1 Synopsis

SetRoleStmt:



Figure 383: SetRoleStmt

SetRoleOpt:

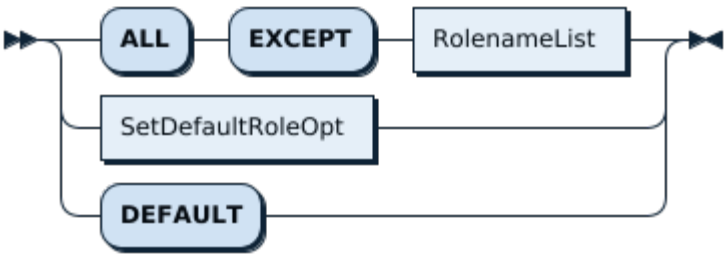


Figure 384: SetRoleOpt

SetDefaultRoleOpt:

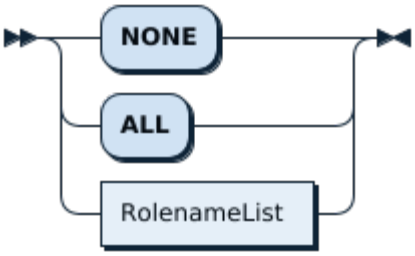


Figure 385: SetDefaultRoleOpt

14.10.2.81.2 Examples

Create a user `'u1'@'%'` and three roles: `'r1'@'%'`, `'r2'@'%'` and `'r3'@'%'`. Grant these roles to `'u1'@'%'` and set `'r1'@'%'` as the default role of `'u1'@'%'`.

```
CREATE USER 'u1'@'%';
CREATE ROLE 'r1', 'r2', 'r3';
GRANT 'r1', 'r2', 'r3' TO 'u1'@'%';
SET DEFAULT ROLE 'r1' TO 'u1'@'%';
```

Log in as `'u1'@'%'` and execute the following `SET ROLE` statement to enable all roles.

```
SET ROLE ALL;
SELECT CURRENT_ROLE();
```

```
+----------------------------+
| CURRENT_ROLE()             |
+----------------------------+
| `r1`@`%`,`r2`@`%`,`r3`@`%` |
+----------------------------+
1 row in set (0.000 sec)
```

Execute the following `SET ROLE` statement to enable `'r2'` and `'r3'`.

```
SET ROLE 'r2', 'r3';
SELECT CURRENT_ROLE();
```

```
+-------------------+
| CURRENT_ROLE()  |
+-------------------+
| `r2`@`%`,`r3`@`%` |
+-------------------+
1 row in set (0.000 sec)
```

Execute the following `SET ROLE` statement to enable the default role(s).

```
SET ROLE DEFAULT;
SELECT CURRENT_ROLE();
```

```
+----------------+
| CURRENT_ROLE() |
+----------------+
| `r1`@`%`     |
+----------------+
1 row in set (0.000 sec)
```

Execute the following `SET ROLE` statement to cancel all enabled role(s).

```
SET ROLE NONE;
SELECT CURRENT_ROLE();
```

```
+----------------+
| CURRENT_ROLE() |
+----------------+
|              |
+----------------+
1 row in set (0.000 sec)
```

14.10.2.81.3 MySQL compatibility

This statement is understood to be fully compatible with roles, which are a feature of MySQL 8.0. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.81.4 See also

- CREATE ROLE
- DROP ROLE
- `GRANT <role>`
- `REVOKE <role>`
- SET DEFAULT ROLE

- Role-Based Access Control

14.10.2.82 SET TRANSACTION

The `SET TRANSACTION` statement can be used to change the current isolation level on a `GLOBAL` or `SESSION` basis. This syntax is an alternative to `SET transaction_isolation='new-value'` and is included for compatibility with both MySQL, and the SQL standards.

14.10.2.82.1 Synopsis

```
SetStmt ::=
    'SET' ( VariableAssignmentList |
    'PASSWORD' ('FOR' Username)? '=' PasswordOpt |
    ( 'GLOBAL'| 'SESSION' )? 'TRANSACTION' TransactionChars |
    'CONFIG' ( Identifier | stringLit) ConfigItemName EqOrAssignmentEq
       ↪ SetExpr )

TransactionChars ::=
    ( 'ISOLATION' 'LEVEL' IsolationLevel | 'READ' 'WRITE' | 'READ' 'ONLY'
       ↪ AsOfClause? )

IsolationLevel ::=
    ( 'REPEATABLE' 'READ' | 'READ' ( 'COMMITTED' | 'UNCOMMITTED' ) | '
       ↪ SERIALIZABLE' )

AsOfClause ::=
    ( 'AS' 'OF' 'TIMESTAMP' Expression)
```

14.10.2.82.2 Examples

```
mysql> SHOW SESSION VARIABLES LIKE 'transaction_isolation';
+-----------------------+----------------+
| Variable_name         | Value          |
+-----------------------+----------------+
| transaction_isolation | REPEATABLE-READ |
+-----------------------+----------------+
1 row in set (0.00 sec)

mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW SESSION VARIABLES LIKE 'transaction_isolation';
+-----------------------+----------------+
| Variable_name         | Value          |
+-----------------------+----------------+
| transaction_isolation | READ-COMMITTED |
+-----------------------+----------------+
```

```
1 row in set (0.01 sec)

mysql> SET SESSION transaction_isolation = 'REPEATABLE-READ';
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW SESSION VARIABLES LIKE 'transaction_isolation';
+-----------------------+----------------+
| Variable_name         | Value          |
+-----------------------+----------------+
| transaction_isolation | REPEATABLE-READ |
+-----------------------+----------------+
1 row in set (0.00 sec)
```

14.10.2.82.3 MySQL compatibility

- TiDB supports the ability to set a transaction as read-only in syntax only.
- The isolation levels `READ-UNCOMMITTED` and `SERIALIZABLE` are not supported.
- The `REPEATABLE-READ` isolation level is achieved through using the snapshot isolation technology, which is partly compatible with MySQL.
- In pessimistic transactions, TiDB supports two isolation levels compatible with MySQL: `REPEATABLE-READ` and `READ-COMMITTED`. For a detailed description, see Isolation Levels.

14.10.2.82.4 See also

- SET [GLOBAL|SESSION] <variable>
- Isolation Levels

14.10.2.83 SET [GLOBAL|SESSION] <variable>

The statement `SET [GLOBAL|SESSION]` modifies one of TiDB's built in variables, of either `SESSION` or `GLOBAL` scope.

> **Note:**
>
> Similar to MySQL, changes to `GLOBAL` variables do not apply to either existing connections, or the local connection. Only new sessions reflect the changes to the value.

14.10.2.83.1 Synopsis

SetStmt:

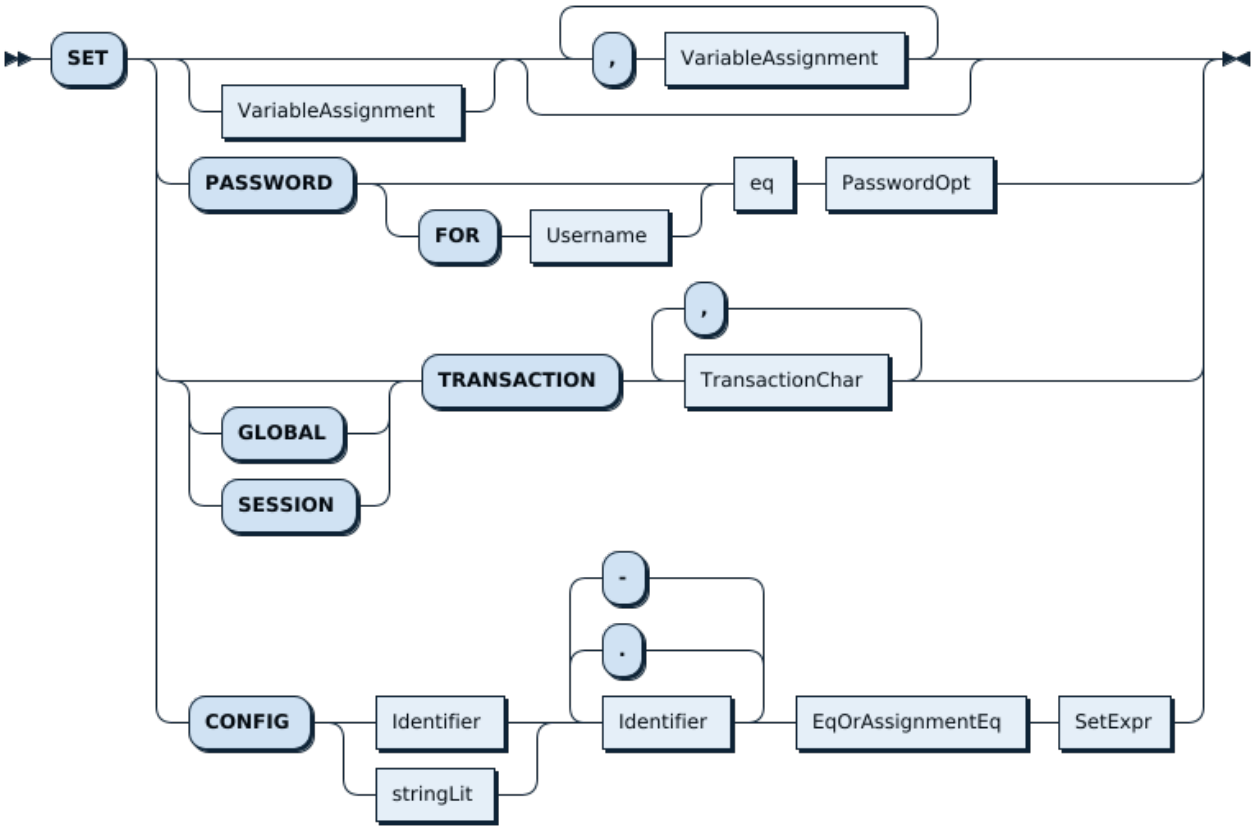


Figure 386: SetStmt

VariableAssignment:

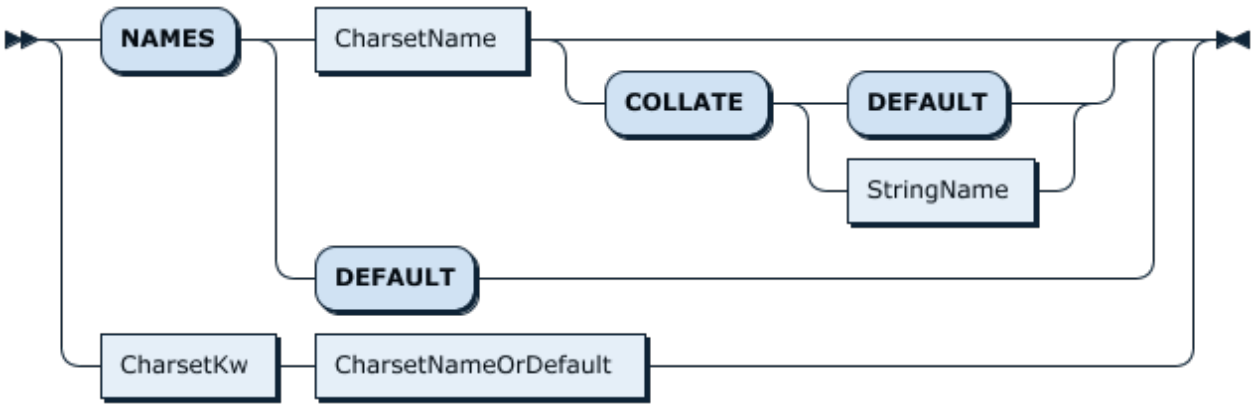


Figure 387: VariableAssignment

14.10.2.83.2 Examples

Get the value of `sql_mode`.

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sql_mode';
+--
    ↪ -------------+-----------------------------------------------------------------
    ↪
| Variable_name | Value
    ↪
    ↪ |
+--
    ↪ -------------+-----------------------------------------------------------------
    ↪
| sql_mode      | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,
    ↪ NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,
    ↪ NO_ENGINE_SUBSTITUTION |
+--
    ↪ -------------+-----------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> SHOW SESSION VARIABLES LIKE 'sql_mode';
+--
    ↪ -------------+-----------------------------------------------------------------
    ↪
| Variable_name | Value
    ↪
    ↪ |
+--
    ↪ -------------+-----------------------------------------------------------------
    ↪
| sql_mode      | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,
    ↪ NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,
    ↪ NO_ENGINE_SUBSTITUTION |
+--
    ↪ -------------+-----------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

Update the value of `sql_mode` globally. If you check the value of `SQL_mode` after the update, you can see that the value of `SESSION` level has not been updated:

```
mysql> SET GLOBAL sql_mode = 'STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER';
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW GLOBAL VARIABLES LIKE 'sql_mode';
```

```
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| sql_mode      | STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER |
+---------------+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SHOW SESSION VARIABLES LIKE 'sql_mode';
+--
   ↪ -------------+-----------------------------------------------------------------
   ↪
| Variable_name | Value
   ↪
   ↪ |
+--
   ↪ -------------+-----------------------------------------------------------------
   ↪
| sql_mode      | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,
   ↪ NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,
   ↪ NO_ENGINE_SUBSTITUTION |
+--
   ↪ -------------+-----------------------------------------------------------------
   ↪
1 row in set (0.00 sec)
```

Using `SET SESSION` takes effect immediately:

```
mysql> SET SESSION sql_mode = 'STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER';
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW SESSION VARIABLES LIKE 'sql_mode';
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| sql_mode      | STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER |
+---------------+-----------------------------------------+
1 row in set (0.00 sec)
```

14.10.2.83.3 MySQL compatibility

The following behavior differences apply:

- Changes made with `SET GLOBAL` will be propagated to all TiDB instances in the cluster. This differs from MySQL, where changes do not propagate to replicas.

- TiDB presents several variables as both readable and settable. This is required for MySQL compatibility, because it is common for both applications and connectors to read MySQL variables. For example: JDBC connectors both read and set query cache settings, despite not relying on the behavior.
- Changes made with `SET GLOBAL` will persist through TiDB server restarts. This means that `SET GLOBAL` in TiDB behaves more similar to `SET PERSIST` as available in MySQL 8.0 and above.

14.10.2.83.4 See also

- SHOW [GLOBAL|SESSION] VARIABLES

14.10.2.84 SHOW ANALYZE STATUS

The `SHOW ANALYZE STATUS` statement shows the statistics collection tasks being executed by TiDB and a limited number of historical task records.

Starting from TiDB v6.1.0, the `SHOW ANALYZE STATUS` statement supports showing cluster-level tasks. Even after a TiDB restart, you can still view task records before the restart using this statement. Before TiDB v6.1.0, the `SHOW ANALYZE STATUS` statement can only show instance-level tasks, and task records are cleared after a TiDB restart.

Starting from TiDB v6.1.0, you can view the history tasks within the last 7 days through the system table `mysql.analyze_jobs`.

14.10.2.84.1 Synopsis

```
ShowAnalyzeStatusStmt ::= 'SHOW' 'ANALYZE' 'STATUS' ShowLikeOrWhereOpt

ShowLikeOrWhereOpt ::= 'LIKE' SimpleExpr | 'WHERE' Expression
```

14.10.2.84.2 Examples

```
mysql> create table t(x int, index idx(x)) partition by hash(x) partitions
    ↪ 2;
Query OK, 0 rows affected (0.69 sec)

mysql> set @@tidb_analyze_version = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> analyze table t;
Query OK, 0 rows affected (0.20 sec)

mysql> show analyze status;
```

```
+--
   ↪ -----------+-----------+---------------+------------------+----------------+
   ↪
| Table_schema | Table_name | Partition_name | Job_info  | Processed_rows |
   ↪ Start_time      | End_time          | State   | Fail_reason | Instance
   ↪   | Process_ID |
+--
   ↪ -----------+-----------+---------------+------------------+----------------+
   ↪
| test       | t         | p1           | analyze index idx |          0 |
   ↪ 2022-05-27 11:29:46 | 2022-05-27 11:29:46 | finished | NULL |
   ↪ 127.0.0.1:4000 |  NULL |
| test       | t         | p0           | analyze index idx |          0 |
   ↪ 2022-05-27 11:29:46 | 2022-05-27 11:29:46 | finished | NULL |
   ↪ 127.0.0.1:4000 |  NULL |
| test       | t         | p1           | analyze columns |            0 |
   ↪ 2022-05-27 11:29:46 | 2022-05-27 11:29:46 | finished | NULL |
   ↪ 127.0.0.1:4000 |  NULL |
| test       | t         | p0           | analyze columns |            0 |
   ↪ 2022-05-27 11:29:46 | 2022-05-27 11:29:46 | finished | NULL |
   ↪ 127.0.0.1:4000 |  NULL |
+--
   ↪ -----------+-----------+---------------+------------------+----------------+
   ↪
4 rows in set (0.01 sec)

mysql> set @@tidb_analyze_version = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> analyze table t;
Query OK, 0 rows affected, 2 warnings (0.03 sec)

mysql> show analyze status;
+--
   ↪ -----------+-----------+---------------+-------------------------------------
   ↪
| Table_schema | Table_name | Partition_name | Job_info
   ↪                                                 | Processed_rows |
   ↪ Start_time      | End_time          | State   | Fail_reason | Instance
   ↪   | Process_ID |
+--
   ↪ -----------+-----------+---------------+-------------------------------------
   ↪
| test       | t         | p1           | analyze table all columns with 256
   ↪  buckets, 500 topn, 1 samplerate |    0 | 2022-05-27 11:30:12 |
```

```
   ↪ 2022-05-27 11:30:12 | finished | NULL | 127.0.0.1:4000 |  NULL |
| test       | t         | p0           | analyze table all columns with 256
   ↪  buckets, 500 topn, 1 samplerate |    0 | 2022-05-27 11:30:12 |
   ↪ 2022-05-27 11:30:12 | finished | NULL | 127.0.0.1:4000 |  NULL |
| test       | t         | p1           | analyze index idx
   ↪                                          |            0 | 2022-05-27
   ↪ 11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪ NULL |
| test       | t         | p0           | analyze index idx
   ↪                                          |            0 | 2022-05-27
   ↪ 11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪ NULL |
| test       | t         | p1           | analyze columns
   ↪                                             |            0 | 2022-05-27
   ↪  11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪  NULL |
| test       | t         | p0           | analyze columns
   ↪                                             |            0 | 2022-05-27
   ↪  11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪  NULL |
+--
   ↪ ------------+------------+---------------+----------------------------------
   ↪
6 rows in set (0.00 sec)
```

14.10.2.84.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.84.4 See also

- ANALYZE_STATUS table

14.10.2.85 SHOW [BACKUPS|RESTORES]

These statements show a list of all queued, running and recently finished `BACKUP` and `RESTORE` tasks that were executed on a TiDB instance.

Both statements require `SUPER` privilege to run.

Use `SHOW BACKUPS` to query `BACKUP` tasks and use `SHOW RESTORES` to query `RESTORE` tasks.

Backups and restores that were started with the `br` commandline tool are not shown.

14.10.2.85.1 Synopsis

```
ShowBRIEStmt ::=
    "SHOW" ("BACKUPS" | "RESTORES") ShowLikeOrWhere?

ShowLikeOrWhere ::=
    "LIKE" SimpleExpr
|   "WHERE" Expression
```

14.10.2.85.2 Examples

In one connection, execute the following statement:

```
BACKUP DATABASE `test` TO 's3://example-bucket/backup-01/?region=us-west-1';
```

Before the backup completes, run `SHOW BACKUPS` in a new connection:

```
SHOW BACKUPS;
```

```
+--
    ↪ ----------------------------+---------+----------+--------------------+------
    ↪
| Destination                  | State   | Progress | Queue_time         |
    ↪ Execution_time    | Finish_time | Connection | Message |
+--
    ↪ ----------------------------+---------+----------+--------------------+------
    ↪
| s3://example-bucket/backup-01/ | Backup | 98.38 | 2020-04-12 23:09:03 |
    ↪ 2020-04-12 23:09:25 | NULL |      4 | NULL   |
+--
    ↪ ----------------------------+---------+----------+--------------------+------
    ↪
1 row in set (0.00 sec)
```

The first row of the result above is described as follows:

Column	Description
`Destination` ↪	The destination URL (with all parameters stripped to avoid leaking secret keys)

Column	Description
`State`	State of the task
`Progress`	Estimated progress in the current state as a percentage
`Queue_time`	When the task was queued
`Execution_time`	When the task was started; the value is `0000-00-00 00:00:00` for queueing tasks
`Finish_time`	The timestamp when the task finished; the value is `0000-00-00 00:00:00` for queueing and running tasks
`Connection`	Connection ID running this task
`Message`	Message with details

The possible states are:

State	Description
Backup	Making a backup
Wait	Waiting for execution
Checksum	Running a checksum operation

The connection ID can be used to cancel a backup/restore task via the `KILL TIDB QUERY` statement.

```
KILL TIDB QUERY 4;
```

```
Query OK, 0 rows affected (0.00 sec)
```

Filtering

Use the `LIKE` clause to filter out tasks by matching the destination URL against a wildcard expression.

```
SHOW BACKUPS LIKE 's3://%';
```

Use the `WHERE` clause to filter by columns.

```
SHOW BACKUPS WHERE `Progress` < 25.0;
```

14.10.2.85.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.85.4 See also

- BACKUP
- RESTORE

14.10.2.86 SHOW [GLOBAL|SESSION] BINDINGS

The `SHOW BINDINGS` statement is used to display information about created SQL bindings. A `BINDING` can be on either a `GLOBAL` or `SESSION` basis. The default is `SESSION`.

14.10.2.86.1 Synopsis

ShowStmt:

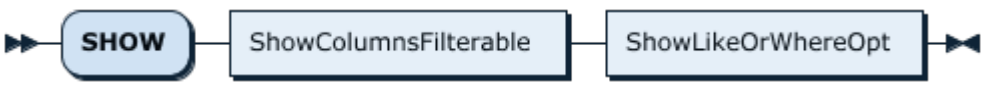


Figure 388: ShowStmt

ShowTargetFilterable:

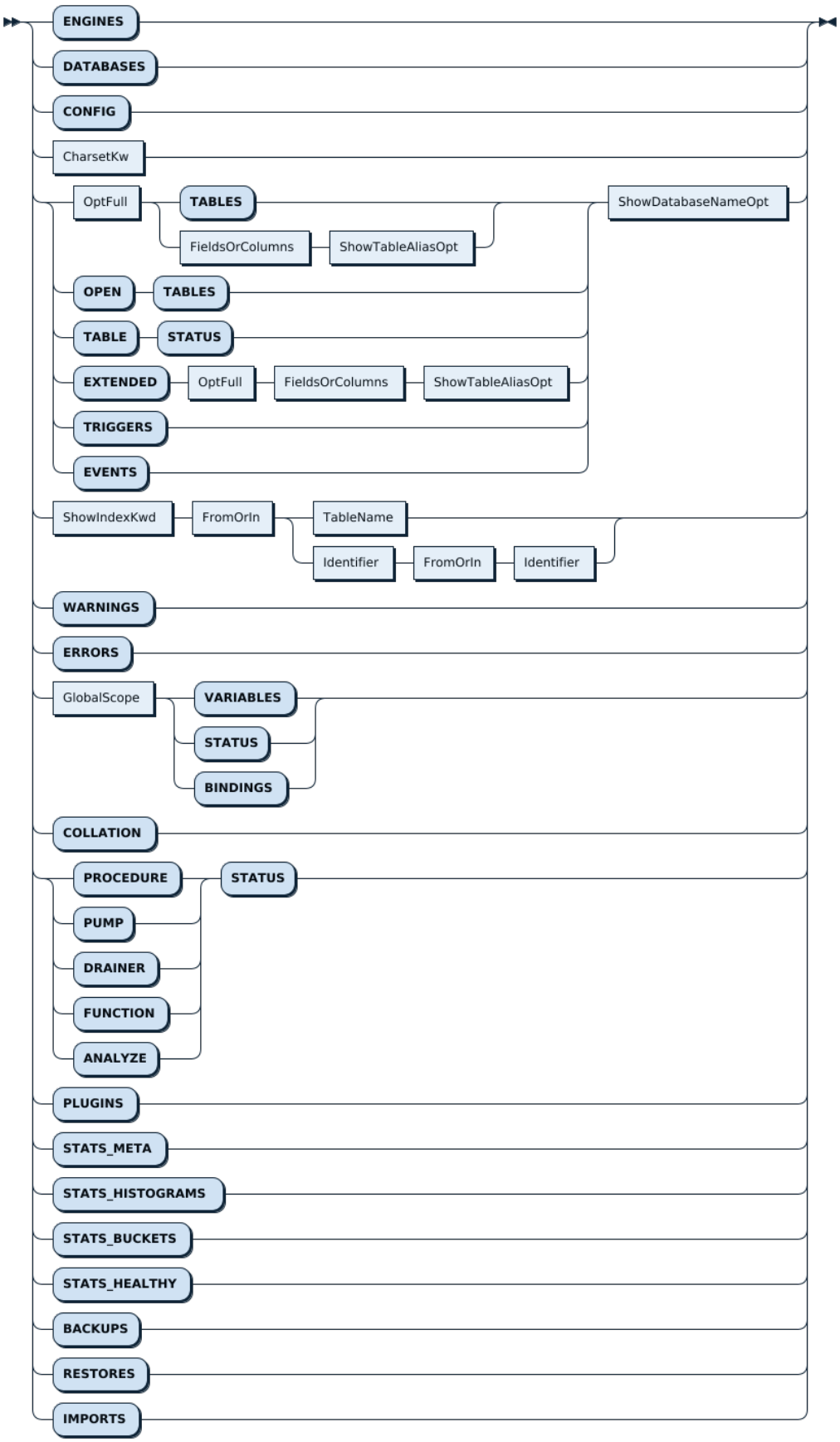


Figure 389: ShowTargetFilterable

GlobalScope:



Figure 390: GlobalScope

ShowLikeOrWhereOpt

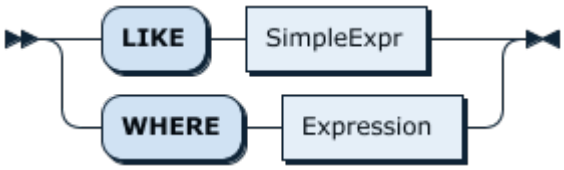


Figure 391: ShowLikeOrWhereOpt

14.10.2.86.2 Syntax description

```
SHOW [GLOBAL | SESSION] BINDINGS [ShowLikeOrWhereOpt];
```

This statement outputs the execution plan bindings at the GLOBAL or SESSION level. The default scope is SESSION. Currently `SHOW BINDINGS` outputs eight columns, as shown below:

Column Name	Description
original_sql	Original SQL statement after parameterization
bind_sql	Bound SQL statement with hints
default_db	Default database
status	Status including 'Using', 'Deleted', 'Invalid', 'Rejected', and 'Pending verification'

Column Name	Description
create_time	Created time
update_time	Updated time
charset	Character set
collation	Sorting rule
source	The way in which a binding is created, including `manual` (created by the `create [global] binding` SQL statement), `capture` (captured automatically by TiDB), and `evolve` (evolved automatically by TiDB)

14.10.2.86.3 Examples

```
mysql> CREATE TABLE t1 (
    -> id INT NOT NULL PRIMARY KEY auto_increment,
    -> b INT NOT NULL,
    -> pad VARBINARY(255),
    -> INDEX(b)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM dual;
Query OK, 1 row affected (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 1 row affected (0.00 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 8 rows affected (0.00 sec)
Records: 8 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 1000 rows affected (0.04 sec)
Records: 1000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (1.74 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (2.15 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255)
    ↪ FROM t1 a JOIN t1 b JOIN t1 c LIMIT 100000;
Query OK, 100000 rows affected (2.64 sec)
Records: 100000 Duplicates: 0 Warnings: 0

mysql> SELECT SLEEP(1);
+----------+
| SLEEP(1) |
+----------+
|        0 |
+----------+
1 row in set (1.00 sec)

mysql> ANALYZE TABLE t1;
Query OK, 0 rows affected (1.33 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
    ↪ ----------------------------+---------+---------+-----------+-----------------
    ↪
| id                          | estRows | actRows | task      | access object
    ↪      | execution info
    ↪                                                                | operator info
    ↪                  | memory       | disk |
+--
```

```
    ↪ ---------------------------+---------+---------+----------+-----------------
    ↪
| IndexLookUp_10              | 583.00 | 297    | root     |
    ↪                    | time:10.545072ms, loops:2, rpc num: 1, rpc time
    ↪ :398.359µs, proc keys:297 |                       | 109.1484375 KB | N/
    ↪ A |
|  ─IndexRangeScan_8(Build)  | 583.00 | 297    | cop[tikv] | table:t1, index
    ↪ :b(b) | time:0s, loops:4                                                |
    ↪  range:[123,123], keep order:false | N/A      | N/A |
|  ─TableRowIDScan_9(Probe)  | 583.00 | 297    | cop[tikv] | table:t1
    ↪         | time:12ms, loops:4
    ↪                                                | keep order:false
    ↪              | N/A          | N/A |
+--
    ↪ ---------------------------+---------+---------+----------+-----------------
    ↪
3 rows in set (0.02 sec)

mysql> CREATE SESSION BINDING FOR
    ->  SELECT * FROM t1 WHERE b = 123
    -> USING
    ->  SELECT * FROM t1 IGNORE INDEX (b) WHERE b = 123;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM t1 WHERE b = 123;
+--
    ↪ ----------------------+----------+---------+----------+-------------+------
    ↪
| id                    | estRows  | actRows | task    | access object |
    ↪ execution info                                                    |
    ↪ operator info    | memory      | disk |
+--
    ↪ ----------------------+----------+---------+----------+-------------+------
    ↪
| TableReader_7         | 583.00   | 297    | root     |             | time
    ↪ :222.32506ms, loops:2, rpc num: 1, rpc time:222.078952ms, proc keys
    ↪ :301010 | data:Selection_6 | 88.6640625 KB | N/A |
|  ─Selection_6         | 583.00   | 297    | cop[tikv] |             | time
    ↪ :224ms, loops:298                                                  | eq(
    ↪ test.t1.b, 123) | N/A       | N/A |
|    ─TableFullScan_5   | 301010.00 | 301010 | cop[tikv] | table:t1 | time
    ↪ :220ms, loops:298                                                  |
    ↪ keep order:false | N/A         | N/A |
+--
    ↪ ----------------------+----------+---------+----------+-------------+------
```

```
    ↪
3 rows in set (0.22 sec)

mysql> SHOW SESSION BINDINGS\G
*************************** 1. row ***************************
Original_sql: select * from t1 where b = ?
    Bind_sql: SELECT * FROM t1 IGNORE INDEX (b) WHERE b = 123
  Default_db: test
      Status: using
 Create_time: 2020-05-22 14:38:03.456
 Update_time: 2020-05-22 14:38:03.456
     Charset: utf8mb4
   Collation: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

14.10.2.86.4 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.86.5 See also

- CREATE [GLOBAL|SESSION] BINDING
- DROP [GLOBAL|SESSION] BINDING
- ANALYZE TABLE
- Optimizer Hints
- SQL Plan Management

14.10.2.87 SHOW BUILTINS

`SHOW BUILTINS` is used to list all supported builtin functions in TiDB.

14.10.2.87.1 Synopsis

ShowBuiltinsStmt:

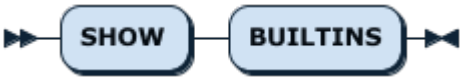


Figure 392: ShowBuiltinsStmt

14.10.2.87.2 Examples

```
SHOW BUILTINS;
```

```
+-----------------------------+
| Supported_builtin_functions |
+-----------------------------+
| abs                         |
| acos                        |
| adddate                     |
| addtime                     |
| aes_decrypt                 |
| aes_encrypt                 |
| and                         |
| any_value                   |
| ascii                       |
| asin                        |
| atan                        |
| atan2                       |
| benchmark                   |
| bin                         |
| bit_count                   |
| bit_length                  |
| bitand                      |
| bitneg                      |
| bitor                       |
| bitxor                      |
| case                        |
| ceil                        |
| ceiling                     |
| char_func                   |
| char_length                 |
| character_length            |
| charset                     |
| coalesce                    |
| coercibility                |
| collation                   |
| compress                    |
| concat                      |
| concat_ws                   |
| connection_id               |
| conv                        |
| convert                     |
| convert_tz                  |
| cos                         |
| cot                         |
| crc32                       |
| curdate                     |
```

```
| current_date            |
| current_role            |
| current_time            |
| current_timestamp       |
| current_user            |
| curtime                 |
| database                |
| date                    |
| date_add                |
| date_format             |
| date_sub                |
| datediff                |
| day                     |
| dayname                 |
| dayofmonth              |
| dayofweek               |
| dayofyear               |
| decode                  |
| default_func            |
| degrees                 |
| des_decrypt             |
| des_encrypt             |
| div                     |
| elt                     |
| encode                  |
| encrypt                 |
| eq                      |
| exp                     |
| export_set              |
| extract                 |
| field                   |
| find_in_set             |
| floor                   |
| format                  |
| format_bytes            |
| format_nano_time        |
| found_rows              |
| from_base64             |
| from_days               |
| from_unixtime           |
| ge                      |
| get_format              |
| get_lock                |
| getparam                |
| getvar                  |
```

```
| greatest                |
| gt                      |
| hex                     |
| hour                    |
| if                      |
| ifnull                  |
| in                      |
| inet6_aton              |
| inet6_ntoa              |
| inet_aton               |
| inet_ntoa               |
| insert_func             |
| instr                   |
| intdiv                  |
| interval                |
| is_free_lock            |
| is_ipv4                 |
| is_ipv4_compat          |
| is_ipv4_mapped          |
| is_ipv6                 |
| is_used_lock            |
| isfalse                 |
| isnull                  |
| istrue                  |
| json_array              |
| json_array_append       |
| json_array_insert       |
| json_contains           |
| json_contains_path      |
| json_depth              |
| json_extract            |
| json_insert             |
| json_keys               |
| json_length             |
| json_merge              |
| json_merge_patch        |
| json_merge_preserve     |
| json_object             |
| json_pretty             |
| json_quote              |
| json_remove             |
| json_replace            |
| json_search             |
| json_set                |
| json_storage_size       |
```

```
| json_type                |
| json_unquote             |
| json_valid               |
| last_day                 |
| last_insert_id           |
| lastval                  |
| lcase                    |
| le                       |
| least                    |
| left                     |
| leftshift                |
| length                   |
| like                     |
| ln                       |
| load_file                |
| localtime                |
| localtimestamp           |
| locate                   |
| log                      |
| log10                    |
| log2                     |
| lower                    |
| lpad                     |
| lt                       |
| ltrim                    |
| make_set                 |
| makedate                 |
| maketime                 |
| master_pos_wait          |
| md5                      |
| microsecond              |
| mid                      |
| minus                    |
| minute                   |
| mod                      |
| month                    |
| monthname                |
| mul                      |
| name_const               |
| ne                       |
| nextval                  |
| not                      |
| now                      |
| nulleq                   |
| oct                      |
```

```
| octet_length            |
| old_password            |
| or                      |
| ord                     |
| password_func           |
| period_add              |
| period_diff             |
| pi                      |
| plus                    |
| position                |
| pow                     |
| power                   |
| quarter                 |
| quote                   |
| radians                 |
| rand                    |
| random_bytes            |
| regexp                  |
| release_all_locks       |
| release_lock            |
| repeat                  |
| replace                 |
| reverse                 |
| right                   |
| rightshift              |
| round                   |
| row_count               |
| rpad                    |
| rtrim                   |
| schema                  |
| sec_to_time             |
| second                  |
| session_user            |
| setval                  |
| setvar                  |
| sha                     |
| sha1                    |
| sha2                    |
| sign                    |
| sin                     |
| sleep                   |
| space                   |
| sqrt                    |
| str_to_date             |
| strcmp                  |
```

```
| subdate                    |
| substr                     |
| substring                  |
| substring_index            |
| subtime                    |
| sysdate                    |
| system_user                |
| tan                        |
| tidb_decode_key            |
| tidb_decode_plan           |
| tidb_is_ddl_owner          |
| tidb_parse_tso             |
| tidb_version               |
| time                       |
| time_format                |
| time_to_sec                |
| timediff                   |
| timestamp                  |
| timestampadd               |
| timestampdiff              |
| to_base64                  |
| to_days                    |
| to_seconds                 |
| trim                       |
| truncate                   |
| ucase                      |
| unaryminus                 |
| uncompress                 |
| uncompressed_length        |
| unhex                      |
| unix_timestamp             |
| upper                      |
| user                       |
| utc_date                   |
| utc_time                   |
| utc_timestamp              |
| uuid                       |
| uuid_short                 |
| validate_password_strength |
| version                    |
| week                       |
| weekday                    |
| weekofyear                 |
| weight_string              |
| xor                        |
```

```
| year                        |
| yearweek                    |
+-----------------------------+
268 rows in set (0.00 sec)
```

14.10.2.87.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.88 SHOW CHARACTER SET

This statement provides a static list of available character sets in TiDB. The output does not reflect any attributes of the current connection or user.

14.10.2.88.1 Synopsis

ShowCharsetStmt:



Figure 393: ShowCharsetStmt

CharsetKw:

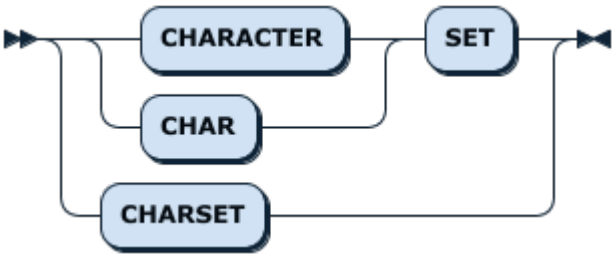


Figure 394: CharsetKw

14.10.2.88.2 Examples

```
SHOW CHARACTER SET;
```

```
+---------+---------------------------------+-------------------+--------+
   ↪
| Charset | Description                     | Default collation | Maxlen |
+---------+---------------------------------+-------------------+--------+
   ↪
| ascii   | US ASCII                        | ascii_bin         |      1 |
```

```
| binary | binary                              | binary          |      1 |
| gbk    | Chinese Internal Code Specification | gbk_chinese_ci  |      2 |
| latin1 | Latin1                              | latin1_bin      |      1 |
| utf8   | UTF-8 Unicode                       | utf8_bin        |      3 |
| utf8mb4 | UTF-8 Unicode                      | utf8mb4_bin     |      4 |
+---------+-------------------------------------+-----------------+--------+
   ↪
6 rows in set (0.00 sec)
```

14.10.2.88.3 MySQL compatibility

The usage of this statement is understood to be fully compatible with MySQL. However, charsets in TiDB may have different default collations compared with MySQL. For details, refer to Compatibility with MySQL. Any other compatibility differences should be reported via an issue on GitHub.

14.10.2.88.4 See also

- SHOW COLLATION
- Character Set and Collation

14.10.2.89 SHOW COLLATION

This statement provides a static list of collations, and is included to provide compatibility with MySQL client libraries.

> **Note:**
>
> Results of `SHOW COLLATION` vary when the "new collation framework" is enabled. For new collation framework details, refer to Character Set and Collation.

14.10.2.89.1 Synopsis

ShowCollationStmt:

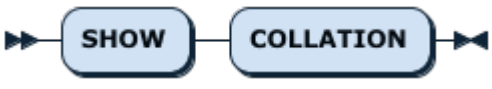


Figure 395: ShowCollationStmt

14.10.2.89.2 Examples

When new collation framework is disabled, only binary collations are displayed.

```
mysql> SHOW COLLATION;
+-------------+---------+------+---------+----------+---------+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-------------+---------+------+---------+----------+---------+
| utf8mb4_bin | utf8mb4 | 46 | Yes  | Yes     |      1 |
| latin1_bin | latin1 |  47 | Yes    | Yes     |      1 |
| binary     | binary |   63 | Yes    | Yes     |      1 |
| ascii_bin | ascii  |   65 | Yes    | Yes     |      1 |
| utf8_bin   | utf8   |   83 | Yes    | Yes     |      1 |
+-------------+---------+------+---------+----------+---------+
5 rows in set (0.02 sec)
```

When new collation framework is enabled, `utf8_general_ci` and `utf8mb4_general_ci` are additionally supported.

```
mysql> SHOW COLLATION;
+--------------------+---------+------+---------+----------+---------+
| Collation        | Charset | Id | Default | Compiled | Sortlen |
+--------------------+---------+------+---------+----------+---------+
| ascii_bin        | ascii  |   65 | Yes    | Yes     |      1 |
| binary           | binary |   63 | Yes    | Yes     |      1 |
| gbk_bin          | gbk    |   87 |        | Yes     |      1 |
| gbk_chinese_ci   | gbk    |   28 | Yes    | Yes     |      1 |
| latin1_bin       | latin1 |   47 | Yes    | Yes     |      1 |
| utf8_bin         | utf8   |   83 | Yes    | Yes     |      1 |
| utf8_general_ci  | utf8   |   33 |        | Yes     |      1 |
| utf8_unicode_ci  | utf8   |  192 |        | Yes     |      1 |
| utf8mb4_bin      | utf8mb4 | 46 | Yes    | Yes     |      1 |
| utf8mb4_general_ci | utf8mb4 | 45 |     | Yes     |      1 |
| utf8mb4_unicode_ci | utf8mb4 | 224 |    | Yes     |      1 |
+--------------------+---------+------+---------+----------+---------+
11 rows in set (0.001 sec)
```

14.10.2.89.3 MySQL compatibility

The usage of this statement is understood to be fully compatible with MySQL. However, charsets in TiDB may have different default collations compared with MySQL. For details, refer to Compatibility with MySQL. Any other compatibility differences should be reported via an issue on GitHub.

14.10.2.89.4 See also

- SHOW CHARACTER SET
- Character Set and Collation

14.10.2.90 SHOW [FULL] COLUMNS FROM

The statement `SHOW [FULL] COLUMNS FROM <table_name>` describes the columns of a table or view in a useful tabular format. The optional keyword `FULL` displays the privileges the current user has to that column, and the `comment` from the table definition.

The statements `SHOW [FULL] FIELDS FROM <table_name>`, `DESC <table_name>`, `DESCRIBE <table_name>`, and `EXPLAIN <table_name>` are aliases of this statement.

> **Note:**
>
> `DESC TABLE <table_name>`, `DESCRIBE TABLE <table_name>`, and `EXPLAIN TABLE <table_name>` are not equivalent to the above statements. They are aliases of `DESC SELECT * FROM <table_name>`.

14.10.2.90.1 Synopsis

ShowStmt:

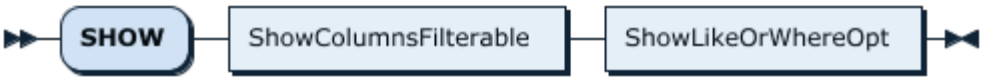


Figure 396: ShowStmt

ShowColumnsFilterable:



Figure 397: ShowColumnsFilterable

OptFull:



Figure 398: OptFull

FieldsOrColumns:



Figure 399: FieldsOrColumns

ShowTableAliasOpt:



Figure 400: ShowTableAliasOpt

FromOrIn:

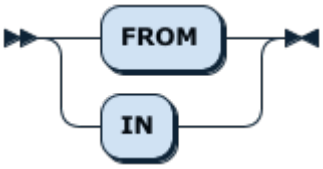


Figure 401: FromOrIn

TableName:

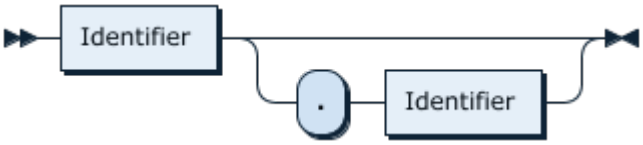


Figure 402: TableName

ShowDatabaseNameOpt:

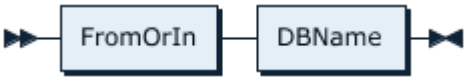


Figure 403: ShowDatabaseNameOpt

DBName:



Figure 404: DBName

ShowLikeOrWhereOpt:

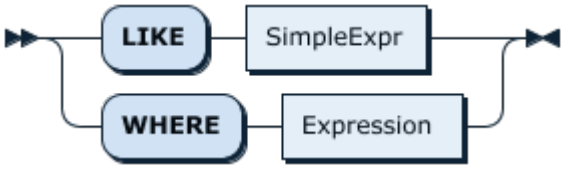


Figure 405: ShowLikeOrWhereOpt

14.10.2.90.2 Examples

```
mysql> create view v1 as select 1;
Query OK, 0 rows affected (0.11 sec)

mysql> show columns from v1;
+------+----------+------+------+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+------+----------+------+------+---------+-------+
| 1    | bigint(1) | YES |    | NULL   |       |
+------+----------+------+------+---------+-------+
1 row in set (0.00 sec)

mysql> desc v1;
+------+----------+------+------+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+------+----------+------+------+---------+-------+
| 1    | bigint(1) | YES |    | NULL   |       |
+------+----------+------+------+---------+-------+
1 row in set (0.00 sec)

mysql> describe v1;
+------+----------+------+------+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+------+----------+------+------+---------+-------+
| 1    | bigint(1) | YES |    | NULL   |       |
+------+----------+------+------+---------+-------+
1 row in set (0.00 sec)

mysql> explain v1;
+------+----------+------+------+---------+-------+
```

```
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| 1     | bigint(1) | YES |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> show fields from v1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| 1     | bigint(1) | YES |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql> show full columns from v1;
+--
   ↪ -----+----------+----------+------+-----+---------+-------+-----------------
   ↪
| Field | Type    | Collation | Null | Key | Default | Extra | Privileges
   ↪              | Comment |
+--
   ↪ -----+----------+----------+------+-----+---------+-------+-----------------
   ↪
| 1     | bigint(1) | NULL    | YES |     | NULL    |       | select,insert,
   ↪ update,references |    |
+--
   ↪ -----+----------+----------+------+-----+---------+-------+-----------------
   ↪
1 row in set (0.00 sec)

mysql> show full columns from mysql.user;
+--
   ↪ --------------------+--------------+------------+------+-----+---------+----
   ↪
| Field                  | Type         | Collation  | Null | Key | Default |
   ↪ Extra | Privileges                | Comment |
+--
   ↪ --------------------+--------------+------------+------+-----+---------+----
   ↪
| Host                   | char(64)     | utf8mb4_bin | NO | PRI | NULL    |
   ↪ | select,insert,update,references |    |
| User                   | char(32)     | utf8mb4_bin | NO | PRI | NULL    |
   ↪ | select,insert,update,references |    |
| authentication_string | text         | utf8mb4_bin | YES |     | NULL    |
   ↪ | select,insert,update,references |    |
```

```
| Select_priv          | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Insert_priv          | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Update_priv          | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Delete_priv          | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Create_priv          | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Drop_priv            | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Process_priv         | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Grant_priv           | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| References_priv      | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Alter_priv           | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Show_db_priv         | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Super_priv           | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Create_tmp_table_priv | enum('N','Y') | utf8mb4_bin | NO | | N     |
   ↪ | select,insert,update,references |    |
| Lock_tables_priv     | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Execute_priv         | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Create_view_priv     | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Show_view_priv       | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Create_routine_priv  | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Alter_routine_priv   | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Index_priv           | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Create_user_priv     | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Event_priv           | enum('N','Y') | utf8mb4_bin | NO |   | N      |
   ↪ | select,insert,update,references |    |
| Trigger_priv         | enum('N','Y') | utf8mb4_bin | NO |   | N      |
```

```
    ↪ | select,insert,update,references |    |
| Create_role_priv    | enum('N','Y') | utf8mb4_bin | NO |    | N      |
    ↪ | select,insert,update,references |    |
| Drop_role_priv      | enum('N','Y') | utf8mb4_bin | NO |    | N      |
    ↪ | select,insert,update,references |    |
| Account_locked      | enum('N','Y') | utf8mb4_bin | NO |    | N      |
    ↪ | select,insert,update,references |    |
| Shutdown_priv       | enum('N','Y') | utf8mb4_bin | NO |    | N      |
    ↪ | select,insert,update,references |    |
| Reload_priv         | enum('N','Y') | utf8mb4_bin | NO |    | N      |
    ↪ | select,insert,update,references |    |
| FILE_priv           | enum('N','Y') | utf8mb4_bin | NO |    | N      |
    ↪ | select,insert,update,references |    |
| Config_priv         | enum('N','Y') | utf8mb4_bin | NO |    | N      |
    ↪ | select,insert,update,references |    |
+--
    ↪ --------------------+--------------+------------+------+------+----------+----
    ↪
33 rows in set (0.01 sec)
```

14.10.2.90.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.90.4 See also

- SHOW CREATE TABLE

14.10.2.91 SHOW CONFIG

The `SHOW CONFIG` statement is used to show the current configuration of various components of TiDB. Note that the configuration and system variables act on different dimensions and should not be mixed up. If you want to obtain the system variable information, use the SHOW VARIABLES syntax.

14.10.2.91.1 Synopsis

ShowStmt:

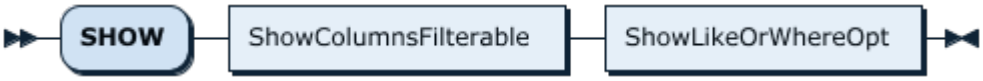


Figure 406: ShowStmt

ShowTargetFilterable:

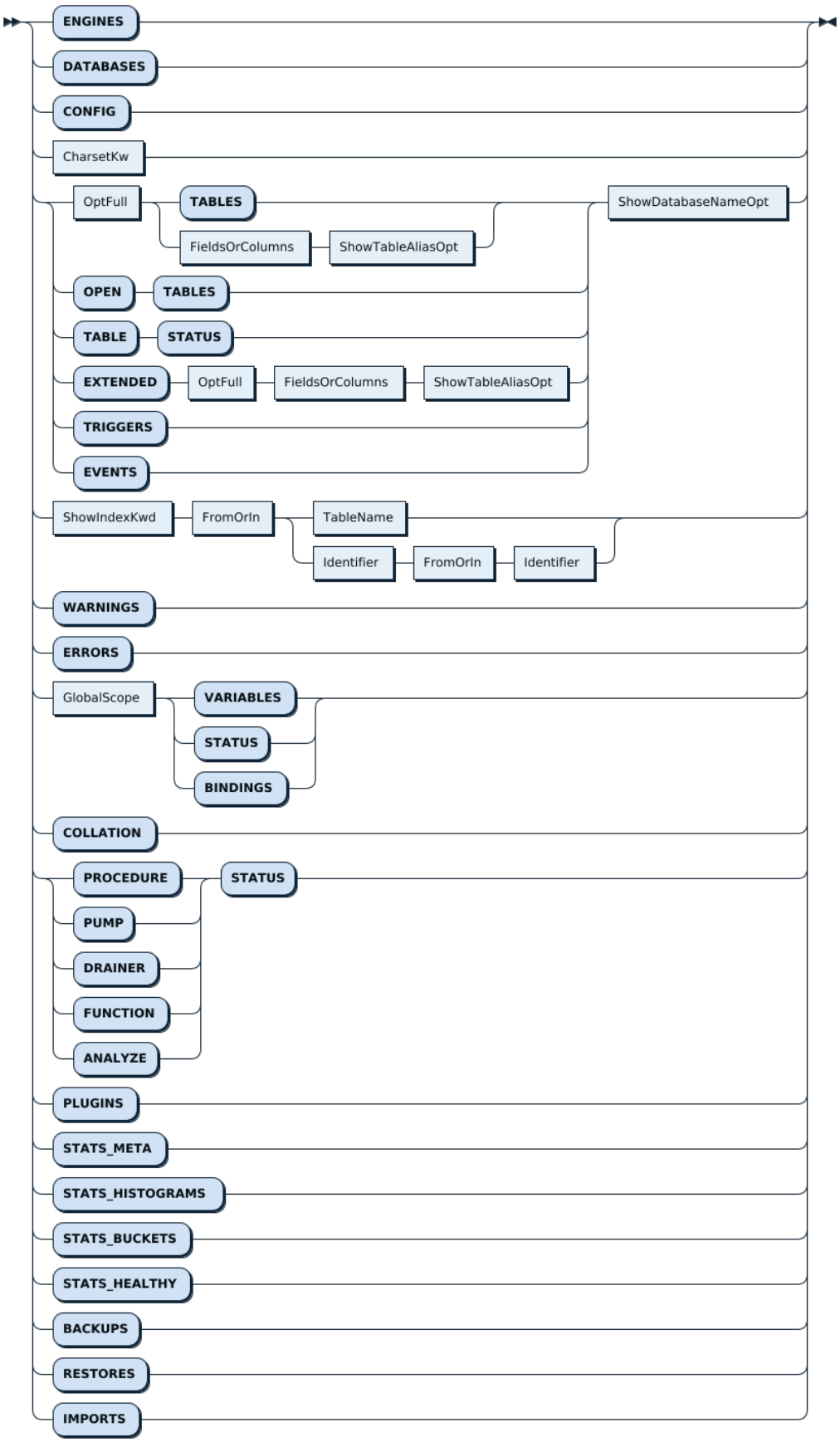


Figure 407: ShowTargetFilterable

14.10.2.91.2 Examples

Show all configurations:

```
SHOW CONFIG;
```

```
+------+----------------+-----------------------------------------------------+-------------
   ↪
| Type | Instance     | Name                                          | Value
   ↪                                                              |
+------+----------------+-----------------------------------------------------+-------------
   ↪
| tidb | 127.0.0.1:4000 | advertise-address                             | 127.0.0.1
   ↪                                                          |
| tidb | 127.0.0.1:4000 | binlog.binlog-socket                          |
   ↪                                                                  |
| tidb | 127.0.0.1:4000 | binlog.enable                                 | false
   ↪                                                              |
...
120 rows in set (0.01 sec)
```

Show the configuration where the `type` is `tidb`:

```
SHOW CONFIG WHERE type = 'tidb' AND name = 'advertise-address';
```

```
+------+----------------+-------------------+-----------+
| Type | Instance     | Name            | Value    |
+------+----------------+-------------------+-----------+
| tidb | 127.0.0.1:4000 | advertise-address | 127.0.0.1 |
+------+----------------+-------------------+-----------+
1 row in set (0.05 sec)
```

You can also use the `LIKE` clause to show the configuration where the `type` is `tidb`:

```
SHOW CONFIG LIKE 'tidb';
```

```
+------+----------------+-----------------------------------------------------+-------------
   ↪
| Type | Instance     | Name                                          | Value
   ↪                                                              |
+------+----------------+-----------------------------------------------------+-------------
   ↪
| tidb | 127.0.0.1:4000 | advertise-address                             | 127.0.0.1
   ↪                                                          |
| tidb | 127.0.0.1:4000 | binlog.binlog-socket                          |
   ↪                                                                  |
```

```
| tidb | 127.0.0.1:4000 | binlog.enable                                  | false
    ↪                                                                |
...
40 rows in set (0.01 sec)
```

14.10.2.91.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.91.4 See also

- SHOW VARIABLES

14.10.2.92 SHOW CREATE DATABASE

`SHOW CREATE DATABASE` is used to show the exact SQL statement for re-creating an existing database. `SHOW CREATE SCHEMA` is a synonym for it.

14.10.2.92.1 Synopsis

ShowCreateDatabaseStmt:

```
ShowCreateDatabaseStmt ::=
    "SHOW" "CREATE" "DATABASE" | "SCHEMA" ("IF" "NOT" "EXISTS")? DBName
```

14.10.2.92.2 Examples

```
CREATE DATABASE test;
```

```
Query OK, 0 rows affected (0.12 sec)
```

```
SHOW CREATE DATABASE test;
```

```
+--
    ↪ --------+-----------------------------------------------------------------+
    ↪
| Database | Create Database                                                  |
+--
    ↪ --------+-----------------------------------------------------------------+
    ↪
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4
    ↪ */ |
+--
    ↪ --------+-----------------------------------------------------------------+
    ↪
```

```
1 row in set (0.00 sec)
```

```
SHOW CREATE SCHEMA IF NOT EXISTS test;
```

```
+--
    ↪ --------+-----------------------------------------------------------------------
    ↪
| Database | Create Database
    ↪                                                                          |
+--
    ↪ --------+-----------------------------------------------------------------------
    ↪
| test     | CREATE DATABASE /*!32312 IF NOT EXISTS*/ `test` /*!40100
    ↪ DEFAULT CHARACTER SET utf8mb4 */ |
+--
    ↪ --------+-----------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.2.92.3 MySQL compatibility

`SHOW CREATE DATABASE` is expected to be fully compatible with MySQL. If you find any compatibility differences, submit a GitHub issue.

14.10.2.92.4 See also

- CREATE TABLE
- DROP TABLE
- SHOW TABLES
- SHOW COLUMNS FROM

14.10.2.93 SHOW CREATE PLACEMENT POLICY

`SHOW CREATE PLACEMENT POLICY` is used to show the definition of a placement policy. This can be used to see the current definition of a placement policy and recreate it in another TiDB cluster.

14.10.2.93.1 Synopsis

```
ShowCreatePlacementPolicyStmt ::=
    "SHOW" "CREATE" "PLACEMENT" "POLICY" PolicyName

PolicyName ::=
    Identifier
```

14.10.2.93.2 Examples

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us
    ↪ -west-1" FOLLOWERS=4;
CREATE TABLE t1 (a INT) PLACEMENT POLICY=p1;
SHOW CREATE PLACEMENT POLICY p1\G;
```

```
Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.10 sec)

***************************[ 1. row ]***************************
Policy      | p1
Create Policy | CREATE PLACEMENT POLICY `p1` PRIMARY_REGION="us-east-1"
    ↪ REGIONS="us-east-1,us-west-1" FOLLOWERS=4
1 row in set (0.00 sec)
```

14.10.2.93.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.93.4 See also

- Placement Rules in SQL
- SHOW PLACEMENT
- CREATE PLACEMENT POLICY
- ALTER PLACEMENT POLICY
- DROP PLACEMENT POLICY

14.10.2.94 SHOW CREATE SEQUENCE

The `SHOW CREATE SEQUENCE` shows the detailed information of a sequence, which is similar to `SHOW CREATE TABLE`.

14.10.2.94.1 Synopsis

ShowCreateSequenceStmt:



Figure 408: ShowCreateSequenceStmt

TableName:

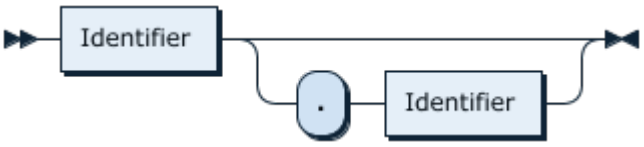


Figure 409: TableName

14.10.2.94.2 Examples

```
CREATE SEQUENCE seq;
```

```
Query OK, 0 rows affected (0.03 sec)
```

```
SHOW CREATE SEQUENCE seq;
```

```
+-------+-------------------------------------------------------------------------------
   ↪
| Table | Create Table
   ↪
   ↪ |
+-------+-------------------------------------------------------------------------------
   ↪
| seq   | CREATE SEQUENCE `seq` start with 1 minvalue 1 maxvalue
   ↪ 9223372036854775806 increment by 1 cache 1000 nocycle ENGINE=InnoDB |
+-------+-------------------------------------------------------------------------------
   ↪
1 row in set (0.00 sec)
```

14.10.2.94.3 MySQL compatibility

This statement is a TiDB extension. The implementation is modeled on sequences available in MariaDB.

14.10.2.94.4 See also

- CREATE SEQUENCE
- DROP SEQUENCE

14.10.2.95 SHOW CREATE TABLE

This statement shows the exact statement to recreate an existing table using SQL.

14.10.2.95.1 Synopsis

ShowCreateTableStmt:

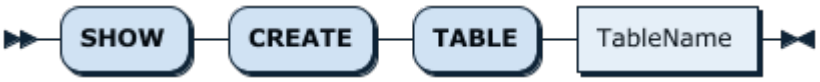


Figure 410: ShowCreateTableStmt

TableName:

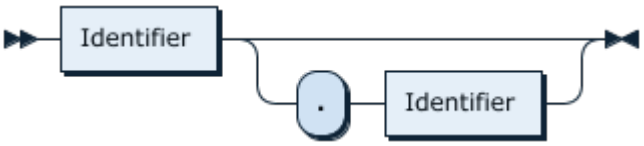


Figure 411: TableName

14.10.2.95.2 Examples

```
mysql> CREATE TABLE t1 (a INT);
Query OK, 0 rows affected (0.12 sec)

mysql> SHOW CREATE TABLE t1;
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| t1    | CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.2.95.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.95.4 See also

- CREATE TABLE
- DROP TABLE
- SHOW TABLES
- SHOW COLUMNS FROM

14.10.2.96 SHOW CREATE USER

This statement shows how to re-create a user using the `CREATE USER` syntax.

14.10.2.96.1 Synopsis

ShowCreateUserStmt:

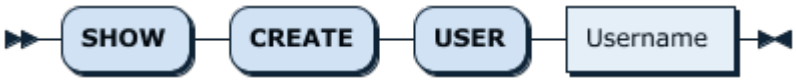


Figure 412: ShowCreateUserStmt

Username:

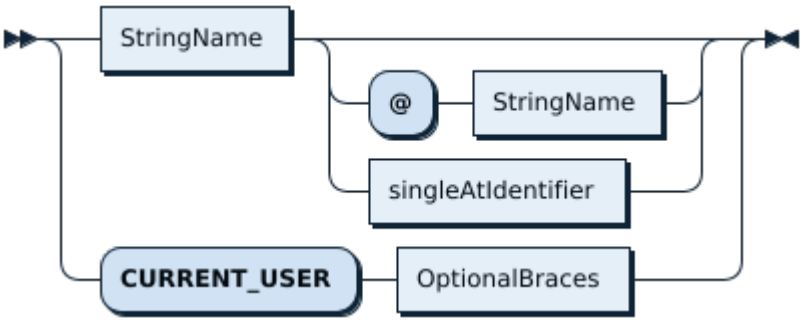


Figure 413: Username

14.10.2.96.2 Examples

```
mysql> SHOW CREATE USER 'root';
+--
    ↪ ------------------------------------------------------------------------
    ↪
| CREATE USER for root@%
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------
    ↪
```

```
| CREATE USER 'root'@'%' IDENTIFIED WITH 'mysql_native_password' AS ''
    ↪ REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK |
+--
    ↪ ----------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> SHOW GRANTS FOR 'root';
+----------------------------------------+
| Grants for root@%                      |
+----------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' |
+----------------------------------------+
1 row in set (0.00 sec)
```

14.10.2.96.3 MySQL compatibility

- The output of `SHOW CREATE USER` is designed to match MySQL, but several of the `CREATE` options are not yet supported by TiDB. Not yet supported options will be parsed but ignored. See [security compatibility] for more details.

14.10.2.96.4 See also

- CREATE USER
- SHOW GRANTS
- DROP USER

14.10.2.97 SHOW DATABASES

This statement shows a list of databases that the current user has privileges to. Databases which the current user does not have access to will appear hidden from the list. The `information_schema` database always appears first in the list of databases.

`SHOW SCHEMAS` is an alias of this statement.

14.10.2.97.1 Synopsis

ShowDatabasesStmt:



Figure 414: ShowDatabasesStmt

ShowLikeOrWhereOpt:

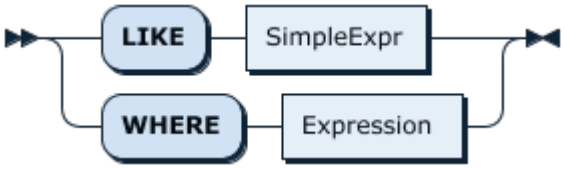


Figure 415: ShowLikeOrWhereOpt

14.10.2.97.2 Examples

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql> CREATE DATABASE mynewdb;
Query OK, 0 rows affected (0.10 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| mynewdb            |
| mysql              |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

14.10.2.97.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.97.4 See also

- SHOW SCHEMAS
- DROP DATABASE
- CREATE DATABASE

14.10.2.98 SHOW DRAINER STATUS

The `SHOW DRAINER STATUS` statement displays the status information for all Drainer nodes in the cluster.

14.10.2.98.1 Examples

```
SHOW DRAINER STATUS;
```

```
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
|  NodeID |     Address    | State |   Max_Commit_Ts   |    Update_Time     |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
| drainer1 | 127.0.0.3:8249 | Online | 408553768673342532 | 2019-05-01
    ↪ 00:00:03 |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
| drainer2 | 127.0.0.4:8249 | Online | 408553768673345531 | 2019-05-01
    ↪ 00:00:04 |
+--
    ↪ --------/----------------/--------/--------------------/--------------------/
    ↪
2 rows in set (0.00 sec)
```

14.10.2.98.2 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.98.3 See also

- SHOW PUMP STATUS
- CHANGE PUMP STATUS
- CHANGE DRAINER STATUS

14.10.2.99 SHOW ENGINES

This statement is used to list all supported storage engines. The syntax is included only for compatibility with MySQL.

14.10.2.99.1 Synopsis

ShowEnginesStmt:

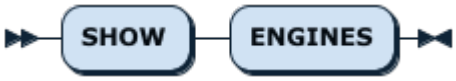


Figure 416: ShowEnginesStmt

```
SHOW ENGINES;
```

14.10.2.99.2 Examples

```
mysql> SHOW ENGINES;
+--
    ↪ ------+---------+----------------------------------------------------------------+--
    ↪
| Engine | Support | Comment                                                        |
    ↪ Transactions | XA | Savepoints |
+--
    ↪ ------+---------+----------------------------------------------------------------+--
    ↪
| InnoDB | DEFAULT | Supports transactions, row-level locking, and foreign
    ↪ keys | YES | YES | YES    |
+--
    ↪ ------+---------+----------------------------------------------------------------+--
    ↪
1 row in set (0.00 sec)
```

14.10.2.99.3 MySQL compatibility

- This statement will always only return InnoDB as the supported engine. Internally, TiDB will typically use TiKV as the storage engine.

14.10.2.100 SHOW ERRORS

This statement shows errors from previously executed statements. The error buffer is cleared as soon as a statement executes successfully. In which case, `SHOW ERRORS` will return an empty set.

The behavior of which statements generate errors vs. warnings is highly influenced by the current `sql_mode`.

14.10.2.100.1 Synopsis

ShowErrorsStmt:

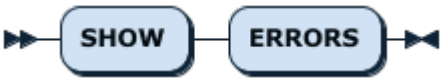


Figure 417: ShowErrorsStmt

14.10.2.100.2 Examples

```
mysql> select invalid;
ERROR 1054 (42S22): Unknown column 'invalid' in 'field list'
mysql> create invalid;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
    ↪ that corresponds to your TiDB version for the right syntax to use
    ↪ line 1 column 14 near "invalid"
mysql> SHOW ERRORS;
+--
    ↪ -----+------+-----------------------------------------------------------------
    ↪
| Level | Code | Message
    ↪
    ↪ |
+--
    ↪ -----+------+-----------------------------------------------------------------
    ↪
| Error | 1054 | Unknown column 'invalid' in 'field list'
    ↪
    ↪ |
| Error | 1064 | You have an error in your SQL syntax; check the manual
    ↪ that corresponds to your TiDB version for the right syntax to use
    ↪ line 1 column 14 near "invalid" |
+--
    ↪ -----+------+-----------------------------------------------------------------
    ↪
2 rows in set (0.00 sec)

mysql> CREATE invalid2;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
    ↪ that corresponds to your TiDB version for the right syntax to use
    ↪ line 1 column 15 near "invalid2"
mysql> SELECT 1;
+------+
| 1    |
+------+
```

```
|    1 |
+------+
1 row in set (0.00 sec)

mysql> SHOW ERRORS;
Empty set (0.00 sec)
```

14.10.2.100.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.100.4 See also

- SHOW WARNINGS

14.10.2.101 SHOW [FULL] FIELDS FROM

This statement is an alias to `SHOW [FULL] COLUMNS FROM`. It is included for compatibility with MySQL.

14.10.2.102 SHOW GRANTS

This statement shows a list of privileges associated with a user. As in MySQL, the `USAGE` privileges denotes the ability to login to TiDB.

14.10.2.102.1 Synopsis

ShowGrantsStmt:

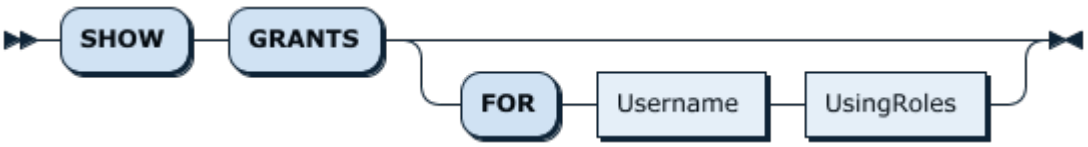


Figure 418: ShowGrantsStmt

Username:

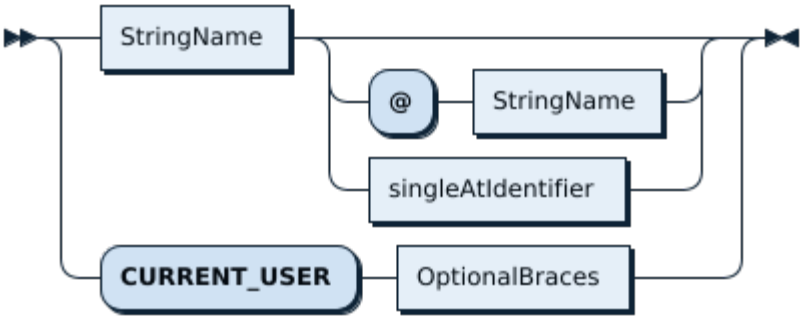


Figure 419: Username

UsingRoles:



Figure 420: UsingRoles

RolenameList:

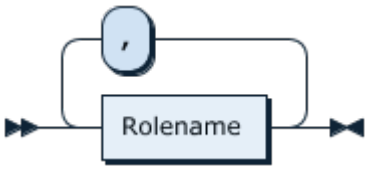


Figure 421: RolenameList

Rolename:

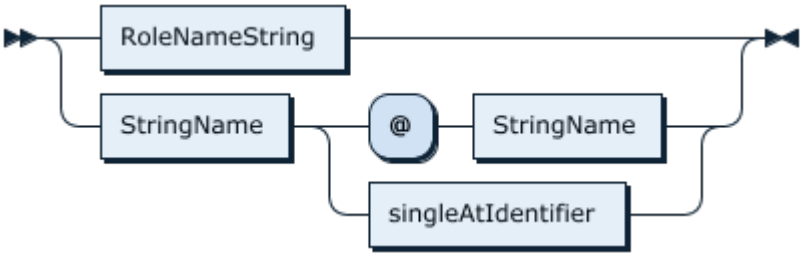


Figure 422: Rolename

14.10.2.102.2 Examples

```
mysql> SHOW GRANTS;
+----------------------------------------+
```

```
| Grants for User                                |
+------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' |
+------------------------------------------------+
1 row in set (0.00 sec)

mysql> SHOW GRANTS FOR 'u1';
ERROR 1141 (42000): There is no such grant defined for user 'u1' on host '%
    ↪ '
mysql> CREATE USER u1;
Query OK, 1 row affected (0.04 sec)

mysql> GRANT SELECT ON test.* TO u1;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW GRANTS FOR u1;
+-----------------------------------+
| Grants for u1@%                   |
+-----------------------------------+
| GRANT USAGE ON *.* TO 'u1'@'%' |
| GRANT Select ON test.* TO 'u1'@'%' |
+-----------------------------------+
2 rows in set (0.00 sec)
```

14.10.2.102.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.102.4 See also

- SHOW CREATE USER
- GRANT

14.10.2.103 SHOW INDEX [FROM|IN]

This statement is an alias to `SHOW INDEXES [FROM|IN]`. It is included for compatibility with MySQL.

14.10.2.104 SHOW INDEXES [FROM|IN]

The statement `SHOW INDEXES [FROM|IN]` lists the indexes on a specified table. The statements `SHOW INDEX [FROM|IN]`, `SHOW KEYS [FROM|IN]` are aliases of this statement, and included for compatibility with MySQL.

14.10.2.104.1 Synopsis

ShowIndexStmt:

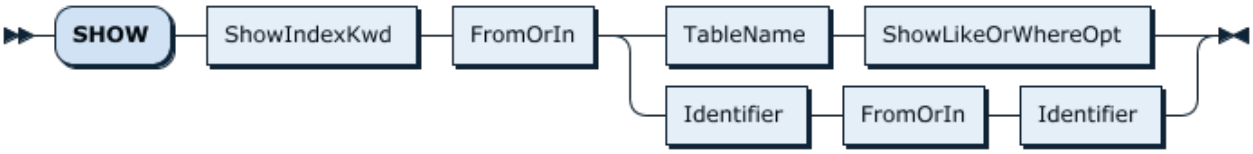


Figure 423: ShowIndexStmt

ShowIndexKwd:

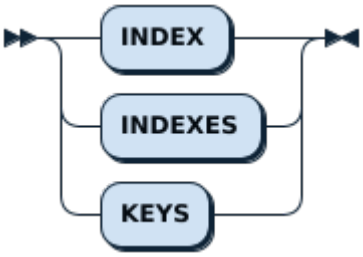


Figure 424: ShowIndexKwd

FromOrIn:

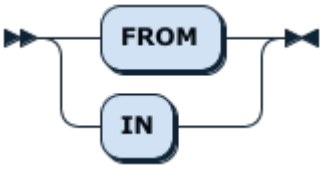


Figure 425: FromOrIn

TableName:

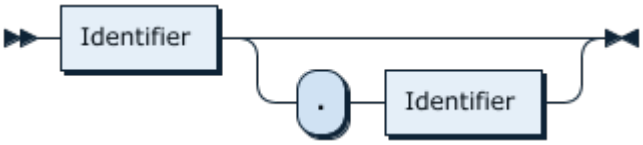


Figure 426: TableName

ShowLikeOrWhereOpt:

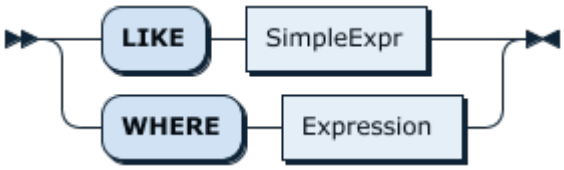


Figure 427: ShowLikeOrWhereOpt

14.10.2.104.2 Examples

```
mysql> CREATE TABLE t1 (id int not null primary key AUTO_INCREMENT, col1
    ↪ INT, INDEX(col1));
Query OK, 0 rows affected (0.12 sec)

mysql> SHOW INDEXES FROM t1;
+--
    ↪ -----+------------+---------+-------------+------------+---------+--------
    ↪
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation |
    ↪ Cardinality | Sub_part | Packed | Null | Index_type | Comment |
    ↪ Index_comment | Visible | Expression |
+--
    ↪ -----+------------+---------+-------------+------------+---------+--------
    ↪
| t1    |          0 | PRIMARY |           1 | id         | A         |          0
    ↪  |     NULL | NULL |      | BTREE      |         |               | YES
    | NULL       |
| t1    |          1 | col1    |           1 | col1       | A         |          0
    ↪  |     NULL | NULL | YES | BTREE      |         |               | YES
    | NULL       |
+--
    ↪ -----+------------+---------+-------------+------------+---------+--------
    ↪
2 rows in set (0.00 sec)

mysql> SHOW INDEX FROM t1;
+--
    ↪ -----+------------+---------+-------------+------------+---------+--------
    ↪
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation |
    ↪ Cardinality | Sub_part | Packed | Null | Index_type | Comment |
    ↪ Index_comment | Visible | Expression |
+--
    ↪ -----+------------+---------+-------------+------------+---------+--------
    ↪
| t1    |          0 | PRIMARY |           1 | id         | A         |          0 |
```

```
    ↪  |    NULL | NULL |      | BTREE     |         |               | YES
    | NULL       |
| t1    |          1 | col1    |           1 | col1       | A         |          0
    ↪  |    NULL | NULL | YES | BTREE     |         |               | YES
    | NULL       |
+--
    ↪ -----+------------+----------+-------------+------------+----------+--------
    ↪
2 rows in set (0.00 sec)

mysql> SHOW KEYS FROM t1;
+--
    ↪ -----+------------+----------+-------------+------------+----------+--------
    ↪
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation |
    ↪ Cardinality | Sub_part | Packed | Null | Index_type | Comment |
    ↪ Index_comment | Visible | Expression |
+--
    ↪ -----+------------+----------+-------------+------------+----------+--------
    ↪
| t1    |          0 | PRIMARY |           1 | id         | A         |          0
    ↪  |    NULL | NULL |      | BTREE     |         |               | YES
    | NULL       |
| t1    |          1 | col1    |           1 | col1       | A         |          0
    ↪  |    NULL | NULL | YES | BTREE     |         |               | YES
    | NULL       |
+--
    ↪ -----+------------+----------+-------------+------------+----------+--------
    ↪
2 rows in set (0.00 sec)
```

14.10.2.104.3 MySQL compatibility

The `Cardinality` column in MySQL shows the number of different values on the index. In TiDB, the `Cardinality` column always shows `0`.

14.10.2.104.4 See also

- SHOW CREATE TABLE
- DROP INDEX
- CREATE INDEX

14.10.2.105 SHOW KEYS [FROM|IN]

This statement is an alias to SHOW INDEXES [FROM|IN]. It is included for compatibility with MySQL.

14.10.2.106 SHOW MASTER STATUS

The SHOW MASTER STATUS statement displays the latest TSO in the cluster.

14.10.2.106.1 Examples

```
SHOW MASTER STATUS;
```

```
+--
    ↪ ----------+--------------------+-------------+-----------------+-----------
    ↪
| File       | Position          | Binlog_Do_DB | Binlog_Ignore_DB |
    ↪ Executed_Gtid_Set |
+--
    ↪ ----------+--------------------+-------------+-----------------+-----------
    ↪
| tidb-binlog | 416916363252072450 |        |                 |
    ↪                  |
+--
    ↪ ----------+--------------------+-------------+-----------------+-----------
    ↪
1 row in set (0.00 sec)
```

14.10.2.106.2 MySQL compatibility

The output of SHOW MASTER STATUS is designed to match MySQL. However, the execution results are different in that the MySQL result is the binlog location information and the TiDB result is the latest TSO information.

14.10.2.106.3 See also

- SHOW PUMP STATUS
- SHOW DRAINER STATUS
- CHANGE PUMP STATUS
- CHANGE DRAINER STATUS

14.10.2.107 SHOW PLACEMENT

SHOW PLACEMENT summarizes all placement options from placement policies, and presents them in canonical form.

The statement returns a result set in which the Scheduling_State field indicates the current progress that the Placement Driver (PD) has made in scheduling the placement:

- `PENDING`: The PD has not yet started scheduling the placement. This might indicate that that the placement rules are semantically correct, but cannot currently be satisfied by the cluster. For example, if `FOLLOWERS=4` but there are only 3 TiKV stores which are candidates for followers.
- `INPROGRESS`: The PD is currently scheduling the placement.
- `SCHEDULED`: The PD has successfully scheduled the placement.

14.10.2.107.1 Synopsis

```
ShowStmt ::=
    "PLACEMENT"
```

14.10.2.107.2 Examples

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us
    ↪ -west-1" FOLLOWERS=4;
CREATE TABLE t1 (a INT) PLACEMENT POLICY=p1;
SHOW PLACEMENT;
```

```
Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

+---------------+-------------------------------------------------------------------------+
    ↪
| Target        | Placement
    ↪                                                         | Scheduling_State
    ↪ |
+---------------+-------------------------------------------------------------------------+
    ↪
| POLICY p1     | PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-west-1"
    ↪ FOLLOWERS=4 | NULL  |
| DATABASE test | PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-west-1"
    ↪ FOLLOWERS=4 | INPROGRESS |
| TABLE test.t1 | PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-west-1"
    ↪ FOLLOWERS=4 | INPROGRESS |
+---------------+-------------------------------------------------------------------------+
    ↪
4 rows in set (0.00 sec)
```

14.10.2.107.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.107.4 See also

- Placement Rules in SQL
- SHOW PLACEMENT FOR
- CREATE PLACEMENT POLICY

14.10.2.108 SHOW PLACEMENT FOR

`SHOW PLACEMENT FOR` summarizes all placement options, and presents them in the canonical form for a specific table, database schema, or partition.

The statement returns a result set in which the `Scheduling_State` field indicates the current progress that the Placement Driver (PD) has made in scheduling the placement:

- `PENDING`: The PD has not yet started scheduling the placement. This might indicate that that the placement rules are semantically correct, but cannot currently be satisfied by the cluster. For example, if `FOLLOWERS=4` but there are only 3 TiKV stores that are candidates for followers.
- `INPROGRESS`: The PD is currently scheduling the placement.
- `SCHEDULED`: The PD has successfully scheduled the placement.

14.10.2.108.1 Synopsis

```
ShowStmt ::=
    "PLACEMENT" "FOR" ShowPlacementTarget

ShowPlacementTarget ::=
    DatabaseSym DBName
|   "TABLE" TableName
|   "TABLE" TableName "PARTITION" Identifier
```

14.10.2.108.2 Examples

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us
    ↪ -west-1" FOLLOWERS=4;
ALTER DATABASE test PLACEMENT POLICY=p1;
CREATE TABLE t1 (a INT);
SHOW PLACEMENT FOR DATABASE test;
SHOW PLACEMENT FOR TABLE t1;
SHOW CREATE TABLE t1\G;
CREATE TABLE t3 (a INT) PARTITION BY RANGE (a) (PARTITION p1 VALUES LESS
    ↪ THAN (10), PARTITION p2 VALUES LESS THAN (20));
SHOW PLACEMENT FOR TABLE t3 PARTITION p1\G;
```

```
Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

+---------------+-------------------------------------------------------------------------+
   ↪
| Target      | Placement
   ↪                                                          | Scheduling_State
   ↪ |
+---------------+-------------------------------------------------------------------------+
   ↪
| DATABASE test | PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-west-1"
   ↪ FOLLOWERS=4 | INPROGRESS |
+---------------+-------------------------------------------------------------------------+
   ↪
1 row in set (0.00 sec)

+---------------+-------------+------------------+
| Target      | Placement | Scheduling_State |
+---------------+-------------+------------------+
| TABLE test.t1 | FOLLOWERS=4 | INPROGRESS |
+---------------+-------------+------------------+
1 row in set (0.00 sec)

***************************[ 1. row ]***************************
Table      | t1
Create Table | CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin /*T![placement]
   ↪ PLACEMENT POLICY=`p1` */
1 row in set (0.00 sec)

***************************[ 1. row ]***************************
Target         | TABLE test.t3 PARTITION p1
Placement      | PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-west-1"
   ↪ FOLLOWERS=4
Scheduling_State | PENDING
1 row in set (0.00 sec)
```

14.10.2.108.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.108.4 See also

- Placement Rules in SQL
- SHOW PLACEMENT
- CREATE PLACEMENT POLICY

14.10.2.109 SHOW PLACEMENT LABELS

`SHOW PLACEMENT LABELS` is used to summarize the labels and values that are available for Placement Rules.

14.10.2.109.1 Synopsis

```
ShowStmt ::=
    "PLACEMENT" "LABELS"
```

14.10.2.109.2 Examples

```
SHOW PLACEMENT LABELS;
```

```
+--------+----------------+
| Key    | Values         |
+--------+----------------+
| region | ["us-east-1"] |
| zone   | ["us-east-1a"] |
+--------+----------------+
2 rows in set (0.00 sec)
```

14.10.2.109.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.109.4 See also

- Placement Rules in SQL
- SHOW PLACEMENT
- CREATE PLACEMENT POLICY

14.10.2.110 SHOW PLUGINS

`SHOW PLUGINS` shows all plugins installed in TiDB, including each plugin's status and version information.

14.10.2.110.1 Synopsis

ShowStmt:

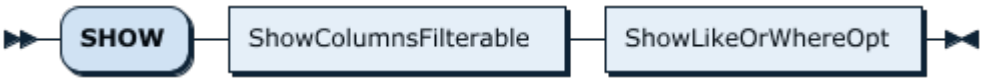


Figure 428: ShowStmt

ShowTargetFilterable:

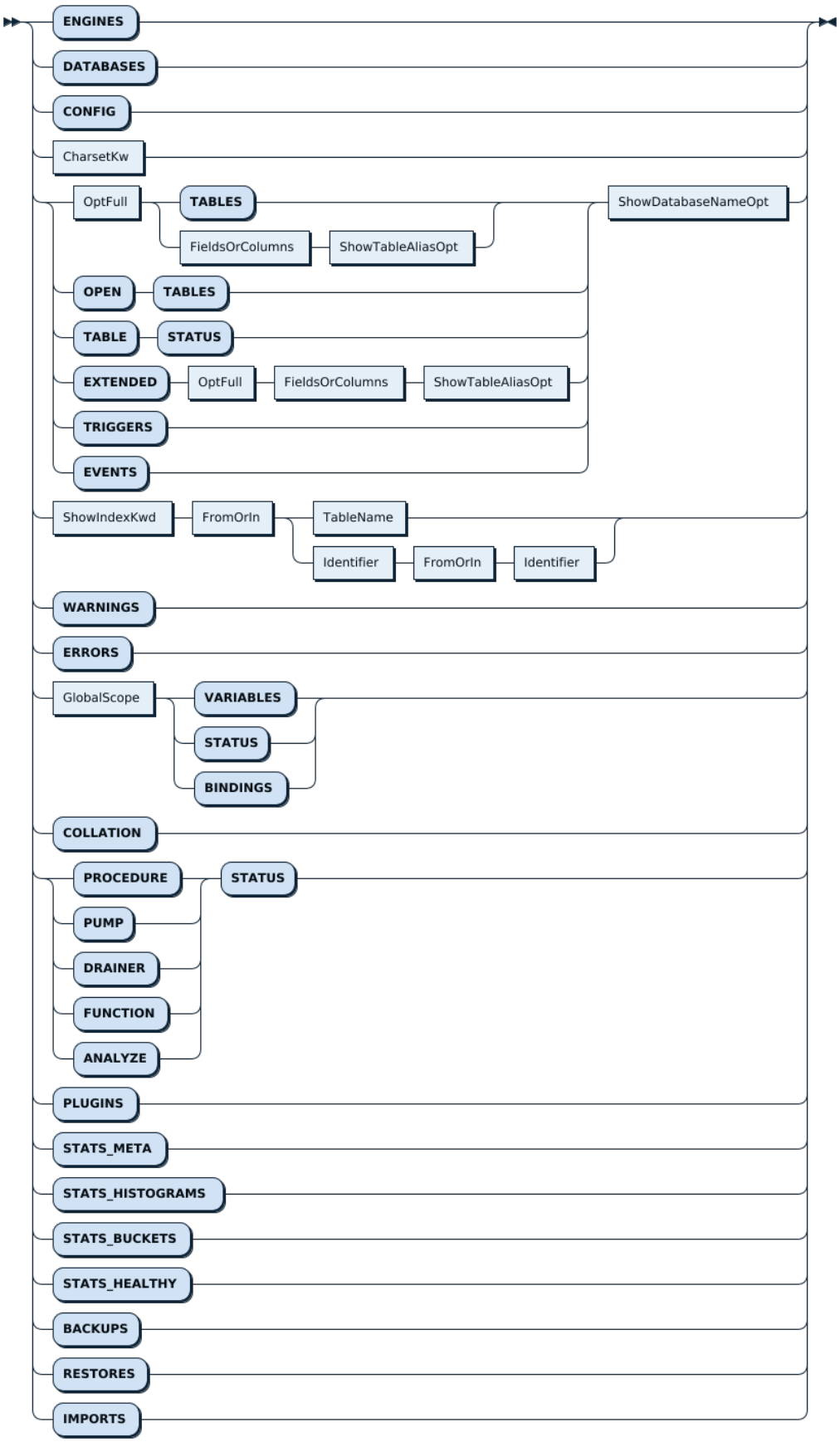


Figure 429: ShowTargetFilterable

14.10.2.110.2 Examples

```
SHOW PLUGINS;
```

```
+-------+-------------+-------+----------------------------+---------+---------+
    ↪
| Name | Status      | Type | Library                     | License | Version |
+-------+-------------+-------+----------------------------+---------+---------+
    ↪
| audit | Ready-enable | Audit | /tmp/tidb/plugin/audit-1.so | | 1     |
+-------+-------------+-------+----------------------------+---------+---------+
    ↪
1 row in set (0.000 sec)
```

```
SHOW PLUGINS LIKE 'a%';
```

```
+-------+-------------+-------+----------------------------+---------+---------+
    ↪
| Name | Status      | Type | Library                     | License | Version |
+-------+-------------+-------+----------------------------+---------+---------+
    ↪
| audit | Ready-enable | Audit | /tmp/tidb/plugin/audit-1.so | | 1     |
+-------+-------------+-------+----------------------------+---------+---------+
    ↪
1 row in set (0.000 sec)
```

14.10.2.110.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.111 SHOW PRIVILEGES

This statement shows a list of assignable privileges in TiDB. It is a static list, and does not reflect the privileges of the current user.

14.10.2.111.1 Synopsis

ShowStmt:

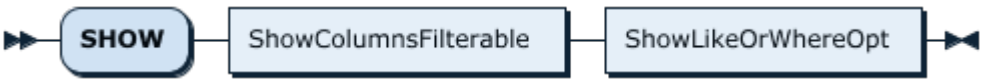


Figure 430: ShowStmt

14.10.2.111.2 Examples

```
mysql> show privileges;
+--
   ↪ ----------------------+-----------------------------------------+---------------
   ↪
| Privilege             | Context                                 | Comment
   ↪                                          |
+--
   ↪ ----------------------+-----------------------------------------+---------------
   ↪
| Alter                 | Tables                                  | To alter the
   ↪ table                                  |
| Alter                 | Tables                                  | To alter the
   ↪ table                                  |
| Alter routine         | Functions,Procedures                    | To alter or drop
   ↪  stored functions/procedures   |
| Create                | Databases,Tables,Indexes                | To create new
   ↪ databases and tables                |
| Create routine        | Databases                               | To use CREATE
   ↪ FUNCTION/PROCEDURE                  |
| Create temporary tables | Databases                             | To use CREATE
   ↪ TEMPORARY TABLE                     |
| Create view           | Tables                                  | To create new
   ↪ views                               |
| Create user           | Server Admin                            | To create new
   ↪ users                               |
| Delete                | Tables                                  | To delete
   ↪ existing rows                          |
| Drop                  | Databases,Tables                        | To drop
   ↪ databases, tables, and views           |
| Event                 | Server Admin                            | To create, alter
   ↪ , drop and execute events      |
| Execute               | Functions,Procedures                    | To execute
   ↪ stored routines                        |
| File                  | File access on server                   | To read and
   ↪ write files on the server          |
| Grant option          | Databases,Tables,Functions,Procedures | To give to
   ↪ other users those privileges you possess |
| Index                 | Tables                                  | To create or
   ↪ drop indexes                        |
| Insert                | Tables                                  | To insert data
   ↪ into tables                         |
| Lock tables           | Databases                               | To use LOCK
   ↪ TABLES (together with SELECT privilege) |
```

```
| Process              | Server Admin                        | To view the
   ↪ plain text of currently executing queries |
| Proxy                | Server Admin                        | To make proxy
   ↪ user possible                     |
| References           | Databases,Tables                    | To have
   ↪ references on tables                    |
| Reload               | Server Admin                        | To reload or
   ↪ refresh tables, logs and privileges |
| Replication client   | Server Admin                        | To ask where the
   ↪  slave or master servers are   |
| Replication slave    | Server Admin                        | To read binary
   ↪ log events from the master      |
| Select               | Tables                              | To retrieve rows
   ↪  from table                     |
| Show databases       | Server Admin                        | To see all
   ↪ databases with SHOW DATABASES       |
| Show view            | Tables                              | To see views
   ↪ with SHOW CREATE VIEW             |
| Shutdown             | Server Admin                        | To shut down the
   ↪  server                         |
| Super                | Server Admin                        | To use KILL
   ↪ thread, SET GLOBAL, CHANGE MASTER, etc. |
| Trigger              | Tables                              | To use triggers
   ↪                                 |
| Create tablespace    | Server Admin                        | To create/alter/
   ↪ drop tablespaces                |
| Update               | Tables                              | To update
   ↪ existing rows                          |
| Usage                | Server Admin                        | No privileges -
   ↪ allow connect only              |
+--
   ↪ ----------------------+-----------------------------------------+----------------
   ↪
32 rows in set (0.00 sec)
```

14.10.2.111.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.111.4 See also

- SHOW GRANTS
- GRANT <privileges>

14.10.2.112 SHOW [FULL] PROCESSLIST

This statement lists the current sessions connected to the same TiDB server. The `Info` column contains the query text, which will be truncated unless the optional keyword `FULL` is specified.

14.10.2.112.1 Synopsis

ShowProcesslistStmt:



Figure 431: ShowProcesslistStmt

OptFull:



Figure 432: OptFull

14.10.2.112.2 Examples

```
mysql> SHOW PROCESSLIST;
+--
    ↪ ----+------+-----------------+------+---------+------+-----------+------------
    ↪
| Id   | User | Host          | db   | Command | Time | State   | Info
    ↪           |
+--
    ↪ ----+------+-----------------+------+---------+------+-----------+------------
    ↪
|    5 | root | 127.0.0.1:45970 | test | Query | 0 | autocommit | SHOW
    ↪ PROCESSLIST |
+--
    ↪ ----+------+-----------------+------+---------+------+-----------+------------
    ↪
1 rows in set (0.00 sec)
```

14.10.2.112.3 MySQL compatibility

- The `State` column in TiDB is non-descriptive. Representing state as a single value is more complex in TiDB, since queries are executed in parallel and each goroutine will have a different state at any one time.

14.10.2.112.4 See also

- KILL [TIDB]

14.10.2.113 SHOW PROFILES

The `SHOW PROFILES` statement currently only returns an empty result.

14.10.2.113.1 Synopsis

ShowStmt:

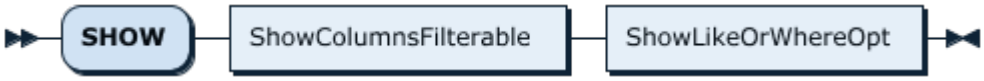


Figure 433: ShowStmt

14.10.2.113.2 Examples

```
SHOW PROFILES;
```

```
Empty set (0.00 sec)
```

14.10.2.113.3 MySQL compatibility

This statement is included only for compatibility with MySQL. Executing `SHOW PROFILES` always returns an empty result.

14.10.2.114 SHOW PUMP STATUS

The `SHOW PUMP STATUS` statement displays the status information for all Pump nodes in the cluster.

14.10.2.114.1 Examples

```
SHOW PUMP STATUS;
```

```
+--
    ↪ ------/---------------/--------/-------------------/--------------------/
    ↪
| NodeID |   Address   | State |  Max_Commit_Ts  |    Update_Time     |
+--
    ↪ ------/---------------/--------/-------------------/--------------------/
    ↪
```

```
| pump1 | 127.0.0.1:8250 | Online | 408553768673342237 | 2019-05-01 00:00:01
    ↪  |
+--
    ↪ ------/---------------/--------/--------------------/--------------------/
    ↪
| pump2 | 127.0.0.2:8250 | Online | 408553768673342335 | 2019-05-01 00:00:02
    ↪  |
+--
    ↪ ------/---------------/--------/--------------------/--------------------/
    ↪
2 rows in set (0.00 sec)
```

14.10.2.114.2 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.114.3 See also

- SHOW DRAINER STATUS
- CHANGE PUMP STATUS
- CHANGE DRAINER STATUS

14.10.2.115 SHOW SCHEMAS

This statement is an alias to `SHOW DATABASES`. It is included for compatibility with MySQL.

14.10.2.116 SHOW STATS_HEALTHY

The `SHOW STATS_HEALTHY` statement shows an estimation of how accurate statistics are believed to be. Tables with a low percentage health may generate sub-optimal query execution plans.

The health of a table can be improved by running the `ANALYZE` table command. `ANALYZE` runs automatically when the health drops below the `tidb_auto_analyze_ratio` threshold.

14.10.2.116.1 Synopsis

ShowStmt

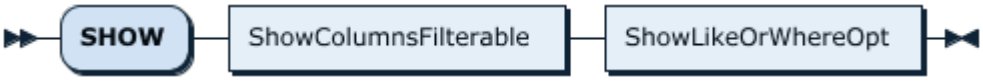


Figure 434: ShowStmt

ShowTargetFiltertable

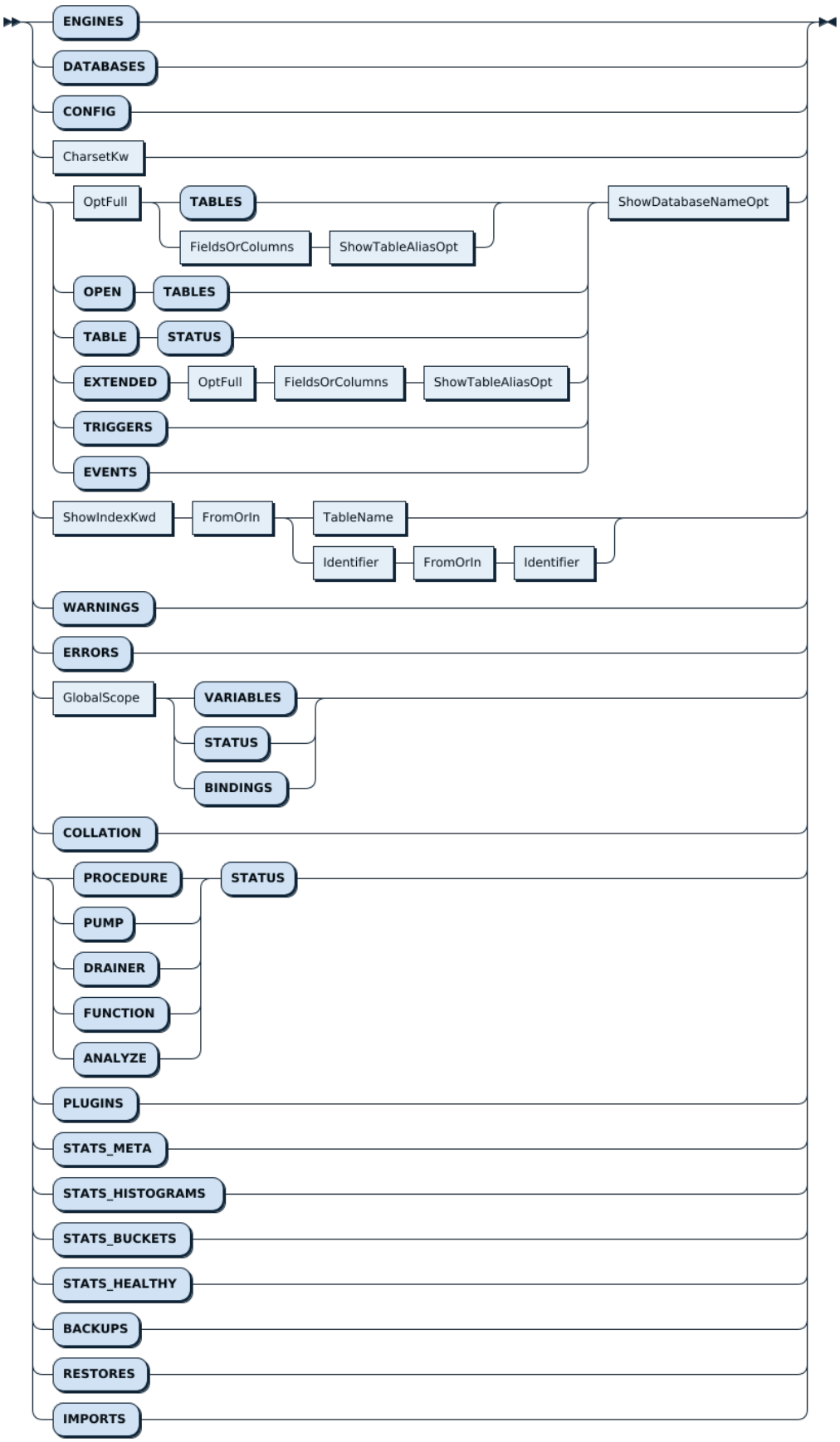


Figure 435: ShowTargetFilterable

ShowLikeOrWhereOpt

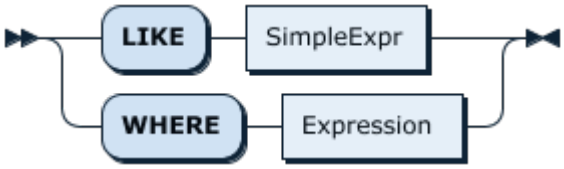


Figure 436: ShowLikeOrWhereOpt

14.10.2.116.2 Examples

Load example data and run `ANALYZE`:

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY auto_increment,
 b INT NOT NULL,
 pad VARBINARY(255),
 INDEX(b)
);

INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255) FROM dual
    ↪ ;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255) FROM t1 a
    ↪  JOIN t1 b JOIN t1 c LIMIT 100000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255) FROM t1 a
    ↪  JOIN t1 b JOIN t1 c LIMIT 100000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255) FROM t1 a
    ↪  JOIN t1 b JOIN t1 c LIMIT 100000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255) FROM t1 a
    ↪  JOIN t1 b JOIN t1 c LIMIT 100000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255) FROM t1 a
    ↪  JOIN t1 b JOIN t1 c LIMIT 100000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(255) FROM t1 a
    ↪  JOIN t1 b JOIN t1 c LIMIT 100000;
SELECT SLEEP(1);
ANALYZE TABLE t1;
SHOW STATS_HEALTHY; # should be 100% healthy
```

```
...
mysql> SHOW STATS_HEALTHY;
+---------+------------+----------------+---------+
| Db_name | Table_name | Partition_name | Healthy |
+---------+------------+----------------+---------+
| test    | t1         |                |     100 |
+---------+------------+----------------+---------+
```

```
1 row in set (0.00 sec)
```

Perform a bulk update deleting approximately 30% of the records. Check the health of the statistics:

```
DELETE FROM t1 WHERE id BETWEEN 101010 AND 201010; # delete about 30% of
    ↪ records
SHOW STATS_HEALTHY;
```

```
mysql> SHOW STATS_HEALTHY;
+---------+------------+----------------+---------+
| Db_name | Table_name | Partition_name | Healthy |
+---------+------------+----------------+---------+
| test    | t1         |                |      50 |
+---------+------------+----------------+---------+
1 row in set (0.00 sec)
```

14.10.2.116.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.116.4 See also

- ANALYZE
- Introduction to Statistics

14.10.2.117 SHOW STATS_HISTOGRAMS

This statement shows the histogram information collected by the `ANALYZE` statement.

14.10.2.117.1 Synopsis

ShowStmt

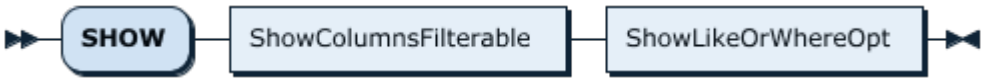


Figure 437: ShowStmt

ShowTargetFiltertable

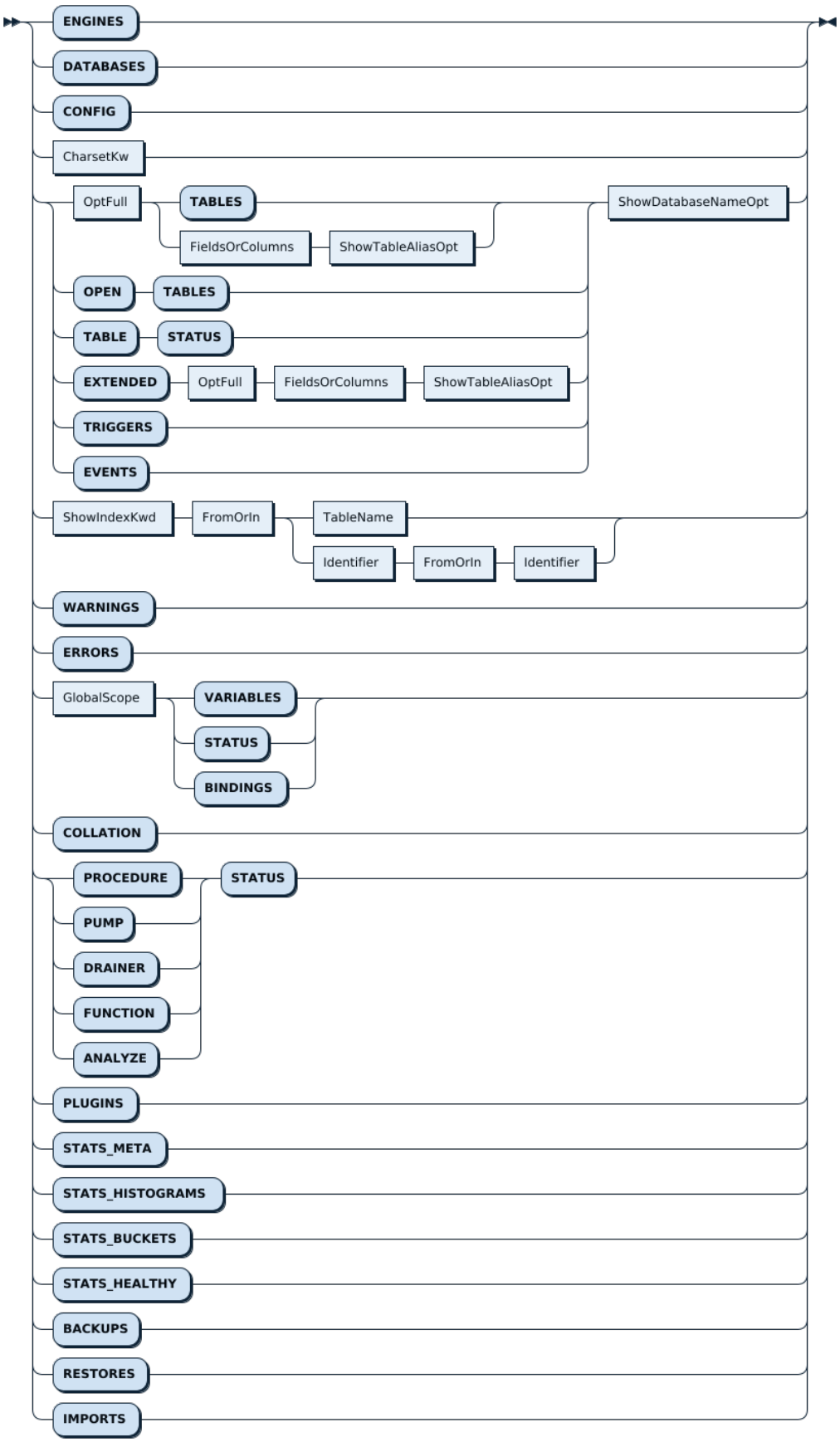


Figure 438: ShowTargetFilterable

ShowLikeOrWhereOpt

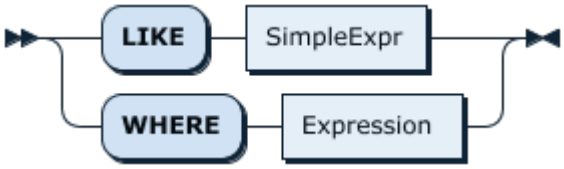


Figure 439: ShowLikeOrWhereOpt

14.10.2.117.2 Examples

```
show stats_histograms;
```

```
+--
   ↪ -------+-----------+---------------+-----------+---------+----------------
   ↪
| Db_name | Table_name | Partition_name | Column_name | Is_index |
   ↪ Update_time | Distinct_count | Null_count | Avg_col_size |
   ↪ Correlation |
+--
   ↪ -------+-----------+---------------+-----------+---------+----------------
   ↪
| test   | t          |                | a           |        0 | 2020-05-25
   ↪ 19:20:00 |         7 |          0 |           1 |          1 |
| test   | t2         |                | a           |        0 | 2020-05-25
   ↪ 19:20:01 |         6 |          0 |           8 |          0 |
| test   | t2         |                | b           |        0 | 2020-05-25
   ↪ 19:20:01 |         6 |          0 |        1.67 |          1 |
+--
   ↪ -------+-----------+---------------+-----------+---------+----------------
   ↪
3 rows in set (0.00 sec)
```

```
show stats_histograms where table_name = 't2';
```

```
+--
   ↪ -------+-----------+---------------+-----------+---------+----------------
   ↪
| Db_name | Table_name | Partition_name | Column_name | Is_index |
   ↪ Update_time | Distinct_count | Null_count | Avg_col_size |
   ↪ Correlation |
+--
   ↪ -------+-----------+---------------+-----------+---------+----------------
   ↪
```

```
| test   | t2       |              | b          |       0 | 2020-05-25
   ↪ 19:20:01 |         6 |         0 |       1.67 |         1 |
| test   | t2       |              | a          |       0 | 2020-05-25
   ↪ 19:20:01 |         6 |         0 |          8 |         0 |
+--
   ↪ -------+-----------+---------------+------------+---------+----------------
   ↪
2 rows in set (0.00 sec)
```

14.10.2.117.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.117.4 See also

- ANALYZE
- Introduction to Statistics

14.10.2.118 SHOW STATS_META

You can use `SHOW STATS_META` to view how many rows are in a table and how many rows are changed in that table. When using this statement, you can filter the needed information by the `ShowLikeOrWhere` clause.

Currently, the `SHOW STATS_META` statement outputs 6 columns:

Column name	Description
db_name	Database name
table_name	Table name
partition_name	Partition name
update_time	Last updated time
modify_count	The number of rows modified
row_count	The total row count

Note:

The `update_time` is updated when TiDB updates the `modify_count` and `row_count` fields according to DML statements. So `update_time` is not the last execution time of the `ANALYZE` statement.

14.10.2.118.1 Synopsis

ShowStmt

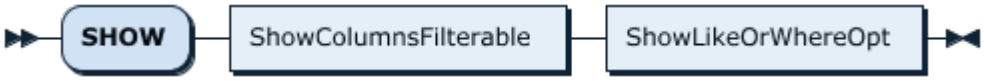


Figure 440: ShowStmt

ShowTargetFiltertable

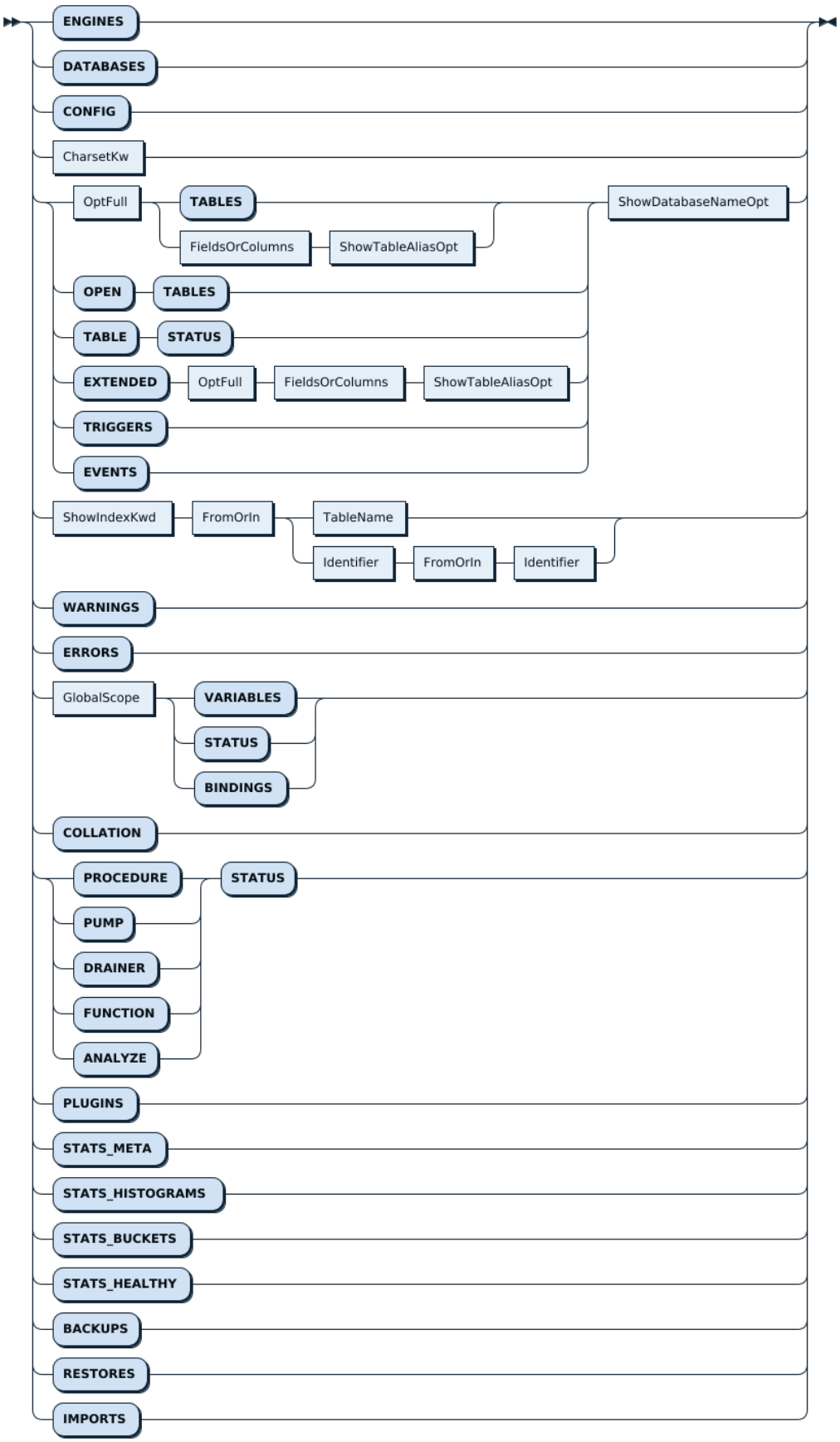


Figure 441: ShowTargetFilterable

ShowLikeOrWhereOpt

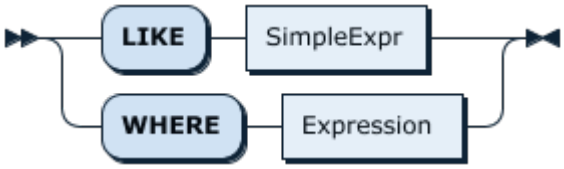


Figure 442: ShowLikeOrWhereOpt

14.10.2.118.2 Examples

```
show stats_meta;
```

```
+--
    ↪ -------+-----------+---------------+--------------------+-------------+-----
    ↪
| Db_name | Table_name | Partition_name | Update_time | Modify_count |
    ↪ Row_count |
+--
    ↪ -------+-----------+---------------+--------------------+-------------+-----
    ↪
| test   | t0         |               | 2020-05-15 16:58:00 |        0 |
    ↪         0 |
| test   | t1         |               | 2020-05-15 16:58:04 |        0 |
    ↪         0 |
| test   | t2         |               | 2020-05-15 16:58:11 |        0 |
    ↪         0 |
| test   | s          |               | 2020-05-22 19:46:43 |        0 |
    ↪         0 |
| test   | t          |               | 2020-05-25 12:04:21 |        0 |
    ↪         0 |
+--
    ↪ -------+-----------+---------------+--------------------+-------------+-----
    ↪
5 rows in set (0.00 sec)
```

```
show stats_meta where table_name = 't2';
```

```
+--
    ↪ -------+-----------+---------------+--------------------+-------------+-----
    ↪
| Db_name | Table_name | Partition_name | Update_time | Modify_count |
    ↪ Row_count |
```

```
+--
    ↪ -------+-----------+---------------+--------------------+----------------+-----
    ↪
| test   | t2        |               | 2020-05-15 16:58:11 |        0 |
    ↪        0 |
+--
    ↪ -------+-----------+---------------+--------------------+----------------+-----
    ↪
1 row in set (0.00 sec)
```

14.10.2.118.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.118.4 See also

- ANALYZE
- Introduction to Statistics

14.10.2.119 SHOW [GLOBAL|SESSION] STATUS

This statement is included for compatibility with MySQL. It has no effect on TiDB, which uses Prometheus and Grafana for centralized metrics collection instead of `SHOW STATUS`.

14.10.2.119.1 Synopsis

ShowStmt:

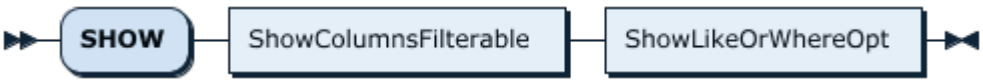


Figure 443: ShowStmt

ShowTargetFilterable:

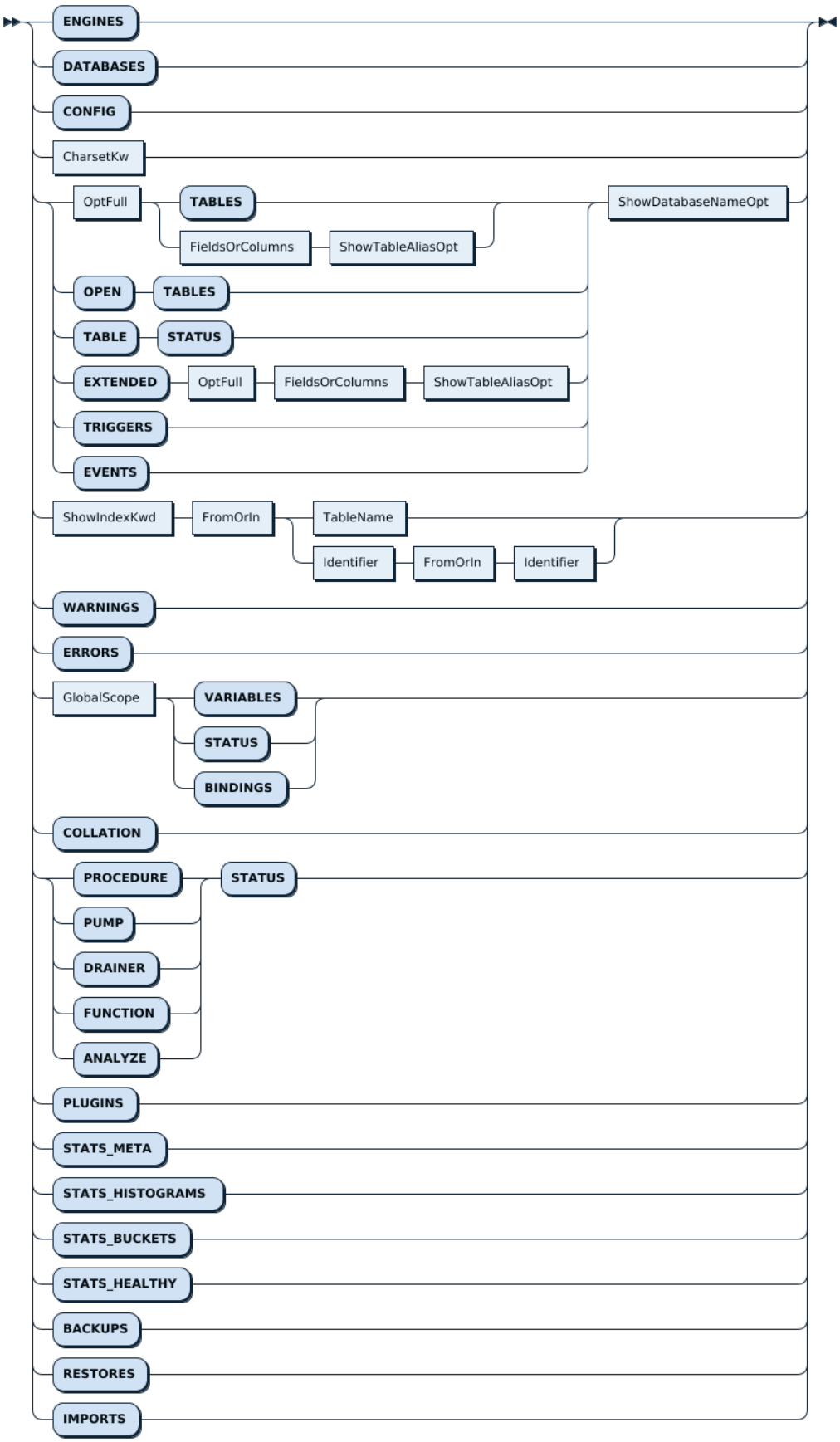


Figure 444: ShowTargetFilterable

GlobalScope:



Figure 445: GlobalScope

14.10.2.119.2 Examples

```
mysql> show status;
+--------------------+--------------------------------------+
| Variable_name      | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher_list    |                                      |
| server_id          | 93e2e07d-6bb4-4a1b-90b7-e035fae154fe |
| ddl_schema_version | 141                                  |
| Ssl_verify_mode    | 0                                    |
| Ssl_version        |                                      |
| Ssl_cipher         |                                      |
+--------------------+--------------------------------------+
6 rows in set (0.01 sec)

mysql> show global status;
+--------------------+--------------------------------------+
| Variable_name      | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher         |                                      |
| Ssl_cipher_list    |                                      |
| Ssl_verify_mode    | 0                                    |
| Ssl_version        |                                      |
| server_id          | 93e2e07d-6bb4-4a1b-90b7-e035fae154fe |
| ddl_schema_version | 141                                  |
+--------------------+--------------------------------------+
6 rows in set (0.00 sec)
```

14.10.2.119.3 MySQL compatibility

- This statement is included only for compatibility with MySQL.

14.10.2.119.4 See also

- FLUSH STATUS

14.10.2.120 SHOW TABLE NEXT_ROW_ID

`SHOW TABLE NEXT_ROW_ID` is used to show the details of some special columns of a table, including:

- `AUTO_INCREMENT` column automatically created by TiDB, namely, `_tidb_rowid` column.
- `AUTO_INCREMENT` column created by users.
- `AUTO_RANDOM` column created by users.
- `SEQUENCE` created by users.

14.10.2.120.1 Synopsis

ShowTableNextRowIDStmt:



Figure 446: ShowTableNextRowIDStmt

TableName:

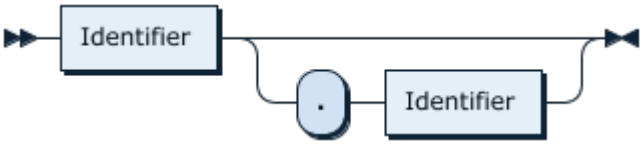


Figure 447: TableName

14.10.2.120.2 Examples

For newly created tables, `NEXT_GLOBAL_ROW_ID` is `1` because no Row ID is allocated.

```
create table t(a int);
Query OK, 0 rows affected (0.06 sec)
```

```
show table t next_row_id;
+---------+------------+-------------+--------------------+
| DB_NAME | TABLE_NAME | COLUMN_NAME | NEXT_GLOBAL_ROW_ID |
+---------+------------+-------------+--------------------+
| test    | t          | _tidb_rowid |                  1 |
+---------+------------+-------------+--------------------+
1 row in set (0.00 sec)
```

Data have been written to the table. The TiDB server that inserts the data allocates and caches 30000 IDs at once. Thus, NEXT_GLOBAL_ROW_ID is 30001 now.

```
insert into t values (), (), ();
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
show table t next_row_id;
+---------+------------+-------------+--------------------+
| DB_NAME | TABLE_NAME | COLUMN_NAME | NEXT_GLOBAL_ROW_ID |
+---------+------------+-------------+--------------------+
| test    | t          | _tidb_rowid |              30001 |
+---------+------------+-------------+--------------------+
1 row in set (0.00 sec)
```

14.10.2.120.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.120.4 See also

- CREATE TABLE
- AUTO_RANDOM
- CREATE_SEQUENCE

14.10.2.121 SHOW TABLE REGIONS

The `SHOW TABLE REGIONS` statement is used to show the Region information of a table in TiDB.

14.10.2.121.1 Syntax

```
SHOW TABLE [table_name] REGIONS [WhereClauseOptional];
SHOW TABLE [table_name] INDEX [index_name] REGIONS [WhereClauseOptional];
```

14.10.2.121.2 Synopsis

ShowTableRegionStmt:

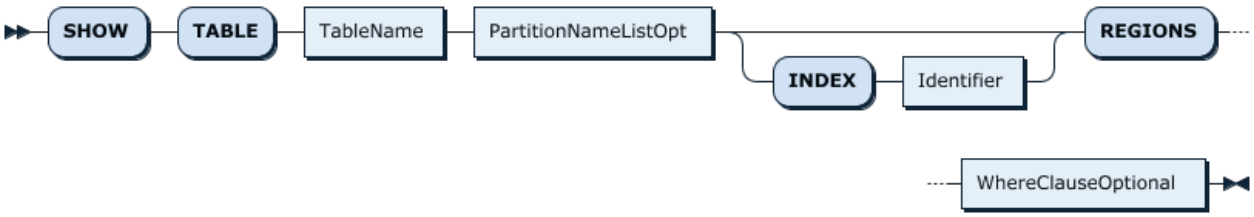


Figure 448: ShowTableRegionStmt

TableName:

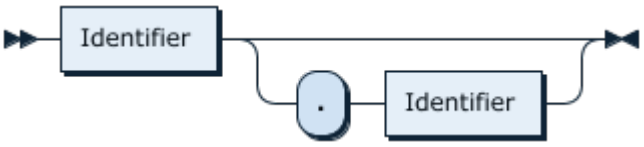


Figure 449: TableName

PartitionNameListOpt:

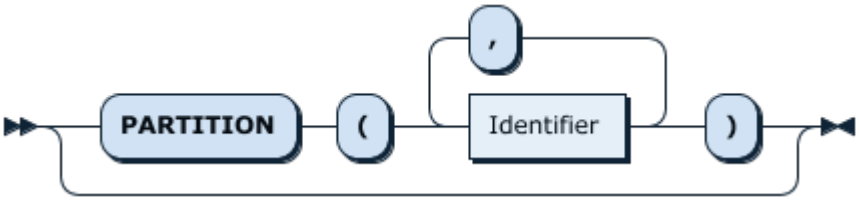


Figure 450: PartitionNameListOpt

WhereClauseOptional:

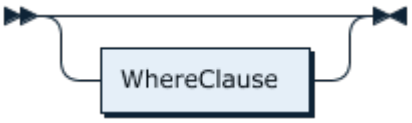


Figure 451: WhereClauseOptional

WhereClause:



Figure 452: WhereClause

Executing `SHOW TABLE REGIONS` returns the following columns:

- `REGION_ID`: The Region ID.
- `START_KEY`: The start key of the Region.
- `END_KEY`: The end key of the Region.
- `LEADER_ID`: The Leader ID of the Region.
- `LEADER_STORE_ID`: The ID of the store (TiKV) where the Region leader is located.
- `PEERS`: The IDs of all Region replicas.
- `SCATTERING`: Whether the Region is being scheduled. `1` means true.

- `WRITTEN_BYTES`: The estimated amount of data written into the Region within one heartbeat cycle. The unit is byte.
- `READ_BYTES`: The estimated amount of data read from the Region within one heartbeat cycle. The unit is byte.
- `APPROXIMATE_SIZE(MB)`: The estimated amount of data in the Region. The unit is megabytes (MB).
- `APPROXIMATE_KEYS`: The estimated number of Keys in the Region.

> **Note:**
>
> The values of `WRITTEN_BYTES`, `READ_BYTES`, `APPROXIMATE_SIZE(MB)`, `APPROXIMATE_KEYS` are not accurate data. They are estimated data from PD based on the heartbeat information that PD receives from the Region.

14.10.2.121.3 Examples

Create an example table with enough data that fills a few Regions:

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY auto_increment,
 b INT NOT NULL,
 pad1 VARBINARY(1024),
 pad2 VARBINARY(1024),
 pad3 VARBINARY(1024)
);
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM dual;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
```

```
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1000), RANDOM_BYTES(1024),
    ↪ RANDOM_BYTES(1024), RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c
    ↪ LIMIT 10000;
SELECT SLEEP(5);
SHOW TABLE t1 REGIONS;
```

The output should show that the table is split into Regions. The `REGION_ID`, `START_KEY` and `END_KEY` may not match exactly:

```
...
mysql> SHOW TABLE t1 REGIONS;
+--
    ↪ ---------+-------------+-------------+----------+----------------+-------+--
    ↪
| REGION_ID | START_KEY | END_KEY    | LEADER_ID | LEADER_STORE_ID | PEERS |
    ↪  SCATTERING | WRITTEN_BYTES | READ_BYTES | APPROXIMATE_SIZE(MB) |
    ↪ APPROXIMATE_KEYS |
+--
    ↪ ---------+-------------+-------------+----------+----------------+-------+--
    ↪
|       94 | t_75_      | t_75_r_31717 |     95 |             1 | 95    |
    ↪         0 |           0 |         0 |                112 |
    ↪ 207465 |
|       96 | t_75_r_31717 | t_75_r_63434 |   97 |             1 | 97    |
    ↪         0 |           0 |         0 |                 97 |
    ↪ 0 |
|        2 | t_75_r_63434 |              |        3 |             1 | 3     |
    ↪         0 |    269323514 | 66346110 |                245 |
    ↪ 162020 |
+--
```

```
    ↪ ---------+-------------+-------------+----------+----------------+-------+--
    ↪
3 rows in set (0.00 sec)
```

In the output above, a `START_KEY` of `t_75_r_31717` and `END_KEY` of `t_75_r_63434` shows that data with a PRIMARY KEY between `31717` and `63434` is stored in this Region. The prefix `t_75_` indicates that this is the Region for a table (`t`) which has an internal table ID of `75`. An empty key value for `START_KEY` or `END_KEY` indicates negative infinity or positive infinity respectively.

TiDB automatically rebalances Regions as needed. For manual rebalancing, use the `SPLIT TABLE REGION` statement:

```
mysql> SPLIT TABLE t1 BETWEEN (31717) AND (63434) REGIONS 2;
+--------------------+----------------------+
| TOTAL_SPLIT_REGION | SCATTER_FINISH_RATIO |
+--------------------+----------------------+
|                  1 |                    1 |
+--------------------+----------------------+
1 row in set (42.34 sec)

mysql> SHOW TABLE t1 REGIONS;
+--
    ↪ ---------+-------------+-------------+----------+----------------+-------+--
    ↪
| REGION_ID | START_KEY | END_KEY    | LEADER_ID | LEADER_STORE_ID | PEERS |
    ↪  SCATTERING | WRITTEN_BYTES | READ_BYTES | APPROXIMATE_SIZE(MB) |
    ↪ APPROXIMATE_KEYS |
+--
    ↪ ---------+-------------+-------------+----------+----------------+-------+--
    ↪
|       94 | t_75_      | t_75_r_31717 |     95 |             1 | 95    |
    ↪         0 |           0 |         0 |                112 |
    ↪ 207465 |
|       98 | t_75_r_31717 | t_75_r_47575 |   99 |             1 | 99    |
    ↪         0 |         1325 |         0 |                 53 |
    ↪ 12052 |
|       96 | t_75_r_47575 | t_75_r_63434 |   97 |             1 | 97    |
    ↪         0 |         1526 |         0 |                 48 |
    ↪ 0 |
|        2 | t_75_r_63434 |           |        3 |             1 | 3     |
    ↪         0 |           0 |   55752049 |                 60 |
    ↪ 0 |
+--
    ↪ ---------+-------------+-------------+----------+----------------+-------+--
    ↪
```

```
4 rows in set (0.00 sec)
```

The above output shows that Region 96 was split, with a new Region 98 being created. The remaining Regions in the table were unaffected by the split operation. This is confirmed by the output statistics:

- `TOTAL_SPLIT_REGION` indicates the number of newly split Regions. In this example, the number is 1.
- `SCATTER_FINISH_RATIO` indicates the rate at which the newly split Regions are successfully scattered. `1.0` means that all Regions are scattered.

For a more detailed example:

```
mysql> show table t regions;
+--
   ↪ ---------+-------------+-------------+----------+----------------+----------
   ↪
| REGION_ID | START_KEY | END_KEY    | LEADER_ID | LEADER_STORE_ID | PEERS
   ↪   | SCATTERING | WRITTEN_BYTES | READ_BYTES | APPROXIMATE_SIZE(MB) |
   ↪ APPROXIMATE_KEYS |
+--
   ↪ ---------+-------------+-------------+----------+----------------+----------
   ↪
| 102      | t_43_r      | t_43_r_20000 | 118    | 7              | 105, 118,
   ↪ 119 | 0       | 0           | 0         | 1                | 0
   ↪               |
| 106      | t_43_r_20000 | t_43_r_40000 | 120  | 7              | 107, 108,
   ↪ 120 | 0       | 23          | 0         | 1                | 0
   ↪               |
| 110      | t_43_r_40000 | t_43_r_60000 | 112  | 9              | 112, 113,
   ↪ 121 | 0       | 0           | 0         | 1                | 0
   ↪               |
| 114      | t_43_r_60000 | t_43_r_80000 | 122  | 7              | 115, 122,
   ↪ 123 | 0       | 35          | 0         | 1                | 0
   ↪               |
| 3        | t_43_r_80000 |           | 93       | 8              | 5, 73, 93
   ↪   | 0         | 0           | 0         | 1                | 0
   ↪               |
| 98       | t_43_       | t_43_r      | 99       | 1              | 99, 100,
   ↪ 101 | 0        | 0           | 0         | 1                | 0
   ↪               |
+--
   ↪ ---------+-------------+-------------+----------+----------------+----------
   ↪
6 rows in set
```

In the above example:

- Table t corresponds to six Regions. In these Regions, 102, 106, 110, 114, and 3 store the row data and 98 stores the index data.
- For START_KEY and END_KEY of Region 102, t_43 indicates the table prefix and ID. _r is the prefix of the record data in table t. _i is the prefix of the index data.
- In Region 102, START_KEY and END_KEY mean that record data in the range of [-inf, 20000) is stored. In similar way, the ranges of data storage in Regions (106, 110, 114, 3) can also be calculated.
- Region 98 stores the index data. The start key of table t's index data is t_43_i, which is in the range of Region 98.

To check the Region that corresponds to table t in store 1, use the WHERE clause:

```
test> show table t regions where leader_store_id =1;
+--
   ↪ ---------+-----------+---------+-----------+----------------+--------------+---
   ↪
| REGION_ID | START_KEY | END_KEY | LEADER_ID | LEADER_STORE_ID | PEERS |
   ↪ SCATTERING | WRITTEN_BYTES | READ_BYTES | APPROXIMATE_SIZE(MB) |
   ↪ APPROXIMATE_KEYS |
+--
   ↪ ---------+-----------+---------+-----------+----------------+--------------+---
   ↪
| 98        | t_43_     | t_43_r | 99        | 1              | 99, 100, 101 | 0
   ↪        | 0            | 0         | 1                 | 0              |
+--
   ↪ ---------+-----------+---------+-----------+----------------+--------------+---
   ↪
```

Use SPLIT TABLE REGION to split the index data into Regions. In the following example, the index data name of table t is split into two Regions in the range of [a,z].

```
test> split table t index name between ("a") and ("z") regions 2;
+--------------------+----------------------+
| TOTAL_SPLIT_REGION | SCATTER_FINISH_RATIO |
+--------------------+----------------------+
| 2                  | 1.0                  |
+--------------------+----------------------+
1 row in set
```

Now table t corresponds to seven Regions. Five of them (102, 106, 110, 114, 3) store the record data of table t and another two (135, 98) store the index data name.

```
test> show table t regions;
```

```
+--
    ↪ ---------+----------------------------+----------------------------+---------
    ↪
| REGION_ID | START_KEY                | END_KEY                    | LEADER_ID |
    ↪  LEADER_STORE_ID | PEERS    | SCATTERING | WRITTEN_BYTES | READ_BYTES
    ↪ | APPROXIMATE_SIZE(MB) | APPROXIMATE_KEYS |
+--
    ↪ ---------+----------------------------+----------------------------+---------
    ↪
| 102      | t_43_r                   | t_43_r_20000               | 118      |
    ↪ 7              | 105, 118, 119 | 0       | 0           | 0         | 1
    ↪                  | 0              |
| 106      | t_43_r_20000             | t_43_r_40000               | 120      |
    ↪ 7              | 108, 120, 126 | 0       | 0           | 0         | 1
    ↪                  | 0              |
| 110      | t_43_r_40000             | t_43_r_60000               | 112      |
    ↪ 9              | 112, 113, 121 | 0       | 0           | 0         | 1
    ↪                  | 0              |
| 114      | t_43_r_60000             | t_43_r_80000               | 122      |
    ↪ 7              | 115, 122, 123 | 0       | 35          | 0         | 1
    ↪                  | 0              |
| 3        | t_43_r_80000             |                            | 93       |
    ↪ 8              | 73, 93, 128 | 0         | 0           | 0         | 1
    ↪                  | 0              |
| 135      | t_43_i_1_                | t_43_i_1_016d80000000000000 | 139  |
    ↪ 2              | 138, 139, 140 | 0       | 35          | 0         | 1
    ↪                  | 0              |
| 98       | t_43_i_1_016d80000000000000 | t_43_r                  | 99       |
    ↪ 1              | 99, 100, 101 | 0        | 0           | 0         | 1
    ↪                  | 0              |
+--
    ↪ ---------+----------------------------+----------------------------+---------
    ↪
7 rows in set
```

14.10.2.121.4 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.121.5 See also

- SPLIT REGION
- CREATE TABLE

14.10.2.122 SHOW TABLE STATUS

This statement shows various statistics about tables in TiDB. If the statistics appear out of date, it is recommended to run `ANALYZE TABLE`.

14.10.2.122.1 Synopsis

ShowTableStatusStmt:

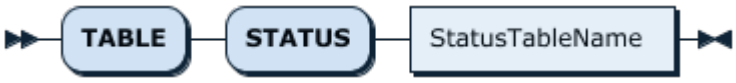


Figure 453: ShowTableStatusStmt

FromOrIn:

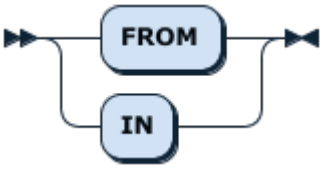


Figure 454: FromOrIn

StatusTableName:



Figure 455: StatusTableName

14.10.2.122.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.02 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SHOW TABLE STATUS LIKE 't1'\G
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Compact
```

```
          Rows: 0
 Avg_row_length: 0
    Data_length: 0
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 30001
    Create_time: 2019-04-19 08:32:06
    Update_time: NULL
     Check_time: NULL
      Collation: utf8mb4_bin
       Checksum:
 Create_options:
        Comment:
1 row in set (0.00 sec)

mysql> analyze table t1;
Query OK, 0 rows affected (0.12 sec)

mysql> SHOW TABLE STATUS LIKE 't1'\G
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 5
 Avg_row_length: 16
    Data_length: 80
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: 30001
    Create_time: 2019-04-19 08:32:06
    Update_time: NULL
     Check_time: NULL
      Collation: utf8mb4_bin
       Checksum:
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

14.10.2.122.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility

differences should be reported via an issue on GitHub.

14.10.2.122.4 See also

- SHOW TABLES
- CREATE TABLE
- DROP TABLE
- SHOW CREATE TABLE

14.10.2.123 SHOW [FULL] TABLES

This statement shows a list of tables and views in the currently selected database. The optional keyword `FULL` indicates if a table is of type `BASE TABLE` or `VIEW`.

To show tables in a different database, use `SHOW TABLES IN DatabaseName`.

14.10.2.123.1 Synopsis

ShowTablesStmt:



Figure 456: ShowTablesStmt

OptFull:

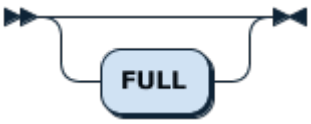


Figure 457: OptFull

ShowDatabaseNameOpt:

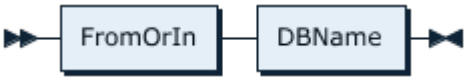


Figure 458: ShowDatabaseNameOpt

ShowLikeOrWhereOpt:

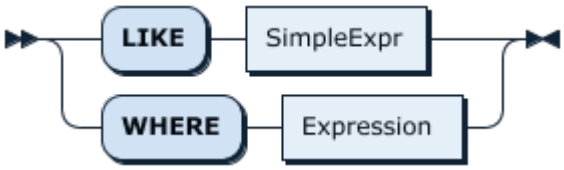


Figure 459: ShowLikeOrWhereOpt

14.10.2.123.2 Examples

```
mysql> CREATE TABLE t1 (a int);
Query OK, 0 rows affected (0.12 sec)

mysql> CREATE VIEW v1 AS SELECT 1;
Query OK, 0 rows affected (0.10 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
| v1             |
+----------------+
2 rows in set (0.00 sec)

mysql> SHOW FULL TABLES;
+----------------+------------+
| Tables_in_test | Table_type |
+----------------+------------+
| t1             | BASE TABLE |
| v1             | VIEW       |
+----------------+------------+
2 rows in set (0.00 sec)

mysql> SHOW TABLES IN mysql;
+------------------------+
| Tables_in_mysql        |
+------------------------+
| GLOBAL_VARIABLES       |
| bind_info              |
| columns_priv           |
| db                     |
| default_roles          |
| expr_pushdown_blacklist |
| gc_delete_range        |
| gc_delete_range_done   |
```

```
| global_priv           |
| help_topic            |
| opt_rule_blacklist    |
| role_edges            |
| stats_buckets         |
| stats_feedback        |
| stats_histograms      |
| stats_meta            |
| stats_top_n           |
| tables_priv           |
| tidb                  |
| user                  |
+-----------------------+
20 rows in set (0.00 sec)
```

14.10.2.123.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.123.4 See also

- CREATE TABLE
- DROP TABLE
- SHOW CREATE TABLE

14.10.2.124 SHOW [GLOBAL|SESSION] VARIABLES

This statement shows a list of variables for the scope of either `GLOBAL` or `SESSION`. If no scope is specified, the default scope of `SESSION` will apply.

14.10.2.124.1 Synopsis

ShowStmt:

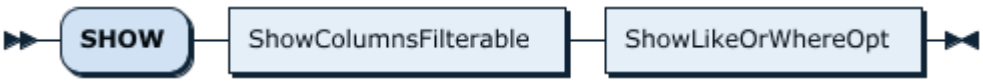


Figure 460: ShowStmt

ShowTargetFilterable:

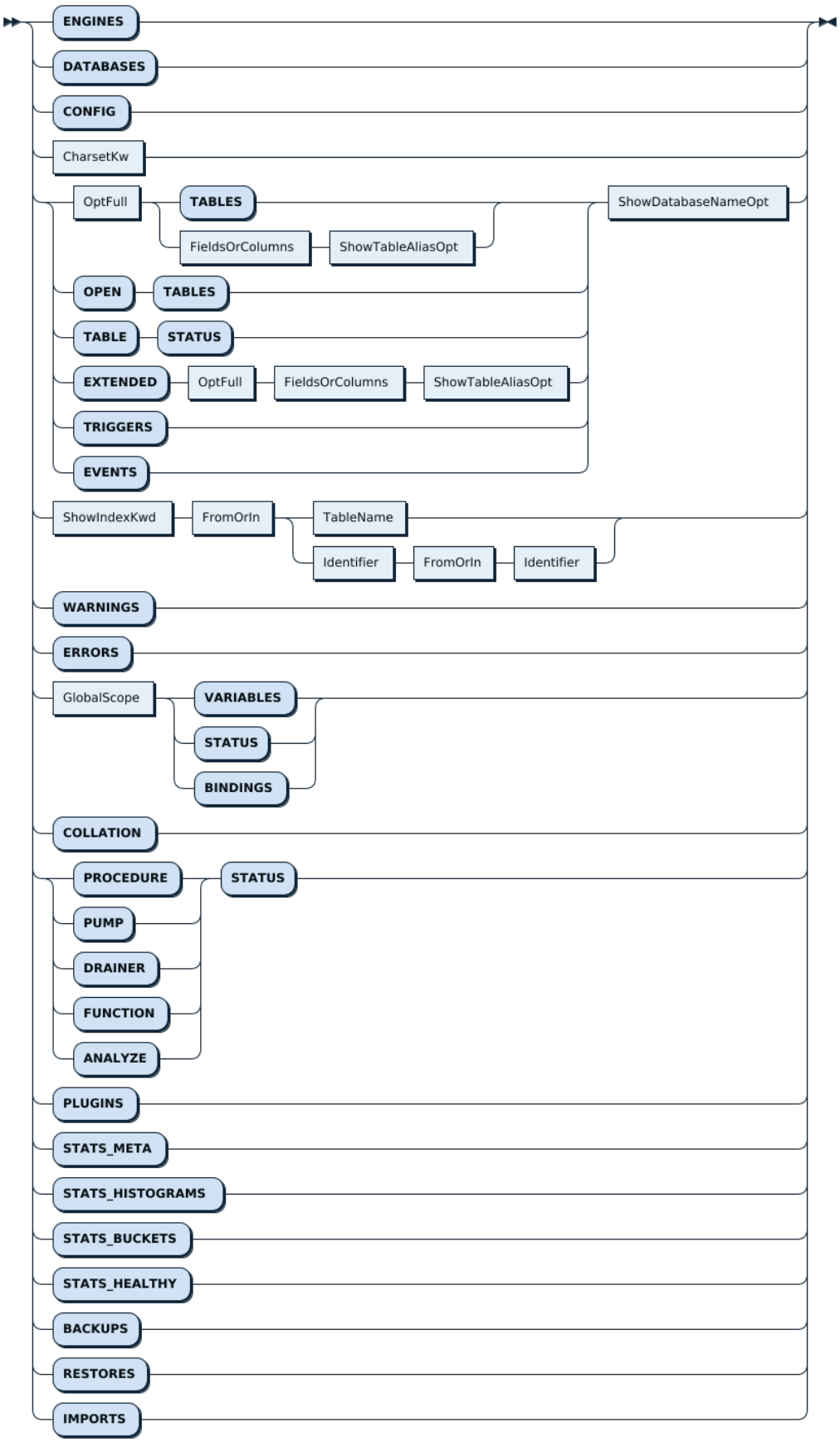


Figure 461: ShowTargetFilterable

GlobalScope:



Figure 462: GlobalScope

14.10.2.124.2 Examples

List all TiDB specific variables. For detailed description, refer to System Variables.

```
mysql> SHOW GLOBAL VARIABLES LIKE 'tidb%';
+-----------------------------------+---------------------+
| Variable_name                     | Value               |
+-----------------------------------+---------------------+
| tidb_allow_batch_cop              | 0                   |
| tidb_allow_remove_auto_inc        | 0                   |
| tidb_auto_analyze_end_time        | 23:59 +0000         |
| tidb_auto_analyze_ratio           | 0.5                 |
| tidb_auto_analyze_start_time      | 00:00 +0000         |
| tidb_backoff_lock_fast            | 100                 |
| tidb_backoff_weight               | 2                   |
| tidb_batch_commit                 | 0                   |
| tidb_batch_delete                 | 0                   |
| tidb_build_stats_concurrency      | 4                   |
| tidb_capture_plan_baselines       | off                 |
| tidb_check_mb4_value_in_utf8      | 1                   |
| tidb_checksum_table_concurrency | 4                   |
| tidb_config                       |                     |
| tidb_constraint_check_in_place  | 0                   |
| tidb_current_ts                   | 0                   |
| tidb_ddl_error_count_limit        | 512                 |
| tidb_ddl_reorg_batch_size         | 256                 |
| tidb_ddl_reorg_priority           | PRIORITY_LOW        |
| tidb_ddl_reorg_worker_cnt         | 4                   |
| tidb_disable_txn_auto_retry       | 1                   |
| tidb_distsql_scan_concurrency     | 15                  |
| tidb_dml_batch_size               | 20000               |
| tidb_enable_cascades_planner      | 0                   |
| tidb_enable_chunk_rpc             | 1                   |
| tidb_enable_collect_execution_info | 1                   |
| tidb_enable_fast_analyze          | 0                   |
```

```
| tidb_enable_index_merge         | 0                  |
| tidb_enable_noop_functions      | 0                  |
| tidb_enable_radix_join          | 0                  |
| tidb_enable_slow_log            | 1                  |
| tidb_enable_stmt_summary        | 1                  |
| tidb_enable_table_partition     | on                 |
| tidb_enable_vectorized_expression | 1                |
| tidb_enable_window_function     | 1                  |
| tidb_evolve_plan_baselines      | off                |
| tidb_evolve_plan_task_end_time  | 23:59 +0000        |
| tidb_evolve_plan_task_max_time  | 600                |
| tidb_evolve_plan_task_start_time | 00:00 +0000       |
| tidb_expensive_query_time_threshold | 60             |
| tidb_force_priority             | NO_PRIORITY        |
| tidb_general_log                | 0                  |
| tidb_hash_join_concurrency      | 5                  |
| tidb_hashagg_final_concurrency  | 4                  |
| tidb_hashagg_partial_concurrency | 4                 |
| tidb_index_join_batch_size      | 25000              |
| tidb_index_lookup_concurrency   | 4                  |
| tidb_index_lookup_join_concurrency | 4               |
| tidb_index_lookup_size          | 20000              |
| tidb_index_serial_scan_concurrency | 1               |
| tidb_init_chunk_size            | 32                 |
| tidb_isolation_read_engines     | tikv, tiflash, tidb |
| tidb_low_resolution_tso         | 0                  |
| tidb_max_chunk_size             | 1024               |
| tidb_max_delta_schema_count     | 1024               |
| tidb_mem_quota_hashjoin         | 34359738368        |
| tidb_mem_quota_indexlookupjoin  | 34359738368        |
| tidb_mem_quota_indexlookupreader | 34359738368       |
| tidb_mem_quota_mergejoin        | 34359738368        |
| tidb_mem_quota_nestedloopapply  | 34359738368        |
| tidb_mem_quota_query            | 1073741824         |
| tidb_mem_quota_sort             | 34359738368        |
| tidb_mem_quota_topn             | 34359738368        |
| tidb_metric_query_range_duration | 60                |
| tidb_metric_query_step          | 60                 |
| tidb_opt_agg_push_down          | 0                  |
| tidb_opt_concurrency_factor     | 3                  |
| tidb_opt_copcpu_factor          | 3                  |
| tidb_opt_correlation_exp_factor | 1                  |
| tidb_opt_correlation_threshold  | 0.9                |
| tidb_opt_cpu_factor             | 3                  |
| tidb_opt_desc_factor            | 3                  |
```

```
| tidb_opt_disk_factor            | 1.5               |
| tidb_opt_distinct_agg_push_down | 0                 |
| tidb_opt_insubq_to_join_and_agg | 1                 |
| tidb_opt_join_reorder_threshold | 0                 |
| tidb_opt_memory_factor          | 0.001             |
| tidb_opt_network_factor         | 1                 |
| tidb_opt_scan_factor            | 1.5               |
| tidb_opt_seek_factor            | 20                |
| tidb_opt_write_row_id           | 0                 |
| tidb_optimizer_selectivity_level | 0                |
| tidb_pprof_sql_cpu              | 0                 |
| tidb_projection_concurrency     | 4                 |
| tidb_query_log_max_len          | 4096              |
| tidb_record_plan_in_slow_log    | 1                 |
| tidb_replica_read               | leader            |
| tidb_retry_limit                | 10                |
| tidb_row_format_version         | 2                 |
| tidb_scatter_region             | 0                 |
| tidb_skip_isolation_level_check | 0                 |
| tidb_skip_utf8_check            | 0                 |
| tidb_slow_log_threshold         | 300               |
| tidb_slow_query_file            | tidb-slow.log     |
| tidb_snapshot                   |                   |
| tidb_stmt_summary_history_size  | 24                |
| tidb_stmt_summary_internal_query | 0                |
| tidb_stmt_summary_max_sql_length | 4096             |
| tidb_stmt_summary_max_stmt_count | 3000             |
| tidb_stmt_summary_refresh_interval | 1800           |
| tidb_store_limit                | 0                 |
| tidb_txn_mode                   |                   |
| tidb_use_plan_baselines         | on                |
| tidb_wait_split_region_finish   | 1                 |
| tidb_wait_split_region_timeout  | 300               |
| tidb_window_concurrency         | 4                 |
+---------------------------------+-------------------+
108 rows in set (0.01 sec)

mysql> SHOW GLOBAL VARIABLES LIKE 'time_zone%';
+---------------+--------+
| Variable_name | Value  |
+---------------+--------+
| time_zone     | SYSTEM |
+---------------+--------+
1 row in set (0.00 sec)
```

14.10.2.124.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.124.4 See also

- SET [GLOBAL|SESSION]

14.10.2.125 SHOW WARNINGS

This statement shows a list of warnings that occurred for previously executed statements in the current client connection. As in MySQL, the `sql_mode` impacts which statements will cause errors vs. warnings considerably.

14.10.2.125.1 Synopsis

ShowWarningsStmt:



Figure 463: ShowWarningsStmt

14.10.2.125.2 Examples

```
mysql> CREATE TABLE t1 (a INT UNSIGNED);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 VALUES (0);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT 1/a FROM t1;
+------+
| 1/a  |
+------+
| NULL |
+------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+---------------+
| Level   | Code | Message       |
+---------+------+---------------+
| Warning | 1365 | Division by 0 |
```

```
+---------+------+---------------+
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES (-1);
ERROR 1264 (22003): Out of range value for column 'a' at row 1
mysql> SELECT * FROM t1;
+------+
| a    |
+------+
|    0 |
+------+
1 row in set (0.00 sec)

mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (-1);
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+--------------------------+
| Level   | Code | Message                  |
+---------+------+--------------------------+
| Warning | 1690 | constant -1 overflows int |
+---------+------+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t1;
+------+
| a    |
+------+
|    0 |
|    0 |
+------+
2 rows in set (0.00 sec)
```

14.10.2.125.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.125.4 See also

- SHOW ERRORS

14.10.2.126 SHUTDOWN

The `SHUTDOWN` statement is used to perform a shutdown operation in TiDB. Execution of the `SHUTDOWN` statement requires the user to have `SHUTDOWN privilege`.

14.10.2.126.1 Synopsis

Statement:



Figure 464: Statement

14.10.2.126.2 Examples

```
SHUTDOWN;
```

```
Query OK, 0 rows affected (0.00 sec)
```

14.10.2.126.3 MySQL compatibility

> **Note:**
>
> Because TiDB is a distributed database, the shutdown operation in TiDB stops the client-connected TiDB instance, not the entire TiDB cluster.

The `SHUTDOWN` statement is partly compatible with MySQL. If you encounter any compatibility issues, you can report the issues on GitHub.

14.10.2.127 Split Region

For each new table created in TiDB, one Region is segmented by default to store the data of this table. This default behavior is controlled by `split-table` in the TiDB configuration file. When the data in this Region exceeds the default Region size limit, the Region starts to split into two.

In the above case, because there is only one Region at the beginning, all write requests occur on the TiKV where the Region is located. If there are a large number of writes for the newly created table, hotspots are caused.

To solve the hotspot problem in the above scenario, TiDB introduces the pre-split function, which can pre-split multiple Regions for a certain table according to the specified parameters and scatter them to each TiKV node.

14.10.2.127.1 Synopsis

SplitRegionStmt:



Figure 465: SplitRegionStmt

SplitSyntaxOption:



Figure 466: SplitSyntaxOption

TableName:

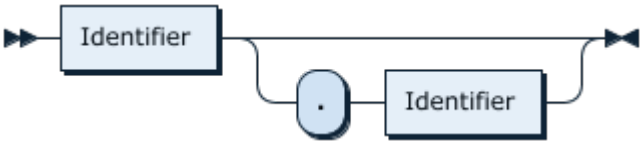


Figure 467: TableName

PartitionNameListOpt:

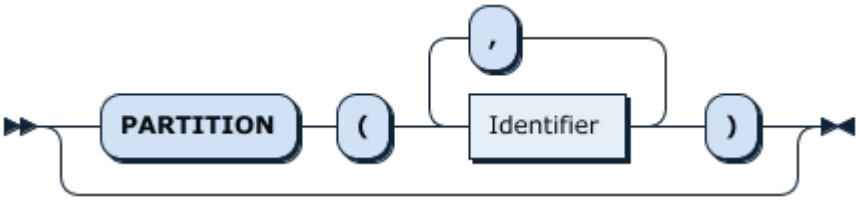


Figure 468: PartitionNameListOpt

SplitOption:

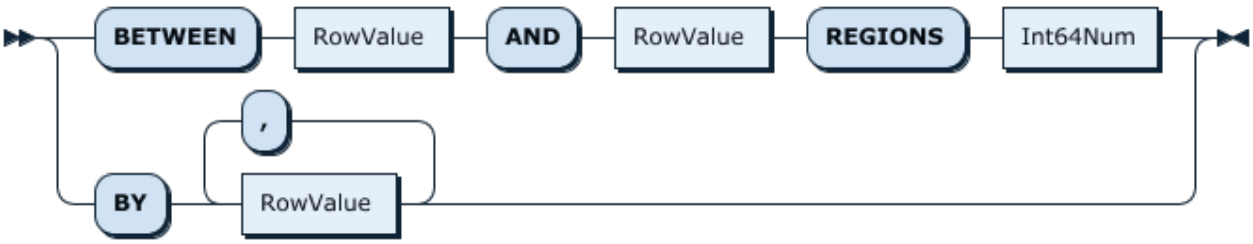


Figure 469: SplitOption

RowValue:



Figure 470: RowValue

Int64Num:

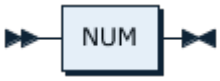


Figure 471: Int64Num

14.10.2.127.2 Usage of Split Region

There are two types of Split Region syntax:

- The syntax of even split:

```
SPLIT TABLE table_name [INDEX index_name] BETWEEN (lower_value) AND (
    ↪ upper_value) REGIONS region_num
```

`BETWEEN lower_value AND upper_value REGIONS region_num` defines the upper boundary, the lower boundary, and the Region amount. Then the current region will be evenly spilt into the number of regions (as specified in `region_num`) between the upper and lower boundaries.

- The syntax of uneven split:

```
SPLIT TABLE table_name [INDEX index_name] BY (value_list) [, (
    ↪ value_list)] ...
```

`BY value_list⋯` specifies a series of points manually, based on which the current Region is spilt. It is suitable for scenarios with unevenly distributed data.

The following example shows the result of the `SPLIT` statement:

```
+--------------------+----------------------+
| TOTAL_SPLIT_REGION | SCATTER_FINISH_RATIO |
+--------------------+----------------------+
| 4                  | 1.0                  |
+--------------------+----------------------+
```

- `TOTAL_SPLIT_REGION`: the number of newly split Regions.
- `SCATTER_FINISH_RATIO`: the completion rate of scattering for newly split Regions. `1.0` means that all Regions are scattered. `0.5` means that only half of the Regions are scattered and the rest are being scattered.

Note:

The following two session variables might affect the behavior of the `SPLIT` statement:

- `tidb_wait_split_region_finish`: It might take a while to scatter the Regions. This duration depends on PD scheduling and TiKV loads. This variable is used to control when executing the `SPLIT REGION` statement whether to return the results to the client until all Regions are scattered. If its value is set to `1` (by default), TiDB returns the results only after the scattering is completed. If its value is set to `0`, TiDB returns the results regardless of the scattering status.
- `tidb_wait_split_region_timeout`: This variable is to set the execution timeout of the `SPLIT REGION` statement, in seconds. The default value is 300s. If the `split` operation is not completed within the duration, TiDB returns a timeout error.

Split Table Region

The key of row data in each table is encoded by `table_id` and `row_id`. The format is as follows:

```
t[table_id]_r[row_id]
```

For example, when `table_id` is 22 and `row_id` is 11:

```
t22_r11
```

Row data in the same table have the same `table_id`, but each row has its unique `row_id` that can be used for Region split.

Even Split

Because `row_id` is an integer, the value of the key to be split can be calculated according to the specified `lower_value`, `upper_value`, and `region_num`. TiDB first calculates the step value (`step = (upper_value - lower_value)/region_num`). Then split will be done evenly per each "step" between `lower_value` and `upper_value` to generate the number of Regions as specified by `region_num`.

For example, if you want 16 evenly split Regions split from key range`minInt64~maxInt64` for table t, you can use this statement:

```
SPLIT TABLE t BETWEEN (-9223372036854775808) AND (9223372036854775807)
    ↪ REGIONS 16;
```

This statement splits table t into 16 Regions between minInt64 and maxInt64. If the given primary key range is smaller than the specified one, for example, 0~1000000000, you can use 0 and 1000000000 take place of minInt64 and maxInt64 respectively to split Regions.

```
SPLIT TABLE t BETWEEN (0) AND (1000000000) REGIONS 16;
```

Uneven split

If the known data is unevenly distributed, and you want a Region to be split respectively in key ranges -inf ~ 10000, 10000 ~ 90000, and 90000 ~ +inf, you can achieve this by setting fixed points, as shown below:

```
SPLIT TABLE t BY (10000), (90000);
```

Split index Region

The key of the index data in the table is encoded by `table_id`, `index_id`, and the value of the index column. The format is as follows:

```
t[table_id]_i[index_id][index_value]
```

For example, when `table_id` is 22, `index_id` is 5, and `index_value` is abc:

```
t22_i5abc
```

The `table_id` and `index_id` of the same index data in one table is the same. To split index Regions, you need to split Regions based on `index_value`.

Even Spilt

The way to split index evenly works the same as splitting data evenly. However, calculating the value of step is more complicated, because `index_value` might not be an integer.

The values of `upper` and `lower` are encoded into a byte array firstly. After removing the longest common prefix of `lower` and `upper` byte array, the first 8 bytes of `lower` and `upper` are converted into the uint64 format. Then `step = (upper - lower)/num` is calculated. After that, the calculated step is encoded into a byte array, which is appended to the longest common prefix of the `lower` and `upper` byte array for index split. Here is an example:

If the column of the `idx` index is of the integer type, you can use the following SQL statement to split index data:

```
SPLIT TABLE t INDEX idx BETWEEN (-9223372036854775808) AND
    ↪ (9223372036854775807) REGIONS 16;
```

This statement splits the Region of index idx in table t into 16 Regions from `minInt64` to `maxInt64`.

If the column of index idx1 is of varchar type, and you want to split index data by prefix letters.

```
SPLIT TABLE t INDEX idx1 BETWEEN ("a") AND ("z") REGIONS 25;
```

This statement splits index idx1 into 25 Regions from a~z. The range of Region 1 is `[minIndexValue, b)`; the range of Region 2 is `[b, c)`; … the range of Region 25 is `[y, minIndexValue]`. For the `idx` index, data with the `a` prefix is written into Region 1, and data with the `b` prefix is written into Region 2.

In the split method above, both data with the `y` and `z` prefixes are written into Region 25, because the upper bound is not `z`, but `{` (the character next to `z` in ASCII). Therefore, a more accurate split method is as follows:

```
SPLIT TABLE t INDEX idx1 BETWEEN ("a") AND ("{") REGIONS 26;
```

This statement splits index idx1 of the table `t` into 26 Regions from a~`{`. The range of Region 1 is `[minIndexValue, b)`; the range of Region 2 is `[b, c)`; … the range of Region 25 is `[y, z)`, and the range of Region 26 is `[z, maxIndexValue)`.

If the column of index `idx2` is of time type like timestamp/datetime, and you want to split index Region by year:

```
SPLIT TABLE t INDEX idx2 BETWEEN ("2010-01-01 00:00:00") AND ("2020-01-01
    ↪ 00:00:00") REGIONS 10;
```

This statement splits the Region of index `idx2` in table `t` into 10 Regions from `2010-01-01 00:00:00` to `2020-01-01 00:00:00`. The range of Region 1 is `[minIndexValue , 2011-01-01 00:00:00)`; the range of Region 2 is `[2011-01-01 00:00:00, 2012-01-01 00:00:00)`.

If you want to split the index Region by day, see the following example:

```
SPLIT TABLE t INDEX idx2 BETWEEN ("2020-06-01 00:00:00") AND ("2020-07-01
    ↪ 00:00:00") REGIONS 30;
```

This statement splits the data of June 2020 of index `idex2` in table `t` into 30 Regions, each Region representing 1 day.

Region split methods for other types of index columns are similar.

For data Region split of joint indexes, the only difference is that you can specify multiple columns values.

For example, index `idx3 (a, b)` contains 2 columns, with column `a` of timestamp type and column `b` int. If you just want to do a time range split according to column `a`, you can use the SQL statement for splitting time index of a single column. In this case, do not specify the value of column `b` in `lower_value` and `upper_velue`.

```
SPLIT TABLE t INDEX idx3 BETWEEN ("2010-01-01 00:00:00") AND ("2020-01-01
    ↪ 00:00:00") REGIONS 10;
```

Within the same range of time, if you want to do one more split according to column b column. Just specify the value for column b when splitting.

```
SPLIT TABLE t INDEX idx3 BETWEEN ("2010-01-01 00:00:00", "a") AND ("
    ↪ 2010-01-01 00:00:00", "z") REGIONS 10;
```

This statement splits 10 Regions in the range of a~z according to the value of column b, with the same time prefix as column a. If the value specified for column a is different, the value of column b might not be used in this case.

If the primary key of the table is a non-clustered index, you need to use backticks ` to escape the `PRIMARY` keyword when splitting Regions. For example:

```
SPLIT TABLE t INDEX `PRIMARY` BETWEEN (-9223372036854775808) AND
    ↪ (9223372036854775807) REGIONS 16;
```

Uneven Split

Index data can also be split by specified index values.

For example, there is `idx4 (a,b)`, with column `a` of the varchar type and column `b` of the timestamp type.

```
SPLIT TABLE t1 INDEX idx4 BY ("a", "2000-01-01 00:00:01"), ("b", "
    ↪ 2019-04-17 14:26:19"), ("c", "");
```

This statement specifies 3 values to split 4 Regions. The range of each Region is as follows:

```
region1 [ minIndexValue           , ("a", "2000-01-01 00:00:01"))
region2 [("a", "2000-01-01 00:00:01") , ("b", "2019-04-17 14:26:19"))
region3 [("b", "2019-04-17 14:26:19") , ("c", "")           )
region4 [("c", "")                , maxIndexValue           )
```

Split Regions for partitioned tables

Splitting Regions for partitioned tables is the same as splitting Regions for ordinary tables. The only difference is that the same split operation is performed for every partition.

- The syntax of even split:

```
SPLIT [PARTITION] TABLE t [PARTITION] [(partition_name_list...)] [INDEX
    ↪  index_name] BETWEEN (lower_value) AND (upper_value) REGIONS
    ↪ region_num
```

- The syntax of uneven split:

```
SPLIT [PARTITION] TABLE table_name [PARTITION (partition_name_list...)]
    ↪  [INDEX index_name] BY (value_list) [, (value_list)] ...
```

Examples of Split Regions for partitioned tables

1. Create a partitioned table `t`. Suppose that you want to create a Hash table divided into two partitions. The example statement is as follows:

```
create table t (a int,b int,index idx(a)) partition by hash(a)
    ↪ partitions 2;
```

After creating the table `t`, a Region is split for each partition. Use the `SHOW TABLE ↪ REGIONS` syntax to view the Regions of this table:

```
show table t regions;
```

```
+--
    ↪ ---------+-----------+---------+-----------+-----------------+------------
    ↪
| REGION_ID | START_KEY | END_KEY | LEADER_ID | LEADER_STORE_ID | PEERS
    ↪   | SCATTERING | WRITTEN_BYTES | READ_BYTES | APPROXIMATE_SIZE(MB
    ↪ ) | APPROXIMATE_KEYS |
+--
    ↪ ---------+-----------+---------+-----------+-----------------+------------
    ↪
| 1978     | t_1400_  | t_1401_ | 1979    | 4             | 1979, 1980,
    ↪ 1981 | 0       | 0           | 0         | 1                   | 0
    ↪              |
| 6        | t_1401_  |         | 17       | 4             | 17, 18, 21
    ↪     | 0         | 223         | 0         | 1                   | 0
    ↪              |
+--
    ↪ ---------+-----------+---------+-----------+-----------------+------------
    ↪
```

2. Use the `SPLIT` syntax to split a Region for each partition. Suppose that you want to split the data in the `[0,10000]` range of each partition into four Regions. The example statement is as follows:

```
split partition table t between (0) and (10000) regions 4;
```

In the above statement, `0` and `10000` respectively represent the `row_id` of the upper and lower boundaries corresponding to the hotspot data you want to scatter.

> **Note:**
>
> This example only applies to scenarios where hotspot data is evenly distributed. If the hotspot data is unevenly distributed in a specified data range, refer to the syntax of uneven split in Split Regions for partitioned tables.

3. Use the `SHOW TABLE REGIONS` syntax to view the Regions of this table again. You can see that this table now has ten Regions, each partition with five Regions, four of which are the row data and one is the index data.

```
show table t regions;
```

```
+--
   ↪ ---------+---------------+---------------+-----------+-----------------+---
   ↪
| REGION_ID | START_KEY  | END_KEY      | LEADER_ID | LEADER_STORE_ID |
   ↪ PEERS      | SCATTERING | WRITTEN_BYTES | READ_BYTES |
   ↪ APPROXIMATE_SIZE(MB) | APPROXIMATE_KEYS |
+--
   ↪ ---------+---------------+---------------+-----------+-----------------+---
   ↪
| 1998     | t_1400_r     | t_1400_r_2500 | 2001   | 5               |
   ↪ 2000, 2001, 2015 | 0       | 132           | 0          | 1
   ↪                  | 0               |
| 2006     | t_1400_r_2500 | t_1400_r_5000 | 2016 | 1               |
   ↪ 2007, 2016, 2017 | 0       | 35            | 0          | 1
   ↪                  | 0               |
| 2010     | t_1400_r_5000 | t_1400_r_7500 | 2012 | 2               |
   ↪ 2011, 2012, 2013 | 0       | 35            | 0          | 1
   ↪                  | 0               |
| 1978     | t_1400_r_7500 | t_1401_    | 1979     | 4               |
   ↪ 1979, 1980, 1981 | 0       | 621           | 0          | 1
   ↪                  | 0               |
| 1982     | t_1400_      | t_1400_r     | 2014     | 3               |
   ↪ 1983, 1984, 2014 | 0       | 35            | 0          | 1
   ↪                  | 0               |
| 1990     | t_1401_r     | t_1401_r_2500 | 1992   | 2               |
   ↪ 1991, 1992, 2020 | 0       | 120           | 0          | 1
   ↪                  | 0               |
```

```
| 1994     | t_1401_r_2500 | t_1401_r_5000 | 1997 | 5            |
   ↪ 1996, 1997, 2021 | 0       | 129          | 0         | 1
   ↪                  | 0              |
| 2002     | t_1401_r_5000 | t_1401_r_7500 | 2003 | 4            |
   ↪ 2003, 2023, 2022 | 0       | 141          | 0         | 1
   ↪                  | 0              |
| 6        | t_1401_r_7500 |               | 17       | 4            | 17,
   ↪ 18, 21     | 0         | 601           | 0         | 1
   ↪                  | 0              |
| 1986     | t_1401_      | t_1401_r     | 1989     | 5            |
   ↪ 1989, 2018, 2019 | 0       | 123          | 0         | 1
   ↪                  | 0              |
+--
   ↪ ---------+---------------+--------------+----------+-------------------+---
   ↪
```

4. You can also split Regions for the index of each partition. For example, you can split the `[1000,10000]` range of the `idx` index into two Regions. The example statement is as follows:

```
split partition table t index idx between (1000) and (10000) regions
   ↪ 2;
```

Examples of Split Region for a single partition

You can specify the partition to be split.

1. Create a partitioned table. Suppose that you want to create a Range partitioned table split into three partitions. The example statement is as follows:

```
create table t ( a int, b int, index idx(b)) partition by range( a ) (
   partition p1 values less than (10000),
   partition p2 values less than (20000),
   partition p3 values less than (MAXVALUE) );
```

2. Suppose that you want to split the data in the `[0,10000]` range of the `p1` partition into two Regions. The example statement is as follows:

```
split partition table t partition (p1) between (0) and (10000) regions
   ↪  2;
```

3. Suppose that you want to split the data in the `[10000,20000]` range of the `p2` partition into two Regions. The example statement is as follows:

```
split partition table t partition (p2) between (10000) and (20000)
   ↪ regions 2;
```

4. You can use the `SHOW TABLE REGIONS` syntax to view the Regions of this table:

```
show table t regions;
```

```
+--
    ↪ ---------+----------------+----------------+----------+-----------------+-
    ↪
| REGION_ID | START_KEY   | END_KEY      | LEADER_ID | LEADER_STORE_ID |
    ↪  PEERS      | SCATTERING | WRITTEN_BYTES | READ_BYTES |
    ↪ APPROXIMATE_SIZE(MB) | APPROXIMATE_KEYS |
+--
    ↪ ---------+----------------+----------------+----------+-----------------+-
    ↪
| 2040     | t_1406_       | t_1406_r_5000 | 2045    | 3             |
    ↪ 2043, 2045, 2044 | 0       | 0            | 0         | 1
    ↪                  | 0              |
| 2032     | t_1406_r_5000 | t_1407_      | 2033     | 4             |
    ↪ 2033, 2034, 2035 | 0       | 0            | 0         | 1
    ↪                  | 0              |
| 2046     | t_1407_       | t_1407_r_15000 | 2048   | 2             |
    ↪ 2047, 2048, 2050 | 0       | 35           | 0         | 1
    ↪                  | 0              |
| 2036     | t_1407_r_15000 | t_1408_     | 2037     | 4             |
    ↪ 2037, 2038, 2039 | 0       | 0            | 0         | 1
    ↪                  | 0              |
| 6        | t_1408_       |               | 17       | 4             |
    ↪ 17, 18, 21     | 0         | 214          | 0         | 1
    ↪                  | 0              |
+--
    ↪ ---------+----------------+----------------+----------+-----------------+-
    ↪
```

5. Suppose that you want to split the `[0,20000]` range of the `idx` index of the `p1` and `p2` partitions into two Regions. The example statement is as follows:

```
split partition table t partition (p1,p2) index idx between (0) and
    ↪ (20000) regions 2;
```

14.10.2.127.3 pre__split__regions

To have evenly split Regions when a table is created, it is recommended you use `SHARD_ROW_ID_BITS` together with `PRE_SPLIT_REGIONS`. When a table is created successfully, `PRE_SPLIT_REGIONS` pre-spilts tables into the number of Regions as specified by `2^(PRE_SPLIT_REGIONS)`.

Note:

The value of `PRE_SPLIT_REGIONS` must be less than or equal to that of `SHARD_ROW_ID_BITS`.

The `tidb_scatter_region` global variable affects the behavior of `PRE_SPLIT_REGIONS` ↪ . This variable controls whether to wait for Regions to be pre-split and scattered before returning results after the table creation. If there are intensive writes after creating the table, you need to set the value of this variable to `1`, then TiDB will not return the results to the client until all the Regions are split and scattered. Otherwise, TiDB writes the data before the scattering is completed, which will have a significant impact on write performance.

Examples of pre_split_regions

```
create table t (a int, b int,index idx1(a)) shard_row_id_bits = 4
    ↪ pre_split_regions=2;
```

After building the table, this statement splits `4 + 1` Regions for table t. `4 (2^2)` Regions are used to save table row data, and 1 Region is for saving the index data of `idx1`.

The ranges of the 4 table Regions are as follows:

```
region1:  [ -inf     ,  1<<61 )
region2:  [ 1<<61    ,  2<<61 )
region3:  [ 2<<61    ,  3<<61 )
region4:  [ 3<<61    ,  +inf  )
```

Note:

The Region split by the Split Region statement is controlled by the Region merge scheduler in PD. To avoid PD re-merging the newly split Region soon after, you need to dynamically modify configuration items related to the Region merge feature.

14.10.2.127.4 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.127.5 See also

- SHOW TABLE REGIONS

- Session variables: tidb_scatter_region, tidb_wait_split_region_finish and tidb_wait_split_region_timeout.

14.10.2.128 START TRANSACTION

This statement starts a new transaction inside of TiDB. It is similar to the statement BEGIN.

In the absence of a START TRANSACTION statement, every statement will by default autocommit in its own transaction. This behavior ensures MySQL compatibility.

14.10.2.128.1 Synopsis

BeginTransactionStmt:

```
BeginTransactionStmt ::=
    'BEGIN' ( 'PESSIMISTIC' | 'OPTIMISTIC' )?
|   'START' 'TRANSACTION' ( 'READ' ( 'WRITE' | 'ONLY' ( ( 'WITH' 'TIMESTAMP'
    ↪  'BOUND' TimestampBound )? | AsOfClause ) ) | 'WITH' 'CONSISTENT' '
    ↪ SNAPSHOT' | 'WITH' 'CAUSAL' 'CONSISTENCY' 'ONLY' )?

AsOfClause ::=
    ( 'AS' 'OF' 'TIMESTAMP' Expression)
```

14.10.2.128.2 Examples

```
mysql> CREATE TABLE t1 (a int NOT NULL PRIMARY KEY);
Query OK, 0 rows affected (0.12 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

14.10.2.128.3 MySQL compatibility

- START TRANSACTION immediately starts a transaction inside TiDB. This differs from MySQL, where START TRANSACTION lazily creates a transaction. But START ↪ TRANSACTION in TiDB is equivalent to MySQL's START TRANSACTION WITH ↪ CONSISTENT SNAPSHOT.
- The statement START TRANSACTION READ ONLY is parsed for compatibility with MySQL, but still allows write operations.

14.10.2.128.4 See also

- COMMIT
- ROLLBACK
- BEGIN
- START TRANSACTION WITH CAUSAL CONSISTENCY ONLY

14.10.2.129 TABLE

The `TABLE` statement can be used instead of `SELECT * FROM` when no aggregation or complex filtering is needed.

14.10.2.129.1 Synopsis

```
TableStmt ::=
    "TABLE" Table ( "ORDER BY" Column )? ( "LIMIT" NUM )?
```

14.10.2.129.2 Examples

Create table `t1`:

```
CREATE TABLE t1(id INT PRIMARY KEY);
```

Insert some data into `t1`:

```
INSERT INTO t1 VALUES (1),(2),(3);
```

View the data in table `t1`:

```
TABLE t1;
```

```
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
3 rows in set (0.01 sec)
```

Query `t1` and sort the result by the `id` field in descending order:

```
TABLE t1 ORDER BY id DESC;
```

```
+----+
| id |
+----+
|  3 |
|  2 |
|  1 |
+----+
3 rows in set (0.01 sec)
```

Query the first record in `t1`:

```
TABLE t1 LIMIT 1;
```

```
+----+
| id |
+----+
|  1 |
+----+
1 row in set (0.01 sec)
```

14.10.2.129.3 MySQL compatibility

The `TABLE` statement was introduced in MySQL 8.0.19.

14.10.2.129.4 See also

- SELECT
- `TABLE` statements in MySQL

14.10.2.130 TRACE

The `TRACE` statement provides detailed information about query execution. It is intended to be viewed through a Graphical interface exposed by the TiDB server's status port.

14.10.2.130.1 Synopsis

TraceStmt:

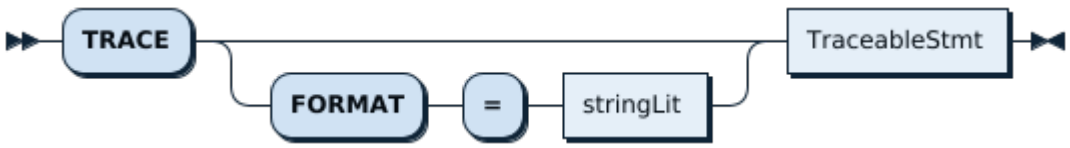


Figure 472: TraceStmt

TraceableStmt:

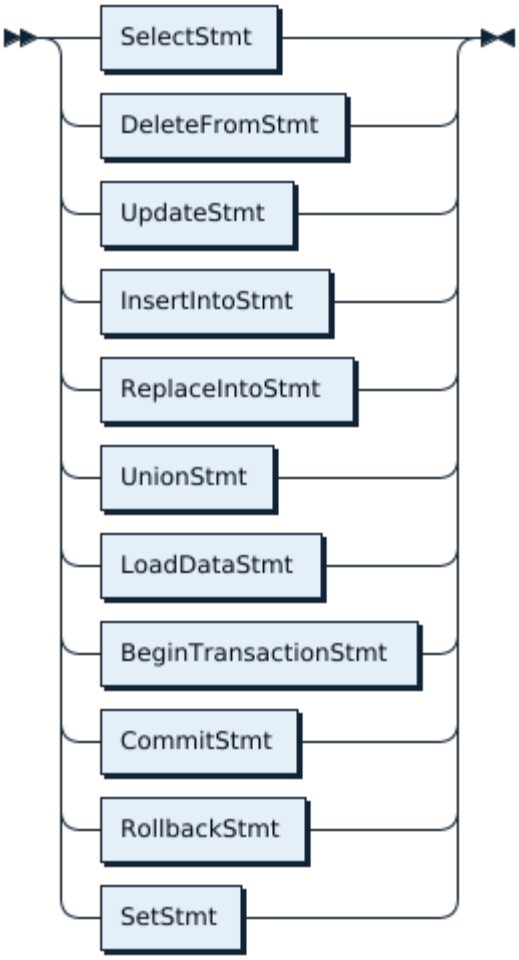


Figure 473: TraceableStmt

14.10.2.130.2 Examples

```
trace format='row' select * from mysql.user;
```

```
+-----------------------------------------+-----------------+------------+
   ↪
| operation                               | startTS         | duration   |
+-----------------------------------------+-----------------+------------+
   ↪
| trace                                   | 17:03:31.938237 | 886.086µs  |
|    ─session.Execute                     | 17:03:31.938247 | 507.812µs  |
|      ─session.ParseSQL                  | 17:03:31.938254 | 22.504µs   |
|      ─executor.Compile                  | 17:03:31.938321 | 278.931µs  |
|        ─session.getTxnFuture            | 17:03:31.938337 | 1.515µs    |
|      ─session.runStmt                   | 17:03:31.938613 | 109.578µs  |
```

```
|       ─TableReaderExecutor.Open       | 17:03:31.938645 | 50.657µs |
|         ─distsql.Select               | 17:03:31.938666 | 21.066µs |
|           ─RPCClient.SendRequest      | 17:03:31.938799 | 158.411µs |
|       ─session.CommitTxn              | 17:03:31.938705 | 12.06µs |
|         ─session.doCommitWitRetry     | 17:03:31.938709 | 2.437µs |
|   ─* executor.TableReaderExecutor.Next | 17:03:31.938781 | 224.327µs |
|   ─* executor.TableReaderExecutor.Next | 17:03:31.939019 | 6.266µs |
+----------------------------------------+-----------------+------------+
    ↪
13 rows in set (0.00 sec)
```

```
trace format='json' select * from mysql.user;
```

The JSON formatted trace can be pasted into the trace viewer, which is accessed via the TiDB status port:

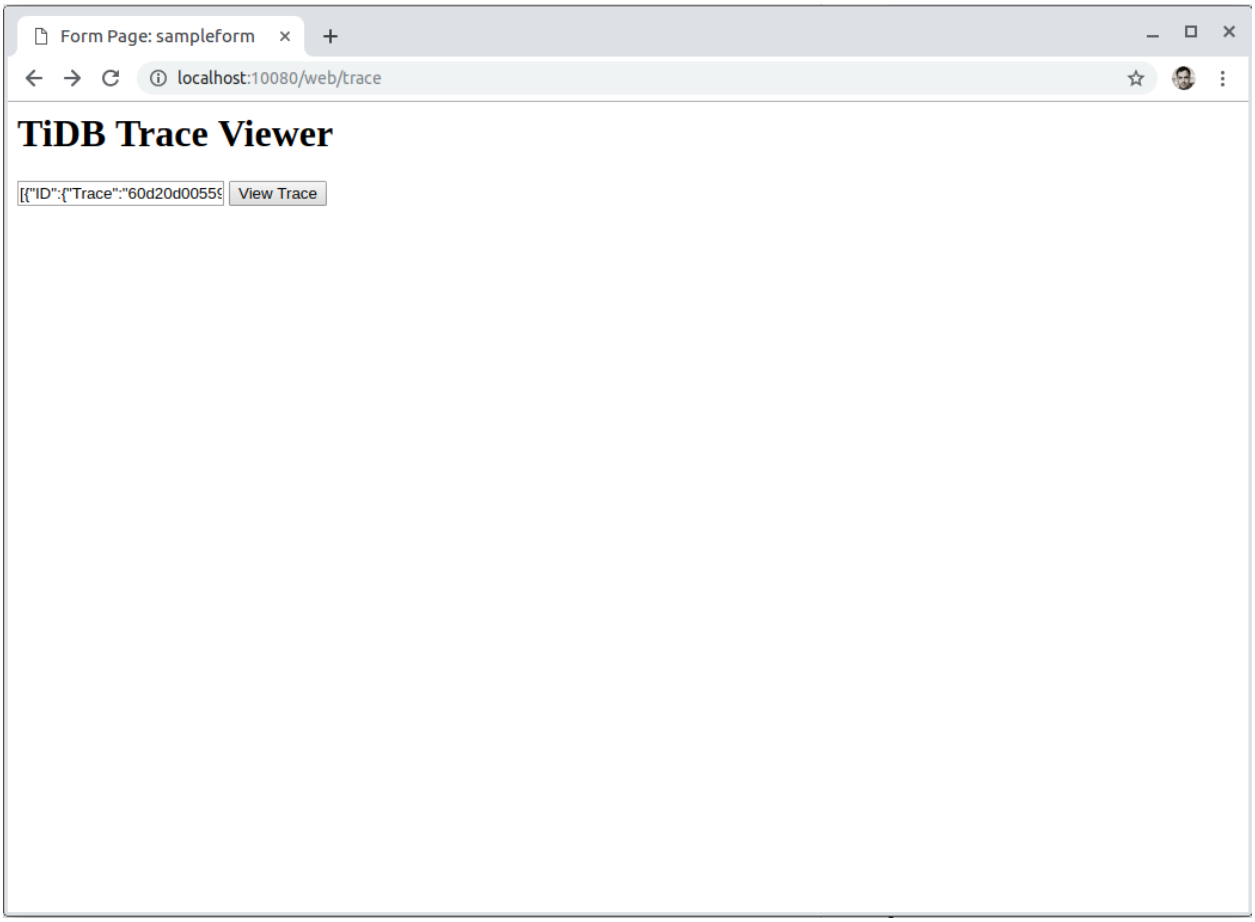


Figure 474: TiDB Trace Viewer-1

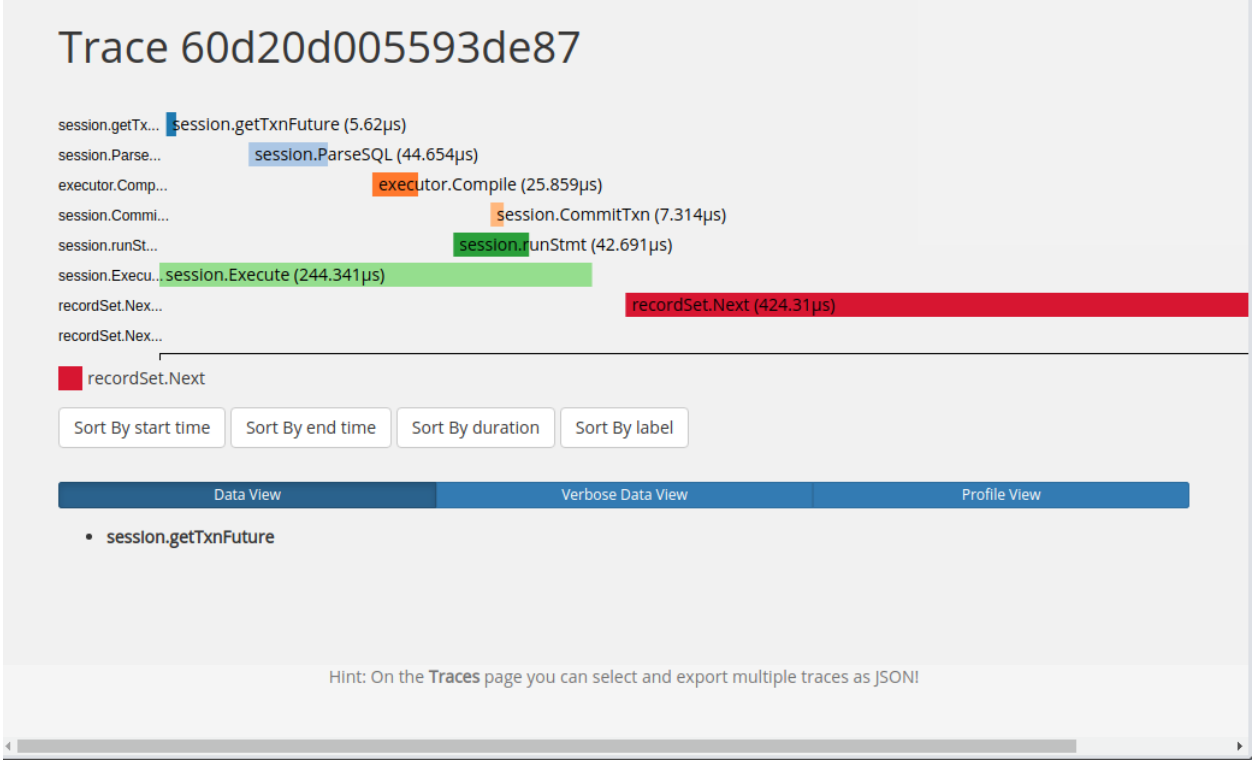


Figure 475: TiDB Trace Viewer-2

14.10.2.130.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

14.10.2.130.4 See also

- EXPLAIN ANALYZE

14.10.2.131 TRUNCATE

The `TRUNCATE` statement removes all data from the table in a non-transactional way. `TRUNCATE` can be thought of as semantically the same as `DROP TABLE` + `CREATE TABLE` with the previous definition.

Both `TRUNCATE TABLE tableName` and `TRUNCATE tableName` are valid syntax.

14.10.2.131.1 Synopsis

TruncateTableStmt:



Figure 476: TruncateTableStmt

OptTable:



Figure 477: OptTable

TableName:

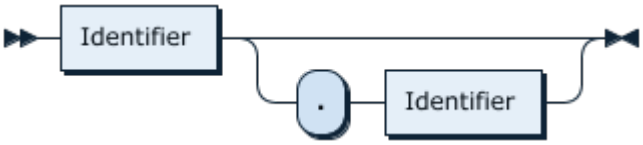


Figure 478: TableName

14.10.2.131.2 Examples

```
mysql> CREATE TABLE t1 (a INT NOT NULL PRIMARY KEY);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+---+
5 rows in set (0.00 sec)

mysql> TRUNCATE t1;
Query OK, 0 rows affected (0.11 sec)
```

```
mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> INSERT INTO t1 VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> TRUNCATE TABLE t1;
Query OK, 0 rows affected (0.11 sec)
```

14.10.2.131.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.131.4 See also

- DROP TABLE
- DELETE
- CREATE TABLE
- SHOW CREATE TABLE

14.10.2.132 UPDATE

The `UPDATE` statement is used to modify data in a specified table.

14.10.2.132.1 Synopsis

UpdateStmt:

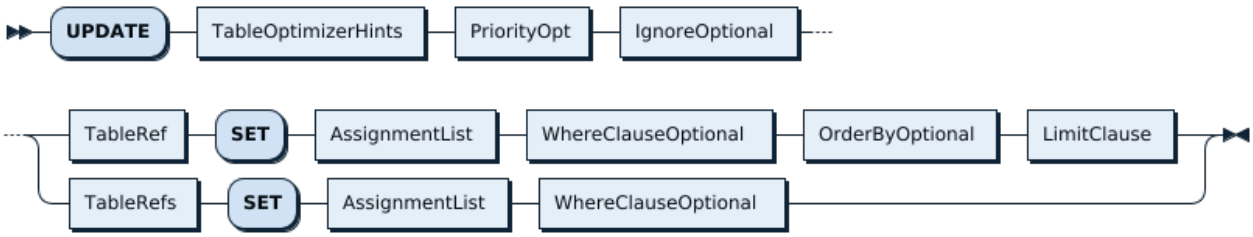


Figure 479: UpdateStmt

PriorityOpt:

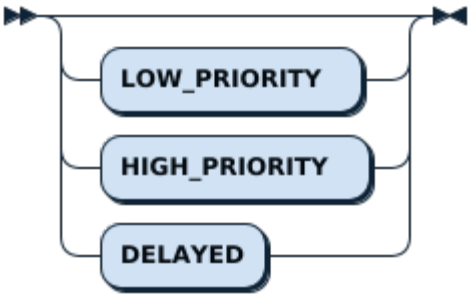


Figure 480: PriorityOpt

TableRef:

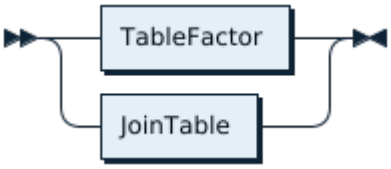


Figure 481: TableRef

TableRefs:



Figure 482: TableRefs

AssignmentList:

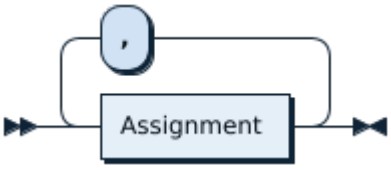


Figure 483: AssignmentList

WhereClauseOptional:

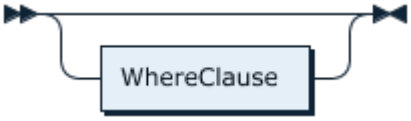


Figure 484: WhereClauseOptional

14.10.2.132.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
    ↪ NOT NULL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO t1 (c1) VALUES (1), (2), (3);
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  3 |
+----+----+
3 rows in set (0.00 sec)

mysql> UPDATE t1 SET c1=5 WHERE c1=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM t1;
+----+----+
| id | c1 |
+----+----+
|  1 |  1 |
|  2 |  2 |
|  3 |  5 |
+----+----+
3 rows in set (0.00 sec)
```

14.10.2.132.3 MySQL compatibility

TiDB always uses the original value of a column when evaluating expressions. For example:

```
CREATE TABLE t (a int, b int);
INSERT INTO t VALUES (1,2);
UPDATE t SET a = a+1,b=a;
```

In MySQL, the column `b` is updated to 2 because it is set to the value of `a`, and the value of `a` (which is 1) is updated to `a+1` (which is 2) in the same statement.

TiDB follows the more standard SQL behavior, and updates `b` to 1.

14.10.2.132.4 See also

- INSERT
- SELECT
- DELETE
- REPLACE

14.10.2.133 USE

The `USE` statement selects a current database for the user session.

14.10.2.133.1 Synopsis

UseStmt:



Figure 485: UseStmt

DBName:



Figure 486: DBName

14.10.2.133.2 Examples

```
mysql> USE mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_mysql         |
+-------------------------+
| GLOBAL_VARIABLES        |
| bind_info               |
| columns_priv            |
| db                      |
| default_roles           |
| expr_pushdown_blacklist |
| gc_delete_range         |
```

```
| gc_delete_range_done |
| global_priv          |
| help_topic           |
| opt_rule_blacklist   |
| role_edges           |
| stats_buckets        |
| stats_feedback       |
| stats_histograms     |
| stats_meta           |
| stats_top_n          |
| tables_priv          |
| tidb                 |
| user                 |
+----------------------+
20 rows in set (0.01 sec)

mysql> CREATE DATABASE newtest;
Query OK, 0 rows affected (0.10 sec)

mysql> USE newtest;
Database changed
mysql> SHOW TABLES;
Empty set (0.00 sec)

mysql> CREATE TABLE t1 (a int);
Query OK, 0 rows affected (0.10 sec)

mysql> SHOW TABLES;
+------------------+
| Tables_in_newtest |
+------------------+
| t1               |
+------------------+
1 row in set (0.00 sec)
```

14.10.2.133.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be reported via an issue on GitHub.

14.10.2.133.4 See also

- CREATE DATABASE
- SHOW TABLES

14.10.2.134 WITH

A Common Table Expression (CTE) is a temporary result set that can be referred multiple times within a SQL statement to improve the statement's readability and execution efficiency. You can apply the `WITH` statement to use Common Table Expressions.

14.10.2.134.1 Synopsis

WithClause:

```
WithClause ::=
       "WITH" WithList
|      "WITH" recursive WithList
```

WithList:

```
WithList ::=
       WithList ',' CommonTableExpr
|      CommonTableExpr
```

CommonTableExpr:

```
CommonTableExpr ::=
       Identifier IdentListWithParenOpt "AS" SubSelect
```

IdentListWithParenOpt:

```
IdentListWithParenOpt ::=
( '(' IdentList ')' )?
```

14.10.2.134.2 Examples

Non-recursive CTE:

```
WITH CTE AS (SELECT 1, 2) SELECT * FROM cte t1, cte t2;
```

```
+---+---+---+---+
| 1 | 2 | 1 | 2 |
+---+---+---+---+
| 1 | 2 | 1 | 2 |
+---+---+---+---+
1 row in set (0.00 sec)
```

Recursive CTE:

```
WITH RECURSIVE cte(a) AS (SELECT 1 UNION SELECT a+1 FROM cte WHERE a < 5)
    ↪ SELECT * FROM cte;
```

```
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+---+
5 rows in set (0.00 sec)
```

14.10.2.134.3 MySQL compatibility

- In strict mode, when the data length recursively calculated exceeds the data length of the seed part, TiDB returns a warning while MySQL returns an error. In non-strict mode, the behavior of TiDB is consistent with that of MySQL.
- The data type for recursive CTE is determined by the seed part. The data type of the seed part is not completely consistent with MySQL in some cases (such as functions).
- In the case of multiple `UNION` / `UNION ALL` operators, MySQL does not allow `UNION` to be followed by `UNION ALL`, but TiDB does.
- If there is a problem with the definition of a CTE, TiDB will report an error, while MySQL will not if the CTE is not referred.

14.10.2.134.4 See also

- SELECT
- INSERT
- DELETE
- UPDATE
- REPLACE

14.10.3 Data Types

14.10.3.1 Data Types

TiDB supports all the data types in MySQL except the `SPATIAL` type. This includes all the numeric types, string types, date & time types, and the JSON type.

The definitions used for datatypes are specified as `T(M[, D])`. Where by:

- `T` indicates the specific data type.

- `M` indicates the maximum display width for integer types. For floating-point and fixed-point types, `M` is the total number of digits that can be stored (the precision). For string types, `M` is the maximum length. The maximum permissible value of M depends on the data type.
- `D` applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale).
- `fsp` applies to the `TIME`, `DATETIME`, and `TIMESTAMP` types and represents the fractional seconds precision. The `fsp` value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

14.10.3.2 Default Values

The `DEFAULT` value clause in a data type specification indicates a default value for a column. The default value must be a constant and cannot be a function or an expression. But for the time type, you can specify the `NOW`, `CURRENT_TIMESTAMP`, `LOCALTIME`, and `LOCALTIMESTAMP` functions as the default for `TIMESTAMP` and `DATETIME` columns.

The `BLOB`, `TEXT`, and `JSON` columns **cannot** be assigned a default value.

If a column definition includes no explicit `DEFAULT` value, TiDB determines the default value as follows:

- If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause.
- If the column cannot take `NULL` as the value, TiDB defines the column with no explicit `DEFAULT` clause.

For data entry into a `NOT NULL` column that has no explicit `DEFAULT` clause, if an `INSERT` or `REPLACE` statement includes no value for the column, TiDB handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is enabled, an error occurs for transactional tables, and the statement is rolled back. For nontransactional tables, an error occurs.
- If strict mode is not enabled, TiDB sets the column to the implicit default value for the column data type.

Implicit defaults are defined as follows:

- For numeric types, the default is 0. If declared with the `AUTO_INCREMENT` attribute, the default is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate "zero" value for the type. For `TIMESTAMP`, the default value is the current date and time.
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

14.10.3.3 Numeric Types

TiDB supports all the MySQL numeric types, including:

- Integer Types (Exact Value)
- Floating-Point Types (Approximate Value)
- Fixed-Point Types (Exact Value)

14.10.3.3.1 Integer types

TiDB supports all the MySQL integer types, including `INTEGER/INT`, `TINYINT`, `SMALLINT` ↪ , `MEDIUMINT`, and `BIGINT`. For more information, see Integer Data Type Syntax in MySQL.

The following table summarizes field descriptions:

Syntax Element	Description
M	the display width of the type. Optional.
UNSIGNED	UNSIGNED. If omitted, it is SIGNED.
ZEROFILL	If you specify ZEROFILL for a numeric column, TiDB automatically adds the UNSIGNED attribute to the column.

The following table summarizes the required storage and range for integer types supported by TiDB:

Data Type	Storage Required (bytes)	Minimum Value (signed/unsigned)	Maximum value (signed/unsigned)
`TINYINT` ↪	1	-128 / 0	127 / 255
`SMALLINT` ↪	2	-32768 / 0	32767 / 65535
`MEDIUMINT` ↪	3	-8388608 / 0	8388607 / 16777215
`INT`	4	-2147483648 / 0	2147483647 / 4294967295

Data Type	Storage Required (bytes)	Minimum Value (signed/unsigned)	Maximum value (signed/unsigned)
BIGINT	8	-9223372036854775808 / 0	9223372036854775807 / 18446744073709551615

BIT type

The BIT data type. A type of BIT(M) enables the storage of M-bit values. M can range from 1 to 64, with the default value of 1:

```
BIT[(M)]
```

BOOLEAN type

The BOOLEAN type and its alias BOOL are equivalent to TINYINT(1). If the value is 0, it is considered as False; otherwise, it is considered True. As in MySQL, True is 1 and False is 0:

```
BOOLEAN
```

TINYINT type

The TINYINT data type stores signed values of range [-128, 127] and unsigned values of range [0, 255]:

```
TINYINT[(M)] [UNSIGNED] [ZEROFILL]
```

SMALLINT type

The SMALLINT data type stores signed values of range [-32768, 32767], and unsigned values of range [0, 65535]:

```
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]
```

MEDIUMINT type

The MEDIUMINT data type stores signed values of range [-8388608, 8388607], and unsigned values of range [0, 16777215]:

```
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]
```

INTEGER type

The INTEGER type and its alias INT stores signed values of range [-2147483648, 2147483647], and unsigned values of range [0, 4294967295]:

```
INT[(M)] [UNSIGNED] [ZEROFILL]
```

You can also use another form:

```
INTEGER[(M)] [UNSIGNED] [ZEROFILL]
```

`BIGINT` type

The `BIGINT` data type stores signed values of range [-9223372036854775808, 9223372036854775807], and unsigned values of range [0, 18446744073709551615]:

```
BIGINT[(M)] [UNSIGNED] [ZEROFILL]
```

14.10.3.3.2 Floating-point types

TiDB supports all the MySQL floating-point types, including `FLOAT`, and `DOUBLE`. For more information, see Floating-Point Types (Approximate Value) - FLOAT, DOUBLE in MySQL.

The following table summarizes field descriptions:

Syntax Element	Description
M	the total number of digits
D	the number of digits following the decimal point
UNSIGNED	UNSIGNED. If omitted, it is SIGNED.
ZEROFILL	If you specify ZEROFILL for a numeric column, TiDB automatically adds the UNSIGNED attribute to the column.

The following table summarizes the required storage for floating-point types supported by TiDB:

Data Type	Storage Required (bytes)
`FLOAT`	4
`FLOAT(p)`	If 0 <= p <= 24, it is 4; if 25 <= p <= 53, it is 8
`DOUBLE`	8

`FLOAT` type

The `FLOAT` type stores a single-precision floating-point number. Permissible values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly

smaller depending on your hardware or operating system.

`FLOAT(p)` can be used to represent the required precision in bits. TiDB uses this value only to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT with no M or D values. If p is from 25 to 53, the data type becomes `DOUBLE` with no M or D values. The range of the resulting column is the same as for the single-precision `FLOAT` or double-precision `DOUBLE` data type.

```
FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]
FLOAT(p) [UNSIGNED] [ZEROFILL]
```

Note:

As in MySQL, the `FLOAT` data type stores approximate values. For values such as currency, it is recommended to use the `DECIMAL` type instead.

In TiDB, the default precision of the `FLOAT` data type is 8 digits, but in MySQL, the default precision is 6 digits. For example, assuming that you insert `123456789` and `1.23456789` into columns of the `FLOAT` type in both TiDB and MySQL, when you query the corresponding values in MySQL, you get `123457000` and `1.23457`, while in TiDB, you get `123456790` and `1.2345679`.

`DOUBLE` type

The `DOUBLE` type, and its alias `DOUBLE PRECISION` stores a double-precision floating-point number. Permissible values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

```
DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]
DOUBLE PRECISION [(M,D)] [UNSIGNED] [ZEROFILL], REAL[(M,D)] [UNSIGNED] [
    ↪ ZEROFILL]
```

Warning:

As in MySQL, the `DOUBLE` data type stores approximate values. For values such as currency, it is recommended to use the `DECIMAL` type instead.

14.10.3.3.3 Fixed-point types

TiDB supports all the MySQL floating-point types, including DECIMAL, and NUMERIC. For more information, Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC in MySQL.

The meaning of the fields:

Syntax Element	Description
M	the total number of decimal digits
D	the number of digits after the decimal point
UNSIGNED	UNSIGNED. If omitted, it is SIGNED.
ZEROFILL	If you specify ZEROFILL for a numeric column, TiDB automatically adds the UNSIGNED attribute to the column.

`DECIMAL` type

`DECIMAL` and its alias `NUMERIC` store a packed "exact" fixed-point number. M is the total number of decimal digits (the precision), and D is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the - sign are not counted in M. If D is 0, values have no decimal point or fractional part. The maximum number of digits (M) for DECIMAL is 65. The maximum number of supported decimals (D) is 30. If D is omitted, the default is 0. If M is omitted, the default is 10.

```
DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]
NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]
```

14.10.3.4 Date and Time Types

TiDB supports all MySQL date and time data types to store temporal values: `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, and `YEAR`. For more information, see Date and Time Data Types in MySQL.

Each of these types has its range of valid values, and uses a zero value to indicate that it is an invalid value. In addition, the `TIMESTAMP` and `DATETIME` types can automatically generate new time values on modification.

When dealing with date and time value types, note:

- Although TiDB tries to interpret different formats, the date-portion must be in the format of year-month-day (for example, '1998-09-04'), rather than month-day-year or day-month-year.

- If the year-portion of a date is specified as 2 digits, TiDB converts it based on specific rules.
- If a numeric value is needed in the context, TiDB automatically converts the date or time value into a numeric type. For example:

```
mysql> SELECT NOW(), NOW()+0, NOW(3)+0;
+---------------------+---------------+-------------------+
| NOW()               | NOW()+0       | NOW(3)+0          |
+---------------------+---------------+-------------------+
| 2012-08-15 09:28:00 | 20120815092800 | 20120815092800.889 |
+---------------------+---------------+-------------------+
```

- TiDB might automatically convert invalid values or values beyond the supported range to a zero value of that type. This behavior is dependent on the SQL Mode set. For example:

```
mysql> show create table t1;
+--
    ↪ -----+-----------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+-----------------------------------------------------------------
    ↪
| t1    | CREATE TABLE `t1` (
  `a` time DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin |
+--
    ↪ -----+-----------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> select @@sql_mode;
+--
    ↪ -----------------------------------------------------------------------
    ↪
| @@sql_mode
    ↪
    ↪ |
+--
    ↪ -----------------------------------------------------------------------
    ↪
```

```
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
    ↪ ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,
    ↪ NO_ENGINE_SUBSTITUTION |
+--
    ↪ ----------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)

mysql> insert into t1 values ('2090-11-32:22:33:44');
ERROR 1292 (22007): Truncated incorrect time value: '
    ↪ 2090-11-32:22:33:44'
mysql> set @@sql_mode='';
    ↪
    ↪ Query OK, 0 rows affected (0.01 sec)

mysql> insert into t1 values ('2090-11-32:22:33:44');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> select * from t1;
+----------+
| a        |
+----------+
| 00:00:00 |
+----------+
1 row in set (0.01 sec)
```

- Setting different SQL modes can change TiDB behaviors.
- If the SQL mode `NO_ZERO_DATE` is not enabled, TiDB allows month or day in the columns of `DATE` and `DATETIME` to be zero value, for example, '2009-00-00' or '2009-01-00'. If this date type is to be calculated in a function, for example, in `DATE_SUB()` or `DATE_ADD()`, the result can be incorrect.
- By default, TiDB enables the SQL mode `NO_ZERO_DATE`. This mode prevents storing zero values such as '0000-00-00'.

Different types of zero value are shown in the following table:

Date Type	"Zero" Value
DATE	'0000-00-00'
TIME	'00:00:00'
DATETIME	'0000-00-00 00:00:00'
TIMESTAMP	'0000-00-00 00:00:00'
YEAR	0000

Invalid `DATE`, `DATETIME`, `TIMESTAMP` values are automatically converted to the corresponding type of zero value ('0000-00-00' or '0000-00-00 00:00:00') if the SQL mode permits such usage.

14.10.3.4.1 Supported types

`DATE` type

`DATE` only contains date-portion and no time-portion, displayed in `YYYY-MM-DD` format. The supported range is '0000-01-01' to '9999-12-31':

```
DATE
```

`TIME` type

For the `TIME` type, the format is `HH:MM:SS[.fraction]` and valid values range from '-838:59:59.000000' to '838:59:59.000000'. `TIME` is used not only to indicate the time within a day but also to indicate the time interval between 2 events. An optional `fsp` value in the range from 0 to 6 may be given to specify fractional seconds precision. If omitted, the default precision is 0:

```
TIME[(fsp)]
```

> **Note:**
>
> Pay attention to the abbreviated form of `TIME`. For example, '11:12' means '11:12:00' instead of '00:11:12'. However, '1112' means '00:11:12'. These differences are caused by the presence or absence of the `:` character.

`DATETIME` type

`DATETIME` contains both date-portion and time-portion. Valid values range from '0000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'.

TiDB displays `DATETIME` values in `YYYY-MM-DD HH:MM:SS[.fraction]` format, but permits assignment of values to `DATETIME` columns using either strings or numbers. An optional fsp value in the range from 0 to 6 may be given to specify fractional seconds precision. If omitted, the default precision is 0:

```
DATETIME[(fsp)]
```

`TIMESTAMP` type

`TIMESTAMP` contains both date-portion and time-portion. Valid values range from '1970-01-01 00:00:01.000000' to '2038-01-19 03:14:07.999999' in UTC time. An optional fsp value

in the range from 0 to 6 may be given to specify fractional seconds precision. If omitted, the default precision is 0.

In `TIMESTAMP`, zero is not permitted to appear in the month-portion or day-portion. The only exception is zero value itself '0000-00-00 00:00:00'.

```
TIMESTAMP[(fsp)]
```

Timezone Handling

When `TIMESTAMP` is to be stored, TiDB converts the `TIMESTAMP` value from the current time zone to UTC time zone. When `TIMESTAMP` is to be retrieved, TiDB converts the stored `TIMESTAMP` value from UTC time zone to the current time zone (Note: `DATETIME` is not handled in this way). The default time zone for each connection is the server's local time zone, which can be modified by the environment variable `time_zone`.

> **Warning:**
>
> As in MySQL, the `TIMESTAMP` data type suffers from the Year 2038 Problem. For storing values that may span beyond 2038, please consider using the `DATETIME` type instead.

`YEAR` type

The `YEAR` type is specified in the format 'YYYY'. Supported values range from 1901 to 2155, or the zero value of 0000:

```
YEAR[(4)]
```

`YEAR` follows the following format rules:

- Four-digit numeral ranges from 1901 to 2155
- Four-digit string ranges from '1901' to '2155'
- One-digit or two-digit numeral ranges from 1 to 99. Accordingly, 1-69 is converted to 2001-2069 and 70-99 is converted to 1970-1999
- One-digit or two-digit string ranges from '0' to '99'
- Value 0 is taken as 0000 whereas the string '0' or '00' is taken as 2000

Invalid `YEAR` value is automatically converted to 0000 (if users are not using the `NO_ZERO_DATE` SQL mode).

14.10.3.4.2 Automatic initialization and update of TIMESTAMP and DATETIME

Columns with TIMESTAMP or DATETIME value type can be automatically initialized or updated to the current time.

For any column with TIMESTAMP or DATETIME value type in the table, you can set the default or auto-update value as current timestamp.

These properties can be set by setting DEFAULT CURRENT_TIMESTAMP and ON UPDATE ↪ CURRENT_TIMESTAMP when the column is being defined. DEFAULT can also be set as a specific value, such as DEFAULT 0 or DEFAULT '2000-01-01 00:00:00'.

```
CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

The default value for DATETIME is NULL unless it is specified as NOT NULL. For the latter situation, if no default value is set, the default value is be 0.

```
CREATE TABLE t1 (
    dt1 DATETIME ON UPDATE CURRENT_TIMESTAMP,   -- default NULL
    dt2 DATETIME NOT NULL ON UPDATE CURRENT_TIMESTAMP -- default 0
);
```

14.10.3.4.3 Decimal part of time value

DATETIME and TIMESTAMP values can contain a fractional part of up to 6 digits which is accurate to microseconds. In any column of DATETIME or TIMESTAMP types, a fractional part is stored instead of being discarded. With a fractional part, the value is in the format of 'YYYY-MM-DD HH:MM:SS[.fraction]', and the fraction ranges from 000000 to 999999. A decimal point must be used to separate the fraction from the rest.

- Use type_name(fsp) to define a column that supports fractional precision, where type_name can be TIME, DATETIME or TIMESTAMP. For example,

  ```
  CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
  ```

  fsp must range from 0 to 6.

  0 means there is no fractional part. If fsp is omitted, the default is 0.

- When inserting TIME, DATETIME or TIMESTAMP which contain a fractional part, if the number of digit of the fraction is too few, or too many, rounding might be needed in the situation. For example:

```
mysql> CREATE TABLE fractest( c1 TIME(2), c2 DATETIME(2), c3 TIMESTAMP
    ↪ (2) );
Query OK, 0 rows affected (0.33 sec)

mysql> INSERT INTO fractest VALUES
    > ('17:51:04.777', '2014-09-08 17:51:04.777', '2014-09-08
        ↪ 17:51:04.777');
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM fractest;
+-------------/-----------------------/-----------------------+
| c1          | c2                    | c3                    |
+-------------/-----------------------/-----------------------+
| 17:51:04.78 | 2014-09-08 17:51:04.78 | 2014-09-08 17:51:04.78 |
+-------------/-----------------------/-----------------------+
1 row in set (0.00 sec)
```

14.10.3.4.4 Conversions between date and time types

Sometimes we need to make conversions between date and time types. But some conversions might lead to information loss. For example, `DATE`, `DATETIME` and `TIMESTAMP` values all have their own respective ranges. `TIMESTAMP` should be no earlier than the year 1970 in UTC time or no later than UTC time '2038-01-19 03:14:07'. Based on this rule, '1968-01-01' is a valid date value of `DATE` or `DATETIME`, but becomes 0 when it is converted to `TIMESTAMP`.

The conversions of `DATE`:

- When `DATE` is converted to `DATETIME` or `TIMESTAMP`, a time-portion '00:00:00' is added, because DATE does not contain any time information
- When `DATE` is converted to `TIME`, the result is '00:00:00'

Conversions of `DATETIME` or `TIMESTAMP`:

- When `DATETIME` or `TIMESTAMP` is converted to `DATE`, the time and fractional part is discarded. For example, '1999-12-31 23:59:59.499' is converted to '1999-12-31'
- When `DATETIME` or `TIMESTAMP` is converted to TIME, the date-portion is discarded, because `TIME` does not contain any date information

When we convert `TIME` to other time and date formats, the date-portion is automatically specified as `CURRENT_DATE()`. The final converted result is a date that consists of `TIME` and `CURRENT_DATE()`. This is to say that if the value of TIME is beyond the range from '00:00:00' to '23:59:59', the converted date-portion does not indicate the current day.

When `TIME` is converted to `DATE`, the process is similar, and the time-portion is discarded.

Using the `CAST()` function can explicitly convert a value to a `DATE` type. For example:

```
date_col = CAST(datetime_col AS DATE)
```

Converting TIME and DATETIME to numeric format. For example:

```
mysql> SELECT CURTIME(), CURTIME()+0, CURTIME(3)+0;
+-----------/-------------/--------------+
| CURTIME() | CURTIME()+0 | CURTIME(3)+0 |
+-----------/-------------/--------------+
| 09:28:00 |      92800 |   92800.887 |
+-----------/-------------/--------------+
mysql> SELECT NOW(), NOW()+0, NOW(3)+0;
+---------------------/----------------/--------------------+
| NOW()               | NOW()+0       | NOW(3)+0           |
+---------------------/----------------/--------------------+
| 2012-08-15 09:28:00 | 20120815092800 | 20120815092800.889 |
+---------------------/----------------/--------------------+
```

14.10.3.4.5 Two-digit year-portion contained in the date

The two-digit year-portion contained in date does not explicitly indicate the actual year and is ambiguous.

For DATETIME, DATE and TIMESTAMP types, TiDB follows the following rules to eliminate ambiguity:

- Values between 01 and 69 is converted to a value between 2001 and 2069
- Values between 70 and 99 is converted to a value between 1970 and 1999

These rules also apply to the YEAR type, with one exception:

When numeral 00 is inserted to YEAR(4), the result is 0000 rather than 2000.

If you want the result to be 2000, specify the value to be 2000.

The two-digit year-portion might not be properly calculated in some functions such MIN() and MAX(). For these functions, the four-digit format suites better.

14.10.3.5 String Types

TiDB supports all the MySQL string types, including CHAR, VARCHAR, BINARY, VARBINARY , BLOB, TEXT, ENUM, and SET. For more information, see String Types in MySQL.

14.10.3.5.1 Supported types

CHAR type

CHAR is a fixed length string. M represents the column-length in characters (not bytes). The range of M is 0 to 255. Different from the VARCHAR type, when data is inserted into a CHAR column, the trailing spaces are truncated.

```
[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]
```

`VARCHAR` type

`VARCHAR` is a string of variable-length. M represents the maximum column length in characters (not bytes). The maximum size of `VARCHAR` cannot exceed 65,535 bytes. The maximum row length and the character set being used determine the `VARCHAR` length.

The space occupied by a single character might differ for different character sets. The following table shows the bytes consumed by a single character, and the range of the `VARCHAR` column length in each character set:

Character Set	Byte(s) per Character	Range of the Maximum `VARCHAR` Column Length
ascii	1	(0, 65535]
latin1	1	(0, 65535]
binary	1	(0, 65535]
utf8	3	(0, 21845]
utf8mb4	4	(0, 16383]

```
[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]
```

`TEXT` type

`TEXT` is a string of variable-length. The maximum column length is 65,535 bytes. The optional M argument is in characters and is used to automatically select the fittest type of a `TEXT` column. For example `TEXT(60)` will yield a `TINYTEXT` data type that can hold up to 255 bytes, which fits a 60-character UTF-8 string that has up to 4 bytes per character (4×60=240). Using the M argument is not recommended.

```
TEXT[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]
```

`TINYTEXT` type

The `TINYTEXT` type is similar to the TEXT type. The difference is that the maximum column length of `TINYTEXT` is 255.

```
TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

`MEDIUMTEXT` type

The `MEDIUMTEXT` type is similar to the TEXT type. The difference is that the maximum column length of `MEDIUMTEXT` is 16,777,215. But due to the limitation of txn-entry-size-limit, the maximum storage size of a single row in TiDB is 6 MiB by default and can be increased to 120 MiB by changing the configuration.

```
MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

`LONGTEXT` type

The `LONGTEXT` type is similar to the `TEXT` type. The difference is that the maximum column length of `LONGTEXT` is 4,294,967,295. But due to the limitation of `txn-entry-size-limit`, the maximum storage size of a single row in TiDB is 6 MiB by default and can be increased to 120 MiB by changing the configuration.

```
LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

`BINARY` type

The `BINARY` type is similar to the `CHAR` type. The difference is that `BINARY` stores binary byte strings.

```
BINARY(M)
```

`VARBINARY` type

The `VARBINARY` type is similar to the `VARCHAR` type. The difference is that the `VARBINARY` stores binary byte strings.

```
VARBINARY(M)
```

`BLOB` type

`BLOB` is a large binary file. M represents the maximum column length in bytes, ranging from 0 to 65,535.

```
BLOB[(M)]
```

`TINYBLOB` type

The `TINYBLOB` type is similar to the `BLOB` type. The difference is that the maximum column length of `TINYBLOB` is 255.

```
TINYBLOB
```

`MEDIUMBLOB` type

The `MEDIUMBLOB` type is similar to the `BLOB` type. The difference is that the maximum column length of `MEDIUMBLOB` is 16,777,215. But due to the limitation of `txn-entry-size-limit`, the maximum storage size of a single row in TiDB is 6 MiB by default and can be increased to 120 MiB by changing the configuration.

```
MEDIUMBLOB
```

`LONGBLOB` type

The `LONGBLOB` type is similar to the `BLOB` type. The difference is that the maximum column length of `LONGBLOB` is 4,294,967,295. But due to the limitation of `txn-entry-size-limit`, the maximum storage size of a single row in TiDB is 6 MiB by default and can be increased to 120 MiB by changing the configuration.

```
LONGBLOB
```

`ENUM` type

An `ENUM` is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification when the table is created. The syntax is:

```
ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE
    ↪ collation_name]

#### For example:
ENUM('apple', 'orange', 'pear')
```

The value of the `ENUM` data type is stored as numbers. Each value is converted to a number according the definition order. In the previous example, each string is mapped to a number:

Value	Number
NULL	NULL
”	0
‘apple’	1
‘orange’	2
‘pear’	3

For more information, see the ENUM type in MySQL.

`SET` type

A `SET` is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. The syntax is:

```
SET('value1','value2',...) [CHARACTER SET charset_name] [COLLATE
    ↪ collation_name]

#### For example:
SET('1', '2') NOT NULL
```

In the example, any of the following values can be valid:

```
''
'1'
'2'
'1,2'
```

In TiDB, the values of the `SET` type is internally converted to `Int64`. The existence of each element is represented using a binary: 0 or 1. For a column specified as `SET('a','b ↪ ','c','d')`, the members have the following decimal and binary values.

Member	Decimal Value	Binary Value
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

In this case, for an element of `('a', 'c')`, it is `0101` in binary.

For more information, see the SET type in MySQL.

14.10.3.6 JSON Type

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

TiDB supports the `JSON` (JavaScript Object Notation) data type, which is useful for storing semi-structured data. The `JSON` data type provides the following advantages over storing `JSON`-format strings in a string column:

- Use the Binary format for serialization. The internal format permits quick read access to `JSON` document elements.
- Automatic validation of the JSON documents stored in `JSON` columns. Only valid documents can be stored.

`JSON` columns, like columns of other binary types, are not indexed directly, but you can index the fields in the `JSON` document in the form of generated column:

```
CREATE TABLE city (
    id INT PRIMARY KEY,
    detail JSON,
    population INT AS (JSON_EXTRACT(detail, '$.population')),
    index index_name (population)
    );
INSERT INTO city (id,detail) VALUES (1, '{"name": "Beijing", "population":
    ↪ 100}');
SELECT id FROM city WHERE population >= 100;
```

For more information, see JSON Functions and Generated Columns.

14.10.4 Functions and Operators

14.10.4.1 Function and Operator Reference

The usage of the functions and operators in TiDB is similar to MySQL. See Functions and Operators in MySQL.

In SQL statements, expressions can be used on the `ORDER BY` and `HAVING` clauses of the `SELECT` statement, the `WHERE` clause of `SELECT`/`DELETE`/`UPDATE` statements, and `SET` statements.

You can write expressions using literals, column names, NULL, built-in functions, operators and so on. For expressions that TiDB supports pushing down to TiKV, see List of Expressions for Pushdown.

14.10.4.2 Type Conversion in Expression Evaluation

TiDB behaves the same as MySQL: https://dev.mysql.com/doc/refman/5.7/en/type-conversion.html

14.10.4.3 Operators

This document describes the operators precedence, comparison functions and operators, logical operators, and assignment operators.

- Operator precedence
- Comparison functions and operators
- Logical operators
- Assignment operators

Name	Description
AND, &&	Logical AND
=	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
:=	Assign a value
BETWEEN … AND …	Check whether a value is within a range of values
BINARY	Cast a string to a binary string
&	Bitwise AND
~	Bitwise inversion
|	Bitwise OR
[^](https://dev.mysql.com/doc/refman/5.7/en/bit-functions.html#operator_bitwise-xor)	Bitwise XOR
CASE	Case operator
DIV	Integer division

Name	Description
/	Division operator
=	Equal operator
<=>	NULL-safe equal to operator
>	Greater than operator
>=	Greater than or equal operator
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
->	Return value from JSON column after evaluating path; equivalent to `JSON_EXTRACT()`
->>	Return value from JSON column after evaluating path and unquoting the result; equivalent to `JSON_UNQUOTE(JSON_EXTRACT())`
<<	Left shift
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching
-	Minus operator
%, MOD	Modulo operator
NOT, !	Negates value
NOT BETWEEN … AND …	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
||, OR	Logical OR
+	Addition operator
REGEXP	Pattern matching using regular expressions
>>	Right shift
RLIKE	Synonym for REGEXP
*	Multiplication operator
-	Change the sign of the argument
XOR	Logical XOR

14.10.4.3.1 Unsupported operators

- `SOUNDS LIKE`

14.10.4.3.2 Operator precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
AND, &&
XOR
OR, ||
= (assignment), :=
```

For details, see Operator Precedence.

14.10.4.3.3 Comparison functions and operators

Name	Description
BETWEEN … AND …	Check whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
=	Equal operator
<=>	NULL-safe equal to operator
>	Greater than operator
>=	Greater than or equal operator
GREATEST()	Return the largest argument
IN()	Check whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching
NOT BETWEEN … AND …	Check whether a value is not within a range of values

Name	Description
!=, <>	Not equal operator
NOT IN()	Check whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

For details, see Comparison Functions and Operators.

14.10.4.3.4 Logical operators

Name	Description
AND, &&	Logical AND
NOT, !	Negates value
||, OR	Logical OR
XOR	Logical XOR

For details, see MySQL Handling of GROUP BY.

14.10.4.3.5 Assignment operators

Name	Description
=	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
:=	Assign a value

For details, see Detection of Functional Dependence.

14.10.4.4 Control Flow Functions

TiDB supports all of the control flow functions available in MySQL 5.7.

Name	Description
`CASE`	Case operator
`IF()`	If/else construct
`IFNULL()`	Null if/else construct
`NULLIF()`	Return NULL if expr1 = expr2

14.10.4.5 String Functions

TiDB supports most of the string functions available in MySQL 5.7 and some of the functions available in Oracle 21.

For comparisons between functions and syntax of Oracle and TiDB, see Comparisons between Functions and Syntax of Oracle and TiDB.

14.10.4.5.1 Supported functions

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Return the index position of the first argument within the second argument
FORMAT()	Return a number formatted to specified number of decimal places
FROM_BASE64()	Decode to a base-64 string and return result
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOCATE()	Return the position of the first occurrence of substring

Name	Description
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
TO_BASE64()	Return the argument converted to a base-64 string

Name	Description
TRANSLATE()	Replace all occurrences of characters by other characters in a string. It does not treat empty strings as NULL as Oracle does.
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for the input string

14.10.4.5.2 Unsupported functions

- LOAD_FILE()
- MATCH()
- SOUNDEX()

14.10.4.6 Numeric Functions and Operators

TiDB supports all of the numeric functions and operators available in MySQL 5.7.

14.10.4.6.1 Arithmetic operators

Name	Description
+	Addition operator
-	Minus operator
*	Multiplication operator
/	Division operator
DIV	Integer division
lstinlineMOD!	Modulo operator
-	Change the sign of the argument

14.10.4.6.2 Mathematical functions

Name	Description
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
EXP()	Raise to the power of
SQRT()	Return the square root of the argument

Name	Description
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG2()	Return the base-2 logarithm of the argument
LOG10()	Return the base-10 logarithm of the argument
PI()	Return the value of pi
TAN()	Return the tangent of the argument
COT()	Return the cotangent
SIN()	Return the sine of the argument
COS()	Return the cosine
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
ASIN()	Return the arc sine
ACOS()	Return the arc cosine
RADIANS()	Return argument converted to radians
DEGREES()	Convert radians to degrees
MOD()	Return the remainder
ABS()	Return the absolute value
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
FLOOR()	Return the largest integer value not greater than the argument
ROUND()	Round the argument
RAND()	Return a random floating-point value
SIGN()	Return the sign of the argument
CONV()	Convert numbers between different number bases
TRUNCATE()	Truncate to specified number of decimal places
CRC32()	Compute a cyclic redundancy check value

14.10.4.7 Date and Time Functions

TiDB supports all of the date and time functions available in MySQL 5.7.

Note:

- MySQL will often accept the incorrectly formatted date and time values. For example, `'2020-01-01\n\t01:01:01'` and `'2020-01_01\n\t01:01'` are treated as valid date and time values.
- TiDB makes the best effort to match MySQL's behavior, but it might not match in all instances. It is recommended to correctly format dates, because the intended behavior for incorrectly formatted values is not documented, and is often inconsistent.

Date/Time functions:

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CONVERT_TZ()	Convert from one time zone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
GET_FORMAT()	Return a date format string
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods

Name	Description
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

For details, see Date and Time Functions.

14.10.4.7.1 MySQL compatibility

The function `str_to_date()` is supported by TiDB, but is unable to parse all date and time values. In addition, the following date and time formatting options are **not implemented**:

Format	Description
“%a”	Abbreviated weekday name (Sun..Sat)
“%D”	Day of the month with English suffix (0th, 1st, 2nd, 3rd)
“%U”	Week (00..53), where Sunday is the first day of the week; WEEK() mode 0
“%u”	Week (00..53), where Monday is the first day of the week; WEEK() mode 1
“%V”	Week (01..53), where Sunday is the first day of the week; WEEK() mode 2; used with %X
“%v”	Week (01..53), where Monday is the first day of the week; WEEK() mode 3; used with %x
“%W”	Weekday name (Sunday..Saturday)
“%w”	Day of the week (0=Sunday..6=Saturday)
“%X”	Year for the week where Sunday is the first day of the week, numeric, four digits.
“%x”	Year for the week, where Monday is the first day of the week, numeric, four digits.

See issue #30082 for more details.

14.10.4.7.2 Related system variables

The `default_week_format` variable affects the `WEEK()` function.

14.10.4.8 Bit Functions and Operators

TiDB supports all of the bit functions and operators available in MySQL 5.7.

Bit functions and operators:

Name	Description
`BIT_COUNT ↪ ()`	Return the number of bits that are set as 1
&	Bitwise AND
~	Bitwise inversion
|	Bitwise OR
[^](https://dev.mysql.com/doc/refman/5.7/en/bit-functions.html#operator_bitwise-xor)	Bitwise XOR
<<	Left shift
>>	Right shift

14.10.4.9 Cast Functions and Operators

TiDB supports all of the cast functions and operators available in MySQL 5.7.

Name	Description
BINARY	Cast a string to a binary string
CAST()	Cast a value as a certain type
CONVERT()	Cast a value as a certain type

Cast functions and operators enable conversion of values from one data type to another.

14.10.4.10 Encryption and Compression Functions

TiDB supports most of the encryption and compression functions available in MySQL 5.7.

14.10.4.10.1 Supported functions

Name	Description
MD5()	Calculate MD5 checksum
PASSWORD()	Calculate and return a password string
RANDOM_BYTES()	Return a random byte vector
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
AES_DECRYPT()	Decrypt using AES
AES_ENCRYPT()	Encrypt using AES
COMPRESS()	Return result as a binary string
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression

14.10.4.10.2 Related system variables

The `block_encryption_mode` variable sets the encryption mode that is used for `AES_ENCRYPT()` and `AES_DECRYPT()`.

14.10.4.10.3 Unsupported functions

- `DES_DECRYPT()`, `DES_ENCRYPT()`, `OLD_PASSWORD()`, `ENCRYPT()`: these functions were deprecated in MySQL 5.7 and removed in 8.0.
- `VALIDATE_PASSWORD_STRENGTH()`.
- Functions only available in MySQL Enterprise Issue #2632.

14.10.4.11 Locking Functions

TiDB supports most of the user-level locking functions available in MySQL 5.7.

14.10.4.11.1 Supported functions

Name	Description
`GET_LOCK(lockName, timeout)`	Acquires an advisory lock. The `lockName` parameter must be NO longer than 64 characters. Waits maximum `timeout` seconds before timing out and returns a failure.
`RELEASE_LOCK(lockName)`	Releases a previously acquired lock. The `lockName` parameter must be NO longer than 64 characters.
`RELEASE_ALL_LOCKS()`	Releases all locks held by the current session.

14.10.4.11.2 MySQL compatibility

- The minimum timeout permitted by TiDB is 1 second, and the maximum timeout is 1 hour (3600 seconds). This differs from MySQL, where both 0 second and unlimited timeouts (`timeout=-1`) are permitted. TiDB will automatically convert out-of-range values to the nearest permitted value and convert `timeout=-1` to 3600 seconds.
- TiDB does not automatically detect deadlocks caused by user-level locks. Deadlocked sessions will timeout after a maximum of 1 hour, but can also be manually resolved by using `KILL` on one of the affected sessions. You can also prevent deadlocks by always acquiring user-level locks in the same order.
- Locks take effect on all TiDB servers in the cluster. This differs from MySQL Cluster and Group Replication where locks are local to a single server.

14.10.4.11.3 Unsupported functions

- `IS_FREE_LOCK()`
- `IS_USED_LOCK()`

14.10.4.12 Information Functions

TiDB supports most of the information functions available in MySQL 5.7.

14.10.4.12.1 Supported functions

Name	Description
BENCHMARK()	Execute an expression in a loop
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
CURRENT_USER(), CURRENT_USER	Return the authenticated user name and host name
DATABASE()	Return the default (current) database name
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of the rows that are returned if there is no LIMIT clause
LAST_INSERT_ID()	Return the value of the AUTOINCREMENT column for the last INSERT
ROW_COUNT()	The number of rows affected
SCHEMA()	Synonym for DATABASE()
SESSION_USER()	Synonym for USER()
SYSTEM_USER()	Synonym for USER()

Name	Description
USER()	Return the user name and host name provided by the client
VERSION ()	Return a string that indicates the MySQL server version

14.10.4.12.2 Unsupported functions

- CHARSET()
- COERCIBILITY()
- COLLATION()

14.10.4.13 JSON Functions

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

TiDB supports most of the JSON functions that shipped with the GA release of MySQL 5.7.

14.10.4.13.1 Functions that create JSON values

Function Name	Description
JSON_ARRAY([val[, val] ...])	Evaluates a (possibly empty) list of values and returns a JSON array containing those values

Function Name	Description
JSON_OBJECT(key, val[, key, val] …)	Evaluates a (possibly empty) list of key-value pairs and returns a JSON object containing those pairs
JSON_QUOTE(string)	Returns a string as a JSON value with quotes

14.10.4.13.2 Functions that search JSON values

Function Name	Description
JSON_CONTAINS(target, candidate[, path])	Indicates by returning 1 or 0 whether a given candidate JSON document is contained within a target JSON document
JSON_CONTAINS_PATH(json_doc, one_or_all, path[, path] …)	Returns 0 or 1 to indicate whether a JSON document contains data at a given path or paths

Function Name	Description
JSON_EXTRACT(json_doc, path[, path] ...)	Returns data from a JSON document, selected from the parts of the document matched by the `path` arguments
->	Returns the value from a JSON column after the evaluating path; an alias for `JSON_EXTRACT ↪ (doc, ↪ path_literal ↪ )`
->>	Returns the value from a JSON column after the evaluating path and unquoting the result; an alias for `JSON_UNQUOTE ↪ ( ↪ JSON_EXTRACT ↪ (doc, ↪ path_literal ↪ ))`

Function Name	Description
JSON_KEYS(json_doc[, path])	Returns the keys from the top-level value of a JSON object as a JSON array, or, if a path argument is given, the top-level keys from the selected path
JSON_SEARCH(json_doc, one_or_all, search_string)	Search a JSON document for one or all matches of a string

14.10.4.13.3 Functions that modify JSON values

Function Name	Description
JSON_APPEND(json_doc, path, value)	An alias to `JSON_ARRAY_APPEND` ↪
JSON_ARRAY_APPEND(json_doc, path, value)	Appends a value to the end of a JSON array at a specified path
JSON_ARRAY_INSERT(json_doc, path, val[, path, val] ...)	Inserts an array into the json document and returns the modified document

Function Name	Description
JSON_INSERT(json_doc, path, val[, path, val] …)	Inserts data into a JSON document and returns the result
JSON_MERGE(json_doc, json_doc[, json_doc] …)	A deprecated alias for `JSON_MERGE_PRESERVE` ↪
JSON_MERGE_PATCH(json_doc, json_doc[, json_doc] …)	Merge JSON documents
JSON_MERGE_PRESERVE(json_doc, json_doc[, json_doc] …)	Merges two or more JSON documents and returns the merged result
JSON_REMOVE(json_doc, path[, path] …)	Removes data from a JSON document and returns the result
JSON_REPLACE(json_doc, path, val[, path, val] …)	Replaces existing values in a JSON document and returns the result
JSON_SET(json_doc, path, val[, path, val] …)	Inserts or updates data in a JSON document and returns the result

Function Name	Description
JSON_UNQUOTE(json_val)	Unquotes a JSON value and returns the result as a string
JSON_ARRAY_APPEND(json_doc, path, val[, path, val] ...)	Appends values to the end of the indicated arrays within a JSON document and returns the result
JSON_ARRAY_INSERT(json_doc, path, val[, path, val] ...)	Insert values into the specified location of a JSON document and returns the result

14.10.4.13.4 Functions that return JSON value attributes

Function Name	Description
JSON_DEPTH(json_doc)	Returns the maximum depth of a JSON document
JSON_LENGTH(json_doc[, path])	Returns the length of a JSON document, or, if a path argument is given, the length of the value within the path

Function Name	Description
JSON_TYPE(json_val)	Returns a string indicating the type of a JSON value
JSON_VALID(json_doc)	Checks if a json_doc is valid JSON. Useful for checking a column before converting it to the json type.

14.10.4.13.5 Utility Functions

Function Name	Description
JSON_PRETTY(json_doc)	Pretty formatting of a JSON document
JSON_STORAGE_SIZE(json_doc)	Returns an approximate size of bytes required to store the json value. As the size does not account for TiKV using compression, the output of this function is not strictly compatible with MySQL.

14.10.4.13.6 Aggregate Functions

Function Name	Description
JSON_ARRAYAGG(key)	Provides an aggregation of keys.
JSON_OBJECTAGG(key, value)	Provides an aggregation of values for a given key.

14.10.4.13.7 See also

- JSON Function Reference
- JSON Data Type

14.10.4.14 Aggregate (GROUP BY) Functions

This document describes details about the supported aggregate functions in TiDB.

14.10.4.14.1 Supported aggregate functions

This section describes the supported MySQL `GROUP BY` aggregate functions in TiDB.

Name	Description
`COUNT()`	Return a count of the number of rows returned
`COUNT(DISTINCT)`	Return the count of a number of different values
`SUM()`	Return the sum
`AVG()`	Return the average value of the argument
`MAX()`	Return the maximum value
`MIN()`	Return the minimum value
`GROUP_CONCAT()`	Return a concatenated string
`VARIANCE()`, `VAR_POP()`	Return the population standard variance
`STD()`, `STDDEV()`, `STDDEV_POP`	Return the population standard deviation
`VAR_SAMP()`	Return the sample variance
`STDDEV_SAMP()`	Return the sample standard deviation

Name	Description
JSON_OBJECTAGG(key, value)	Return the result set as a single JSON object containing key-value pairs

- Unless otherwise stated, group functions ignore NULL values.
- If you use a group function in a statement containing no GROUP BY clause, it is equivalent to grouping on all rows.

In addition, TiDB also provides the following aggregate functions:

- APPROX_PERCENTILE(expr, constant_integer_expr)

  This function returns the percentile of expr. The constant_integer_expr argument indicates the percentage value which is a constant integer in the range of [1,100]. A percentile Pk (k represents percentage) indicates that there are at least k% values in the data set that are less than or equal to Pk.

  This function only supports the numeric type and the date and time type as the returned type of expr. For other returned types, APPROX_PERCENTILE only returns NULL.

  The following example shows how to calculate the fiftieth percentile of a INT column:

```
drop table if exists t;
create table t(a int);
insert into t values(1), (2), (3);
```

```
select approx_percentile(a, 50) from t;
```

```
+--------------------------+
| approx_percentile(a, 50) |
+--------------------------+
|                        2 |
+--------------------------+
1 row in set (0.00 sec)
```

- APPROX_COUNT_DISTINCT(expr, [expr...])

  This function is similar to COUNT(DISTINCT) in counting the number of distinct values but returns an approximate result. It uses the BJKST algorithm, significantly reducing memory consumption when processing large datasets with a power-law distribution. Moreover, for low-cardinality data, this function provides high accuracy while maintaining efficient CPU utilization.

  The following example shows how to use this function:

```
DROP TABLE IF EXISTS t;
CREATE TABLE t(a INT, b INT, c INT);
INSERT INTO t VALUES(1, 1, 1), (2, 1, 1), (2, 2, 1), (3, 1, 1), (5, 1,
    ↪ 2), (5, 1, 2), (6, 1, 2), (7, 1, 2);
```

```
SELECT APPROX_COUNT_DISTINCT(a, b) FROM t GROUP BY c;
```

```
+-----------------------------+
| approx_count_distinct(a, b) |
+-----------------------------+
|                           3 |
|                           4 |
+-----------------------------+
2 rows in set (0.00 sec)
```

Except for the `GROUP_CONCAT()`, `APPROX_PERCENTILE()`, and `APPROX_COUNT_DISTINCT` functions, all the preceding functions can serve as Window functions.

14.10.4.14.2 GROUP BY modifiers

TiDB does not currently support `GROUP BY` modifiers such as `WITH ROLLUP`. We plan to add support in the future. See TiDB #4250.

14.10.4.14.3 SQL mode support

TiDB supports the SQL Mode `ONLY_FULL_GROUP_BY`, and when enabled TiDB will refuse queries with ambiguous non-aggregated columns. For example, this query is illegal with `ONLY_FULL_GROUP_BY` enabled because the non-aggregated column "b" in the `SELECT` list does not appear in the `GROUP BY` statement:

```
drop table if exists t;
create table t(a bigint, b bigint, c bigint);
insert into t values(1, 2, 3), (2, 2, 3), (3, 2, 3);

mysql> select a, b, sum(c) from t group by a;
+------+------+--------+
| a    | b    | sum(c) |
+------+------+--------+
|    1 |    2 |      3 |
|    2 |    2 |      3 |
|    3 |    2 |      3 |
+------+------+--------+
3 rows in set (0.01 sec)
```

```
mysql> set sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> select a, b, sum(c) from t group by a;
ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP BY clause
    ↪ and contains nonaggregated column 'b' which is not functionally
    ↪ dependent on columns in GROUP BY clause; this is incompatible with
    ↪ sql_mode=only_full_group_by
```

TiDB currently enables the ONLY_FULL_GROUP_BY mode by default.

Differences from MySQL

The current implementation of ONLY_FULL_GROUP_BY is less strict than that in MySQL 5.7. For example, suppose that we execute the following query, expecting the results to be ordered by "c":

```
drop table if exists t;
create table t(a bigint, b bigint, c bigint);
insert into t values(1, 2, 1), (1, 2, 2), (1, 3, 1), (1, 3, 2);
select distinct a, b from t order by c;
```

To order the result, duplicates must be eliminated first. But to do so, which row should we keep? This choice influences the retained value of "c", which in turn influences ordering and makes it arbitrary as well.

In MySQL, a query that has DISTINCT and ORDER BY is rejected as invalid if any ORDER
↪ BY expression does not satisfy at least one of these conditions:

- The expression is equal to one in the SELECT list
- All columns referenced by the expression and belonging to the query's selected tables are elements of the SELECT list

But in TiDB, the above query is legal, for more information see #4254.

Another TiDB extension to standard SQL permits references in the HAVING clause to aliased expressions in the SELECT list. For example, the following query returns "name" values that occur only once in table "orders":

```
select name, count(name) from orders
group by name
having count(name) = 1;
```

The TiDB extension permits the use of an alias in the HAVING clause for the aggregated column:

```
select name, count(name) as c from orders
group by name
having c = 1;
```

Standard SQL permits only column expressions in `GROUP BY` clauses, so a statement such as this is invalid because "FLOOR(value/100)" is a noncolumn expression:

```
select id, floor(value/100)
from tbl_name
group by id, floor(value/100);
```

TiDB extends standard SQL to permit noncolumn expressions in `GROUP BY` clauses and considers the preceding statement valid.

Standard SQL also does not permit aliases in `GROUP BY` clauses. TiDB extends standard SQL to permit aliases, so another way to write the query is as follows:

```
select id, floor(value/100) as val
from tbl_name
group by id, val;
```

14.10.4.14.4 Unsupported aggregate functions

The following aggregate functions are currently unsupported in TiDB. You can track our progress in TiDB #7623:

- `JSON_ARRAYAGG`

14.10.4.14.5 Related system variables

The `group_concat_max_len` variable sets the maximum number of items for the `GROUP_CONCAT()` function.

14.10.4.15 Window Functions

The usage of window functions in TiDB is similar to that in MySQL 8.0. For details, see MySQL Window Functions.

Because window functions reserve additional words in the parser, TiDB provides an option to disable window functions. If you receive errors parsing SQL statements after upgrading, try setting `tidb_enable_window_function=0`.

Except for `GROUP_CONCAT()` and `APPROX_PERCENTILE()`, TiDB supports all `GROUP BY` aggregate functions. In addition, TiDB supports the following window functions:

Function name	Feature description
`CUME_DIST()`	Returns the cumulative distribution of a value within a group of values.
`DENSE_RANK()`	Returns the rank of the current row within the partition, and the rank is without gaps.

Function name	Feature description
FIRST_VALUE()	Returns the expression value of the first row in the current window.
LAG()	Returns the expression value from the row that precedes the current row by N rows within the partition.
LAST_VALUE()	Returns the expression value of the last row in the current window.
LEAD()	Returns the expression value from the row that follows the current row by N rows within the partition.
NTH_VALUE()	Returns the expression value from the N-th row of the current window.
NTILE()	Divides a partition into N buckets, assigns the bucket number to each row in the partition, and returns the bucket number of the current row within the partition.
PERCENT_RANK()	Returns the percentage of partition values that are less than the value in the current row.
RANK()	Returns the rank of the current row within the partition. The rank may be with gaps.
ROW_NUMBER()	Returns the number of the current row in the partition.

14.10.4.16 Miscellaneous Functions

TiDB supports most of the miscellaneous functions available in MySQL 5.7.

14.10.4.16.1 Supported functions

Name	Description
ANY_VALUE()	Suppress ONLY_FULL_GROUP_BY value rejection
BIN_TO_UUID()	Convert UUID from binary format to text format
DEFAULT()	Returns the default value for a table column
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
INET6_ATON()	Return the numeric value of an IPv6 address
INET6_NTOA()	Return the IPv6 address from a numeric value
IS_IPV4()	Whether argument is an IPv4 address
IS_IPV4_COMPAT()	Whether argument is an IPv4-compatible address
IS_IPV4_MAPPED()	Whether argument is an IPv4-mapped address

Name	Description
IS_IPV6()	Whether argument is an IPv6 address
NAME_CONST()	Can be used to rename a column name
SLEEP()	Sleep for a number of seconds
UUID()	Return a Universal Unique Identifier (UUID)
UUID_TO_BIN()	Convert UUID from text format to binary format
VALUES()	Defines the values to be used during an INSERT

14.10.4.16.2 Unsupported functions

Name	Description
UUID_SHORT ()	Provides a UUID that is unique given certain assumptions not present in TiDB TiDB #4620
MASTER_WAIT_POS ()	Relates to MySQL replication

14.10.4.17 Precision Math

The precision math support in TiDB is consistent with MySQL. For more information, see Precision Math in MySQL.

14.10.4.17.1 Numeric types

The scope of precision math for exact-value operations includes the exact-value data types (integer and DECIMAL types) and exact-value numeric literals. Approximate-value data types and numeric literals are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation (power-of-10) with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar might be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The DECIMAL data type is a fixed-point type and the calculations are exact. The FLOAT and DOUBLE data types are floating-point types and calculations are approximate.

14.10.4.17.2 DECIMAL data type characteristics

This section discusses the following topics of the characteristics of the DECIMAL data type (and its synonyms):

- Maximum number of digits
- Storage format
- Storage requirements

The declaration syntax for a DECIMAL column is `DECIMAL(M,D)`. The ranges of values for the arguments are as follows:

- M is the maximum number of digits (the precision). 1<= M <= 65.
- D is the number of digits to the right of the decimal point (the scale). 1 <= D <= 30 and D must be no larger than M.

The maximum value of 65 for M means that calculations on DECIMAL values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals.

Values for DECIMAL columns are stored using a binary format that packs 9 decimal digits into 4 bytes. The storage requirements for the integer and fractional parts of each value are determined separately. Each multiple of 9 digits requires 4 bytes, and any remaining digits left over require some fraction of 4 bytes. The storage required for remaining digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1–2	1
3–4	2
5–6	3
7–9	4

For example, a `DECIMAL(18,9)` column has 9 digits on each side of the decimal point, so the integer part and the fractional part each require 4 bytes. A `DECIMAL(20,6)` column has 14 integer digits and 6 fractional digits. The integer digits require 4 bytes for 9 of the digits and 3 bytes for the remaining 5 digits. The 6 fractional digits require 3 bytes.

DECIMAL columns do not store a leading `+` character or `-` character or leading `0` digits. If you insert `+0003.1` into a `DECIMAL(5,1)` column, it is stored as `3.1`. For negative numbers, a literal `-` character is not stored.

DECIMAL columns do not permit values larger than the range implied by the column definition. For example, a `DECIMAL(3,0)` column supports a range of `-999` to `999`. A `DECIMAL(M,D)` column permits at most `M - D` digits to the left of the decimal point.

For more information about the internal format of the DECIMAL values, see `mydecimal.go` in TiDB souce code.

14.10.4.17.3 Expression handling

For expressions with precision math, TiDB uses the exact-value numbers as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, if you add an exact data type into a column, a number is inserted with its exact value if it is within the column range. When retrieved, the value is the same as what is inserted. If strict SQL mode is not enabled, truncation for INSERT is permitted in TiDB.

How to handle a numeric expression depends on the values of the expression:

- If the expression contains any approximate values, the result is approximate. TiDB evaluates the expression using floating-point arithmetic.
- If the expression contains no approximate values are present, which means only exact values are contained, and if any exact value contains a fractional part, the expression is evaluated using DECIMAL exact arithmetic and has a precision of 65 digits.
- Otherwise, the expression contains only integer values. The expression is exact. TiDB evaluates the expression using integer arithmetic and has a precision the same as BIGINT (64 bits).

If a numeric expression contains strings, the strings are converted to double-precision floating-point values and the result of the expression is approximate.

Inserts into numeric columns are affected by the SQL mode. The following discussions mention strict mode and `ERROR_FOR_DIVISION_BY_ZERO`. To turn on all the restrictions, you can simply use the `TRADITIONAL` mode, which includes both strict mode values and `ERROR_FOR_DIVISION_BY_ZERO`:

```
SET sql_mode = 'TRADITIONAL`;
```

If a number is inserted into an exact type column (DECIMAL or integer), it is inserted with its exact value if it is within the column range. For this number:

- If the value has too many digits in the fractional part, rounding occurs and a warning is generated.
- If the value has too many digits in the integer part, it is too large and is handled as follows:
    - If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
    - If strict mode is enabled, an overflow error occurs.

To insert strings into numeric columns, TiDB handles the conversion from string to number as follows if the string has nonnumeric contents:

- In strict mode, a string (including an empty string) that does not begin with a number cannot be used as a number. An error, or a warning occurs.

- A string that begins with a number can be converted, but the trailing nonnumeric portion is truncated. In strict mode, if the truncated portion contains anything other than spaces, an error, or a warning occurs.

By default, the result of the division by 0 is NULL and no warning. By setting the SQL mode appropriately, division by 0 can be restricted. If you enable the `ERROR_FOR_DIVISION_BY_ZERO` SQL mode, TiDB handles division by 0 differently:

- In strict mode, inserts and updates are prohibited, and an error occurs.
- If it's not in the strict mode, a warning occurs.

In the following SQL statement:

```
INSERT INTO t SET i = 1/0;
```

The following results are returned in different SQL modes:

`sql_mode` Value	Result
''	No warning, no error; i is set to NULL.
strict	No warning, no error; i is set to NULL.
`ERROR_FOR_DIVISION_BY_ZERO`	Warning, no error; i is set to NULL.
strict, `ERROR_FOR_DIVISION_BY_ZERO`	Error; no row is inserted.

14.10.4.17.4 Rounding behavior

The result of the `ROUND()` function depends on whether its argument is exact or approximate:

- For exact-value numbers, the `ROUND()` function uses the "round half up" rule.
- For approximate-value numbers, the results in TiDB differs from that in MySQL:

```
TiDB > SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
|          3 |            3 |
+------------+--------------+
1 row in set (0.00 sec)
```

For inserts into a DECIMAL or integer column, the rounding uses round half away from zero.

```
TiDB > CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.01 sec)

TiDB > INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)

TiDB > SELECT d FROM t;
+------+
| d    |
+------+
|    3 |
|    3 |
+------+
2 rows in set (0.00 sec)
```

14.10.4.18 Set Operations

TiDB supports three set operations using the UNION, EXCEPT, and INTERSECT operators. The smallest unit of a set is a SELECT statement.

14.10.4.18.1 UNION operator

In mathematics, the union of two sets A and B consists of all elements that are in A or in B. For example:

```
SELECT 1 UNION SELECT 2;
+---+
| 1 |
+---+
| 2 |
| 1 |
+---+
2 rows in set (0.00 sec)
```

TiDB supports both `UNION DISTINCT` and `UNION ALL` operators. `UNION DISTINCT` removes duplicate records from the result set, while `UNION ALL` keeps all records including duplicates. `UNION DISTINCT` is used by default in TiDB.

```
CREATE TABLE t1 (a int);
CREATE TABLE t2 (a int);
INSERT INTO t1 VALUES (1),(2);
INSERT INTO t2 VALUES (1),(3);
```

Examples for `UNION DISTINCT` and `UNION ALL` queries are respectively as follows:

```
SELECT * FROM t1 UNION DISTINCT SELECT * FROM t2;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
+---+
3 rows in set (0.00 sec)

SELECT * FROM t1 UNION ALL SELECT * FROM t2;
+---+
| a |
+---+
| 1 |
| 2 |
| 1 |
| 3 |
+---+
4 rows in set (0.00 sec)
```

14.10.4.18.2 EXCEPT operator

If A and B are two sets, EXCEPT returns the difference set of A and B which consists of elements that are in A but not in B.

```
SELECT * FROM t1 EXCEPT SELECT * FROM t2;
+---+
| a |
+---+
| 2 |
+---+
1 rows in set (0.00 sec)
```

`EXCEPT ALL` operator is not yet supported.

14.10.4.18.3 INTERSECT operator

In mathematics, the intersection of two sets A and B consists of all elements that are both in A and B, and no other elements.

```
SELECT * FROM t1 INTERSECT SELECT * FROM t2;
+---+
| a |
+---+
```

```
| 1 |
+---+
1 rows in set (0.00 sec)
```

`INTERSECT ALL` operator is not yet supported. INTERSECT operator has higher precedence over EXCEPT and UNION operators.

```
SELECT * FROM t1 UNION ALL SELECT * FROM t1 INTERSECT SELECT * FROM t2;
+---+
| a |
+---+
| 1 |
| 1 |
| 2 |
+---+
3 rows in set (0.00 sec)
```

14.10.4.18.4 Parentheses

TiDB supports using parentheses to specify the precedence of set operations. Expressions in parentheses are processed first.

```
(SELECT * FROM t1 UNION ALL SELECT * FROM t1) INTERSECT SELECT * FROM t2;
+---+
| a |
+---+
| 1 |
+---+
1 rows in set (0.00 sec)
```

14.10.4.18.5 Use `ORDER BY` and `LIMIT`

TiDB supports using `ORDER BY` or `LIMIT` clause in set operations. These two clauses must be at the end of the entire statement.

```
(SELECT * FROM t1 UNION ALL SELECT * FROM t1 INTERSECT SELECT * FROM t2)
    ↪ ORDER BY a LIMIT 2;
+---+
| a |
+---+
| 1 |
| 1 |
+---+
2 rows in set (0.00 sec)
```

14.10.4.19 List of Expressions for Pushdown

When TiDB reads data from TiKV, TiDB tries to push down some expressions (including calculations of functions or operators) to be processed to TiKV. This reduces the amount of transferred data and offloads processing from a single TiDB node. This document introduces the expressions that TiDB already supports pushing down and how to prohibit specific expressions from being pushed down using blocklist.

14.10.4.19.1 Supported expressions for pushdown

Expression Type	Operations
Logical operators	AND (&&), OR (||), NOT (!)
Comparison functions and operators	<, <=, =, != (<>), >, >=, <=>, IN(), IS NULL, LIKE, IS TRUE, IS FALSE, COALESCE()
Numeric functions and operators	+, -, *, /, ABS(), CEIL(), CEILING(), FLOOR(), MOD()
Control flow functions	CASE, IF(), IFNULL()
JSON functions	JSON_TYPE(json_val), JSON_EXTRACT(json_doc, path[, path] ...), JSON_OBJECT(key, val[, key, val] ...), JSON_ARRAY([val[, val] ...]), JSON_MERGE(json_doc, json_doc[, json_doc] ...), JSON_SET(json_doc, path, val[, path, val] ...), JSON_INSERT(json_doc, path, val[, path, val] ...), JSON_REMOVE(json_doc, path[, path] ...)
Date and time functions	DATE_FORMAT(), SYSDATE()
String functions	RIGHT()

14.10.4.19.2 Blocklist specific expressions

If unexpected behavior occurs in the calculation process when pushing down the supported expressions or specific data types (**only** the ENUM type and the BIT type), you can restore the application quickly by prohibiting the pushdown of the corresponding functions, operators, or data types. Specifically, you can prohibit the functions, operators, or data types from being pushed down by adding them to the blocklist `mysql.expr_pushdown_blacklist`. For details, refer to Add to the blocklist.

The schema of `mysql.expr_pushdown_blacklist` is as follows:

```
tidb> desc mysql.expr_pushdown_blacklist;
+------------+--------------+------+------+-------------------+-------+
| Field      | Type         | Null | Key | Default           | Extra |
+------------+--------------+------+------+-------------------+-------+
| name       | char(100)    | NO   |     | NULL              |       |
| store_type | char(100)  | NO   |     | tikv,tiflash,tidb |     |
| reason     | varchar(200) | YES |    | NULL              |       |
+------------+--------------+------+------+-------------------+-------+
3 rows in set (0.00 sec)
```

Field description:

- `name`: the name of the function, operator, or data type that is prohibited from being pushed down.
- `store_type`: specifies to which storage engine the function, operator, or data type is prohibited from being pushed down. Currently, TiDB supports the three storage engines: `tikv`, `tidb`, and `tiflash`. `store_type` is case-insensitive. If a function is prohibited from being pushed down to multiple storage engines, use a comma to separate each engine.
- `reason` : The reason why the function is blocklisted.

Add to the blocklist

To add one or more functions, operators, or data types (**only** the `ENUM` type and the `BIT` type) to the blocklist, perform the following steps:

1. Insert the followings to `mysql.expr_pushdown_blacklist`:
    - the name of the function, operator, or data type to be prohibited from being pushed down
    - the storage engine to be prohibited from being pushed down
2. Execute the `admin reload expr_pushdown_blacklist;` command.

Remove from the blocklist

To remove one or more functions, operators, or data types from the blocklist, perform the following steps:

1. Delete the name of the function, operator, or data type in `mysql.expr_pushdown_blacklist` ↪ .
2. Execute the `admin reload expr_pushdown_blacklist;` command.

Blocklist usage examples

The following example demonstrates how to add the `DATE_FORMAT()` function, `>` operator, and `BIT` data type to the blocklist, then remove `>` from the blocklist.

You can see whether the blocklist takes effect by checking the results returned by `EXPLAIN` statement (See Understanding `EXPLAIN` results).

```
tidb> create table t(a int);
Query OK, 0 rows affected (0.06 sec)

tidb> explain select * from t where a < 2 and a > 2;
+--
   ↪ ----------------------+----------+-----------+--------------+----------------
   ↪
| id                    | estRows | task    | access object | operator info
   ↪                |
+--
   ↪ ----------------------+----------+-----------+--------------+----------------
   ↪
| TableReader_7         | 0.00    | root    |               | data:Selection_6
   ↪                |
|  ─Selection_6         | 0.00    | cop[tikv] |             | gt(ssb_1.t.a, 2)
   ↪ , lt(ssb_1.t.a, 2) |
|    ─TableFullScan_5   | 10000.00 | cop[tikv] | table:t   | keep order:false
   ↪ , stats:pseudo |
+--
   ↪ ----------------------+----------+-----------+--------------+----------------
   ↪
3 rows in set (0.00 sec)

tidb> insert into mysql.expr_pushdown_blacklist values('date_format()', '
   ↪ tikv',''), ('>','tikv',''), ('bit','tikv','');
Query OK, 2 rows affected (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 0

tidb> admin reload expr_pushdown_blacklist;
Query OK, 0 rows affected (0.00 sec)

tidb> explain select * from t where a < 2 and a > 2;
+--
   ↪ ----------------------+----------+-----------+--------------+----------------
   ↪
| id                    | estRows | task    | access object | operator info
   ↪                |
+--
   ↪ ----------------------+----------+-----------+--------------+----------------
```

```
    ↪
| Selection_7          | 10000.00 | root    |              | gt(ssb_1.t.a, 2),
    ↪  lt(ssb_1.t.a, 2) |
|  ─TableReader_6      | 10000.00 | root    |              | data:
    ↪ TableFullScan_5          |
|    ─TableFullScan_5  | 10000.00 | cop[tikv] | table:t  | keep order:false
    ↪ , stats:pseudo |
+--
    ↪ ---------------------+---------+----------+--------------+---------------
    ↪
3 rows in set (0.00 sec)

tidb> delete from mysql.expr_pushdown_blacklist where name = '>';
Query OK, 1 row affected (0.01 sec)

tidb> admin reload expr_pushdown_blacklist;
Query OK, 0 rows affected (0.00 sec)

tidb> explain select * from t where a < 2 and a > 2;
+--
    ↪ ------------------------+---------+----------+--------------+-------------
    ↪
| id                    | estRows | task     | access object | operator info
    ↪             |
+--
    ↪ ------------------------+---------+----------+--------------+-------------
    ↪
| Selection_8           | 0.00    | root     |              | lt(ssb_1.t.a,
    ↪ 2)           |
|  ─TableReader_7        | 0.00    | root     |              | data:
    ↪ Selection_6           |
|    ─Selection_6        | 0.00    | cop[tikv] |             | gt(ssb_1.t.a,
    ↪ 2)           |
|      ─TableFullScan_5  | 10000.00 | cop[tikv] | table:t  | keep order:
    ↪ false, stats:pseudo |
+--
    ↪ ------------------------+---------+----------+--------------+-------------
    ↪
4 rows in set (0.00 sec)
```

Note:

- `admin reload expr_pushdown_blacklist` only takes effect on the TiDB server that executes this SQL statement. To make it apply to all TiDB servers, execute the SQL statement on each TiDB server.
- The feature of blocklisting specific expressions is supported in TiDB 3.0.0 or later versions.
- TiDB 3.0.3 or earlier versions does not support adding some of the operators (such as ">", "+", "is null") to the blocklist by using their original names. You need to use their aliases (case-sensitive) instead, as shown in the following table:

Operator Name	Aliases
<	lt
>	gt
<=	le
>=	ge
=	eq
!=	ne
<>	ne
<=>	nulleq
|	bitor
&&	bitand
||	or
!	not
in	in
+	plus
-	minus
	mul
/	div
DIV	intdiv
IS NULL	isnull
IS TRUE	istrue
IS FALSE	isfalse

14.10.4.20 TiDB Specific Functions

The following functions are TiDB extensions, and are not present in MySQL:

Function name	Function description
TIDB_BOUNDED_STALENESS ()	The TIDB_BOUNDED_STALENESS function instructs TiDB to read the data as new as possible within the time range. See also: Read Historical Data Using the AS OF TIMESTAMP Clause
TIDB_DECODE_KEY (str)	The TIDB_DECODE_KEY function can be used to decode a TiDB-encoded key entry into a JSON structure containing _tidb_rowid and table_id. These encoded keys can be found in some system tables and in logging outputs.
TIDB_DECODE_PLAN (str)	The TIDB_DECODE_PLAN function can be used to decode a TiDB execution plan.
TIDB_IS_DDL_OWNER ()	The TIDB_IS_DDL_OWNER function can be used to check whether or not the TiDB instance you are connected to is the one that is the DDL Owner. The DDL Owner is the TiDB instance that is tasked with executing DDL statements on behalf of all other nodes in the cluster.
TIDB_PARSE_TSO (num)	The TIDB_PARSE_TSO function can be used to extract the physical timestamp from a TiDB TSO timestamp. See also: tidb_current_ts.
TIDB_VERSION()	The TIDB_VERSION function returns the TiDB version with additional build information.
TIDB_DECODE_SQL_DIGESTS (digests, stmtTruncateLength)	The TIDB_DECODE_SQL_DIGESTS() function is used to query the normalized SQL statements (a form without formats and arguments) corresponding to the set of SQL digests in the cluster.
VITESS_HASH(str)	The VITESS_HASH function returns the hash of a string that is compatible with Vitess' HASH function. This is intended to help the data migration from Vitess.
TIDB_SHARD()	The TIDB_SHARD function can be used to create a shard index to scatter the index hotspot. A shard index is an expression index with a TIDB_SHARD function as the prefix.

14.10.4.20.1 Examples

This section provides examples for some of the functions above.

TIDB_DECODE_KEY

In the following example, the table `t1` has a hidden `rowid` that is generated by TiDB. The `TIDB_DECODE_KEY` is used in the statement. From the result, you can see that the hidden `rowid` is decoded and output, which is a typical result for the non-clustered primary key.

```
SELECT START_KEY, TIDB_DECODE_KEY(START_KEY) FROM information_schema.
   ↪ tikv_region_status WHERE table_name='t1' AND REGION_ID=2\G
```

```
*************************** 1. row ***************************
              START_KEY: 7480000000000000
                  ↪ FF3B5F728000000000FF1DE3F10000000000FA
TIDB_DECODE_KEY(START_KEY): {"_tidb_rowid":1958897,"table_id":"59"}
1 row in set (0.00 sec)
```

In the following example, the table `t2` has a compound clustered primary key. From the JSON output, you can see a `handle` that contains the name and value for both of the columns that are part of the primary key.

```
show create table t2\G
```

```
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE `t2` (
  `id` binary(36) NOT NULL,
  `a` tinyint(3) unsigned NOT NULL,
  `v` varchar(512) DEFAULT NULL,
  PRIMARY KEY (`a`,`id`) /*T![clustered_index] CLUSTERED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
1 row in set (0.001 sec)
```

```
select * from information_schema.tikv_region_status where table_name='t2'
   ↪ limit 1\G
```

```
*************************** 1. row ***************************
               REGION_ID: 48
               START_KEY: 7480000000000000
                   ↪ FF3E5F720400000000FF0000000601633430FF3338646232FF2D64FF3531632D3131
                   ↪
                 END_KEY:
                TABLE_ID: 62
                 DB_NAME: test
              TABLE_NAME: t2
```

```
               IS_INDEX: 0
               INDEX_ID: NULL
             INDEX_NAME: NULL
          EPOCH_CONF_VER: 1
           EPOCH_VERSION: 38
           WRITTEN_BYTES: 0
              READ_BYTES: 0
        APPROXIMATE_SIZE: 136
        APPROXIMATE_KEYS: 479905
  REPLICATIONSTATUS_STATE: NULL
REPLICATIONSTATUS_STATEID: NULL
1 row in set (0.005 sec)
```

```
select tidb_decode_key('7480000000000000
    ↪ FF3E5F720400000000FF0000000601633430FF3338646232FF2D64FF3531632D3131FF65FF622D3863
    ↪ ');
```

```
+--
    ↪ ------------------------------------------------------------------------------
    ↪
| tidb_decode_key('7480000000000000
    ↪ FF3E5F720400000000FF0000000601633430FF3338646232FF2D64FF3531632D3131FF65FF622D3863
    ↪ ') |
+--
    ↪ ------------------------------------------------------------------------------
    ↪
| {"handle":{"a":"6","id":"c4038db2-d51c-11eb-8c75-80e65018a9be"},"table_id"
    ↪ :62}
    ↪
    ↪ |
+--
    ↪ ------------------------------------------------------------------------------
    ↪
1 row in set (0.001 sec)
```

TIDB_DECODE_PLAN

You can find TiDB execution plans in encoded form in the slow query log. The `TIDB_DECODE_PLAN()` function is then used to decode the encoded plans into a human-readable form.

This function is useful because a plan is captured at the time the statement is executed. Re-executing the statement in `EXPLAIN` might produce different results as data distribution and statistics evolves over time.

```
SELECT tidb_decode_plan('8
    ↪ QIYMAkzMV83CQEH8E85LjA0CWRhdGE6U2VsZWN0aW9uXzYJOTYwCXRpbWU6NzEzLjHCtXMsIGxvb3BzOjI
    ↪ ')\G
```

```
*************************** 1. row ***************************
  tidb_decode_plan('8
    ↪ QIYMAkzMV83CQEH8E85LjA0CWRhdGE6U2VsZWN0aW9uXzYJOTYwCXRpbWU6NzEzLjHCtXMsIGxvb3BzO
    ↪ : id task estRows operator info                    actRows
    ↪ execution info
    ↪
    ↪ memory    disk
   TableReader_7       root       319.04    data:Selection_6
      ↪                       960       time:713.1µs, loops:2, cop_task: {
      ↪ num: 1, max: 568.5µs, proc_keys: 0, rpc_num: 1, rpc_time: 549.1µs,
      ↪  copr_cache_hit_ratio: 0.00} 3.99 KB N/A
    └─Selection_6       cop[tikv]   319.04    lt(test.t.a, 10000)
      ↪                    960       tikv_task:{time:313.8µs, loops:960}
      ↪
      ↪ N/A       N/A
      └─TableFullScan_5  cop[tikv]   960       table:t, keep order:false,
        ↪ stats:pseudo 960     tikv_task:{time:153µs, loops:960}
        ↪
        ↪ N/A       N/A
```

TIDB_PARSE_TSO

The `TIDB_PARSE_TSO` function can be used to extract the physical timestamp from a TiDB TSO timestamp. TSO stands for Time Stamp Oracle and is a monotonically increasing timestamp given out by PD (Placement Driver) for every transaction.

A TSO is a number that consists of two parts:

- A physical timestamp
- A logical counter

```
BEGIN;
SELECT TIDB_PARSE_TSO(@@tidb_current_ts);
ROLLBACK;
```

```
+----------------------------------+
| TIDB_PARSE_TSO(@@tidb_current_ts) |
+----------------------------------+
| 2021-05-26 11:33:37.776000    |
+----------------------------------+
1 row in set (0.0012 sec)
```

Here `TIDB_PARSE_TSO` is used to extract the physical timestamp from the timestamp number that is available in the `tidb_current_ts` session variable. Because timestamps are given out per transaction, this function is running in a transaction.

TIDB_VERSION

The `TIDB_VERSION` function can be used to get the version and build details of the TiDB server that you are connected to. You can use this function when reporting issues on GitHub.

```
SELECT TIDB_VERSION()\G
```

```
*************************** 1. row ***************************
TIDB_VERSION(): Release Version: v5.1.0-alpha-13-gd5e0ed0aa-dirty
Edition: Community
Git Commit Hash: d5e0ed0aaed72d2f2dfe24e9deec31cb6cb5fdf0
Git Branch: master
UTC Build Time: 2021-05-24 14:39:20
GoVersion: go1.13
Race Enabled: false
TiKV Min Version: v3.0.0-60965b006877ca7234adaced7890d7b029ed1306
Check Table Before Drop: false
1 row in set (0.00 sec)
```

TIDB_DECODE_SQL_DIGESTS

The `TIDB_DECODE_SQL_DIGESTS()` function is used to query the normalized SQL statements (a form without formats and arguments) corresponding to the set of SQL digests in the cluster. This function accepts 1 or 2 arguments:

- `digests`: A string. This parameter is in the format of a JSON string array, and each string in the array is a SQL digest.
- `stmtTruncateLength`: An integer (optional). It is used to limit the length of each SQL statement in the returned result. If a SQL statement exceeds the specified length, the statement is truncated. `0` means that the length is unlimited.

This function returns a string, which is in the format of a JSON string array. The i-th item in the array is the normalized SQL statement corresponding to the i-th element in the `digests` parameter. If an element in the `digests` parameter is not a valid SQL digest or the system cannot find the corresponding SQL statement, the corresponding item in the returned result is `null`. If the truncation length is specified (`stmtTruncateLength > 0`), for each statement in the returned result that exceeds this length, the first `stmtTruncateLength` characters are retained and the suffix `"..."` is added at the end to indicate the truncation. If the `digests` parameter is `NULL`, the returned value of the function is `NULL`.

Note:

- Only users with the PROCESS privilege can use this function.
- When `TIDB_DECODE_SQL_DIGESTS` is executed, TiDB queries the statement corresponding to each SQL digest from the statement summary tables, so there is no guarantee that the corresponding statement can always be found for any SQL digest. Only the statements that have been executed in the cluster can be found, and whether these SQL statements can be queried or not is also affected by the related configuration of the statement summary tables. For the detailed description of the statement summary table, see Statement Summary Tables.
- This function has a high overhead. In queries with a large number of rows (for example, querying the full table of `information_schema.cluster_tidb_trx` on a large and busy cluster), using this function might cause the queries to run for too long. Use it with caution.
  - This function has a high overhead because every time it is called, it internally queries the `STATEMENTS_SUMMARY`, `STATEMENTS_SUMMARY_HISTORY`, `CLUSTER_STATEMENTS_SUMMARY`, and `CLUSTER_STATEMENTS_SUMMARY_HISTORY` tables, and the query involves the `UNION` operation. This function currently does not support vectorization, that is, when calling this function for multiple rows of data, the above query is performed separately for each row.

```
set @digests = '["
    ↪ e6f07d43b5c21db0fbb9a31feac2dc599787763393dd5acbfad80e247eb02ad5","38
    ↪ b03afa5debbdf0326a014dbe5012a62c51957f1982b3093e748460f8b00821","
    ↪ e5796985ccafe2f71126ed6c0ac939ffa015a8c0744a24b7aee6d587103fd2f7"]';

select tidb_decode_sql_digests(@digests);
```

```
+------------------------------------+
| tidb_decode_sql_digests(@digests) |
+------------------------------------+
| ["begin",null,"select * from `t`"] |
+------------------------------------+
1 row in set (0.00 sec)
```

In the above example, the parameter is a JSON array containing 3 SQL digests, and the corresponding SQL statements are the three items in the query results. But the SQL statement corresponding to the second SQL digest cannot be found from the cluster, so the second item in the result is `null`.

```
select tidb_decode_sql_digests(@digests, 10);
```

```
+----------------------------------------+
| tidb_decode_sql_digests(@digests, 10) |
+----------------------------------------+
| ["begin",null,"select * f..."]   |
+----------------------------------------+
1 row in set (0.01 sec)
```

The above call specifies the second parameter (that is, the truncation length) as 10, and the length of the third statement in the query result is greater than 10. Therefore, only the first 10 characters are retained, and `"..."` is added at the end, which indicates the truncation.

See also:

- Statement Summary Tables
- INFORMATION_SCHEMA.TIDB_TRX

TIDB_SHARD

The `TIDB_SHARD` function can be used to create a shard index to scatter the index hotspot. A shard index is an expression index prefixed with a `TIDB_SHARD` function.

- Creation:

  To create a shard index for the index field `a`, you can use `uk((tidb_shard(a)), a))`. When there is a hotspot caused by monotonically increasing or decreasing data on the index field `a` in the unique secondary index `uk((tidb_shard(a)), a))`, the index's prefix `tidb_shard(a)` can scatter the hotspot to improve the scalability of the cluster.

- Scenarios:

  - There is a write hotspot caused by monotonically increasing or decreasing keys on the unique secondary index, and the index contains integer type fields.
  - The SQL statement executes an equality query based on all fields of the secondary index, either as a separate `SELECT` or as an internal query generated by `UPDATE`, `DELETE` and so on. The equality query includes two ways: `a = 1` or `a IN (1, 2, ↪ ......)`.

- Limitations:

  - Cannot be used in inequality queries.
  - Cannot be used in queries that contain `OR` mixed with an outmost `AND` operator.
  - Cannot be used in the `GROUP BY` clause.
  - Cannot be used in the `ORDER BY` clause.
  - Cannot be used in the `ON` clause.
  - Cannot be used in the `WHERE` subquery.

- Can be used to scatter unique indexes of only the integer fields.
- Might not take effect in composite indexes.
- Cannot go through FastPlan process, which affects optimizer performance.
- Cannot be used to prepare the execution plan cache.

The following example shows how to use the `TIDB_SHARD` function.

- Use the `TIDB_SHARD` function to calculate the SHARD value.

  The following statement shows how to use the `TIDB_SHARD` function to calculate the SHARD value of `12373743746`:

```
SELECT TIDB_SHARD(12373743746);
```

- The SHARD value is:

```
+-------------------------+
| TIDB_SHARD(12373743746) |
+-------------------------+
|                     184 |
+-------------------------+
1 row in set (0.00 sec)
```

- Create a shard index using the `TIDB_SHARD` function:

```
CREATE TABLE test(id INT PRIMARY KEY CLUSTERED, a INT, b INT, UNIQUE
    ↪ KEY uk((tidb_shard(a)), a));
```

14.10.4.21 Comparisons between Functions and Syntax of Oracle and TiDB

This document describes the comparisons between functions and syntax of Oracle and TiDB. It helps you find the corresponding TiDB functions based on the Oracle functions, and understand the syntax differences between Oracle and TiDB.

> **Note:**
>
> The functions and syntax in this document are based on Oracle 12.2.0.1.0 and TiDB v5.4.0. They might be different in other versions.

14.10.4.21.1 Comparisons of functions

The following table shows the comparisons between some Oracle and TiDB functions.

Function	Oracle syntax	TiDB syntax	Note

| Cast a value as a certain type |

`TO_NUMBER(key)`

`TO_CHAR(key)`

| `CONVERT(key,dataType)` | TiDB supports casting a value as one of the following types: `BINARY`, `CHAR`, `DATE`, `DATETIME`, `TIME`, `SIGNED INTEGER`, `UNSIGNED INTEGER` and `DECIMAL`. |
| Convert a date to a string |

`TO_CHAR(SYSDATE,'yyyy-MM-dd hh24:mi:ss')`

`TO_CHAR(SYSDATE,'yyyy-MM-dd')`

|

`DATE_FORMAT(NOW(),'%Y-%m-%d %H:%i:%s')`

`DATE_FORMAT(NOW(),'%Y-%m-%d')`

| The format string of TiDB is case-sensitive. | | Convert a string to a date |

`TO_DATE('2021-05-28 17:31:37','yyyy-MM-dd hh24:mi:ss')`

`TO_DATE('2021-05-28','yyyy-MM-dd hh24:mi:ss')`

|

`STR_TO_DATE('2021-05-28 17:31:37','%Y-%m-%d %H:%i:%s')`

`STR_TO_DATE('2021-05-28','%Y-%m-%d%T')`

| The format string of TiDB is case-sensitive. | | Get the current system time in second precision | `SYSDATE` | `NOW()` | | | Get the current system time in microsecond precision | `SYSTIMESTAMP` | `CURRENT_TIMESTAMP(6)` | | | Get the number of days between two dates | `date1 - date2` | `DATEDIFF(date1, date2)` | | | Get the number of months between two dates | `MONTHS_BETWEEN(ENDDATE,SYSDATE)` | `TIMESTAMPDIFF(MONTH,SYSDATE,ENDDATE)` | The results of `MONTHS_BETWEEN()` in Oracle and `TIMESTAMPDIFF()` in TiDB are different. `TIMESTAMPDIFF()` returns an integer. Note that the parameters in the two functions are swapped. | | Add `n` days to a date | `DATEVAL + n` | `DATE_ADD(dateVal,INTERVAL n DAY)` | `n` can be a negative value.| | Add `n` months to a date | `ADD_MONTHS(dateVal,n)`| `DATE_ADD` ↪ `(dateVal,INTERVAL n MONTH)` | `n` can be a negative value. | | Get the day of a date | `TRUNC(SYSDATE)` |

`CAST(NOW()AS DATE)`

`DATE_FORMAT(NOW(),'%Y-%m-%d')`

| In TiDB, `CAST` and `DATE_FORMAT` return the same result. | | Get the month of a date | `TRUNC(SYSDATE,'mm')` | `DATE_ADD(CURDATE(),interval - day(CURDATE())+ 1 day)` | | | Truncate a value | `TRUNC(2.136)= 2` `TRUNC(2.136,2)= 2.13` | `TRUNCATE(2.136,0)= 2` ↪ `TRUNCATE(2.136,2)= 2.13` | Data precision is preserved. Truncate the corresponding

decimal places without rounding. | | Get the next value in a sequence | `sequence_name ↪ .NEXTVAL` | `NEXTVAL(sequence_name)` | | | Get a random sequence value | `SYS_GUID ↪ ()` | `UUID()` | TiDB returns a Universal Unique Identifier (UUID). | | Left join or right join | `SELECT * FROM a, b WHERE a.id = b.id(+);SELECT * FROM a, b WHERE a ↪ .id(+)= b.id;` | `SELECT * FROM a LEFT JOIN b ON a.id = b.id;SELECT * FROM a ↪ RIGHT JOIN b ON a.id = b.id;` | In a correlated query, TiDB does not support using (+) to left join or right join. You can use `LEFT JOIN` or `RIGHT JOIN` instead. | | `NVL()` | `NVL(key,val)` | `IFNULL(key,val)` | If the value of the field is `NULL`, it returns `val`; otherwise, it returns the value of the field. | | `NVL2()` | `NVL2(key, val1, val2)` | `IF(key is NULL, ↪ val1, val2)` | If the value of the field is not `NULL`, it returns `val1`; otherwise, it returns `val2`. | | `DECODE()` |

`DECODE(key,val1,val2,val3)`

`DECODE(value,if1,val1,if2,val2,...,ifn,valn,val)`

|

`IF(key=val1,val2,val3)`

`CASE WHEN value=if1 THEN val1 WHEN value=if2 THEN val2,...,WHEN value= ↪ ifn THEN valn ELSE val END`

|

If the value of the field is `val1`, then it returns `val2`; otherwise it returns `val3`.

When the value of the field satisfies condition 1 (`if1`), it returns `val1`. When it satisfies condition 2 (`if2`), it returns `val2`. When it satisfies condition 3 (`if3`), it returns `val3`.

| | Concatenate the string `a` and `b` | 'a' || 'b' | `CONCAT('a','b')` | | | Get the length of a string | `LENGTH(str)` | `CHAR_LENGTH(str)` | | | Get the substring as specified | `SUBSTR(' ↪ abcdefg',0,2)= 'ab' SUBSTR('abcdefg',1,2)= 'ab'` | `SUBSTRING('abcdefg',0,2) ↪ = ''SUBSTRING('abcdefg',1,2)= 'ab'` |

In Oracle, the starting position 0 has the same effect as 1.

In TiDB, the starting position 0 returns an empty string. If you want to get a substring from the beginning, the starting position should be 1.

| | Get the position of a substring | `INSTR('abcdefg','b',1,1)` | `INSTR('abcdefg ↪ ','b')` | Search from the first character of `'abcdefg'` and return the position of the first occurrence of `'b'`. | | Get the position of a substring | `INSTR('stst','s',1,2)` | `LENGTH(SUBSTRING_INDEX('stst','s',2))+ 1` | Search from the first character of `'stst ↪ '` and return the position of the second occurrence of `'s'`. | | Get the position of a substring | `INSTR('abcabc','b',2,1)` | `LOCATE('b','abcabc',2)` | Search from the second character of `abcabc` and return the position of the first occurrence of `b`. | | Concatenate values of a column | `LISTAGG(CONCAT(E.dimensionid,'---',E.DIMENSIONNAME),'***')within ↪ GROUP(ORDER BY DIMENSIONNAME)` | `GROUP_CONCAT(CONCAT(E.dimensionid,'---',E. ↪ DIMENSIONNAME)ORDER BY DIMENSIONNAME SEPARATOR '***')` | Concatenate values of a specified column to one row with the `***` delimiter. | | Convert an ASCII code to a character

| CHR(n) | CHAR(n) | The Tab (CHR(9)), LF (CHR(10)), and CR (CHR(13)) characters in Oracle correspond to CHAR(9), CHAR(10), and CHAR(13) in TiDB. |

14.10.4.21.2 Comparisons of syntax

This section describes some syntax differences between Oracle and TiDB.

String syntax

In Oracle, a string can only be enclosed in single quotes (''). For example 'a'.

In TiDB, a string can be enclosed in single quotes ('') or double quotes (""). For example, 'a' and "a".

Difference between NULL and an empty string

Oracle does not distinguish between NULL and an empty string '', that is, NULL is equivalent to ''.

TiDB distinguishes between NULL and an empty string ''.

Read and write to the same table in an INSERT statement

Oracle supports reading and writing to the same table in an INSERT statement. For example:

```
INSERT INTO table1 VALUES (feild1,(SELECT feild2 FROM table1 WHERE...))
```

TiDB does not support reading and writing to the same table in a INSERT statement. For example:

```
INSERT INTO table1 VALUES (feild1,(SELECT T.fields2 FROM table1 T WHERE...)
    ↪ )
```

Get the first n rows from a query

In Oracle, to get the first n rows from a query, you can use the ROWNUM <= n clause. For example ROWNUM <= 10.

In TiDB, to get the first n rows from a query, you can use the LIMIT n clause. For example LIMIT 10. The Hibernate Query Language (HQL) running SQL statements with LIMIT results in an error. You need to change the Hibernate statements to SQL statements.

Update multiple tables in an UPDATE statement

In Oracle, it is not necessary to list the specific field update relationship when updating multiple tables. For example:

```
UPDATE test1 SET(test1.name,test1.age) = (SELECT test2.name,test2.age FROM
    ↪ test2 WHERE test2.id=test1.id)
```

In TiDB, when updating multiple tables, you need to list all the specific field update relationships in SET. For example:

```
UPDATE test1,test2 SET test1.name=test2.name,test1.age=test2.age WHERE
    ↪ test1.id=test2.id
```

Derived table alias

In Oracle, when querying multiple tables, it is unnecessary to add an alias to the derived table. For example:

```
SELECT * FROM (SELECT * FROM test)
```

In TiDB, when querying multiple tables, every derived table must have its own alias. For example:

```
SELECT * FROM (SELECT * FROM test) t
```

Set operations

In Oracle, to get the rows that are in the first query result but not in the second, you can use the `MINUS` set operation. For example:

```
SELECT * FROM t1 MINUS SELECT * FROM t2
```

TiDB does not support the `MINUS` operation. You can use the `EXCEPT` set operation. For example:

```
SELECT * FROM t1 EXCEPT SELECT * FROM t2
```

Comment syntax

In Oracle, the comment syntax is `--Comment`.

In TiDB, the comment syntax is `-- Comment`. Note that there is a white space after `--` in TiDB.

Pagination

In Oracle, you can use the `OFFSET m ROWS` to skip `m` rows and use the `FETCH NEXT n ROWS ONLY`to fetch `n` rows. For example:

```
SELECT * FROM tables OFFSET 0 ROWS FETCH NEXT 2000 ROWS ONLY
```

In TiDB, you can use the `LIMIT n OFFSET m` to replace `OFFSET m ROWS FETCH NEXT n ROWS ONLY`. For example:

```
SELECT * FROM tables LIMIT 2000 OFFSET 0
```

Sorting order on `NULL` values

In Oracle, `NULL` values are sorted by the `ORDER BY` clause in the following cases:

- In the `ORDER BY column ASC` statement, `NULL` values are returned last.

- In the `ORDER BY column DESC` statement, `NULL` values are returned first.
- In the `ORDER BY column [ASC|DESC] NULLS FIRST` statement, `NULL` values are returned before non-NULL values. Non-NULL values are returned in ascending order or descending order specified in `ASC|DESC`.
- In the `ORDER BY column [ASC|DESC] NULLS LAST` statement, `NULL` values are returned after non-NULL values. Non-NULL values are returned in ascending order or descending order specified in `ASC|DESC`.

In TiDB, `NULL` values are sorted by the `ORDER BY` clause in the following cases:

- In the `ORDER BY column ASC` statement, `NULL` values are returned first.
- In the `ORDER BY column DESC` statement, `NULL` values are returned last.

The following table shows some examples of equivalent `ORDER BY` statements in Oracle and TiDB:

`ORDER BY` in Oracle	Equivalent statements in TiDB
`SELECT * FROM t1 ↪ ORDER BY name ↪ NULLS FIRST;`	`SELECT * FROM t1 ↪ ORDER BY ↪ name;`
`SELECT * FROM t1 ↪ ORDER BY name ↪ DESC NULLS LAST ↪ ;`	`SELECT * FROM t1 ↪ ORDER BY ↪ name DESC;`
`SELECT * FROM t1 ↪ ORDER BY name ↪ DESC NULLS ↪ FIRST;`	`SELECT * FROM t1 ↪ ORDER BY ↪ ISNULL(name) ↪ DESC, name ↪ DESC;`
`SELECT * FROM t1 ↪ ORDER BY name ↪ ASC NULLS LAST;`	`SELECT * FROM t1 ↪ ORDER BY ↪ ISNULL(name), ↪ name;`

14.10.5 Clustered Indexes

TiDB supports the clustered index feature since v5.0. This feature controls how data is stored in tables containing primary keys. It provides TiDB the ability to organize tables in a way that can improve the performance of certain queries.

The term *clustered* in this context refers to the *organization of how data is stored* and

not *a group of database servers working together*. Some database management systems refer to clustered indexes as *index-organized tables* (IOT).

Currently, tables containing primary keys in TiDB are divided into the following two categories:

- `NONCLUSTERED`: The primary key of the table is non-clustered index. In tables with non-clustered indexes, the keys for row data consist of internal `_tidb_rowid` implicitly assigned by TiDB. Because primary keys are essentially unique indexes, tables with non-clustered indexes need at least two key-value pairs to store a row, which are:
    - `_tidb_rowid` (key) - row data (value)
    - Primary key data (key) - `_tidb_rowid` (value)
- `CLUSTERED`: The primary key of the table is clustered index. In tables with clustered indexes, the keys for row data consist of primary key data given by the user. Therefore, tables with clustered indexes need only one key-value pair to store a row, which is:
    - Primary key data (key) - row data (value)

> **Note:**
>
> TiDB supports clustering only by a table's `PRIMARY KEY`. With clustered indexes enabled, the terms *the* `PRIMARY KEY` and *the clustered index* might be used interchangeably. `PRIMARY KEY` refers to the constraint (a logical property), and clustered index describes the physical implementation of how the data is stored.

14.10.5.1 User scenarios

Compared to tables with non-clustered indexes, tables with clustered indexes offer greater performance and throughput advantages in the following scenarios:

- When data is inserted, the clustered index reduces one write of the index data from the network.
- When a query with an equivalent condition only involves the primary key, the clustered index reduces one read of index data from the network.
- When a query with a range condition only involves the primary key, the clustered index reduces multiple reads of index data from the network.
- When a query with an equivalent or range condition only involves the primary key prefix, the clustered index reduces multiple reads of index data from the network.

On the other hand, tables with clustered indexes have certain disadvantages. See the following:

- There might be write hotspot issues when inserting a large number of primary keys with close values.
- The table data takes up more storage space if the data type of the primary key is larger than 64 bits, especially when there are multiple secondary indexes.

14.10.5.2 Usages

14.10.5.3 Create a table with clustered indexes

Since TiDB v5.0, you can add non-reserved keywords `CLUSTERED` or `NONCLUSTERED` after `PRIMARY KEY` in a `CREATE TABLE` statement to specify whether the table's primary key is a clustered index. For example:

```
CREATE TABLE t (a BIGINT PRIMARY KEY CLUSTERED, b VARCHAR(255));
CREATE TABLE t (a BIGINT PRIMARY KEY NONCLUSTERED, b VARCHAR(255));
CREATE TABLE t (a BIGINT KEY CLUSTERED, b VARCHAR(255));
CREATE TABLE t (a BIGINT KEY NONCLUSTERED, b VARCHAR(255));
CREATE TABLE t (a BIGINT, b VARCHAR(255), PRIMARY KEY(a, b) CLUSTERED);
CREATE TABLE t (a BIGINT, b VARCHAR(255), PRIMARY KEY(a, b) NONCLUSTERED);
```

Note that keywords `KEY` and `PRIMARY KEY` have the same meaning in the column definition.

You can also use the comment syntax in TiDB to specify the type of the primary key. For example:

```
CREATE TABLE t (a BIGINT PRIMARY KEY /*T![clustered_index] CLUSTERED */, b
    ↪  VARCHAR(255));
CREATE TABLE t (a BIGINT PRIMARY KEY /*T![clustered_index] NONCLUSTERED */
    ↪ , b VARCHAR(255));
CREATE TABLE t (a BIGINT, b VARCHAR(255), PRIMARY KEY(a, b) /*T![
    ↪ clustered_index] CLUSTERED */);
CREATE TABLE t (a BIGINT, b VARCHAR(255), PRIMARY KEY(a, b) /*T![
    ↪ clustered_index] NONCLUSTERED */);
```

For statements that do not explicitly specify the keyword `CLUSTERED`/`NONCLUSTERED`, the default behavior is controlled by the system variable `@@global.tidb_enable_clustered_index` ↪ . Supported values for this variable are as follows:

- `OFF` indicates that primary keys are created as non-clustered indexes by default.
- `ON` indicates that primary keys are created as clustered indexes by default.
- `INT_ONLY` indicates that the behavior is controlled by the configuration item `alter-` ↪ `primary-key`. If `alter-primary-key` is set to `true`, primary keys are created as non-clustered indexes by default. If it is set to `false`, only the primary keys which consist of an integer column are created as clustered indexes.

The default value of `@@global.tidb_enable_clustered_index` is `INT_ONLY`.

14.10.5.3.1 Add or drop clustered indexes

TiDB does not support adding or dropping clustered indexes after tables are created. Nor does it support the mutual conversion between clustered indexes and non-clustered indexes. For example:

```
ALTER TABLE t ADD PRIMARY KEY(b, a) CLUSTERED; -- Currently not supported.
ALTER TABLE t DROP PRIMARY KEY; -- If the primary key is a clustered index
    ↪ , then not supported.
ALTER TABLE t DROP INDEX `PRIMARY`; -- If the primary key is a clustered
    ↪ index, then not supported.
```

14.10.5.3.2 Add or drop non-clustered indexes

TiDB supports adding or dropping non-clustered indexes after tables are created. You can explicitly specify the keyword `NONCLUSTERED` or omit it. For example:

```
ALTER TABLE t ADD PRIMARY KEY(b, a) NONCLUSTERED;
ALTER TABLE t ADD PRIMARY KEY(b, a); -- If you omit the keyword, the
    ↪ primary key is a non-clustered index by default.
ALTER TABLE t DROP PRIMARY KEY;
ALTER TABLE t DROP INDEX `PRIMARY`;
```

14.10.5.3.3 Check whether the primary key is a clustered index

You can check whether the primary key of a table is a clustered index using one of the following methods:

- Execute the command `SHOW CREATE TABLE`.
- Execute the command `SHOW INDEX FROM`.
- Query the `TIDB_PK_TYPE` column in the system table `information_schema.tables`.

By running the command `SHOW CREATE TABLE`, you can see whether the attribute of `PRIMARY KEY` is `CLUSTERED` or `NONCLUSTERED`. For example:

```
mysql> SHOW CREATE TABLE t;
+--
    ↪ -----+-----------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+-----------------------------------------------------------------
    ↪
| t     | CREATE TABLE `t` (
```

```
  `a` bigint(20) NOT NULL,
  `b` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`a`) /*T![clustered_index] CLUSTERED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin |
+--
   ↪ -----+-------------------------------------------------------------------------
   ↪
1 row in set (0.01 sec)
```

By running the command `SHOW INDEX FROM`, you can check whether the result in the column `Clustered` shows `YES` or `NO`. For example:

```
mysql> SHOW INDEX FROM t;
+--
   ↪ -----+-----------+---------+-------------+-------------+----------+--------
   ↪
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation |
   ↪ Cardinality | Sub_part | Packed | Null | Index_type | Comment |
   ↪ Index_comment | Visible | Expression | Clustered |
+--
   ↪ -----+-----------+---------+-------------+-------------+----------+--------
   ↪
| t     |          0 | PRIMARY |           1 | a           | A         |           0
   ↪  |     NULL | NULL |      | BTREE     |         |               | YES     |
   ↪ NULL       | YES      |
+--
   ↪ -----+-----------+---------+-------------+-------------+----------+--------
   ↪
1 row in set (0.01 sec)
```

You can also query the column `TIDB_PK_TYPE` in the system table `information_schema` ↪ `.tables` to see whether the result is `CLUSTERED` or `NONCLUSTERED`. For example:

```
mysql> SELECT TIDB_PK_TYPE FROM information_schema.tables WHERE
   ↪ table_schema = 'test' AND table_name = 't';
+--------------+
| TIDB_PK_TYPE |
+--------------+
| CLUSTERED    |
+--------------+
1 row in set (0.03 sec)
```

14.10.5.4 Limitations

Currently, there are several different types of limitations for the clustered index feature. See the following:

- Situations that are not supported and not in the support plan:
    - Using clustered indexes together with the attribute SHARD_ROW_ID_BITS is not supported. Also, the attribute PRE_SPLIT_REGIONS does not take effect on tables with clustered indexes that are not AUTO_RANDOM.
    - Downgrading tables with clustered indexes is not supported. If you need to downgrade such tables, use logical backup tools to migrate data instead.
- Situations that are not supported yet but in the support plan:
    - Adding, dropping, and altering clustered indexes using `ALTER TABLE` statements are not supported.
- Limitations for specific versions:
    - In v5.0, using the clustered index feature together with TiDB Binlog is not supported. After TiDB Binlog is enabled, TiDB only allows creating a single integer column as the clustered index of a primary key. TiDB Binlog does not replicate data changes (such as insertion, deletion, and update) on existing tables with clustered indexes to the downstream. If you need to replicate tables with clustered indexes to the downstream, upgrade your cluster to v5.1 or use TiCDC for replication instead.

After TiDB Binlog is enabled, if the clustered index you create is not a single integer primary key, TiDB returns the following error:

```
mysql> CREATE TABLE t (a VARCHAR(255) PRIMARY KEY CLUSTERED);
ERROR 8200 (HY000): Cannot create clustered index table when the binlog is
    ↪ ON
```

If you use clustered indexes together with the attribute `SHARD_ROW_ID_BITS`, TiDB reports the following error:

```
mysql> CREATE TABLE t (a VARCHAR(255) PRIMARY KEY CLUSTERED)
    ↪ SHARD_ROW_ID_BITS = 3;
ERROR 8200 (HY000): Unsupported shard_row_id_bits for table with primary
    ↪ key as row id
```

14.10.5.5 Compatibility

14.10.5.5.1 Compatibility with earlier and later TiDB versions

TiDB supports upgrading tables with clustered indexes but not downgrading such tables, which means that data in tables with clustered indexes on a later TiDB version is not available on an earlier one.

The clustered index feature is partially supported in TiDB v3.0 and v4.0. It is enabled by default when the following requirements are fully met:

- The table contains a `PRIMARY KEY`.
- The `PRIMARY KEY` consists of only one column.
- The `PRIMARY KEY` is an `INTEGER`.

Since TiDB v5.0, the clustered index feature is fully supported for all types of primary keys, but the default behavior is consistent with TiDB v3.0 and v4.0. To change the default behavior, you can configure the system variable `@@tidb_enable_clustered_index` to `ON` or `OFF`. For more details, see Create a table with clustered indexes.

14.10.5.5.2 Compatibility with MySQL

TiDB specific comment syntax supports wrapping the keywords `CLUSTERED` and `NONCLUSTERED` in a comment. The result of `SHOW CREATE TABLE` also contains TiDB specific SQL comments. MySQL databases and TiDB databases of an earlier version will ignore these comments.

14.10.5.5.3 Compatibility with TiDB migration tools

The clustered index feature is only compatible with the following migration tools in v5.0 and later versions:

- Backup and restore tools: BR, Dumpling, and TiDB Lightning.
- Data migration and replication tools: DM and TiCDC.

However, you cannot convert a table with non-clustered indexes to a table with clustered indexes by backing up and restoring the table using the v5.0 BR tool, and vice versa.

14.10.5.5.4 Compatibility with other TiDB features

For a table with a combined primary key or a single non-integer primary key, if you change the primary key from a non-clustered index to a clustered index, the keys of its row data change as well. Therefore, `SPLIT TABLE BY/BETWEEN` statements that are executable in TiDB versions earlier than v5.0 are no longer workable in v5.0 and later versions of TiDB. If you want to split a table with clustered indexes using `SPLIT TABLE BY/BETWEEN`, you need to provide the value of the primary key column, instead of specifying an integer value. See the following example:

```
mysql> create table t (a int, b varchar(255), primary key(a, b) clustered);
Query OK, 0 rows affected (0.01 sec)
mysql> split table t between (0) and (1000000) regions 5;
ERROR 1105 (HY000): Split table region lower value count should be 2
mysql> split table t by (0), (50000), (100000);
ERROR 1136 (21S01): Column count doesn't match value count at row 0
mysql> split table t between (0, 'aaa') and (1000000, 'zzz') regions 5;
+--------------------+----------------------+
```

```
| TOTAL_SPLIT_REGION | SCATTER_FINISH_RATIO |
+--------------------+----------------------+
|                  4 |                    1 |
+--------------------+----------------------+
1 row in set (0.00 sec)
mysql> split table t by (0, ''), (50000, ''), (100000, '');
+--------------------+----------------------+
| TOTAL_SPLIT_REGION | SCATTER_FINISH_RATIO |
+--------------------+----------------------+
|                  3 |                    1 |
+--------------------+----------------------+
1 row in set (0.01 sec)
```

The attribute AUTO_RANDOM can only be used on clustered indexes. Otherwise, TiDB returns the following error:

```
mysql> create table t (a bigint primary key nonclustered auto_random);
ERROR 8216 (HY000): Invalid auto random: column a is not the integer
    ↪ primary key, or the primary key is nonclustered
```

14.10.6 Constraints

TiDB supports almost the same constraint as MySQL.

14.10.6.1 NOT NULL

NOT NULL constraints supported by TiDB are the same as those supported by MySQL.

For example:

```
CREATE TABLE users (
 id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 age INT NOT NULL,
 last_login TIMESTAMP
);
```

```
INSERT INTO users (id,age,last_login) VALUES (NULL,123,NOW());
```

```
Query OK, 1 row affected (0.02 sec)
```

```
INSERT INTO users (id,age,last_login) VALUES (NULL,NULL,NOW());
```

```
ERROR 1048 (23000): Column 'age' cannot be null
```

```
INSERT INTO users (id,age,last_login) VALUES (NULL,123,NULL);
```

```
Query OK, 1 row affected (0.03 sec)
```

- The first `INSERT` statement succeeds because it is possible to assign `NULL` to the `AUTO_INCREMENT` column. TiDB generates sequence numbers automatically.
- The second `INSERT` statement fails because the `age` column is defined as `NOT NULL`.
- The third `INSERT` statement succeeds because the `last_login` column is not explicitly defined as `NOT NULL`. NULL values are allowed by default.

14.10.6.2 CHECK

TiDB parses but ignores `CHECK` constraints. This is MySQL 5.7 compatible behavior.

For example:

```
DROP TABLE IF EXISTS users;
CREATE TABLE users (
 id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 username VARCHAR(60) NOT NULL,
 UNIQUE KEY (username),
 CONSTRAINT min_username_length CHECK (CHARACTER_LENGTH(username) >=4)
);
INSERT INTO users (username) VALUES ('a');
SELECT * FROM users;
```

14.10.6.3 UNIQUE KEY

Depending on the transaction mode and the value of `tidb_constraint_check_in_place` ↪ , TiDB might check `UNIQUE` constraints lazily. This helps improve performance by batching network access.

For example:

```
DROP TABLE IF EXISTS users;
CREATE TABLE users (
 id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 username VARCHAR(60) NOT NULL,
 UNIQUE KEY (username)
);
INSERT INTO users (username) VALUES ('dave'), ('sarah'), ('bill');
```

With the default of pessimistic locking:

```
BEGIN;
INSERT INTO users (username) VALUES ('jane'), ('chris'), ('bill');
```

```
ERROR 1062 (23000): Duplicate entry 'bill' for key 'username'
```

With optimistic locking and `tidb_constraint_check_in_place=0`:

```
BEGIN OPTIMISTIC;
INSERT INTO users (username) VALUES ('jane'), ('chris'), ('bill');
```

```
Query OK, 0 rows affected (0.00 sec)

Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
INSERT INTO users (username) VALUES ('steve'),('elizabeth');
```

```
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
COMMIT;
```

```
ERROR 1062 (23000): Duplicate entry 'bill' for key 'username'
```

In the optimistic example, the unique check was delayed until the transaction is committed. This resulted in a duplicate key error, because the value `bill` was already present.

You can disable this behavior by setting `tidb_constraint_check_in_place` to `1`. This variable setting does not take effect on pessimistic transactions, because in the pessimistic transaction mode the constraints are always checked when the statement is executed. When `tidb_constraint_check_in_place=1`, the unique constraint is checked when the statement is executed.

For example:

```
DROP TABLE IF EXISTS users;
CREATE TABLE users (
 id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 username VARCHAR(60) NOT NULL,
 UNIQUE KEY (username)
);
INSERT INTO users (username) VALUES ('dave'), ('sarah'), ('bill');
```

```
SET tidb_constraint_check_in_place = 1;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
BEGIN OPTIMISTIC;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
INSERT INTO users (username) VALUES ('jane'), ('chris'), ('bill');
```

```
ERROR 1062 (23000): Duplicate entry 'bill' for key 'username'
..
```

The first `INSERT` statement caused a duplicate key error. This causes additional network communication overhead and may reduce the throughput of insert operations.

14.10.6.4 PRIMARY KEY

Like MySQL, primary key constraints contain unique constraints, that is, creating a primary key constraint is equivalent to having a unique constraint. In addition, other primary key constraints of TiDB are also similar to those of MySQL.

For example:

```
CREATE TABLE t1 (a INT NOT NULL PRIMARY KEY);
```

```
Query OK, 0 rows affected (0.12 sec)
```

```
CREATE TABLE t2 (a INT NULL PRIMARY KEY);
```

```
ERROR 1171 (42000): All parts of a PRIMARY KEY must be NOT NULL; if you need
    ↪  NULL in a key, use UNIQUE instead
```

```
CREATE TABLE t3 (a INT NOT NULL PRIMARY KEY, b INT NOT NULL PRIMARY KEY);
```

```
ERROR 1068 (42000): Multiple primary key defined
```

```
CREATE TABLE t4 (a INT NOT NULL, b INT NOT NULL, PRIMARY KEY (a,b));
```

```
Query OK, 0 rows affected (0.10 sec)
```

- Table `t2` failed to be created, because column `a` is defined as the primary key and does not allow NULL values.
- Table `t3` failed to be created, because a table can only have one primary key.
- Table `t4` was created successfully, because even though there can be only one primary key, TiDB supports defining multiple columns as the composite primary key.

In addition to the rules above, TiDB currently only supports adding and deleting the primary keys of the `NONCLUSTERED` type. For example:

```
CREATE TABLE t5 (a INT NOT NULL, b INT NOT NULL, PRIMARY KEY (a,b)
    ↪ CLUSTERED);
ALTER TABLE t5 DROP PRIMARY KEY;
```

```
ERROR 8200 (HY000): Unsupported drop primary key when the table is using
    ↪ clustered index
```

```
CREATE TABLE t5 (a INT NOT NULL, b INT NOT NULL, PRIMARY KEY (a,b)
    ↪ NONCLUSTERED);
ALTER TABLE t5 DROP PRIMARY KEY;
```

```
Query OK, 0 rows affected (0.10 sec)
```

For more details about the primary key of the `CLUSTERED` type, refer to clustered index.

14.10.6.5 FOREIGN KEY

> **Note:**
>
> TiDB has limited support for foreign key constraints.

TiDB supports creating `FOREIGN KEY` constraints in DDL commands.

For example:

```
CREATE TABLE users (
 id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 doc JSON
);
CREATE TABLE orders (
 id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 user_id INT NOT NULL,
 doc JSON,
 FOREIGN KEY fk_user_id (user_id) REFERENCES users(id)
);
```

```
SELECT table_name, column_name, constraint_name, referenced_table_name,
    ↪ referenced_column_name
FROM information_schema.key_column_usage WHERE table_name IN ('users', '
    ↪ orders');
```

```
+------------+-------------+-----------------+-----------------------+-----------------
    ↪
| table_name | column_name | constraint_name | referenced_table_name |
    ↪ referenced_column_name |
```

```
+------------+------------+-----------------+----------------------+-----------------
    ↪
| users      | id         | PRIMARY         | NULL                 | NULL
    ↪                 |
| orders     | id         | PRIMARY         | NULL                 | NULL
    ↪                 |
| orders     | user_id    | fk_user_id      | users                | id
    ↪                   |
+------------+------------+-----------------+----------------------+-----------------
    ↪
3 rows in set (0.00 sec)
```

TiDB also supports the syntax to `DROP FOREIGN KEY` and `ADD FOREIGN KEY` via the `ALTER TABLE` command.

```
ALTER TABLE orders DROP FOREIGN KEY fk_user_id;
ALTER TABLE orders ADD FOREIGN KEY fk_user_id (user_id) REFERENCES users(id
    ↪ );
```

14.10.6.5.1 Notes

- TiDB supports foreign keys to avoid errors caused by this syntax when you migrate data from other databases to TiDB.

  However, TiDB does not perform constraint checking on foreign keys in DML statements. For example, even if there is no record with id=123 in the users table, the following transactions can be submitted successfully.

  ```
  START TRANSACTION;
  INSERT INTO orders (user_id, doc) VALUES (123, NULL);
  COMMIT;
  ```

14.10.7 Generated Columns

> **Warning:**
>
> This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

This document introduces the concept and usage of generated columns.

14.10.7.1 Basic concepts

Unlike general columns, the value of the generated column is calculated by the expression in the column definition. When inserting or updating a generated column, you cannot assign a value, but only use `DEFAULT`.

There are two kinds of generated columns: virtual and stored. A virtual generated column occupies no storage and is computed when it is read. A stored generated column is computed when it is written (inserted or updated) and occupies storage. Compared with the virtual generated columns, the stored generated columns have better read performance, but take up more disk space.

You can create an index on a generated column whether it is virtual or stored.

14.10.7.2 Usage

One of the main usage of generated columns is to extract data from the JSON data type and indexing the data.

In both MySQL 5.7 and TiDB, columns of type JSON cannot be indexed directly. That is, the following table schema is **not supported**:

```
CREATE TABLE person (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address_info JSON,
    KEY (address_info)
);
```

To index a JSON column, you must extract it as a generated column first.

Using the `city` field in `address_info` as an example, you can create a virtual generated column and add an index for it:

```
CREATE TABLE person (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address_info JSON,
    city VARCHAR(64) AS (JSON_UNQUOTE(JSON_EXTRACT(address_info, '$.city')))
        ↪ , -- virtual generated column
    -- city VARCHAR(64) AS (JSON_UNQUOTE(JSON_EXTRACT(address_info, '$.city
        ↪ '))) VIRTUAL, -- virtual generated column
    -- city VARCHAR(64) AS (JSON_UNQUOTE(JSON_EXTRACT(address_info, '$.city
        ↪ '))) STORED, -- stored generated column
    KEY (city)
);
```

In this table, the `city` column is a **virtual generated column** and has an index. The following query can use the index to speed up the execution:

```
SELECT name, id FROM person WHERE city = 'Beijing';
```

```
EXPLAIN SELECT name, id FROM person WHERE city = 'Beijing';
```

```
+---------------------------------+---------+-----------+-----------------------------
    ↪
| id                              | estRows | task      | access object
    ↪              | operator info                                          |
+---------------------------------+---------+-----------+-----------------------------
    ↪
| Projection_4                    | 10.00   | root      |
    ↪                                | test.person.name, test.person.id
    ↪                          |
|  ─IndexLookUp_10                | 10.00   | root      |
    ↪                                |
    ↪                                                          |
|    ─IndexRangeScan_8(Build)     | 10.00   | cop[tikv] | table:person, index:
    ↪ city(city) | range:["Beijing","Beijing"], keep order:false, stats:
    ↪ pseudo |
|    ─TableRowIDScan_9(Probe)     | 10.00   | cop[tikv] | table:person
    ↪              | keep order:false, stats:pseudo                         |
+---------------------------------+---------+-----------+-----------------------------
    ↪
```

From the query execution plan, it can be seen that the `city` index is used to read the `HANDLE` of the row that meets the condition `city ='Beijing'`, and then it uses this `HANDLE` to read the data of the row.

If no data exists at path `$.city`, `JSON_EXTRACT` returns `NULL`. If you want to enforce a constraint that `city` must be `NOT NULL`, you can define the virtual generated column as follows:

```
CREATE TABLE person (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address_info JSON,
    city VARCHAR(64) AS (JSON_UNQUOTE(JSON_EXTRACT(address_info, '$.city')))
        ↪  NOT NULL,
    KEY (city)
);
```

14.10.7.3 Validation of generated columns

Both `INSERT` and `UPDATE` statements check virtual column definitions. Rows that do not pass validation return errors:

```
mysql> INSERT INTO person (name, address_info) VALUES ('Morgan',
    ↪ JSON_OBJECT('Country', 'Canada'));
ERROR 1048 (23000): Column 'city' cannot be null
```

14.10.7.4 Generated columns index replacement rule

When an expression in a query is equivalent to a generated column with an index, TiDB replaces the expression with the corresponding generated column, so that the optimizer can take that index into account during execution plan construction.

For example, the following example creates a generated column for the expression `a+1` and adds an index:

```
create table t(a int);
desc select a+1 from t where a+1=3;
+--
    ↪ ------------------------+---------+-----------+--------------+-------------
    ↪
| id                        | estRows | task      | access object | operator info
    ↪             |
+--
    ↪ ------------------------+---------+-----------+--------------+-------------
    ↪
| Projection_4              | 8000.00 | root      |               | plus(test.t.a,
    ↪ 1)->Column#3 |
| ─TableReader_7            | 8000.00 | root      |               | data:
    ↪ Selection_6             |
|   ─Selection_6            | 8000.00 | cop[tikv] |               | eq(plus(test.t
    ↪ .a, 1), 3)   |
|     ─TableFullScan_5      | 10000.00 | cop[tikv] | table:t       | keep order:
    ↪ false, stats:pseudo |
+--
    ↪ ------------------------+---------+-----------+--------------+-------------
    ↪
4 rows in set (0.00 sec)

alter table t add column b bigint as (a+1) virtual;
alter table t add index idx_b(b);
desc select a+1 from t where a+1=3;
+--
    ↪ ---------------------+---------+-----------+-----------------------+--------
    ↪
| id                     | estRows | task      | access object         | operator
    ↪ info                          |
```

```
+--
   ↪ --------------------+---------+-----------+-------------------------+--------
   ↪
| IndexReader_6       | 10.00  | root      |                         | index:
   ↪ IndexRangeScan_5                 |
|  ─IndexRangeScan_5  | 10.00  | cop[tikv] | table:t, index:idx_b(b) |
   ↪ range:[3,3], keep order:false, stats:pseudo |
+--
   ↪ --------------------+---------+-----------+-------------------------+--------
   ↪
2 rows in set (0.01 sec)
```

Note:

Only when the expression type and the generated column type are strictly equal, the replacement is performed.

In the above example, the column type of `a` is int and the column type of `a+1` is bigint. If the type of the generated column is set to int, the replacement will not occur.

For type conversion rules, see Type Conversion of Expression Evaluation.

14.10.7.5 Limitations

The current limitations of JSON and generated columns are as follows:

- You cannot add a stored generated column through `ALTER TABLE`.
- You can neither convert a stored generated column to a normal column through the `ALTER TABLE` statement nor convert a normal column to a stored generated column.
- You cannot modify the expression of a stored generated column through the `ALTER` ↪ `TABLE` statement.
- Not all JSON functions are supported;
- Currently, the generated column index replacement rule is valid only when the generated column is a virtual generated column. It is not valid on the stored generated column, but the index can still be used by directly using the generated column itself.

— title: SQL Mode summary: Learn SQL mode. —

14.10.8 SQL Mode

TiDB servers operate in different SQL modes and apply these modes differently for different clients. SQL mode defines the SQL syntaxes that TiDB supports and the type of

data validation check to perform, as described below:

After TiDB is started, modify `SET [ SESSION | GLOBAL ] sql_mode='modes'` to set SQL mode.

Ensure that you have `SUPER` privilege when setting SQL mode at `GLOBAL` level, and your setting at this level only affects the connections established afterwards. Changes to SQL mode at `SESSION` level only affect the current client.

`modes` is a set of modes separated by commas (','). You can use the `SELECT @@sql_mode` statement to check the current SQL mode. The default value of SQL mode: `ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION`.

14.10.8.1 Important `sql_mode` values

- `ANSI`: This mode complies with standard SQL. In this mode, data is checked. If data does not comply with the defined type or length, the data type is adjusted or trimmed and a `warning` is returned.
- `STRICT_TRANS_TABLES`: Strict mode, where data is strictly checked. If any data is incorrect, it cannot be inserted into a table and an error is returned.
- `TRADITIONAL`: In this mode, TiDB behaves like a "traditional" SQL database system. An error instead of a warning is returned when any incorrect value is inserted into a column. Then, the `INSERT` or `UPDATE` statement is immediately stopped.

14.10.8.2 SQL mode table

Name	Description
`PIPES_AS_CONCAT`	Treat “||” as a string concatenation operator (`+`) (the same as `CONCAT()`), not as an `OR` (full support)

Name	Description
ANSI_QUOTES	Treats " as an identifier. If ANSI_QUOTES is enabled, only single quotes are treated as string literals, and double quotes are treated as identifiers. Therefore, double quotes cannot be used to quote strings. (full support)

Name	Description
IGNORE_SPACE	If this mode is enabled, the system ignores space. For example: “user” and “user” are the same. (full support)

Name	Description
ONLY_FULL_GROUP_BY	A SQL statement is invalid if it refers to a column in SELECT, HAVING, or ORDER BY that is neither aggregated nor included in the GROUP BY clause. This is because displaying such a column in

Name	Description
`NO_UNSIGNED_SUBTRACTION` ↪	Do not mark the result as `UNSIGNED` ↪ if an operand has no symbol in subtraction. (full support)

Name	Description
NO_DIR_IN_CREATE	Ignore all INDEX DIRECTORY and DATA DIRECTORY directives when a table is created. This option is only useful for secondary replication servers (syntax support only)

Name	Description
NO_KEY_OPTIONS	When you use the SHOW CREATE TABLE statement, MySQL-specific syntaxes such as ENGINE are not exported. Consider this option when migrating across DB types using mysqldump. (syntax support only)

Name	Description
NO_FIELD_OPTIONS	Value you use the SHOW CREATE TABLE statement, MySQL-specific syntaxes such as ENGINE are not exported. Consider this option when migrating across DB types using mysqldump. (syntax support only)

Name	Description
NO_TABLE_OPTIONS	When you use the SHOW CREATE TABLE statement, MySQL-specific syntaxes such as ENGINE are not exported. Consider this option when migrating across DB types using mysqldump. (syntax support only)

Name	Description
NO_AUTO_VALUE_ON_ZERO	If this mode is enabled, when the value passed in the AUTO_INCREMENT column is 0 or a specific value, the system directly writes this value to this column. When NULL is passed, the system automatically generates the next serial

Name	Description
NO_BACKSLASH_ESCAPES ↪	In this mode is enabled, the \ backslash symbol only stands for itself. (full support)

Name	Description
STRICT_TRANS_TABLES	Enables the strict mode for the transaction storage engine and rolls back the entire statement after an illegal value is inserted. (full support)

Name	Description
STRICT_ALL_TABLES	All transactional tables, rolls back the entire transaction statement after an illegal value is inserted. (full support)

Name	Description
NO_ZERO_IN_DATE ↪	Strict mode, where dates with a month or day part of 0 are not accepted. If you use the IGNORE ↪ option, TiDB inserts '0000-00-00' for a similar date. In non-strict mode, this date is accepted but a warning is returned. (full sup-

Name	Description
NO_ZERO_DATE	Does not use '0000-00-00' as a legal date in strict mode. You can still insert a zero date with the IGNORE option. In non-strict mode, this date is accepted but a warning is returned. (full support)

Name	Description
ALLOW_INVALID_DATES	In this mode, the system does not check the validity of all dates. It only checks the month value ranging from 1 to 12 and the date value ranging from 1 to 31. The mode only applies to DATE and DATATIME columns. All TIMESTAMP

Name	Description
`ERROR_FOR_DIVISION_BY_ZERO`	If this mode is enabled, the system returns an error when handling division by `0` in data-change operations (`INSERT` or `UPDATE`). If this mode is not enabled, the system returns a warning and `NULL` is used

Name	Description
NO_AUTO_CREATE_USER ↪	Prevent GRANT ↪ from automatically creating new users, except for the specified password (full support)

Name	Description
HIGH_NOT_PRECEDENCE	The precedence of the NOT operator is such that expressions such as NOT a BETWEEN b AND c are parsed as NOT (a BETWEEN b AND c). In some older versions of MySQL,

Name	Description
NO_ENGINE_SUBSTITUTION	Prevents the automatic replacement of storage engines if the required storage engine is disabled or not compiled. (syntax support only)

Name	Description
PAD_CHAR_TO_FULL_LENGTH ↪	If this mode is enabled, the system does not trim the trailing spaces for CHAR types. (syntax support only. This mode has been deprecated in MySQL 8.0.)

Name	Description
REAL_AS_FLOAT	Treat REAL as the synonym of FLOAT , not the synonym of DOUBLE (full support)
POSTGRESQL	Equivalent to PIPES_AS_CONCAT , ANSI_QUOTES , IGNORE_SPACE , NO_KEY_OPTIONS , NO_TABLE_OPTIONS , NO_FIELD_OPTIONS (syntax support only)

Name	Description
MSSQL	Equivalent to PIPES_AS_CONCAT , ANSI_QUOTES , IGNORE_SPACE , NO_KEY_OPTIONS , NO_TABLE_OPTIONS , NO_FIELD_OPTIONS (syntax support only)
DB2	Equivalent to PIPES_AS_CONCAT , ANSI_QUOTES , IGNORE_SPACE , NO_KEY_OPTIONS , NO_TABLE_OPTIONS , NO_FIELD_OPTIONS (syntax support only)

Name	Description
MAXDB	Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER (full support)
MySQL323	Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE (syntax support only)
MYSQL40	Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE (syntax support only)

Name	Description
`ANSI`	Equivalent to `REAL_AS_FLOAT` ↪ , `PIPES_AS_CONCAT` ↪ , `ANSI_QUOTES` ↪ , `IGNORE_SPACE` ↪ (syntax support only)
`TRADITIONAL` ↪	Equivalent to `STRICT_TRANS_TABLES` ↪ , `STRICT_ALL_TABLES` ↪ , `NO_ZERO_IN_DATE` ↪ , `NO_ZERO_DATE` ↪ , `ERROR_FOR_DIVISION_BY_ZERO` ↪ , `NO_AUTO_CREATE_USER` ↪ (syntax support only)

Name	Description
ORACLE	Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER (syntax support only)

14.10.9 Table Attributes

The Table Attributes feature is introduced in TiDB v5.3.0. Using this feature, you can add specific attributes to a table or partition to perform the operations corresponding to the attributes. For example, you can use table attributes to control the Region merge behavior.

Currently, TiDB only supports adding the `merge_option` attribute to a table or partition to control the Region merge behavior. The `merge_option` attribute is only part of how to deal with hotspots. For more information, refer to Troubleshoot Hotspot Issues.

Note:

- When you use TiDB Binlog or TiCDC to perform replication or use BR to perform incremental backup, the replication or backup operations skip the DDL statement that sets table attributes. To use table attributes in the downstream or in the backup cluster, you need to manually execute the DDL statement in the downstream or in the backup cluster.

14.10.9.1 Usage

The table attribute is in the form of `key=value`. Multiple attributes are separated by commas. In the following examples, `t` is the name of the table to be modified, `p` is the name of the partition to be modified. Items in `[]` are optional.

- Set attributes for a table or partition:

```
ALTER TABLE t [PARTITION p] ATTRIBUTES [=] 'key=value[, key1=value1...]
    ↪ ';
```

- Reset attributes for a table or partition:

```
ALTER TABLE t [PARTITION p] ATTRIBUTES [=] DEFAULT;
```

- See the attributes of all tables and partitions:

```
SELECT * FROM information_schema.attributes;
```

- See the attribute configured to a table or partition:

```
SELECT * FROM information_schema.attributes WHERE id='schema/t[/p]';
```

- See all tables and partitions that have a specific attribute:

```
SELECT * FROM information_schema.attributes WHERE attributes LIKE '%
    ↪ key%';
```

14.10.9.2 Attribute override rules

The attribute configured to a table takes effect on all partitions of the table. However, there is one exception: If the table and partition are configured with the same attribute but different attribute values, the partition attribute overrides the table attribute. For example, suppose that the table `t` is configured with the `key=value` attribute, and the partition `p` is configured with `key=value1`.

```
ALTER TABLE t ATTRIBUTES[=]'key=value';
ALTER TABLE t PARTITION p ATTRIBUTES[=]'key=value1';
```

In this case, `key=value1` is the attribute that actually takes effect on the `p1` partition.

14.10.9.3 Control the Region merge behavior using table attributes

14.10.9.3.1 User scenarios

If there is a write hotspot or read hotspot, you can use table attributes to control the Region merge behavior. You can first add the `merge_option` attribute to a table or partition and then set its value to `deny`. The two scenarios are as follows.

Write hotspot on a newly created table or partition

If a hotspot issue occurs when data is written to a newly created table or partition, you usually need to split and scatter Regions. However, if there is a certain time interval between the split/scatter operation and writes, these operations do not truly avoid the write hotspot. This is because the split operation performed when the table or partition is created produces empty Regions, so if the time interval exists, the split Regions might be merged. To handle this case, you can add the `merge_option` attribute to the table or partition and set the attribute value to `deny`.

Periodic read hotspot in read-only scenarios

Suppose that in a read-only scenario, you try to reduce the periodic read hotspot that occurs on a table or partition by manually splitting Regions, and you do not want the manually split Regions to be merged after the hotspot issue is resolved. In this case, you can add the `merge_option` attribute to the table or partition and set its value to `deny`.

14.10.9.3.2 Usage

- Prevent the Regions of a table from merging:

```
ALTER TABLE t ATTRIBUTES 'merge_option=deny';
```

- Allow merging Regions belonging to a table:

```
ALTER TABLE t ATTRIBUTES 'merge_option=allow';
```

- Reset attributes of a table:

```
ALTER TABLE t ATTRIBUTES DEFAULT;
```

- Prevent the Regions of a partition from merging:

```
ALTER TABLE t PARTITION p ATTRIBUTES 'merge_option=deny';
```

- Allow merging Regions belonging to a partition:

```
ALTER TABLE t PARTITION p ATTRIBUTES 'merge_option=allow';
```

- See all tables or partitions configured the `merge_option` attribution:

```
SELECT * FROM information_schema.attributes WHERE attributes LIKE '%
    ↪ merge_option%';
```

14.10.9.3.3 Attribute override rules

```
ALTER TABLE t ATTRIBUTES 'merge_option=deny';
ALTER TABLE t PARTITION p ATTRIBUTES 'merge_option=allow';
```

When the above two attributes are configured at the same time, the Regions belonging to the partition `p` can actually be merged. When the attribute of the partition is reset, the partition `p` inherits the attribute from the table `t`, and the Regions cannot be merged.

Note:

- For a table with partitions, if the `merge_option` attribute is configured at the table level only, even if `merge_option=allow`, the table is still split into multiple Regions by default according to the actual number of partitions. To merge all Regions, you need to reset the attribute of the table.
- When using the `merge_option` attribute, you need to pay attention to the PD configuration parameter `split-merge-interval`. Suppose that the `merge_option` attribute is not configured. In this case, if Regions meet conditions, Regions can be merged after the interval specified by `split-merge-interval`. If the `merge_option` attribute is configured, PD decides whether to merge Regions after the specified interval according to the `merge_option` configuration.

Note:

- For a table with partitions, if the `merge_option` attribute is configured at the table level only, even if `merge_option=allow`, the table is still split into multiple Regions by default according to the actual number of partitions. To merge all Regions, you need to reset the attribute of the table.
- Suppose that the `merge_option` attribute is not configured. In this case, if Regions meet conditions, Regions can be merged after one hour. If the `merge_option` attribute is configured, PD decides whether to merge Regions after one hour according to the `merge_option` configuration.

14.10.10 Transactions

14.10.10.1 Transactions

TiDB supports distributed transactions using either pessimistic or optimistic transaction mode. Starting from TiDB 3.0.8, TiDB uses the pessimistic transaction mode by default.

This document introduces commonly used transaction-related statements, explicit and implicit transactions, isolation levels, lazy check for constraints, and transaction sizes.

The common variables include `autocommit`, `tidb_disable_txn_auto_retry`, `tidb_retry_limit`, and `tidb_txn_mode`.

> **Note:**
>
> The `tidb_disable_txn_auto_retry` and `tidb_retry_limit` variables only apply to optimistic transactions, not to pessimistic transactions.

14.10.10.1.1 Common statements

Starting a transaction

The statements `BEGIN` and `START TRANSACTION` can be used interchangeably to explicitly start a new transaction.

Syntax:

```
BEGIN;
```

```
START TRANSACTION;
```

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

```
START TRANSACTION WITH CAUSAL CONSISTENCY ONLY;
```

If the current session is in the process of a transaction when one of these statements is executed, TiDB automatically commits the current transaction before starting a new transaction.

> **Note:**
>
> Unlike MySQL, TiDB takes a snapshot of the current database after executing the statements above. MySQL's `BEGIN` and `START TRANSACTION` take a snapshot after executing the first `SELECT` statement (not `SELECT FOR UPDATE` ↪) that reads data from InnoDB after a transaction is started. `START` ↪ `TRANSACTION WITH CONSISTENT SNAPSHOT` takes a snapshot during the execution of the statement. As a result, `BEGIN`, `START TRANSACTION`, and `START TRANSACTION WITH CONSISTENT SNAPSHOT` are equivalent to `START` ↪ `TRANSACTION WITH CONSISTENT SNAPSHOT` in MySQL.

Committing a transaction

The statement `COMMIT` instructs TiDB to apply all changes made in the current transaction.

Syntax:

```
COMMIT;
```

> **Tip:**
>
> Make sure that your application correctly handles that a `COMMIT` statement could return an error before enabling optimistic transactions. If you are unsure of how your application handles this, it is recommended to instead use the default of pessimistic transactions.

Rolling back a transaction

The statement `ROLLBACK` rolls back and cancels all changes in the current transaction.

Syntax:

```
ROLLBACK;
```

Transactions are also automatically rolled back if the client connection is aborted or closed.

14.10.10.1.2 Autocommit

As required for MySQL compatibility, TiDB will by default *autocommit* statements immediately following their execution.

For example:

```
mysql> CREATE TABLE t1 (
    -> id INT NOT NULL PRIMARY KEY auto_increment,
    -> pad1 VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
| 1            |
+--------------+
```

```
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES (1, 'test');
Query OK, 1 row affected (0.02 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM t1;
+----+------+
| id | pad1 |
+----+------+
|  1 | test |
+----+------+
1 row in set (0.00 sec)
```

In the above example, the `ROLLBACK` statement has no effect. This is because the `INSERT` statement is executed in autocommit. That is, it was the equivalent of the following single-statement transaction:

```
START TRANSACTION;
INSERT INTO t1 VALUES (1, 'test');
COMMIT;
```

Autocommit will not apply if a transaction has been explicitly started. In the following example, the `ROLLBACK` statement successfully reverts the `INSERT` statement:

```
mysql> CREATE TABLE t2 (
    -> id INT NOT NULL PRIMARY KEY auto_increment,
    -> pad1 VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
| 1            |
+--------------+
1 row in set (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t2 VALUES (1, 'test');
Query OK, 1 row affected (0.02 sec)
```

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

The autocommit system variable can be changed on either a global or session basis.

For example:

```
SET autocommit = 0;
```

```
SET GLOBAL autocommit = 0;
```

14.10.10.1.3 Explicit and implicit transaction

Note:

Some statements are committed implicitly. For example, executing `[BEGIN|START TRANSACTION]` implicitly commits the last transaction and starts a new transaction. This behavior is required for MySQL compatibility. Refer to implicit commit for more details.

TiDB supports explicit transactions (use `[BEGIN|START TRANSACTION]` and `COMMIT` to define the start and end of the transaction) and implicit transactions (`SET autocommit = 1`).

If you set the value of `autocommit` to `1` and start a new transaction through the `[BEGIN|START TRANSACTION]` statement, the autocommit is disabled before `COMMIT` or `ROLLBACK` which makes the transaction becomes explicit.

For DDL statements, the transaction is committed automatically and does not support rollback. If you run the DDL statement while the current session is in the process of a transaction, the DDL statement is executed after the current transaction is committed.

14.10.10.1.4 Lazy check of constraints

By default, optimistic transactions will not check the primary key or unique constraints when a DML statement is executed. These checks are instead performed on transaction `COMMIT`.

For example:

```
CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY);
INSERT INTO t1 VALUES (1);
BEGIN OPTIMISTIC;
INSERT INTO t1 VALUES (1); -- MySQL returns an error; TiDB returns success
    ↪ .
INSERT INTO t1 VALUES (2);
COMMIT; -- It is successfully committed in MySQL; TiDB returns an error
    ↪ and the transaction rolls back.
SELECT * FROM t1; -- MySQL returns 1 2; TiDB returns 1.
```

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY);
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO t1 VALUES (1);
Query OK, 1 row affected (0.02 sec)

mysql> BEGIN OPTIMISTIC;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (1); -- MySQL returns an error; TiDB returns
    ↪ success.
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT; -- It is successfully committed in MySQL; TiDB returns an
    ↪ error and the transaction rolls back.
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> SELECT * FROM t1; -- MySQL returns 1 2; TiDB returns 1.
+----+
| id |
+----+
|  1 |
+----+
1 row in set (0.01 sec)
```

The lazy check optimization improves performance by batching constraint checks and reducing network communication. The behavior can be disabled by setting `tidb_constraint_check_in_place=TRUE`.

Note:

- This optimization only applies to optimistic transactions.
- This optimization does not take effect for `INSERT IGNORE` and `INSERT ON DUPLICATE KEY UPDATE`, but only for normal `INSERT` statements.

14.10.10.1.5 Statement rollback

TiDB supports atomic rollback after statement execution failure. If a statement results in an error, the changes it made will not take effect. The transaction will remain open, and additional changes can be made before issuing a `COMMIT` or `ROLLBACK` statement.

```
CREATE TABLE test (id INT NOT NULL PRIMARY KEY);
BEGIN;
INSERT INTO test VALUES (1);
INSERT INTO tset VALUES (2); -- Statement does not take effect because "
    ↪ test" is misspelled as "tset".
INSERT INTO test VALUES (1),(2); -- Entire statement does not take effect
    ↪ because it violates a PRIMARY KEY constraint
INSERT INTO test VALUES (3);
COMMIT;
SELECT * FROM test;
```

```
mysql> CREATE TABLE test (id INT NOT NULL PRIMARY KEY);
Query OK, 0 rows affected (0.09 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO test VALUES (1);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO tset VALUES (2); -- Statement does not take effect
    ↪ because "test" is misspelled as "tset".
ERROR 1146 (42S02): Table 'test.tset' doesn't exist
mysql> INSERT INTO test VALUES (1),(2); -- Entire statement does not take
    ↪ effect because it violates a PRIMARY KEY constraint
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> INSERT INTO test VALUES (3);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM test;
```

```
+----+
| id |
+----+
|  1 |
|  3 |
+----+
2 rows in set (0.00 sec)
```

In the above example, the transaction remains open after the failed `INSERT` statements. The final insert statement is then successful and changes are committed.

14.10.10.1.6 Transaction size limit

Due to the limitations of the underlying storage engine, TiDB requires a single row to be no more than 6 MB. All columns of a row are converted to bytes according to their data types and summed up to estimate the size of a single row.

TiDB supports both optimistic and pessimistic transactions, and optimistic transactions are the basis for pessimistic transactions. Because optimistic transactions first cache the changes in private memory, TiDB limits the size of a single transaction.

By default, TiDB sets the total size of a single transaction to no more than 100 MB. You can modify this default value via `txn-total-size-limit` in the configuration file. The maximum value of `txn-total-size-limit` is 1 TB. The individual transaction size limit also depends on the size of remaining memory available in the server. This is because when a transaction is executed, the memory usage of the TiDB process is scaled up comparing with the transaction size, up to two to three times or more of the transaction size.

TiDB previously limited the total number of key-value pairs for a single transaction to 300,000. This restriction was removed in TiDB v4.0.

Note:

Usually, TiDB Binlog is enabled to replicate data to the downstream. In some scenarios, message middleware such as Kafka is used to consume binlogs that are replicated to the downstream.

Taking Kafka as an example, the upper limit of Kafka's single message processing capability is 1 GB. Therefore, when `txn-total-size-limit` is set to more than 1 GB, it might happen that the transaction is successfully executed in TiDB, but the downstream Kafka reports an error. To avoid this situation, you need to decide the actual value of `txn-total-size-limit` according to the limit of the end consumer. For example, if Kafka is used downstream, `txn-total-size-limit` must not exceed 1 GB.

14.10.10.1.7 Causal consistency

> **Note:**
>
> Transactions with causal consistency take effect only when the async commit and one-phase commit features are enabled. For details of the two features, see `tidb_enable_async_commit` and `tidb_enable_1pc`.

TiDB supports enabling causal consistency for transactions. Transactions with causal consistency, when committed, do not need to get timestamp from PD and have lower commit latency. The syntax to enable causal consistency is as follows:

```
START TRANSACTION WITH CAUSAL CONSISTENCY ONLY;
```

By default, TiDB guarantees linear consistency. In the case of linear consistency, if transaction 2 is committed after transaction 1 is committed, logically, transaction 2 should occur after transaction 1. Causal consistency is weaker than linear consistency. In the case of causal consistency, the commit order and occurrence order of two transactions can be guaranteed consistent only when the data locked or written by transaction 1 and transaction 2 have an intersection, which means that the two transactions have a causal relationship known to the database. Currently, TiDB does not support passing in external causal relationship.

Two transactions with causal consistency enabled have the following characteristics:

- Transactions with potential causal relationship have the consistent logical order and physical commit order
- Transactions with no causal relationship do not guarantee consistent logical order and physical commit order
- Reads without lock do not create causal relationship

Transactions with potential causal relationship have the consistent logical order and physical commit order

Assume that both transaction 1 and transaction 2 adopt causal consistency and have the following statements executed:

Transaction 1	Transaction 2
START TRANSACTION WITH CAUSAL CONSISTENCY ONLY	START TRANSACTION WITH CAUSAL CONSISTENCY ONLY
x = SELECT v FROM t WHERE id = 1 FOR UPDATE	
UPDATE t set v = $(x + 1) WHERE id = 2	
COMMIT	
	UPDATE t SET v = 2 WHERE id = 1
	COMMIT

In the example above, transaction 1 locks the `id = 1` record and transaction 2 modifies the `id = 1` record. Therefore, transaction 1 and transaction 2 have a potential causal relationship. Even with the causal consistency enabled, as long as transaction 2 is committed after transaction 1 is successfully committed, logically, transaction 2 must occur after transaction 1. Therefore, it is impossible that a transaction reads transaction 2's modification on the `id = 1` record without reading transaction 1's modification on the `id = 2` record.

Transactions with no causal relationship do not guarantee consistent logical order and physical commit order

Assume that the initial values of `id = 1` and `id = 2` are both `0`. Assume that both transaction 1 and transaction 2 adopt causal consistency and have the following statements executed:

Transaction 1	Transaction 2	Transaction 3
START TRANSACTION WITH CAUSAL CONSISTENCY ONLY	START TRANSACTION WITH CAUSAL CONSISTENCY ONLY	
UPDATE t set v = 3 WHERE id = 2		
	UPDATE t SET v = 2 WHERE id = 1	
		BEGIN
COMMIT		
	COMMIT	
		SELECT v FROM t WHERE id IN (1, 2)

In the example above, transaction 1 does not read the `id = 1` record, so transaction 1 and transaction 2 have no causal relationship known to the database. With causal consistency enabled for the transactions, even if transaction 2 is committed after transaction 1 is committed in terms of physical time order, TiDB does not guarantee that transaction 2 logically occurs after transaction 1.

If transaction 3 begins before transaction 1 is committed, and if transaction 3 reads the `id = 1` and `id = 2` records after transaction 2 is committed, transaction 3 might read the

value of `id = 1` to be `2` but the value of `id = 2` to be `0`.

Reads without lock do not create causal relationship

Assume that both transaction 1 and transaction 2 adopt causal consistency and have the following statements executed:

Transaction 1	Transaction 2
START TRANSACTION WITH CAUSAL CONSISTENCY ONLY	START TRANSACTION WITH CAUSAL CONSISTENCY ONLY
	UPDATE t SET v = 2 WHERE id = 1
SELECT v FROM t WHERE id = 1	
UPDATE t set v = 3 WHERE id = 2	
	COMMIT
COMMIT	

In the example above, reads without lock do not create causal relationship. Transaction 1 and transaction 2 have created write skew. In this case, it would have been unreasonable if the two transactions still had causal relationship. Therefore, the two transactions with causal consistency enabled have no definite logical order.

14.10.10.2 TiDB Transaction Isolation Levels

Transaction isolation is one of the foundations of database transaction processing. Isolation is one of the four key properties of a transaction (commonly referred as ACID).

The SQL-92 standard defines four levels of transaction isolation: Read Uncommitted, Read Committed, Repeatable Read, and Serializable. See the following table for details:

Isolation Level	Dirty Write	Dirty Read	Fuzzy Read	Phantom
READ UNCOMMITTED	Not Possible	Possible	Possible	Possible
READ COMMITTED	Not Possible	Not possible	Possible	Possible
REPEATABLE READ	Not Possible	Not possible	Not possible	Possible
SERIALIZABLE	Not Possible	Not possible	Not possible	Not possible

TiDB implements Snapshot Isolation (SI) consistency, which it advertises as `REPEATABLE-READ` for compatibility with MySQL. This differs from the ANSI Repeatable Read isolation level and the MySQL Repeatable Read level.

> **Note:**
>
> Starting from TiDB v3.0, the automatic retry of transactions is disabled by default. It is not recommended to enable the automatic retry because it might **break the transaction isolation level**. Refer to Transaction Retry for details.
>
> Starting from TiDB v3.0.8, newly created TiDB clusters use the pessimistic transaction mode by default. The current read (`for update` read) is **non-repeatable read**. Refer to pessimistic transaction mode for details.

14.10.10.2.1 Repeatable Read isolation level

The Repeatable Read isolation level only sees data committed before the transaction begins, and it never sees either uncommitted data or changes committed during transaction execution by concurrent transactions. However, the transaction statement does see the effects of previous updates executed within its own transaction, even though they are not yet committed.

For transactions running on different nodes, the start and commit order depends on the order that the timestamp is obtained from PD.

Transactions of the Repeatable Read isolation level cannot concurrently update a same row. When committing, if the transaction finds that the row has been updated by another transaction after it starts, then the transaction rolls back. For example:

```
create table t1(id int);
insert into t1 values(0);
```

```
start transaction;          |            start transaction;
select * from t1;           |            select * from t1;
update t1 set id=id+1;      |            update t1 set id=id+1; -- In
   ↪ pessimistic transactions, the `update` statement executed later
   ↪ waits for the lock until the transaction holding the lock commits or
   ↪  rolls back and releases the row lock.
commit;                     |
                            |            commit; -- The transaction commit
                               ↪  fails and rolls back. Pessimistic
                               ↪ transactions can commit successfully.
```

Difference between TiDB and ANSI Repeatable Read

The Repeatable Read isolation level in TiDB differs from ANSI Repeatable Read isolation level, though they sharing the same name. According to the standard described in the A Critique of ANSI SQL Isolation Levels paper, TiDB implements the Snapshot Isolation level. This isolation level does not allow strict phantoms (A3) but allows broad phantoms (P3) and write skews. In contrast, the ANSI Repeatable Read isolation level allows phantom reads but does not allow write skews.

Difference between TiDB and MySQL Repeatable Read

The Repeatable Read isolation level in TiDB differs from that in MySQL. The MySQL Repeatable Read isolation level does not check whether the current version is visible when updating, which means it can continue to update even if the row has been updated after the transaction starts. In contrast, if the row has been updated after the transaction starts, the TiDB optimistic transaction is rolled back and retried. Transaction retries in TiDB's optimistic concurrency control might fail, leading to a final failure of the transaction, while in TiDB's pessimistic concurrency control and MySQL, the updating transaction can be successful.

14.10.10.2.2 Read Committed isolation level

Starting from TiDB v4.0.0-beta, TiDB supports the Read Committed isolation level.

For historical reasons, the Read Committed isolation level of current mainstream databases is essentially the Consistent Read isolation level defined by Oracle. In order to adapt to this situation, the Read Committed isolation level in TiDB pessimistic transactions is also a consistent read behavior in essence.

> **Note:**
>
> The Read Committed isolation level only takes effect in the pessimistic transaction mode. In the optimistic transaction mode, setting the transaction

isolation level to `Read Committed` does not take effect and transactions still use the Repeatable Read isolation level.

Starting from v6.0.0, TiDB supports using the `tidb_rc_read_check_ts` system variable to optimize the timestamp acquisition in scenarios where read-write conflicts are rare. After enabling this variable, TiDB will try to use the previous valid timestamp to read data when `SELECT` is executed. The initial value of this variable is the `start_ts` of the transaction.

- If TiDB does not encounter any data update during the read process, it returns the result to the client and the `SELECT` statement is successfully executed.
- If TiDB encounters data update during the read process:
  - If TiDB has not yet sent the result to the client, TiDB tries to acquire a new timestamp and retry this statement.
  - If TiDB has already sent partial data to the client, TiDB reports an error to the client. The amount of data sent to the client each time is controlled by `tidb_init_chunk_size` and `tidb_max_chunk_size`.

In scenarios where the `READ-COMMITTED` isolation level is used, the `SELECT` statements are many, and read-write conflicts are rare, enabling this variable can avoid the latency and cost of getting the global timestamp.

14.10.10.2.3 Difference between TiDB and MySQL Read Committed

The MySQL Read Committed isolation level is in line with the Consistent Read features in most cases. There are also exceptions, such as semi-consistent read. This special behavior is not supported in TiDB.

14.10.10.3 TiDB Optimistic Transaction Model

With optimistic transactions, conflicting changes are detected as part of a transaction commit. This helps improve the performance when concurrent transactions are infrequently modifying the same rows, because the process of acquiring row locks can be skipped. In the case that concurrent transactions frequently modify the same rows (a conflict), optimistic transactions may perform worse than Pessimistic Transactions.

Before enabling optimistic transactions, make sure that your application correctly handles that a `COMMIT` statement could return errors. If you are unsure of how your application handles this, it is recommended to instead use Pessimistic Transactions.

Note:

Starting from v3.0.8, TiDB uses the pessimistic transaction mode by default. However, this does not affect your existing cluster if you upgrade it from v3.0.7 or earlier to v3.0.8 or later. In other words, **only newly created clusters default to using the pessimistic transaction mode**.

14.10.10.3.1 Principles of optimistic transactions

To support distributed transactions, TiDB adopts two-phase commit (2PC) in optimistic transactions. The procedure is as follows:

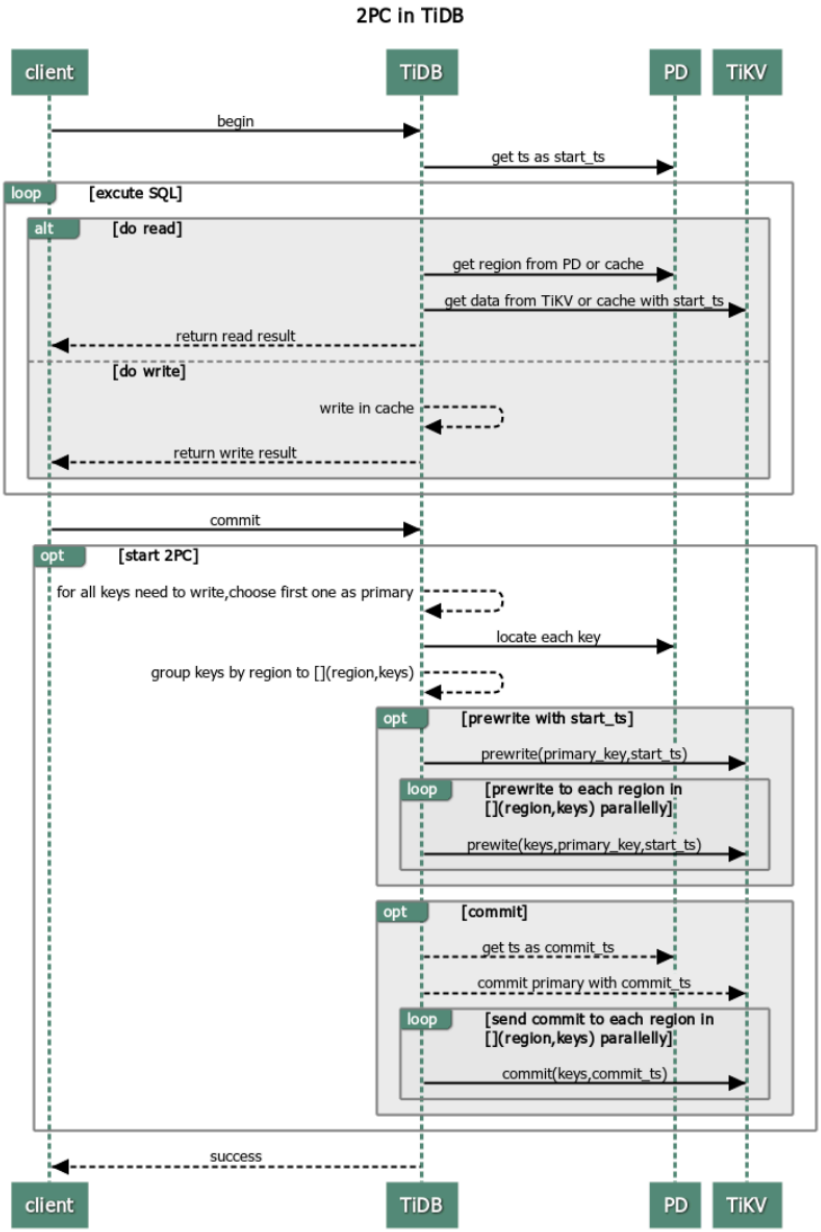


Figure 487: 2PC in TiDB

1. The client begins a transaction.

   TiDB gets a timestamp (monotonically increasing in time and globally unique) from PD as the unique transaction ID of the current transaction, which is called `start_ts`. TiDB implements multi-version concurrency control, so `start_ts` also serves as the version of the database snapshot obtained by this transaction. This means that the transaction can only read the data from the database at `start_ts`.

2. The client issues a read request.

1. TiDB receives routing information (how data is distributed among TiKV nodes) from PD.
2. TiDB receives the data of the `start_ts` version from TiKV.

3. The client issues a write request.

   TiDB checks whether the written data satisfies constraints (to ensure the data types are correct, the NOT NULL constraint is met). **Valid data is stored in the private memory of this transaction in TiDB**.

4. The client issues a commit request.

5. TiDB begins 2PC, and persists data in store while guaranteeing the atomicity of transactions.

   1. TiDB selects a Primary Key from the data to be written.
   2. TiDB receives the information of Region distribution from PD, and groups all keys by Region accordingly.
   3. TiDB sends prewrite requests to all TiKV nodes involved. Then, TiKV checks whether there are conflict or expired versions. Valid data is locked.
   4. TiDB receives all responses in the prewrite phase and the prewrite is successful.
   5. TiDB receives a commit version number from PD and marks it as `commit_ts`.
   6. TiDB initiates the second commit to the TiKV node where Primary Key is located. TiKV checks the data, and cleans the locks left in the prewrite phase.
   7. TiDB receives the message that reports the second phase is successfully finished.

6. TiDB returns a message to inform the client that the transaction is successfully committed.

7. TiDB asynchronously cleans the locks left in this transaction.

14.10.10.3.2 Advantages and disadvantages

From the process of transactions in TiDB above, it is clear that TiDB transactions have the following advantages:

- Simple to understand
- Implement cross-node transaction based on single-row transaction
- Decentralized lock management

However, TiDB transactions also have the following disadvantages:

- Transaction latency due to 2PC
- In need of a centralized timestamp allocation service
- OOM (out of memory) when extensive data is written in the memory

14.10.10.3.3 Transaction retries

In the optimistic transaction model, transactions might fail to be committed because of write–write conflict in heavy contention scenarios. TiDB uses optimistic concurrency control by default, whereas MySQL applies pessimistic concurrency control. This means that MySQL adds locks during the execution of write-type SQL statements, and its Repeatable Read isolation level allows for current reads, so commits generally do not encounter exceptions. To lower the difficulty of adapting applications, TiDB provides an internal retry mechanism.

Automatic retry

If a write-write conflict occurs during the transaction commit, TiDB automatically retries the SQL statement that includes write operations. You can enable the automatic retry by setting `tidb_disable_txn_auto_retry` to `OFF` and set the retry limit by configuring `tidb_retry_limit`:

```
#### Whether to disable automatic retry. ("on" by default)
tidb_disable_txn_auto_retry = OFF
#### Set the maximum number of the retires. ("10" by default)
#### When "tidb_retry_limit = 0", automatic retry is completely disabled.
tidb_retry_limit = 10
```

You can enable the automatic retry in either session level or global level:

1. Session level:

```
SET tidb_disable_txn_auto_retry = OFF;
```

```
SET tidb_retry_limit = 10;
```

2. Global level:

```
SET GLOBAL tidb_disable_txn_auto_retry = OFF;
```

```
SET GLOBAL tidb_retry_limit = 10;
```

Note:

The `tidb_retry_limit` variable decides the maximum number of retries. When this variable is set to `0`, none of the transactions automatically retries, including the implicit single statement transactions that are automatically committed. This is the way to completely disable the automatic retry mechanism in TiDB. After the automatic retry is disabled, all conflicting transactions report failures (including the `try again later` message) to the application layer in the fastest way.

Limits of retry

By default, TiDB will not retry transactions because this might lead to lost updates and damaged `REPEATABLE READ` isolation.

The reason can be observed from the procedures of retry:

1. Allocate a new timestamp and mark it as `start_ts`.
2. Retry the SQL statements that contain write operations.
3. Implement the two-phase commit.

In Step 2, TiDB only retries SQL statements that contain write operations. However, during retrying, TiDB receives a new version number to mark the beginning of the transaction. This means that TiDB retries SQL statements with the data in the new `start_ts` version. In this case, if the transaction updates data using other query results, the results might be inconsistent because the `REPEATABLE READ` isolation is violated.

If your application can tolerate lost updates, and does not require `REPEATABLE READ` isolation consistency, you can enable this feature by setting `tidb_disable_txn_auto_retry = OFF`.

14.10.10.3.4 Conflict detection

As a distributed database, TiDB performs in-memory conflict detection in the TiKV layer, mainly in the prewrite phase. TiDB instances are stateless and unaware of each other, which means they cannot know whether their writes result in conflicts across the cluster. Therefore, conflict detection is performed in the TiKV layer.

The configuration is as follows:

```
#### Controls the number of slots. ("2048000" by default)
scheduler-concurrency = 2048000
```

In addition, TiKV supports monitoring the time spent on waiting latches in the scheduler.

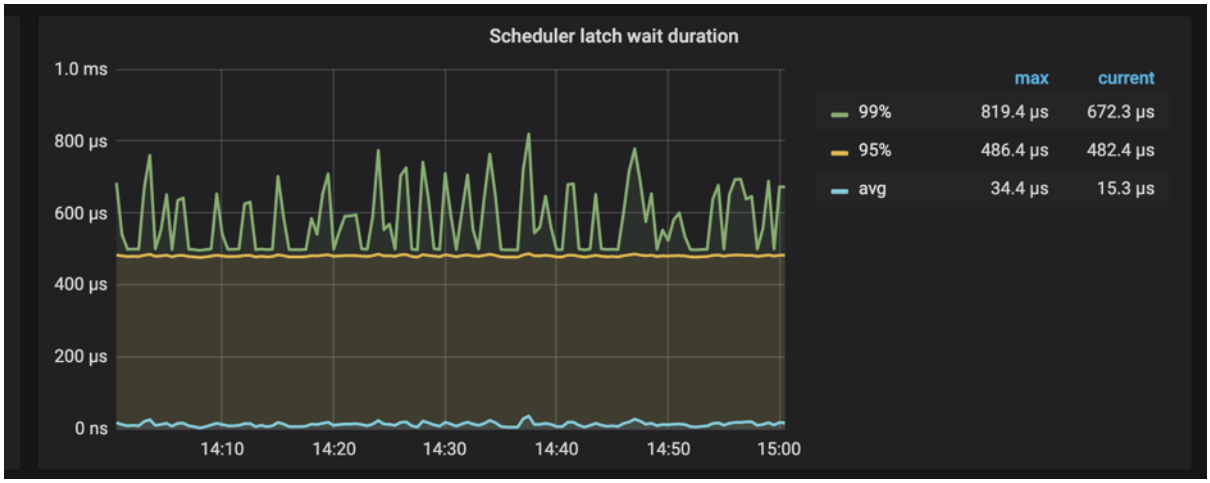


Figure 488: Scheduler latch wait duration

When `Scheduler latch wait duration` is high and there are no slow writes, it can be safely concluded that there are many write conflicts at this time.

14.10.10.4 TiDB Pessimistic Transaction Mode

To make the usage of TiDB closer to traditional databases and reduce the cost of migration, starting from v3.0, TiDB supports the pessimistic transaction mode on top of the optimistic transaction model. This document describes the features of the TiDB pessimistic transaction mode.

> **Note:**
>
> Starting from v3.0.8, newly created TiDB clusters use the pessimistic transaction mode by default. However, this does not affect your existing cluster if you upgrade it from v3.0.7 or earlier to v3.0.8 or later. In other words, **only newly created clusters default to using the pessimistic transaction mode**.

14.10.10.4.1 Switch transaction mode

You can set the transaction mode by configuring the `tidb_txn_mode` system variable. The following command sets all explicit transactions (that is, non-autocommit transactions) executed by newly created sessions in the cluster to the pessimistic transaction mode:

```
SET GLOBAL tidb_txn_mode = 'pessimistic';
```

You can also explicitly enable the pessimistic transaction mode by executing the following SQL statements:

```
BEGIN PESSIMISTIC;
```

```
BEGIN /*T! PESSIMISTIC */;
```

The `BEGIN PESSIMISTIC;` and `BEGIN OPTIMISTIC;` statements take precedence over the `tidb_txn_mode` system variable. Transactions started with these two statements ignore the system variable and support using both the pessimistic and optimistic transaction modes.

14.10.10.4.2 Behaviors

Pessimistic transactions in TiDB behave similarly to those in MySQL. See the minor differences in Difference with MySQL InnoDB.

- For pessimistic transactions, TiDB introduces snapshot read and current read.

- Snapshot read: it is an unlocked read that reads a version committed before the transaction starts. The read in the `SELECT` statement is a snapshot read.
- Current read: it is a locked read that reads the latest committed version. The read in the `UPDATE`, `DELETE`, `INSERT`, or `SELECT FOR UPDATE` statement is a current read.

The following examples provide a detailed description of snapshot read and current read.

Session 1	Session 2	Session 3
CREATE TABLE t (a INT);		
INSERT INTO T VALUES(1);		
BEGIN PESSIMISTIC;		
UPDATE t SET a = a + 1;		
	BEGIN PESSIMISTIC;	

Session 1	Session 2	Session 3
	SELECT * FROM t; – Use the snapshot read to read the version committed before the current transaction starts. The result returns a=1.	
		BEGIN PESSIMISTIC;

Session 1	Session 2	Session 3
		SELECT * FROM t FOR UPDATE; – Use the current read. Wait for the lock.

Session 1	Session 2	Session 3
COMMIT; – Release the lock. The SELECT FOR UPDATE operation of session 3 obtains the lock and TiDB uses the current read to read the latest committed version. The result returns a=2.		

Session 1	Session 2	Session 3
	SELECT * FROM t; – Use the snapshot read to read the version committed before the current transaction starts. The result returns a=1.	

- When you execute `UPDATE`, `DELETE` or `INSERT` statements, the **latest** committed data is read, data is modified, and a pessimistic lock is applied on the modified rows.
- For `SELECT FOR UPDATE` statements, a pessimistic lock is applied on the latest version of the committed data, instead of on the modified rows.
- Locks will be released when the transaction is committed or rolled back. Other transactions attempting to modify the data are blocked and have to wait for the lock to be released. Transactions attempting to *read* the data are not blocked, because TiDB uses multi-version concurrency control (MVCC).
- If several transactions are trying to acquire each other's respective locks, a deadlock will occur. This is automatically detected, and one of the transactions will randomly

be terminated with a MySQL-compatible error code `1213` returned.

- Transactions will wait up to `innodb_lock_wait_timeout` seconds (default: 50) to acquire new locks. When this timeout is reached, a MySQL-compatible error code `1205` is returned. If multiple transactions are waiting for the same lock, the order of priority is approximately based on the `start ts` of the transaction.
- TiDB supports both the optimistic transaction mode and pessimistic transaction mode in the same cluster. You can specify either mode for transaction execution.
- TiDB supports the `FOR UPDATE NOWAIT` syntax and does not block and wait for locks to be released. Instead, a MySQL-compatible error code `3572` is returned.
- If the `Point Get` and `Batch Point Get` operators do not read data, they still lock the given primary key or unique key, which blocks other transactions from locking or writing data to the same primary key or unique key.
- TiDB supports the `FOR UPDATE OF TABLES` syntax. For a statement that joins multiple tables, TiDB only applies pessimistic locks on the rows that are associated with the tables in `OF TABLES`.

14.10.10.4.3 Difference with MySQL InnoDB

1. When TiDB executes DML or `SELECT FOR UPDATE` statements that use range in the WHERE clause, concurrent DML statements within the range are not blocked.

   For example:

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY,
 pad1 VARCHAR(100)
);
INSERT INTO t1 (id) VALUES (1),(5),(10);
```

```
BEGIN /*T! PESSIMISTIC */;
SELECT * FROM t1 WHERE id BETWEEN 1 AND 10 FOR UPDATE;
```

```
BEGIN /*T! PESSIMISTIC */;
INSERT INTO t1 (id) VALUES (6); -- blocks only in MySQL
UPDATE t1 SET pad1='new value' WHERE id = 5; -- blocks waiting in both
    ↪ MySQL and TiDB
```

   This behavior is because TiDB does not currently support *gap locking*.

2. TiDB does not support `SELECT LOCK IN SHARE MODE`.

   When `SELECT LOCK IN SHARE MODE` is executed, it has the same effect as that without the lock, so the read or write operation of other transactions is not blocked.

3. DDL may result in failure of the pessimistic transaction commit.

   When DDL is executed in MySQL, it might be blocked by the transaction that is being executed. However, in this scenario, the DDL operation is not blocked in TiDB, which leads to failure of the pessimistic transaction commit: `ERROR 1105 (HY000) ↪ : Information schema is changed. [try again later]`. TiDB executes the `TRUNCATE TABLE` statement during the transaction execution, which might result in the `table doesn't exist` error.

4. After executing `START TRANSACTION WITH CONSISTENT SNAPSHOT`, MySQL can still read the tables that are created later in other transactions, while TiDB cannot.

5. The autocommit transactions prefer the optimistic locking.

   When using the pessimistic model, the autocommit transactions first try to commit the statement using the optimistic model that has less overhead. If a write conflict occurs, the pessimistic model is used for transaction retry. Therefore, if `tidb_retry_limit` is set to `0`, the autocommit transaction still reports the `Write Conflict` error when a write conflict occurs.

   The autocommit `SELECT FOR UPDATE` statement does not wait for lock.

6. The data read by `EMBEDDED SELECT` in the statement is not locked.

7. Open transactions in TiDB do not block garbage collection (GC). By default, this limits the maximum execution time of pessimistic transactions to 1 hour. You can modify this limit by editing `max-txn-ttl` under `[performance]` in the TiDB configuration file.

14.10.10.4.4 Isolation level

TiDB supports the following two isolation levels in the pessimistic transaction mode:

- Repeatable Read by default, which is the same as MySQL.

> **Note:**
>
> In this isolation level, DML operations are performed based on the latest committed data. The behavior is the same as MySQL, but differs from the optimistic transaction mode in TiDB. See Difference between TiDB and MySQL Repeatable Read.

- Read Committed. You can set this isolation level using the `SET TRANSACTION` statement.

14.10.10.4.5 Pessimistic transaction commit process

In the transaction commit process, pessimistic transactions and optimistic transactions have the same logic. Both transactions adopt the two-phase commit (2PC) mode. The important adaptation of pessimistic transactions is DML execution.

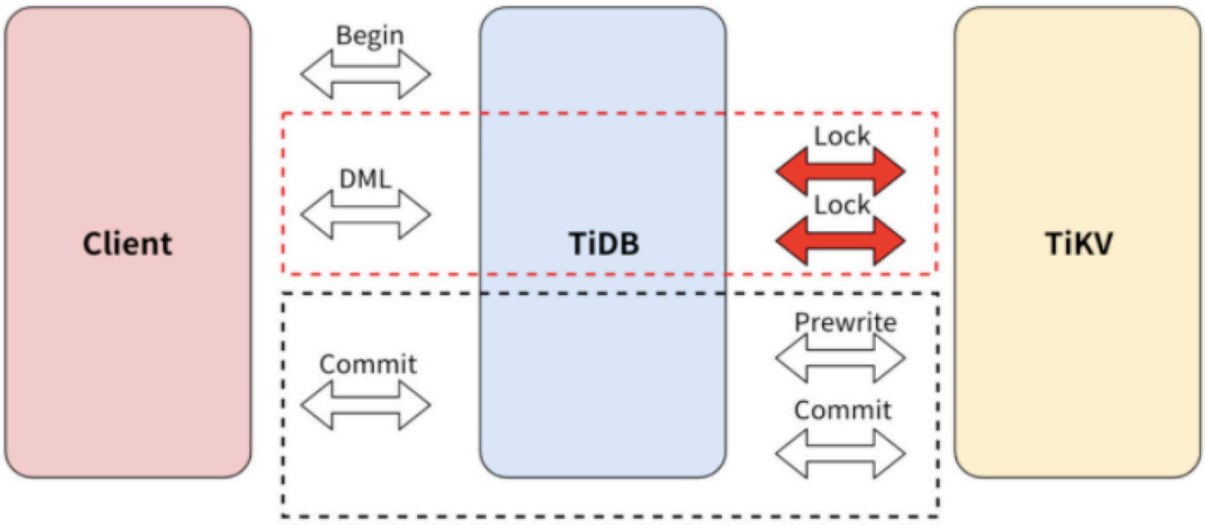


Figure 489: TiDB pessimistic transaction commit process

The pessimistic transaction adds an `Acquire Pessimistic Lock` phase before 2PC. This phase includes the following steps:

1. (Same as the optimistic transaction mode) TiDB receives the `begin` request from the client, and the current timestamp is this transaction's start_ts.
2. When the TiDB server receives a writing request from the client, the TiDB server initiates a pessimistic lock request to the TiKV server, and the lock is persisted to the TiKV server.
3. (Same as the optimistic transaction mode) When the client sends the commit request, TiDB starts to perform the two-phase commit similar to the optimistic transaction mode.

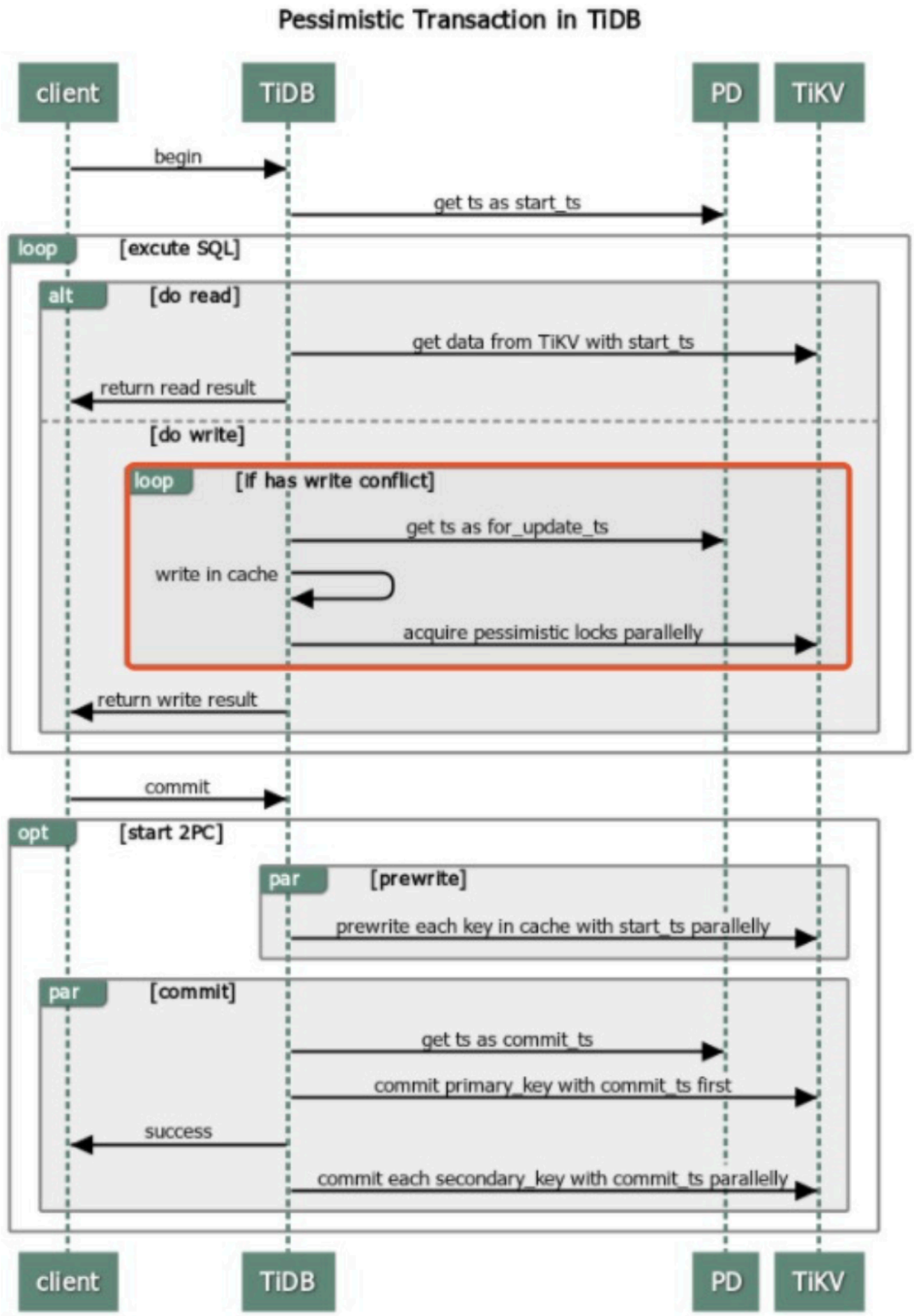


Figure 490: Pessimistic transactions in TiDB

14.10.10.4.6 Pipelined locking process

Adding a pessimistic lock requires writing data into TiKV. The response of successfully adding a lock can only be returned to TiDB after commit and apply through Raft. Therefore, compared with optimistic transactions, the pessimistic transaction mode inevitably has higher latency.

To reduce the overhead of locking, TiKV implements the pipelined locking process: when the data meets the requirements for locking, TiKV immediately notifies TiDB to execute subsequent requests and writes into the pessimistic lock asynchronously. This process reduces most latency and significantly improves the performance of pessimistic transactions. However, when network partition occurs in TiKV or a TiKV node is down, the asynchronous write into the pessimistic lock might fail and affect the following aspects:

- Other transactions that modify the same data cannot be blocked. If the application logic relies on locking or lock waiting mechanisms, the correctness of the application logic is affected.
- There is a low probability that the transaction commit fails, but it does not affect the correctness of the transactions.

If the application logic relies on the locking or lock waiting mechanisms, or if you want to guarantee as much as possible the success rate of transaction commits even in the case of TiKV cluster anomalies, you should disable the pipelined locking feature.

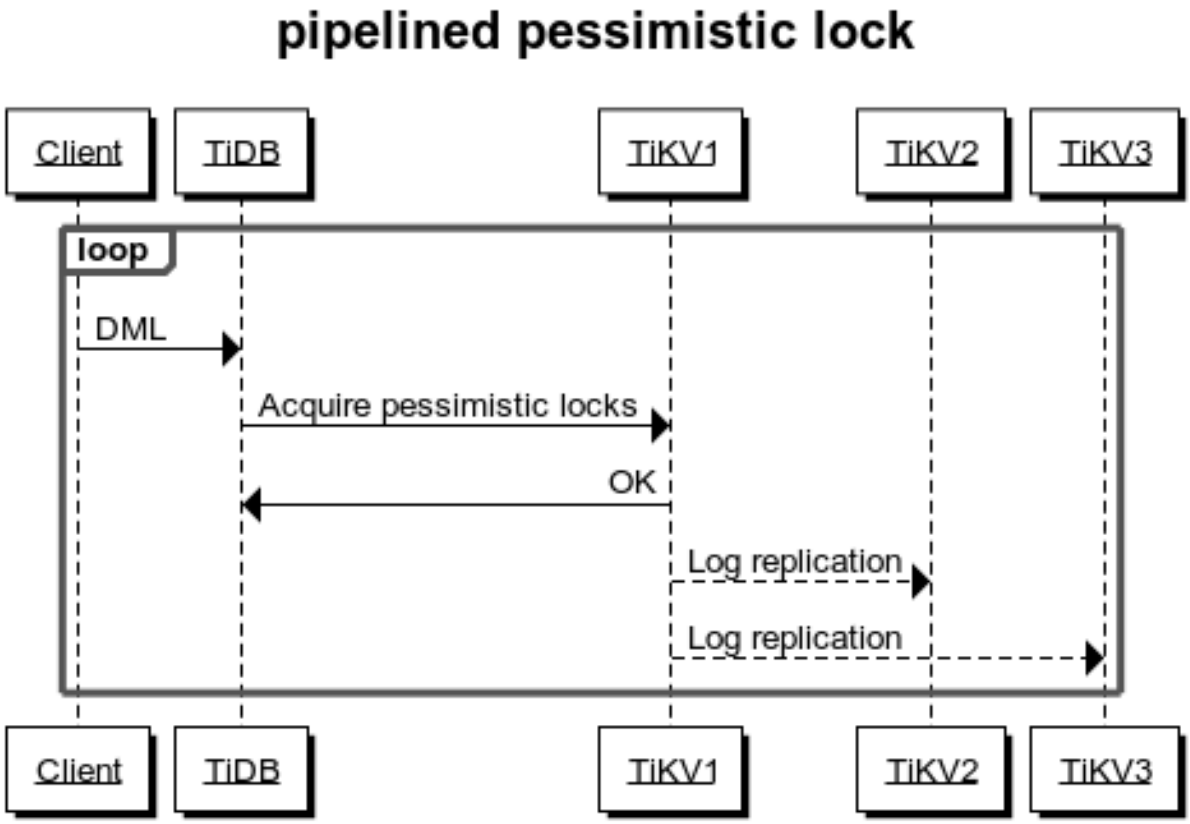


Figure 491: Pipelined pessimistic lock

This feature is enabled by default. To disable it, modify the TiKV configuration:

```
[pessimistic-txn]
pipelined = false
```

If the TiKV cluster is v4.0.9 or later, you can also dynamically disable this feature by modifying TiKV configuration dynamically:

```
set config tikv pessimistic-txn.pipelined='false';
```

14.10.10.4.7 In-memory pessimistic lock

In v6.0.0, TiKV introduces the feature of in-memory pessimistic lock. When this feature is enabled, pessimistic locks are usually stored in the memory of the Region leader only, and are not persisted to disk or replicated through Raft to other replicas. This feature can greatly reduce the overhead of acquiring pessimistic locks and improve the throughput of pessimistic transactions.

When the memory usage of in-memory pessimistic locks exceeds the memory threshold of the Region or the TiKV node, the acquisition of pessimistic locks turns to the pipelined locking process. When the Region is merged or the leader is transferred, to avoid the loss of the pessimistic lock, TiKV writes the in-memory pessimistic lock to disk and replicates it to other replicas.

The in-memory pessimistic lock performs similarly to the pipelined locking process, which does not affect the lock acquisition when the cluster is healthy. However, when network isolation occurs in TiKV or a TiKV node is down, the acquired pessimistic lock might be lost.

If the application logic relies on the lock acquiring or lock waiting mechanism, or if you want to guarantee as much as possible the success rate of transaction commits even when the cluster is in an abnormal state, you need to **disable** the in-memory pessimistic lock feature.

This feature is enabled by default. To disable it, modify the TiKV configuration:

```
[pessimistic-txn]
in-memory = false
```

To dynamically disable this feature, modify the TiKV configuration dynamically:

```
set config tikv pessimistic-txn.in-memory='false';
```

14.10.10.5 Non-Transactional DML Statements

This document describes the usage scenarios, usage methods, and restrictions of non-transactional DML statements in TiDB. In addition, the implementation principle and common issues are also explained.

A non-transactional DML statement is a DML statement split into multiple SQL statements (which is, multiple batches) to be executed in sequence. It enhances the performance and ease of use in batch data processing at the expense of transactional atomicity and isolation.

Non-transactional DML statements include `INSERT`, `UPDATE`, and `DELETE`, of which TiDB currently only supports `DELETE`. For detailed syntax, see `BATCH`.

> **Note:**
>
> A non-transactional DML statement does not guarantee the atomicity and isolation of the statement, and is not equivalent to the original DML statement.

14.10.10.5.1 Usage scenarios

In the scenarios of large data processing, you might often need to perform same operations on a large batch of data. If the operation is performed directly using a single SQL statement, the transaction size might exceed the limit and affect the execution performance.

Batch data processing often has no overlap of time or data with the online application operations. Isolation (I in ACID) is unnecessary when no concurrent operations exist. Atomicity is also unnecessary if bulk data operations are idempotent or easily retryable. If your application needs neither data isolation nor atomicity, you can consider using non-transactional DML statements.

Non-transactional DML statements are used to bypass the size limit on large transactions in certain scenarios. One statement is used to complete tasks that would otherwise require manually splitting of transactions, with higher execution efficiency and less resource consumption.

For example, to delete expired data, if you ensure that no application will access the expired data, you can use a non-transactional DML statement to improve the `DELETE` performance.

14.10.10.5.2 Prerequisites

Before using non-transactional DML statements, make sure that the following conditions are met:

- The statement does not require atomicity, which permits some rows to be modified and some rows to remain unmodified in the execution result.
- The statement is idempotent, or you are prepared to retry on a part of the data according to the error message. If the system variables are set to `tidb_redact_log = 1` and `tidb_nontransactional_ignore_error = 1`, this statement must be idempotent. Otherwise, when the statement partially fails, the failed part cannot be accurately located.
- The data to be operated on has no other concurrent writes, which means it is not updated by other statements at the same time. Otherwise, unexpected results such as missing deletions and wrong deletions might occur.
- The statement does not modify the data to be read by the statement itself. Otherwise, the following batch will read the data written by the previous batch and easily causes unexpected results.
- The statement meets the restrictions.
- It is not recommended to perform concurrent DDL operations on the table to be read or written by this DML statement.

Warning:

> If `tidb_redact_log` and `tidb_nontransactional_ignore_error` are enabled at the same time, you might not get the complete error information of each batch, and you cannot retry the failed batch only. Therefore, if both of the system variables are turned on, the non-transactional DML statement must be idempotent.

14.10.10.5.3 Usage examples

Use a non-transactional DML statement

The following sections describe the use of non-transactional DML statements with examples:

Create a table `t` with the following schema:

```
CREATE TABLE t (id INT, v INT, KEY(id));
```

```
Query OK, 0 rows affected
```

Insert some data into table `t`.

```
INSERT INTO t VALUES (1, 2), (2, 3), (3, 4), (4, 5), (5, 6);
```

```
Query OK, 5 rows affected
```

The following operation uses a non-transactional DML statement to delete rows with values less than the integer 6 on column `v` of table `t`. This statement is split into two SQL statements, with a batch size of 2, divided by the `id` column and executed.

```
BATCH ON id LIMIT 2 DELETE FROM t WHERE v < 6;
```

```
+---------------+--------------+
| number of jobs | job status |
+---------------+--------------+
| 2             | all succeeded |
+---------------+--------------+
1 row in set
```

Check the deletion results of the above non-transactional DML statement.

```
SELECT * FROM t;
```

```
+----+---+
| id | v |
+----+---+
```

```
| 5  | 6 |
+----+---+
1 row in set
```

Check the execution progress

During the execution of a non-transactional DML statement, you can view the progress using `SHOW PROCESSLIST`. The `Time` field in the returned result indicates the time consumption of the current batch execution. Logs and slow logs also record the progress of each split statement throughout the non-transactional DML execution. For example:

```
SHOW PROCESSLIST;
```

```
+--
   ↪ ----+------+--------------------+--------+---------+------+-----------+--------
   ↪
| Id   | User | Host               | db     | Command | Time | State   | Info
   ↪
   ↪ |
+--
   ↪ ----+------+--------------------+--------+---------+------+-----------+--------
   ↪
| 1203 | root | 100.64.10.62:52711 | test | Query | 0 | autocommit | /* job
   ↪  506/500000 */ DELETE FROM `test`.`t1` WHERE `test`.`t1`.`_tidb_rowid
   ↪ ` BETWEEN 2271 AND 2273 |
| 1209 | root | 100.64.10.62:52735 | <null> | Query | 0 | autocommit | show
   ↪  full processlist
   ↪                                                                    |
+--
   ↪ ----+------+--------------------+--------+---------+------+-----------+--------
   ↪
```

Terminate a non-transactional DML statement

To terminate a non-transactional DML statement, you can use `KILL TIDB <
↪ processlist_id>`. Then TiDB will cancel all batches after the batch that is currently being executed. You can get the execution result from the log.

For more information about `KILL TIDB`, see the reference KILL.

Query the batch-dividing statement

During the execution of a non-transactional DML statement, a statement is internally used to divide the DML statement into multiple batches. To query this batch-dividing statement, you can add `DRY RUN QUERY` to this non-transactional DML statement. Then TiDB will not execute this query and the subsequent DML operations.

The following statement queries the batch-dividing statement during the execution of `BATCH ON id LIMIT 2 DELETE FROM t WHERE v < 6`:

```
BATCH ON id LIMIT 2 DRY RUN QUERY DELETE FROM t WHERE v < 6;
```

```
+--
    ↪ ----------------------------------------------------------------------------+
    ↪
| query statement                                                                  |
+--
    ↪ ----------------------------------------------------------------------------+
    ↪
| SELECT `id` FROM `test`.`t` WHERE (`v` < 6) ORDER BY IF(ISNULL(`id`),0,1)
    ↪ ,`id` |
+--
    ↪ ----------------------------------------------------------------------------+
    ↪
1 row in set
```

Query the statements corresponding to the first and the last batches

To query the actual DML statements corresponding to the first and the last batches in a non-transactional DML statement, you can add `DRY RUN` to this non-transactional DML statement. Then, TiDB only divides batches and does not execute these SQL statements. Because there might be many batches, not all batches are displayed, and only the first one and the last one are displayed.

```
BATCH ON id LIMIT 2 DRY RUN DELETE FROM t WHERE v < 6;
```

```
+-----------------------------------------------------------------------+
| split statement examples                                              |
+-----------------------------------------------------------------------+
| DELETE FROM `test`.`t` WHERE (`id` BETWEEN 1 AND 2 AND (`v` < 6)) |
| DELETE FROM `test`.`t` WHERE (`id` BETWEEN 3 AND 4 AND (`v` < 6)) |
+-----------------------------------------------------------------------+
2 rows in set
```

Use the optimizer hint

If an optimizer hint is originally supported in the `DELETE` statement, the optimizer hint is also supported in the non-transactional `DELETE` statement. The position of the hint is the same as that in the ordinary `DELETE` statement:

```
BATCH ON id LIMIT 2 DELETE /*+ USE_INDEX(t)*/ FROM t WHERE v < 6;
```

14.10.10.5.4 Best practices

To use a non-transactional DML statement, the following steps are recommended:

1. Select an appropriate dividing column. Integer or string types are recommended.
2. (Optional) Add `DRY RUN QUERY` to the non-transactional DML statement, execute the query manually, and confirm whether the data range affected by the DML statement is roughly correct.
3. (Optional) Add `DRY RUN` to the non-transactional DML statement, execute the query manually, and check the split statements and the execution plans. You need to pay attention to the index selection efficiency.
4. Execute the non-transactional DML statement.
5. If an error is reported, get the specific failed data range from the error message or log, and retry or handle it manually.

14.10.10.5.5 Parameter description

Parameter	Description	Default value	Required or not	Recommended value
Dividing column	The column used to divide batches, such as the `id` column in the above non-transactional DML statement `BATCH ON id LIMIT 2 DELETE FROM t WHERE v < 3503`	TiDB tries to automatically select a dividing column.	No	Select a column that can meet the `WHERE` condition in the most efficient way.

Parameter	Description	Default value	Required or not	Recommended value
Batch size	Used to control the size of each batch. The number of batches is the number of SQL statements into which DML operations are split, such as `LIMIT` ↪ ↪ 2 ↪ in the above non-transactional DML statement 3504 `BATCH` ↪	N/A	Yes	1000-1000000. Too small or too large a batch will lead to performance degradation.

Parameter	Description	Default value	Required or not	Recommended value

How to select a dividing column

A non-transactional DML statement uses a column as the basis for data batching, which is the dividing column. For higher execution efficiency, a dividing column is required to use index. The execution efficiency brought by different indexes and dividing columns might vary by dozens of times. When choosing the dividing column, consider the following suggestions:

- If you know the application data distribution, according to the `WHERE` condition, choose the column that divides data with smaller ranges after the batching.
    - Ideally, the `WHERE` condition can take advantage of the index of the dividing column to reduce the amount of data to be scanned per batch. For example, there is a transaction table that records the start and end time of each transaction, and you want to delete all transaction records whose end time is before one month. If there is an index on the start time of the transaction, and the start and end times of the transaction are relatively close, then you can choose the start time column as the dividing column.
    - In a less-than-ideal case, the data distribution of the dividing column is completely independent of the `WHERE` condition, and the index of the dividing column cannot be used to reduce the scope of the data scan.
- When a clustered index exists, it is recommended to use the primary key (including an `INT` primary key and `_tidb_rowid`) as the dividing column, so that the execution efficiency is higher.
- Choose the column with fewer duplicate values.

You can also choose not to specify a dividing column. Then, TiDB will use the first column of `handle` as the dividing column by default. But if the first column of the primary key of the clustered index is of a data type not supported by non-transactional DML statements (which is `ENUM`, `BIT`, `SET`, `JSON`), TiDB will report an error. You can choose an appropriate dividing column according to your application needs.

How to set batch size

In non-transactional DML statements, the larger the batch size, the fewer SQL statements are split and the slower each SQL statement is executed. The optimal batch size depends on the workload. It is recommended to start from 50000. Either too small or too large batch sizes will cause decreased execution efficiency.

The information of each batch is stored in memory, so too many batches can significantly increase memory consumption. This explains why the batch size cannot be too small. The upper limit of memory consumed by non-transactional statements for storing batch information is the same as `tidb_mem_quota_query`, and the action triggered when this limit is exceeded is determined by the configuration item `tidb_mem_oom_action`.

14.10.10.5.6 Restrictions

The following are hard restrictions on non-transactional DML statements. If these restrictions are not met, TiDB will report an error.

- You can only operate on a single table. Multi-table joins are currently not supported.
- The DML statements cannot contain `ORDER BY` or `LIMIT` clauses.
- The dividing column must be indexed. The index can be a single-column index, or the first column of a joint index.
- Must be used in the `autocommit` mode.
- Cannot be used when batch-dml is enabled.
- Cannot be used when `tidb_snapshot` is set.
- Cannot be used with the `prepare` statement.
- `ENUM`, `BIT`, `SET`, `JSON` types are not supported as the dividing columns.
- Not supported for temporary tables.
- Common Table Expression is not supported.

14.10.10.5.7 Control batch execution failure

Non-transactional DML statements do not satisfy atomicity. Some batches might succeed and some might fail. The system variable `tidb_nontransactional_ignore_error` controls how the non-transactional DML statements handle errors.

An exception is that if the first batch fails, there is a high probability that the statement itself is wrong. In this case, the entire non-transactional statement will directly return an error.

14.10.10.5.8 How it works

The working principle of non-transactional DML statements is to build into TiDB the automatic splitting of SQL statements. Without non-transactional DML statements, you will need to manually split the SQL statements. To understand the behavior of a non-transactional DML statement, think of it as a user script doing the following tasks:

For the non-transactional DML `BATCH ON $C$ LIMIT $N$ DELETE FROM ... WHERE $P$`, C is the column used for dividing, N is the batch size, and P is the filter condition.

1. According to the filter condition P of the original statement and the specified column C for dividing, TiDB queries all C that satisfy P. TiDB sorts these C into groups $B_1 \dots B_k$ according to N. For each of all B_i, TiDB keeps its first and last C as S_i and E_i. The query statement executed in this step can be viewed through `DRY RUN QUERY`.
2. The data involved in B_i is a subset that satisfies P_i: C BETWEEN S_i AND E_i. You can use P_i to narrow down the range of data that each batch needs to process.
3. For B_i, TiDB embeds the above condition into the `WHERE` condition of the original statement, which makes it WHERE (P_i) AND (P). The execution result of this step can be viewed through `DRY RUN`.

4. For all batches, execute new statements in sequence. The errors for each grouping are collected and combined, and returned as the result of the entire non-transactional DML statement after all groupings are complete.

14.10.10.5.9 Comparison with batch-dml

batch-dml is a mechanism for splitting a transaction into multiple transaction commits during the execution of a DML statement.

> **Note:**
>
> It is not recommended to use batch-dml which has been deprecated. When the batch-dml feature is not properly used, there is a risk of data index inconsistency.

Non-transactional DML statements are not yet a replacement for all batch-dml usage scenarios. Their main differences are as follows:

- Performance: When the dividing column is efficient, the performance of non-transactional DML statements is close to that of batch-dml. When the dividing column is less efficient, the performance of non-transactional DML statements is significantly lower than that of batch-dml.
- Stability: batch-dml is prone to data index inconsistencies due to improper use. Non-transactional DML statements do not cause data index inconsistencies. However, when used improperly, non-transactional DML statements are not equivalent to the original statements, and the applications might observe unexpected behavior. See the common issues section for details.

14.10.10.5.10 Common issues

Executing a multiple table joins statement results in the `Unknown column xxx in 'where clause'` error

This error occurs when the `WHERE` clause concatenated in a query involves tables other than the table in which the shard column is defined. For example, in the following SQL statement, the shard column is `t2.id` and it is defined in table `t2`, but the `WHERE` clause involves table `t2` and `t3`.

```
BATCH ON test.t2.id LIMIT 1
INSERT INTO t
SELECT t2.id, t2.v, t3. FROM t2, t3 WHERE t2.id = t3.id
```

```
(1054, "Unknown column 't3.id' in 'where clause'")
```

If the error occurs, you can print the query statement for confirmation by using `DRY RUN QUERY`. For example:

```
BATCH ON test.t2.id LIMIT 1
DRY RUN QUERY INSERT INTO t
SELECT t2.id, t2.v, t3. FROM t2, t3 WHERE t2.id = t3.id
```

To avoid the error, you can move the condition related to other tables in the `WHERE` clause to the `ON` condition in the `JOIN` clause. For example:

```
BATCH ON test.t2.id LIMIT 1
INSERT INTO t
SELECT t2.id, t2.v, t3. FROM t2 JOIN t3 ON t2.id=t3.id
```

```
+----------------+---------------+
| number of jobs | job status |
+----------------+---------------+
| 0              | all succeeded |
+----------------+---------------+
```

The actual batch size is not the same as the specified batch size

During the execution of a non-transactional DML statement, the size of data to be processed in the last batch might be smaller than the specified batch size.

When **duplicated values exist in the dividing column**, each batch will contain all the duplicated values of the last element of the dividing column in this batch. Therefore, the number of rows in this batch might be greater than the specified batch size.

In addition, when other concurrent writes occur, the number of rows processed in each batch might be different from the specified batch size.

The `Failed to restore the delete statement, probably because of unsupported type of the shard column` error occurs during execution

The dividing column does not support `ENUM`, `BIT`, `SET`, `JSON` types. Try to specify a new dividing column. It is recommended to use an integer or string type column.

If the error occurs when the selected shard column is not one of these unsupported types, get support from PingCAP or the community.

Non-transactional `DELETE` has "exceptional" behavior that is not equivalent to ordinary `DELETE`

A non-transactional DML statement is not equivalent to the original form of this DML statement, which might have the following reasons:

- There are other concurrent writes.

- The non-transactional DML statement modifies a value that the statement itself will read.
- The SQL statement executed in each batch might cause a different execution plan and expression calculation order because the `WHERE` condition is changed. Therefore, the execution result might be different from the original statement.
- The DML statements contain non-deterministic operations.

14.10.10.5.11 MySQL compatibility

Non-transactional statements are TiDB-specific and are not compatible with MySQL.

14.10.10.5.12 See also

- The `BATCH` syntax
- `tidb_nontransactional_ignore_error`

14.10.11 Views

TiDB supports views. A view acts as a virtual table, whose schema is defined by the `SELECT` statement that creates the view. Using views has the following benefits:

- Exposing only safe fields and data to users to ensure security of sensitive fields and data stored in the underlying table.
- Defining complex queries that frequently appear as views to make complex queries easier and more convenient.

14.10.11.1 Query views

Querying a view is similar to querying an ordinary table. However, when TiDB queries a view, it actually queries the `SELECT` statement associated with the view.

14.10.11.2 Show metadata

To obtain the metadata of views, choose any of the following methods.

14.10.11.2.1 Use the `SHOW CREATE TABLE view_name` or `SHOW CREATE VIEW view_name` statement

Usage example:

```
show create view v;
```

This statement shows the `CREATE VIEW` statement corresponding to this view and the value of the `character_set_client` and `collation_connection` system variables when the view was created.

```
+--
    ↪ ----+------------------------------------------------------------------------
    ↪
| View | Create View
    ↪
    ↪ | character_set_client | collation_connection |
+--
    ↪ ----+------------------------------------------------------------------------
    ↪
| v    | CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`127.0.0.1` SQL SECURITY
    ↪ DEFINER VIEW `v` (`a`) AS SELECT `s`.`a` FROM `test`.`t` LEFT JOIN `
    ↪ test`.`s` ON `t`.`a`=`s`.`a` | utf8 | utf8_general_ci |
+--
    ↪ ----+------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.11.2.2 Query the INFORMATION_SCHEMA.VIEWS table

Usage example:

```
select * from information_schema.views;
```

You can view the relevant meta information of the view by querying this table, such as TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, VIEW_DEFINITION, CHECK_OPTION, IS_UPDATABLE, DEFINER, SECURITY_TYPE, CHARACTER_SET_CLIENT, and COLLATION_CONNECTION ↪ .

```
+--
    ↪ ------------+-------------+-----------+----------------------------------
    ↪
| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME | VIEW_DEFINITION
    ↪                                          | CHECK_OPTION | IS_UPDATABLE
    ↪  | DEFINER  | SECURITY_TYPE | CHARACTER_SET_CLIENT |
    ↪ COLLATION_CONNECTION |
+--
    ↪ ------------+-------------+-----------+----------------------------------
    ↪
| def          | test        | v         | SELECT `s`.`a` FROM `test`.`t` LEFT
    ↪  JOIN `test`.`s` ON `t`.`a`=`s`.`a` | CASCADED | NO  | root@127.0.0.1
    ↪ | DEFINER   | utf8                | utf8_general_ci    |
+--
    ↪ ------------+-------------+-----------+----------------------------------
    ↪
1 row in set (0.00 sec)
```

14.10.11.2.3 Use the HTTP APIs

Usage example:

```
curl http://127.0.0.1:10080/schema/test/v
```

By visiting `http://{TiDBIP}:10080/schema/{db}/{view}`, you can get all the metadata for the view.


```
{
 "id": 122,
 "name": {
  "O": "v",
  "L": "v"
 },
 "charset": "utf8",
 "collate": "utf8_general_ci",
 "cols": [
  {
   "id": 1,
   "name": {
    "O": "a",
    "L": "a"
   },
   "offset": 0,
   "origin_default": null,
   "default": null,
   "default_bit": null,
   "default_is_expr": false,
   "generated_expr_string": "",
   "generated_stored": false,
   "dependences": null,
   "type": {
    "Tp": 0,
    "Flag": 0,
    "Flen": 0,
    "Decimal": 0,
    "Charset": "",
    "Collate": "",
    "Elems": null
   },
   "state": 5,
   "comment": "",
   "hidden": false,
   "version": 0
  }
 ],
```

```
 "index_info": null,
 "fk_info": null,
 "state": 5,
 "pk_is_handle": false,
 "is_common_handle": false,
 "comment": "",
 "auto_inc_id": 0,
 "auto_id_cache": 0,
 "auto_rand_id": 0,
 "max_col_id": 1,
 "max_idx_id": 0,
 "update_timestamp": 416801600091455490,
 "ShardRowIDBits": 0,
 "max_shard_row_id_bits": 0,
 "auto_random_bits": 0,
 "pre_split_regions": 0,
 "partition": null,
 "compression": "",
 "view": {
  "view_algorithm": 0,
  "view_definer": {
   "Username": "root",
   "Hostname": "127.0.0.1",
   "CurrentUser": false,
   "AuthUsername": "root",
   "AuthHostname": "%"
  },
  "view_security": 0,
  "view_select": "SELECT `s`.`a` FROM `test`.`t` LEFT JOIN `test`.`s` ON `t
     ↪ `.`a`=`s`.`a`",
  "view_checkoption": 1,
  "view_cols": null
 },
 "sequence": null,
 "Lock": null,
 "version": 3,
 "tiflash_replica": null
}
```


14.10.11.3 Example

The following example creates a view, queries this view, and delete this view:

```
create table t(a int, b int);
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
insert into t values(1, 1),(2,2),(3,3);
```

```
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
create table s(a int);
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
insert into s values(2),(3);
```

```
Query OK, 2 rows affected (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
create view v as select s.a from t left join s on t.a = s.a;
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
select * from v;
```

```
+------+
| a    |
+------+
| NULL |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

```
drop view v;
```

```
Query OK, 0 rows affected (0.02 sec)
```

14.10.11.4 Limitations

Currently, views in TiDB are subject to the following limitations:

- Materialized views are not supported yet.
- Views in TiDB are read-only and do not support write operations such as `UPDATE`, `INSERT`, `DELETE`, and `TRUNCATE`.
- For created views, the only supported DDL operation is `DROP [VIEW | TABLE]`

14.10.11.5 See also

- CREATE VIEW
- DROP VIEW

14.10.12 Partitioning

This document introduces TiDB's implementation of partitioning.

14.10.12.1 Partitioning types

This section introduces the types of partitioning in TiDB. Currently, TiDB supports Range partitioning, List partitioning, List COLUMNS partitioning, and Hash partitioning.

Range partitioning, List partitioning and List COLUMNS partitioning are used to resolve the performance issues caused by a large amount of deletions in the application, and support fast drop partition operations. Hash partitioning is used to scatter the data when there are a large amount of writes.

14.10.12.1.1 Range partitioning

When a table is partitioned by Range, each partition contains rows for which the partitioning expression value lies within a given Range. Ranges have to be contiguous but not overlapping. You can define it by using `VALUES LESS THAN`.

Assume you need to create a table that contains personnel records as follows:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
    store_id INT NOT NULL
);
```

You can partition a table by Range in various ways as needed. For example, you can partition it by using the `store_id` column:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
```

```
    store_id INT NOT NULL
)

PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

In this partition scheme, all rows corresponding to employees whose `store_id` is 1 through 5 are stored in the `p0` partition while all employees whose `store_id` is 6 through 10 are stored in `p1`. Range partitioning requires the partitions to be ordered, from lowest to highest.

If you insert a row of data `(72, 'Tom', 'John', '2015-06-25', NULL, NULL, 15)`, it falls in the `p2` partition. But if you insert a record whose `store_id` is larger than 20, an error is reported because TiDB cannot know which partition this record should be inserted into. In this case, you can use `MAXVALUE` when creating a table:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
    store_id INT NOT NULL
)

PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

`MAXVALUE` represents an integer value that is larger than all other integer values. Now, all records whose `store_id` is equal to or larger than 16 (the highest value defined) are stored in the `p3` partition.

You can also partition a table by employees' job codes, which are the values of the `job_code` column. Assume that two-digit job codes stand for regular employees, three-digit codes stand for office and customer support personnel, and four-digit codes stand for managerial personnel. Then you can create a partitioned table like this:

```
CREATE TABLE employees (
```

```
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
    store_id INT NOT NULL
)

PARTITION BY RANGE (job_code) (
    PARTITION p0 VALUES LESS THAN (100),
    PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (10000)
);
```

In this example, all rows relating to regular employees are stored in the `p0` partition, all office and customer support personnel in the `p1` partition, and all managerial personnel in the `p2` partition.

Besides splitting up the table by `store_id`, you can also partition a table by dates. For example, you can partition by employees' separation year:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)

PARTITION BY RANGE ( YEAR(separated) ) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

In Range partitioning, you can partition based on the values of the `timestamp` column and use the `unix_timestamp()` function, for example:

```
CREATE TABLE quarterly_report_status (
    report_id INT NOT NULL,
    report_status VARCHAR(20) NOT NULL,
    report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
        ↪ CURRENT_TIMESTAMP
```

```
)

PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
    PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
    PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
    PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
    PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
    PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
    PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
    PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
    PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
    PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

It is not allowed to use any other partitioning expression that contains the timestamp column.

Range partitioning is particularly useful when one or more of the following conditions are satisfied:

- You want to delete the old data. If you use the `employees` table in the previous example, you can delete all records of employees who left this company before the year 1991 by simply using `ALTER TABLE employees DROP PARTITION p0;`. It is faster than executing the `DELETE FROM employees WHERE YEAR(separated)<= 1990;` operation.
- You want to use a column that contains time or date values, or containing values arising from some other series.
- You need to frequently run queries on the columns used for partitioning. For example, when executing a query like `EXPLAIN SELECT COUNT(*)FROM employees WHERE separated BETWEEN '2000-01-01' AND '2000-12-31' GROUP BY store_id;`, TiDB can quickly know that only the data in the `p2` partition needs to be scanned, because the other partitions do not match the `WHERE` condition.

14.10.12.1.2 List partitioning

Before creating a List partitioned table, you need to set the value of the session variable `tidb_enable_list_partition` to `ON`.

```
set @@session.tidb_enable_list_partition = ON
```

Also, make sure that `tidb_enable_table_partition` is set to `ON`, which is the default setting.

List partitioning is similar to Range partitioning. Unlike Range partitioning, in List partitioning, the partitioning expression values for all rows in each partition are in a given value set. This value set defined for each partition can have any number of values but cannot

have duplicate values. You can use the `PARTITION ... VALUES IN (...)` clause to define a value set.

Suppose that you want to create a personnel record table. You can create a table as follows:

```
CREATE TABLE employees (
    id INT NOT NULL,
    hired DATE NOT NULL DEFAULT '1970-01-01',
    store_id INT
);
```

Suppose that there are 20 stores distributed in 4 districts, as shown in the table below:

```
| Region | Store ID Numbers  |
| ------- | ------------------- |
| North  | 1, 2, 3, 4, 5      |
| East   | 6, 7, 8, 9, 10     |
| West   | 11, 12, 13, 14, 15 |
| Central | 16, 17, 18, 19, 20 |
```

If you want to store the personnel data of employees of the same region in the same partition, you can create a List partitioned table based on `store_id`:

```
CREATE TABLE employees (
    id INT NOT NULL,
    hired DATE NOT NULL DEFAULT '1970-01-01',
    store_id INT
)
PARTITION BY LIST (store_id) (
    PARTITION pNorth VALUES IN (1, 2, 3, 4, 5),
    PARTITION pEast VALUES IN (6, 7, 8, 9, 10),
    PARTITION pWest VALUES IN (11, 12, 13, 14, 15),
    PARTITION pCentral VALUES IN (16, 17, 18, 19, 20)
);
```

After creating the partitions as above, you can easily add or delete records related to a specific region in the table. For example, suppose that all stores in the East region (East) are sold to another company. Then all the row data related to the store employees of this region can be deleted by executing `ALTER TABLE employees TRUNCATE PARTITION pEast`, which is much more efficient than the equivalent statement `DELETE FROM employees WHERE store_id IN (6, 7, 8, 9, 10)`.

You can also execute `ALTER TABLE employees DROP PARTITION pEast` to delete all related rows, but this statement also deletes the `pEast` partition from the table definition. In this situation, you must execute the `ALTER TABLE ... ADD PARTITION` statement to recover the original partitioning scheme of the table.

Unlike Range partitioning, List partitioning does not have a similar `MAXVALUE` partition to store all values that do not belong to other partitions. Instead, all expected values of the partition expression must be included in the `PARTITION ... VALUES IN (...)` clause. If the value to be inserted in an `INSERT` statement does not match the column value set of any partition, the statement fails to execute and an error is reported. See the following example:

```
test> CREATE TABLE t (
    ->   a INT,
    ->   b INT
    -> )
    -> PARTITION BY LIST (a) (
    ->   PARTITION p0 VALUES IN (1, 2, 3),
    ->   PARTITION p1 VALUES IN (4, 5, 6)
    -> );
Query OK, 0 rows affected (0.11 sec)

test> INSERT INTO t VALUES (7, 7);
ERROR 1525 (HY000): Table has no partition for value 7
```

To ignore the error type above, you can use the `IGNORE` keyword. After using this keyword, if a row contains values that do not match the column value set of any partition, this row will not be inserted. Instead, any row with matched values is inserted, and no error is reported:

```
test> TRUNCATE t;
Query OK, 1 row affected (0.00 sec)

test> INSERT IGNORE INTO t VALUES (1, 1), (7, 7), (8, 8), (3, 3), (5, 5);
Query OK, 3 rows affected, 2 warnings (0.01 sec)
Records: 5 Duplicates: 2 Warnings: 2

test> select * from t;
+------+------+
| a    | b    |
+------+------+
|    5 |    5 |
|    1 |    1 |
|    3 |    3 |
+------+------+
3 rows in set (0.01 sec)
```

14.10.12.1.3 List COLUMNS partitioning

List COLUMNS partitioning is a variant of List partitioning. You can use multiple columns as partition keys. Besides the integer data type, you can also use the columns in the string, `DATE`, and `DATETIME` data types as partition columns.

Suppose that you want to divide the store employees from the following 12 cities into 4 regions, as shown in the following table:

```
| Region | Cities                        |
| :----- | ----------------------------- |
| 1      | LosAngeles,Seattle, Houston |
| 2      | Chicago, Columbus, Boston  |
| 3      | NewYork, LongIsland, Baltimore |
| 4      | Atlanta, Raleigh, Cincinnati |
```

You can use List COLUMNS partitioning to create a table and store each row in the partition that corresponds to the employee's city, as shown below:

```
CREATE TABLE employees_1 (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
    store_id INT,
    city VARCHAR(15)
)
PARTITION BY LIST COLUMNS(city) (
    PARTITION pRegion_1 VALUES IN('LosAngeles', 'Seattle', 'Houston'),
    PARTITION pRegion_2 VALUES IN('Chicago', 'Columbus', 'Boston'),
    PARTITION pRegion_3 VALUES IN('NewYork', 'LongIsland', 'Baltimore'),
    PARTITION pRegion_4 VALUES IN('Atlanta', 'Raleigh', 'Cincinnati')
);
```

Unlike List partitioning, in List COLUMNS partitioning, you do not need to use the expression in the `COLUMNS()` clause to convert column values to integers.

List COLUMNS partitioning can also be implemented using columns of the `DATE` and `DATETIME` types, as shown in the following example. This example uses the same names and columns as the previous `employees_1` table, but uses List COLUMNS partitioning based on the `hired` column:

```
CREATE TABLE employees_2 (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
    store_id INT,
    city VARCHAR(15)
```

```
)
PARTITION BY LIST COLUMNS(hired) (
    PARTITION pWeek_1 VALUES IN('2020-02-01', '2020-02-02', '2020-02-03',
        '2020-02-04', '2020-02-05', '2020-02-06', '2020-02-07'),
    PARTITION pWeek_2 VALUES IN('2020-02-08', '2020-02-09', '2020-02-10',
        '2020-02-11', '2020-02-12', '2020-02-13', '2020-02-14'),
    PARTITION pWeek_3 VALUES IN('2020-02-15', '2020-02-16', '2020-02-17',
        '2020-02-18', '2020-02-19', '2020-02-20', '2020-02-21'),
    PARTITION pWeek_4 VALUES IN('2020-02-22', '2020-02-23', '2020-02-24',
        '2020-02-25', '2020-02-26', '2020-02-27', '2020-02-28')
);
```

In addition, you can also add multiple columns in the `COLUMNS()` clause. For example:

```
CREATE TABLE t (
    id int,
    name varchar(10)
)
PARTITION BY LIST COLUMNS(id,name) (
     partition p0 values IN ((1,'a'),(2,'b')),
     partition p1 values IN ((3,'c'),(4,'d')),
     partition p3 values IN ((5,'e'),(null,null))
);
```

14.10.12.1.4 Hash partitioning

Hash partitioning is used to make sure that data is evenly scattered into a certain number of partitions. With Range partitioning, you must specify the range of the column values for each partition when you use Range partitioning, while you just need to specify the number of partitions when you use Hash partitioning.

Partitioning by Hash requires you to append a `PARTITION BY HASH (expr)` clause to the `CREATE TABLE` statement. `expr` is an expression that returns an integer. It can be a column name if the type of this column is integer. In addition, you might also need to append `PARTITIONS num`, where `num` is a positive integer indicating how many partitions a table is divided into.

The following operation creates a Hash partitioned table, which is divided into 4 partitions by `store_id`:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
```

```
    store_id INT
)

PARTITION BY HASH(store_id)
PARTITIONS 4;
```

If `PARTITIONS num` is not specified, the default number of partitions is 1.

You can also use an SQL expression that returns an integer for `expr`. For example, you can partition a table by the hire year:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)

PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;
```

The most efficient Hash function is one which operates upon a single table column, and whose value increases or decreases consistently with the column value.

For example, `date_col` is a column whose type is `DATE`, and the value of the `TO_DAYS (date_col)` expression varies with the value of `date_col`. `YEAR(date_col)` is different from `TO_DAYS(date_col)`, because not every possible change in `date_col` produces an equivalent change in `YEAR(date_col)`.

In contrast, assume that you have an `int_col` column whose type is `INT`. Now consider about the expression `POW(5-int_col,3)+ 6`. It is not a good Hash function though, because as the value of `int_col` changes, the result of the expression does not change proportionally. A value change in `int_col` might result in a huge change in the expression result. For example, when `int_col` changes from 5 to 6, the change of the expression result is -1. But the result change might be -7 when `int_col` changes from 6 to 7.

In conclusion, when the expression has a form that is closer to `y = cx`, it is more suitable to be a Hash function. Because the more non-linear an expression is, the more unevenly scattered the data among the partitions tends to be.

In theory, pruning is also possible for expressions involving more than one column value, but determining which of such expressions are suitable can be quite difficult and time-consuming. For this reason, the use of hashing expressions involving multiple columns is not particularly recommended.

When using `PARTITION BY HASH`, TiDB decides which partition the data should fall into based on the modulus of the result of the expression. In other words, if a partitioning expression is `expr` and the number of partitions is `num`, `MOD(expr, num)` decides the partition in which the data is stored. Assume that `t1` is defined as follows:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY HASH( YEAR(col3) )
    PARTITIONS 4;
```

When you insert a row of data into `t1` and the value of `col3` is '2005-09-15', then this row is inserted into partition 1:

```
MOD(YEAR('2005-09-01'),4)
=  MOD(2005,4)
=  1
```

14.10.12.1.5 How TiDB partitioning handles NULL

It is allowed in TiDB to use `NULL` as the calculation result of a partitioning expression.

> **Note:**
>
> `NULL` is not an integer. TiDB's partitioning implementation treats `NULL` as being less than any other integer values, just as `ORDER BY` does.

Handling of NULL with Range partitioning

When you insert a row into a table partitioned by Range, and the column value used to determine the partition is `NULL`, then this row is inserted into the lowest partition.

```
CREATE TABLE t1 (
    c1 INT,
    c2 VARCHAR(20)
)

PARTITION BY RANGE(c1) (
    PARTITION p0 VALUES LESS THAN (0),
    PARTITION p1 VALUES LESS THAN (10),
    PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

```
Query OK, 0 rows affected (0.09 sec)
```

```
select * from t1 partition(p0);
```

```
+------|--------+
| c1   | c2     |
+------|--------+
| NULL | mothra |
+------|--------+
1 row in set (0.00 sec)
```

```
select * from t1 partition(p1);
```

```
Empty set (0.00 sec)
```

```
select * from t1 partition(p2);
```

```
Empty set (0.00 sec)
```

Drop the p0 partition and verify the result:

```
alter table t1 drop partition p0;
```

```
Query OK, 0 rows affected (0.08 sec)
```

```
select * from t1;
```

```
Empty set (0.00 sec)
```

Handling of NULL with Hash partitioning

When partitioning tables by Hash, there is a different way of handling NULL value - if the calculation result of the partitioning expression is NULL, it is considered as 0.

```
CREATE TABLE th (
    c1 INT,
    c2 VARCHAR(20)
)

PARTITION BY HASH(c1)
PARTITIONS 2;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
INSERT INTO th VALUES (NULL, 'mothra'), (0, 'gigan');
```

```
Query OK, 2 rows affected (0.04 sec)
```

```
select * from th partition (p0);
```

```
+------|--------+
| c1   | c2     |
+------|--------+
| NULL | mothra |
|    0 | gigan  |
+------|--------+
2 rows in set (0.00 sec)
```

```
select * from th partition (p1);
```

```
Empty set (0.00 sec)
```

You can see that the inserted record `(NULL, 'mothra')` falls into the same partition as `(0, 'gigan')`.

> **Note:**
>
> `NULL` values by Hash partitions in TiDB are handled in the same way as described in How MySQL Partitioning Handles NULL, which, however, is not consistent with the actual behavior of MySQL. In other words, MySQL's implementation in this case is not consistent with its documentation.
>
> In this case, the actual behavior of TiDB is in line with the description of this document.

14.10.12.2 Partition management

For `LIST` and `RANGE` partitioned tables, you can add and drop partitions using the `ALTER TABLE <table name> ADD PARTITION (<partition specification>)` or `ALTER TABLE <table name> DROP PARTITION <list of partitions>` statement.

For `LIST` and `RANGE` partitioned tables, `REORGANIZE PARTITION` is not yet supported.

For `HASH` partitioned tables, `COALESCE PARTITION` and `ADD PARTITION` are not yet supported.

`EXCHANGE PARTITION` works by swapping a partition and a non-partitioned table, similar to how renaming a table like `RENAME TABLE t1 TO t1_tmp, t2 TO t1, t1_tmp TO t2` works.

For example, `ALTER TABLE partitioned_table EXCHANGE PARTITION p1 WITH TABLE non_partitioned_table` swaps the `non_partitioned_table` table in the `p1` partition with the `partitioned_table` table.

Ensure that all rows that you are exchanging into the partition match the partition definition; otherwise, these rows will not be found and cause unexpected issues.

> **Warning:**
>
> `EXCHANGE PARTITION` is an experimental feature. It is not recommended to use it in a production environment. To enable it, set the `tidb_enable_exchange_partition` system variable to `ON`.

14.10.12.2.1 Range partition management

Create a partitioned table:

```
CREATE TABLE members (
    id INT,
    fname VARCHAR(25),
    lname VARCHAR(25),
    dob DATE
)

PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION p0 VALUES LESS THAN (1980),
    PARTITION p1 VALUES LESS THAN (1990),
    PARTITION p2 VALUES LESS THAN (2000)
);
```

Drop a partition:

```
ALTER TABLE members DROP PARTITION p2;
```

```
Query OK, 0 rows affected (0.03 sec)
```

Empty a partition:

```
ALTER TABLE members TRUNCATE PARTITION p1;
```

```
Query OK, 0 rows affected (0.03 sec)
```

> **Note:**
>
> `ALTER TABLE ... REORGANIZE PARTITION` is currently unsupported in TiDB.

Add a partition:

```
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2010));
```

When partitioning tables by Range, `ADD PARTITION` can be only appended to the very end of a partition list. If it is appended to an existing Range partition, an error is reported:

```
ALTER TABLE members
    ADD PARTITION (
    PARTITION n VALUES LESS THAN (1970));
```

```
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »
   increasing for each partition
```

14.10.12.2.2 Hash partition management

Unlike Range partitioning, `DROP PARTITION` is not supported in Hash partitioning.

Currently, `ALTER TABLE ... COALESCE PARTITION` is not supported in TiDB as well. For partition management statements that are not currently supported, TiDB returns an error.

```
alter table members optimize partition p0;
```

```
ERROR 8200 (HY000): Unsupported optimize partition
```

14.10.12.3 Partition pruning

Partition pruning is an optimization which is based on a very simple idea - do not scan the partitions that do not match.

Assume that you create a partitioned table `t1`:

```
CREATE TABLE t1 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
```

```
PARTITION BY RANGE( region_code ) (
    PARTITION p0 VALUES LESS THAN (64),
    PARTITION p1 VALUES LESS THAN (128),
    PARTITION p2 VALUES LESS THAN (192),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

If you want to get the result of this `SELECT` statement:

```
SELECT fname, lname, region_code, dob
    FROM t1
    WHERE region_code > 125 AND region_code < 130;
```

It is evident that the result falls in either the `p1` or the `p2` partition, that is, you just need to search for the matching rows in `p1` and `p2`. Excluding the unneeded partitions is so-called "pruning". If the optimizer is able to prune a part of partitions, the execution of the query in the partitioned table will be much faster than that in a non-partitioned table.

The optimizer can prune partitions through `WHERE` conditions in the following two scenarios:

- partition_column = constant
- partition_column IN (constant1, constant2, ..., constantN)

Currently, partition pruning does not work with `LIKE` conditions.

14.10.12.3.1 Some cases for partition pruning to take effect

1. Partition pruning uses the query conditions on the partitioned table, so if the query conditions cannot be pushed down to the partitioned table according to the planner's optimization rules, partition pruning does not apply for this query.

   For example:

   ```
   create table t1 (x int) partition by range (x) (
          partition p0 values less than (5),
          partition p1 values less than (10));
   create table t2 (x int);
   ```

   ```
   explain select * from t1 left join t2 on t1.x = t2.x where t2.x > 5;
   ```

   In this query, the left out join is converted to the inner join, and then `t1.x > 5` is derived from `t1.x = t2.x` and `t2.x > 5`, so it could be used in partition pruning and only the partition `p1` remains.

```
explain select * from t1 left join t2 on t1.x = t2.x and t2.x > 5;
```

In this query, `t2.x > 5` cannot be pushed down to the `t1` partitioned table, so partition pruning would not take effect for this query.

2. Since partition pruning is done during the plan optimizing phase, it does not apply for those cases that filter conditions are unknown until the execution phase.

   For example:

```
create table t1 (x int) partition by range (x) (
       partition p0 values less than (5),
       partition p1 values less than (10));
```

```
explain select * from t2 where x < (select * from t1 where t2.x < t1.x
   ↪  and t2.x < 2);
```

   This query reads a row from `t2` and uses the result for the subquery on `t1`. Theoretically, partition pruning could benefit from `t1.x > val` expression in the subquery, but it does not take effect there as that happens in the execution phase.

3. As a result of a limitation from current implementation, if a query condition cannot be pushed down to TiKV, it cannot be used by the partition pruning.

   Take the `fn(col)` expression as an example. If the TiKV coprocessor supports this `fn` function, `fn(col)` may be pushed down to the the leaf node (that is, partitioned table) according to the predicate push-down rule during the plan optimizing phase, and partition pruning can use it.

   If the TiKV coprocessor does not support this `fn` function, `fn(col)` would not be pushed down to the leaf node. Instead, it becomes a `Selection` node above the leaf node. The current partition pruning implementation does not support this kind of plan tree.

4. For Hash partition, the only query supported by partition pruning is the equal condition.

5. For Range partition, for partition pruning to take effect, the partition expression must be in those forms: `col` or `fn(col)`, and the query condition must be one of `>`, `<`, `=`, `>=`, and `<=`. If the partition expression is in the form of `fn(col)`, the `fn` function must be monotonous.

   If the `fn` function is monotonous, for any `x` and `y`, if `x > y`, then `fn(x) > fn(y)`. Then this `fn` function can be called strictly monotonous. For any `x` and `y`, if `x > y`, then `fn(x) >= fn(y)`. In this case, `fn` could also be called "monotonous". In theory, all monotonous functions are supported by partition pruning.

   Currently, partition pruning in TiDB only support those monotonous functions:

   - `UNIX_TIMESTAMP()`

- `TO_DAYS()`

For example, the partition expression is a simple column:

```
create table t (id int) partition by range (id) (
       partition p0 values less than (5),
       partition p1 values less than (10));
select * from t where id > 6;
```

Or the partition expression is in the form of `fn(col)` where `fn` is `to_days`:

```
create table t (dt datetime) partition by range (to_days(id)) (
       partition p0 values less than (to_days('2020-04-01')),
       partition p1 values less than (to_days('2020-05-01')));
select * from t where dt > '2020-04-18';
```

An exception is `floor(unix_timestamp())` as the partition expression. TiDB does some optimization for that case by case, so it is supported by partition pruning.

```
create table t (ts timestamp(3) not null default current_timestamp(3))
partition by range (floor(unix_timestamp(ts))) (
       partition p0 values less than (unix_timestamp('2020-04-01
          ↪ 00:00:00')),
       partition p1 values less than (unix_timestamp('2020-05-01
          ↪ 00:00:00')));
select * from t where ts > '2020-04-18 02:00:42.123';
```

14.10.12.4 Partition selection

`SELECT` statements support partition selection, which is implemented by using a `PARTITION` option.

```
SET @@sql_mode = '';

CREATE TABLE employees (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    fname VARCHAR(25) NOT NULL,
    lname VARCHAR(25) NOT NULL,
    store_id INT NOT NULL,
    department_id INT NOT NULL
)

PARTITION BY RANGE(id) (
    PARTITION p0 VALUES LESS THAN (5),
    PARTITION p1 VALUES LESS THAN (10),
    PARTITION p2 VALUES LESS THAN (15),
    PARTITION p3 VALUES LESS THAN MAXVALUE
```

```
);

INSERT INTO employees VALUES
    ('', 'Bob', 'Taylor', 3, 2), ('', 'Frank', 'Williams', 1, 2),
    ('', 'Ellen', 'Johnson', 3, 4), ('', 'Jim', 'Smith', 2, 4),
    ('', 'Mary', 'Jones', 1, 1), ('', 'Linda', 'Black', 2, 3),
    ('', 'Ed', 'Jones', 2, 1), ('', 'June', 'Wilson', 3, 1),
    ('', 'Andy', 'Smith', 1, 3), ('', 'Lou', 'Waters', 2, 4),
    ('', 'Jill', 'Stone', 1, 4), ('', 'Roger', 'White', 3, 2),
    ('', 'Howard', 'Andrews', 1, 2), ('', 'Fred', 'Goldberg', 3, 3),
    ('', 'Barbara', 'Brown', 2, 3), ('', 'Alice', 'Rogers', 2, 2),
    ('', 'Mark', 'Morgan', 3, 3), ('', 'Karen', 'Cole', 3, 2);
```

You can view the rows stored in the `p1` partition:

```
SELECT * FROM employees PARTITION (p1);
```

```
+----|-------|--------|----------|---------------+
| id | fname | lname | store_id | department_id |
+----|-------|--------|----------|---------------+
|  5 | Mary  | Jones |        1 |             1 |
|  6 | Linda | Black |        2 |             3 |
|  7 | Ed    | Jones |        2 |             1 |
|  8 | June  | Wilson |       3 |             1 |
|  9 | Andy  | Smith |        1 |             3 |
+----|-------|--------|----------|---------------+
5 rows in set (0.00 sec)
```

If you want to get the rows in multiple partitions, you can use a list of partition names which are separated by commas. For example, `SELECT * FROM employees PARTITION (p1 ↪ , p2)` returns all rows in the `p1` and `p2` partitions.

When you use partition selection, you can still use `WHERE` conditions and options such as `ORDER BY` and `LIMIT`. It is also supported to use aggregation options such as `HAVING` and `GROUP BY`.

```
SELECT * FROM employees PARTITION (p0, p2)
    WHERE lname LIKE 'S%';
```

```
+----|-------|-------|----------|---------------+
| id | fname | lname | store_id | department_id |
+----|-------|-------|----------|---------------+
|  4 | Jim   | Smith |        2 |             4 |
| 11 | Jill  | Stone |        1 |             4 |
+----|-------|-------|----------|---------------+
2 rows in set (0.00 sec)
```

```
SELECT id, CONCAT(fname, ' ', lname) AS name
    FROM employees PARTITION (p0) ORDER BY lname;
```

```
+----|----------------+
| id | name           |
+----|----------------+
|  3 | Ellen Johnson  |
|  4 | Jim Smith      |
|  1 | Bob Taylor     |
|  2 | Frank Williams |
+----|----------------+
4 rows in set (0.06 sec)
```

```
SELECT store_id, COUNT(department_id) AS c
    FROM employees PARTITION (p1,p2,p3)
    GROUP BY store_id HAVING c > 4;
```

```
+---|----------+
| c | store_id |
+---|----------+
| 5 |        2 |
| 5 |        3 |
+---|----------+
2 rows in set (0.00 sec)
```

Partition selection is supported for all types of table partitioning, including Range partitioning and Hash partitioning. For Hash partitions, if partition names are not specified, `p0`, `p1`, `p2`,..., or `pN-1` is automatically used as the partition name.

`SELECT` in `INSERT ... SELECT` can also use partition selection.

14.10.12.5 Restrictions and limitations on partitions

This section introduces some restrictions and limitations on partitioned tables in TiDB.

14.10.12.5.1 Partitioning keys, primary keys and unique keys

This section discusses the relationship of partitioning keys with primary keys and unique keys. The rule governing this relationship can be expressed as follows: **Every unique key on the table must use every column in the table's partitioning expression**. This also includes the table's primary key, because it is by definition a unique key.

For example, the following table creation statements are invalid:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
```

```
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2)
)

PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1),
    UNIQUE KEY (col3)
)

PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

In each case, the proposed table has at least one unique key that does not include all columns used in the partitioning expression.

The valid statements are as follows:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2, col3)
)

PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col3)
)

PARTITION BY HASH(col1 + col3)
```

```
PARTITIONS 4;
```

The following example displays an error:

```
CREATE TABLE t3 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2),
    UNIQUE KEY (col3)
)

PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

```
ERROR 1491 (HY000): A PRIMARY KEY must include all columns in the table's
    ↪ partitioning function
```

The `CREATE TABLE` statement fails because both `col1` and `col3` are included in the proposed partitioning key, but neither of these columns is part of both of unique keys on the table. After the following modifications, the `CREATE TABLE` statement becomes valid:

```
CREATE TABLE t3 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2, col3),
    UNIQUE KEY (col1, col3)
)
PARTITION BY HASH(col1 + col3)
    PARTITIONS 4;
```

The following table cannot be partitioned at all, because there is no way to include in a partitioning key any columns that belong to both unique keys:

```
CREATE TABLE t4 (
    col1 INT NOT NULL,
    col2 INT NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col3),
    UNIQUE KEY (col2, col4)
);
```

Because every primary key is by definition a unique key, so the next two statements are invalid:

```
CREATE TABLE t5 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)

PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t6 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col3),
    UNIQUE KEY(col2)
)

PARTITION BY HASH( YEAR(col2) )
PARTITIONS 4;
```

In the above examples, the primary key does not include all columns referenced in the partitioning expression. After adding the missing column in the primary key, the `CREATE TABLE` statement becomes valid:

```
CREATE TABLE t5 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2, col3)
)
PARTITION BY HASH(col3)
PARTITIONS 4;
CREATE TABLE t6 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2, col3),
    UNIQUE KEY(col2)
```

```
)
PARTITION BY HASH( YEAR(col2) )
PARTITIONS 4;
```

If a table has neither unique keys nor primary keys, then this restriction does not apply.

When you change tables using DDL statements, you also need to consider this restriction when adding a unique index. For example, when you create a partitioned table as shown below:

```
CREATE TABLE t_no_pk (c1 INT, c2 INT)
    PARTITION BY RANGE(c1) (
        PARTITION p0 VALUES LESS THAN (10),
        PARTITION p1 VALUES LESS THAN (20),
        PARTITION p2 VALUES LESS THAN (30),
        PARTITION p3 VALUES LESS THAN (40)
    );
```

```
Query OK, 0 rows affected (0.12 sec)
```

You can add a non-unique index by using `ALTER TABLE` statements. But if you want to add a unique index, the `c1` column must be included in the unique index.

When using a partitioned table, you cannot specify the prefix index as a unique attribute:

```
CREATE TABLE t (a varchar(20), b blob,
    UNIQUE INDEX (a(5)))
    PARTITION by range columns (a) (
    PARTITION p0 values less than ('aaaaa'),
    PARTITION p1 values less than ('bbbbb'),
    PARTITION p2 values less than ('ccccc'));
```

```
ERROR 1503 (HY000): A UNIQUE INDEX must include all columns in the table's
    ↪ partitioning function
```

14.10.12.5.2 Partitioning limitations relating to functions

Only the functions shown in the following list are allowed in partitioning expressions:

```
ABS()
CEILING()
DATEDIFF()
DAY()
DAYOFMONTH()
DAYOFWEEK()
DAYOFYEAR()
EXTRACT() (see EXTRACT() function with WEEK specifier)
```

```
FLOOR()
HOUR()
MICROSECOND()
MINUTE()
MOD()
MONTH()
QUARTER()
SECOND()
TIME_TO_SEC()
TO_DAYS()
TO_SECONDS()
UNIX_TIMESTAMP() (with TIMESTAMP columns)
WEEKDAY()
YEAR()
YEARWEEK()
```

14.10.12.5.3 Compatibility with MySQL

Currently, TiDB supports Range partitioning, List partitioning, List COLUMNS partitioning, and Hash partitioning. Other partitioning types that are available in MySQL such as key partitioning are not supported yet in TiDB.

For a table partitioned by `RANGE COLUMNS`, currently TiDB only supports using a single partitioning column.

With regard to partition management, any operation that requires moving data in the bottom implementation is not supported currently, including but not limited to: adjust the number of partitions in a Hash partitioned table, modify the Range of a Range partitioned table, merge partitions and exchange partitions.

For the unsupported partitioning types, when you create a table in TiDB, the partitioning information is ignored and the table is created in the regular form with a warning reported.

The `LOAD DATA` syntax does not support partition selection currently in TiDB.

```
create table t (id int, val int) partition by hash(id) partitions 4;
```

The regular `LOAD DATA` operation is supported:

```
load local data infile "xxx" into t ...
```

But `Load Data` does not support partition selection:

```
load local data infile "xxx" into t partition (p1)...
```

For a partitioned table, the result returned by `select * from t` is unordered between the partitions. This is different from the result in MySQL, which is ordered between the partitions but unordered inside the partitions.

```
create table t (id int, val int) partition by range (id) (
    partition p0 values less than (3),
    partition p1 values less than (7),
    partition p2 values less than (11));
```

```
Query OK, 0 rows affected (0.10 sec)
```

```
insert into t values (1, 2), (3, 4),(5, 6),(7,8),(9,10);
```

```
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

TiDB returns a different result every time, for example:

```
select * from t;
```

```
+------|------+
| id   | val  |
+------|------+
|    7 |    8 |
|    9 |   10 |
|    1 |    2 |
|    3 |    4 |
|    5 |    6 |
+------|------+
5 rows in set (0.00 sec)
```

The result returned in MySQL:

```
select * from t;
```

```
+------|------+
| id   | val  |
+------|------+
|    1 |    2 |
|    3 |    4 |
|    5 |    6 |
|    7 |    8 |
|    9 |   10 |
+------|------+
5 rows in set (0.00 sec)
```

The `tidb_enable_list_partition` environment variable controls whether to enable the partitioned table feature. If this variable is set to `OFF`, the partition information will be ignored when a table is created, and this table will be created as a normal table.

This variable is only used in table creation. After the table is created, modify this variable value takes no effect. For details, see system variables.

14.10.12.5.4 Dynamic pruning mode

TiDB accesses partitioned tables in one of the two modes: `dynamic` mode and `static` mode. Currently, `static` mode is used by default. If you want to enable `dynamic` mode, you need to manually set the `tidb_partition_prune_mode` variable to `dynamic`.

```
set @@session.tidb_partition_prune_mode = 'dynamic'
```

Manual ANALYZE and normal queries use the session-level `tidb_partition_prune_mode` setting. The `auto-analyze` operation in the background uses the global `tidb_partition_prune_mode` setting.

In `static` mode, partitioned tables use partition-level statistics. In `dynamic` mode, partitioned tables use table-level statistics (that is, GlobalStats). For detailed information about GlobalStats, see Collect statistics of partitioned tables in dynamic pruning mode.

When switching from `static` mode to `dynamic` mode, you need to check and collect statistics manually. This is because after the switch to `dynamic` mode, partitioned tables have only partition-level statistics but no table-level statistics. GlobalStats are collected only upon the next `auto-analyze` operation.

```
set session tidb_partition_prune_mode = 'dynamic';
show stats_meta where table_name like "t";
```

```
+---------+------------+---------------+---------------------+-------------+---------
    ↪
| Db_name | Table_name | Partition_name | Update_time | Modify_count |
    ↪ Row_count |
+---------+------------+---------------+---------------------+-------------+---------
    ↪
| test    | t          | p0            | 2022-05-27 20:23:34 |           1 |
    ↪         2 |
| test    | t          | p1            | 2022-05-27 20:23:34 |           2 |
    ↪         4 |
| test    | t          | p2            | 2022-05-27 20:23:34 |           2 |
    ↪         4 |
+---------+------------+---------------+---------------------+-------------+---------
    ↪
3 rows in set (0.01 sec)
```

To make sure that the statistics used by SQL statements are correct after you enable global `dynamic` pruning mode, you need to manually trigger `analyze` on the tables or on a partition of the table to obtain GlobalStats.

```
analyze table t partition p1;
show stats_meta where table_name like "t";
```

```
+---------+------------+----------------+---------------------+--------------+----------
   ↪
| Db_name | Table_name | Partition_name | Update_time | Modify_count |
   ↪ Row_count |
+---------+------------+----------------+---------------------+--------------+----------
   ↪
| test    | t          | global         | 2022-05-27 20:50:53 |        0 |
   ↪         5 |
| test    | t          | p0             | 2022-05-27 20:23:34 |        1 |
   ↪         2 |
| test    | t          | p1             | 2022-05-27 20:50:52 |        0 |
   ↪         2 |
| test    | t          | p2             | 2022-05-27 20:50:08 |        0 |
   ↪         2 |
+---------+------------+----------------+---------------------+--------------+----------
   ↪
4 rows in set (0.00 sec)
```

If the following warning is displayed during the `analyze` process, partition statistics are inconsistent, and you need to collect statistics of these partitions or the entire table again.

```
| Warning | 8244 | Build table: `t` column: `a` global-level stats failed
   ↪ due to missing partition-level column stats, please run analyze table
   ↪  to refresh columns of all partitions
```

You can also use scripts to update statistics of all partitioned tables. For details, see Update statistics of partitioned tables in dynamic pruning mode.

After table-level statistics are ready, you can enable the global dynamic pruning mode, which is effective to all SQL statements and `auto-analyze` operations.

```
set global tidb_partition_prune_mode = dynamic
```

In `static` mode, TiDB accesses each partition separately using multiple operators, and then merges the results using `Union`. The following example is a simple read operation where TiDB merges the results of two corresponding partitions using `Union`:

```
mysql> create table t1(id int, age int, key(id)) partition by range(id) (
    ->     partition p0 values less than (100),
    ->     partition p1 values less than (200),
    ->     partition p2 values less than (300),
    ->     partition p3 values less than (400));
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> explain select * from t1 where id < 150;
```

```
+-----------------------------+----------+-----------+----------------------+--------
   ↪
| id                          | estRows | task     | access object        |
   ↪ operator info              |
+-----------------------------+----------+-----------+----------------------+--------
   ↪
| PartitionUnion_9            | 6646.67 | root     |                      |
   ↪                             |
| ├─TableReader_12            | 3323.33 | root     |                      | data
   ↪ :Selection_11             |
| │ └─Selection_11            | 3323.33 | cop[tikv] |                      | lt(
   ↪ test.t1.id, 150)          |
| │   └─TableFullScan_10      | 10000.00 | cop[tikv] | table:t1, partition:p0
   ↪ | keep order:false, stats:pseudo |
| ├─TableReader_18            | 3323.33 | root     |                      | data
   ↪ :Selection_17             |
| │ └─Selection_17            | 3323.33 | cop[tikv] |                      | lt(
   ↪ test.t1.id, 150)          |
| │   └─TableFullScan_16      | 10000.00 | cop[tikv] | table:t1, partition:p1
   ↪ | keep order:false, stats:pseudo |
+-----------------------------+----------+-----------+----------------------+--------
   ↪
7 rows in set (0.00 sec)
```

In `dynamic` mode, each operator supports direct access to multiple partitions, so TiDB no longer uses `Union`.

```
mysql> set @@session.tidb_partition_prune_mode = 'dynamic';
Query OK, 0 rows affected (0.00 sec)

mysql> explain select * from t1 where id < 150;
+--
   ↪ ----------------------+----------+-----------+----------------+--------------
   ↪
| id                    | estRows | task     | access object | operator info
   ↪              |
+--
   ↪ ----------------------+----------+-----------+----------------+--------------
   ↪
| TableReader_7         | 3323.33 | root     | partition:p0,p1 | data:
   ↪ Selection_6          |
| └─Selection_6         | 3323.33 | cop[tikv] |                | lt(test.t1.id,
```

```
    ↪  150)        |
|    └─TableFullScan_5   | 10000.00 | cop[tikv] | table:t1   | keep order:
    ↪ false, stats:pseudo |
+--
    ↪ ----------------------+---------+----------+----------------+-------------
    ↪
3 rows in set (0.00 sec)
```

From the above query results, you can see that the `Union` operator in the execution plan disappears while the partition pruning still takes effect and the execution plan only accesses `p0` and `p1`.

`dynamic` mode makes execution plans simpler and clearer. Omitting the Union operation can improve the execution efficiency and avoid the problem of Union concurrent execution. In addition, `dynamic` mode also allows execution plans with IndexJoin which cannot be used in `static` mode. (See examples below)

Example 1: In the following example, a query is performed in `static` mode using the execution plan with IndexJoin:

```
mysql> create table t1 (id int, age int, key(id)) partition by range(id)
    -> (partition p0 values less than (100),
    -> partition p1 values less than (200),
    -> partition p2 values less than (300),
    -> partition p3 values less than (400));
Query OK, 0 rows affected (0,08 sec)

mysql> create table t2 (id int, code int);
Query OK, 0 rows affected (0.01 sec)

mysql> set @@tidb_partition_prune_mode = 'static';
Query OK, 0 rows affected (0.00 sec)

mysql> explain select /*+ TIDB_INLJ(t1, t2) */ t1.* from t1, t2 where t2.
    ↪ code = 0 and t2.id = t1.id;
+--
    ↪ ----------------------------+---------+----------+----------------------+-
    ↪
| id                           | estRows | task     | access object        |
    ↪ operator info                                |
+--
    ↪ ----------------------------+---------+----------+----------------------+-
    ↪
| HashJoin_13                  | 12.49   | root     |                      |
    ↪ inner join, equal:[eq(test.t1.id, test.t2.id)] |
|  ─TableReader_42(Build)      | 9.99    | root     |                      |
    ↪ data:Selection_41                            |
```

```
|   └─Selection_41            | 9.99    | cop[tikv] |                    | eq
    ↪ (test.t2.code, 0), not(isnull(test.t2.id)) |
|     └─TableFullScan_40      | 10000.00 | cop[tikv] | table:t2         |
    ↪ keep order:false, stats:pseudo           |
| └─PartitionUnion_15(Probe) | 39960.00 | root    |                    |
    ↪                                          |
|   └─TableReader_18          | 9990.00 | root     |                    |
    ↪ data:Selection_17                        |
|     └─Selection_17          | 9990.00 | cop[tikv] |                    |
    ↪ not(isnull(test.t1.id))                  |
|       └─TableFullScan_16    | 10000.00 | cop[tikv] | table:t1, partition:
    ↪ p0 | keep order:false, stats:pseudo     |
|   └─TableReader_24          | 9990.00 | root     |                    |
    ↪ data:Selection_23                        |
|     └─Selection_23          | 9990.00 | cop[tikv] |                    |
    ↪ not(isnull(test.t1.id))                  |
|       └─TableFullScan_22    | 10000.00 | cop[tikv] | table:t1, partition:
    ↪ p1 | keep order:false, stats:pseudo     |
|   └─TableReader_30          | 9990.00 | root     |                    |
    ↪ data:Selection_29                        |
|     └─Selection_29          | 9990.00 | cop[tikv] |                    |
    ↪ not(isnull(test.t1.id))                  |
|       └─TableFullScan_28    | 10000.00 | cop[tikv] | table:t1, partition:
    ↪ p2 | keep order:false, stats:pseudo     |
|   └─TableReader_36          | 9990.00 | root     |                    |
    ↪ data:Selection_35                        |
|     └─Selection_35          | 9990.00 | cop[tikv] |                    |
    ↪ not(isnull(test.t1.id))                  |
|       └─TableFullScan_34    | 10000.00 | cop[tikv] | table:t1, partition:
    ↪ p3 | keep order:false, stats:pseudo     |
+--
    ↪ ----------------------------+---------+----------+--------------------+-
    ↪
17 rows in set, 1 warning (0.00 sec)

mysql> show warnings;
+--
    ↪ -------+------+-----------------------------------------------------------
    ↪
| Level   | Code | Message
    ↪                                                                  |
+--
    ↪ -------+------+-----------------------------------------------------------
    ↪
| Warning | 1815 | Optimizer Hint /*+ INL_JOIN(t1, t2) */ or /*+ TIDB_INLJ(
```

```
    ↪ t1, t2) */ is inapplicable |
+--
    ↪ -------+------+-----------------------------------------------------------------
    ↪
1 row in set (0,00 sec)
```

From example 1, you can see that even if the `TIDB_INLJ` hint is used, the query on the partitioned table cannot select the execution plan with IndexJoin.

Example 2: In the following example, the query is performed in `dynamic` mode using the execution plan with IndexJoin:

```
mysql> set @@tidb_partition_prune_mode = 'dynamic';
Query OK, 0 rows affected (0.00 sec)

mysql> explain select /*+ TIDB_INLJ(t1, t2) */ t1.* from t1, t2 where t2.
    ↪ code = 0 and t2.id = t1.id;
+--
    ↪ ------------------------------+---------+----------+-----------------------+
    ↪
| id                            | estRows | task     | access object         |
    ↪ operator info
    ↪
    ↪ |
+--
    ↪ ------------------------------+---------+----------+-----------------------+
    ↪
| IndexJoin_11                  | 12.49   | root     |                       |
    ↪ inner join, inner:IndexLookUp_10, outer key:test.t2.id, inner key:
    ↪ test.t1.id, equal cond:eq(test.t2.id, test.t1.id) |
|  ─TableReader_16(Build)       | 9.99    | root     |                       |
    ↪ data:Selection_15
    ↪
    ↪ |
|    ─Selection_15              | 9.99    | cop[tikv] |                      |
    ↪ eq(test.t2.code, 0), not(isnull(test.t2.id))
    ↪                                                                  |
|      ─TableFullScan_14        | 10000.00 | cop[tikv] | table:t2         |
    ↪ keep order:false, stats:pseudo
    ↪
    ↪ |
|  ─IndexLookUp_10(Probe)       | 1.25    | root     | partition:all         |
    ↪
    ↪ |
|    ─Selection_9(Build)        | 1.25    | cop[tikv] |                      |
    ↪ not(isnull(test.t1.id))
```

```
    ↪
    ↪ |
|      └─IndexRangeScan_7       | 1.25    | cop[tikv] | table:t1, index:id(id
    ↪ ) | range: decided by [eq(test.t1.id, test.t2.id)], keep order:false,
    ↪  stats:pseudo                    |
|    └─TableRowIDScan_8(Probe)  | 1.25    | cop[tikv] | table:t1          |
    ↪ keep order:false, stats:pseudo
    ↪
    ↪ |
+--
    ↪ ------------------------------+---------+-----------+--------------------+
    ↪
8 rows in set (0.00 sec)
```

From example 2, you can see that in `dynamic` mode, the execution plan with IndexJoin is selected when you execute the query.

Currently, neither `static` nor `dynamic` pruning mode supports prepared statements plan cache.

Update statistics of partitioned tables in dynamic pruning mode

1. Locate all partitioned tables:

```
SELECT DISTINCT CONCAT(TABLE_SCHEMA,'.', TABLE_NAME)
    FROM information_schema.PARTITIONS
    WHERE TIDB_PARTITION_ID IS NOT NULL
    AND TABLE_SCHEMA NOT IN ('INFORMATION_SCHEMA', 'mysql', 'sys', '
        ↪ PERFORMANCE_SCHEMA', 'METRICS_SCHEMA');
```

```
+-------------------------------------+
| concat(TABLE_SCHEMA,'.',TABLE_NAME) |
+-------------------------------------+
| test.t                              |
+-------------------------------------+
1 row in set (0.02 sec)
```

2. Generate the statements for updating the statistics of all partitioned tables:

```
select distinct concat('ANALYZE TABLE ',TABLE_SCHEMA,'.',TABLE_NAME,'
    ↪ ALL COLUMNS;')
    from information_schema.PARTITIONS
    where TABLE_SCHEMA not in ('INFORMATION_SCHEMA','mysql','sys','
        ↪ PERFORMANCE_SCHEMA','METRICS_SCHEMA');
+--
    ↪ ------------------------------------------------------------------------+
    ↪
```

```
| concat('ANALYZE TABLE ',TABLE_SCHEMA,'.',TABLE_NAME,' ALL COLUMNS;')
    ↪ |
+--
    ↪ ----------------------------------------------------------------------+
    ↪
| ANALYZE TABLE test.t ALL COLUMNS;                                          |
+--
    ↪ ----------------------------------------------------------------------+
    ↪
1 row in set (0.01 sec)
```

You can change `ALL COLUMNS` to the columns you need.

3. Export the batch update statements to a file:

```
mysql --host xxxx --port xxxx -u root -p -e "select distinct concat('
    ↪ ANALYZE TABLE ',TABLE_SCHEMA,'.',TABLE_NAME,' ALL COLUMNS;') \
    from information_schema.PARTITIONS \
    where TABLE_SCHEMA not in ('INFORMATION_SCHEMA','mysql','sys','
       ↪ PERFORMANCE_SCHEMA','METRICS_SCHEMA');" | tee
       ↪ gatherGlobalStats.sql
```

4. Execute a batch update:

   Process SQL statements before executing the `source` command:

```
sed -i "" '1d' gatherGlobalStats.sql --- mac
sed -i '1d' gatherGlobalStats.sql --- linux
```

```
SET session tidb_partition_prune_mode = dynamic;
source gatherGlobalStats.sql
```

14.10.13 Temporary Tables

The temporary tables feature is introduced in TiDB v5.3.0. This feature solves the issue of temporarily storing the intermediate results of an application, which frees you from frequently creating and dropping tables. You can store the intermediate calculation data in temporary tables. When the intermediate data is no longer needed, TiDB automatically cleans up and recycles the temporary tables. This avoids user applications being too complicated, reduces table management overhead, and improves performance.

This document introduces the user scenarios and the types of temporary tables, provides usage examples and instruction on how to limit the memory usage of temporary tables, and explains compatibility restrictions with other TiDB features.

14.10.13.1 User scenarios

You can use TiDB temporary tables in the following scenarios:

- Cache the intermediate temporary data of an application. After the calculation is completed, the data is dumped to the ordinary table, and the temporary table is automatically released.
- Perform multiple DML operations on the same data in a short period of time. For example, in an e-commerce shopping cart application, add, modify, and delete products, complete the payment, and remove the shopping cart information.
- Quickly import intermediate temporary data in batches to improve the performance of importing temporary data.
- Update data in batches. Import data into temporary tables in the database in batches, and export the data to files after finishing modify the data.

14.10.13.2 Types of temporary tables

Temporary tables in TiDB are divided into two types: local temporary tables and global temporary tables.

- For a local temporary table, the table definition and data in the table are visible only to the current session. This type is suitable for temporarily storing intermediate data in the session.
- For a global temporary table, the table definition is visible to the entire TiDB cluster, and the data in the table is visible only to the current transaction. This type is suitable for temporarily storing intermediate data in the transaction.

14.10.13.3 Local temporary tables

The semantics of the local temporary table in TiDB is consistent with that of the MySQL temporary table. The characteristics are as follows:

- The table definition of a local temporary table is not persistent. A local temporary table is visible only to the session in which the table is created, and other sessions cannot access the table.
- You can create local temporary tables with the same name in different sessions, and each session reads only from and writes only to the local temporary table created in the session.
- The data of a local temporary table is visible to all transactions in the session.
- After a session ends, the local temporary table created in the session is automatically dropped.
- A local temporary table can have the same name as an ordinary table. In this case, in the DDL and DML statements, the ordinary table is hidden until the local temporary table is dropped.

To create a local temporary table, you can use the `CREATE TEMPORARY TABLE` statement. To drop a local temporary table, you can use the `DROP TABLE` or `DROP TEMPORARY TABLE` statement.

Different from MySQL, the local temporary tables in TiDB are all external tables, and no internal temporary tables will be created automatically when SQL statements are executed.

14.10.13.3.1 Usage examples of local temporary tables

> **Note:**
>
> - Before you use the temporary table in TiDB, pay attention to the compatibility restrictions with other TiDB features and the compatibility with MySQL temporary tables.
> - If you have created local temporary tables on a cluster earlier than TiDB v5.3.0, these tables are actually ordinary tables, and treated as ordinary tables after the cluster is upgraded to TiDB v5.3.0 or a later version.

Assume that there is an ordinary table `users`:

```
CREATE TABLE users (
    id BIGINT,
    name VARCHAR(100),
    PRIMARY KEY(id)
);
```

In session A, creating a local temporary table `users` does not conflict with the ordinary table `users`. When session A accesses the `users` table, it accesses the local temporary table `users`.

```
CREATE TEMPORARY TABLE users (
    id BIGINT,
    name VARCHAR(100),
    city VARCHAR(50),
    PRIMARY KEY(id)
);
```

```
Query OK, 0 rows affected (0.01 sec)
```

If you insert data into `users`, data is inserted to the local temporary table `users` in session A.

```
INSERT INTO users(id, name, city) VALUES(1001, 'Davis', 'LosAngeles');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
SELECT * FROM users;
```

```
+------+-------+------------+
| id   | name | city       |
+------+-------+------------+
| 1001 | Davis | LosAngeles |
+------+-------+------------+
1 row in set (0.00 sec)
```

In session B, creating a local temporary table `users` does not conflict with the ordinary table `users` or the local temporary table `users` in session A. When session B accesses the `users` table, it accesses the local temporary table `users` in session B.

```
CREATE TEMPORARY TABLE users (
    id BIGINT,
    name VARCHAR(100),
    city VARCHAR(50),
    PRIMARY KEY(id)
);
```

```
Query OK, 0 rows affected (0.01 sec)
```

If you insert data into `users`, data is inserted to the local temporary table `users` in session B.

```
INSERT INTO users(id, name, city) VALUES(1001, 'James', 'NewYork');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
SELECT * FROM users;
```

```
+------+-------+---------+
| id   | name | city    |
+------+-------+---------+
| 1001 | James | NewYork |
+------+-------+---------+
1 row in set (0.00 sec)
```

14.10.13.3.2 Compatibility with MySQL temporary tables

The following features and limitations of TiDB local temporary tables are the same with those of MySQL temporary tables:

- When you create or drop local temporary tables, the current transaction is not automatically committed.
- After dropping the schema where a local temporary table is located, the temporary table is not dropped and is still readable and writable.
- Creating a local temporary table requires the `CREATE TEMPORARY TABLES` permission. All subsequent operations on the table do not require any permission.
- Local temporary tables do not support foreign keys and partitioned tables.
- Does not support creating views based on local temporary tables.
- `SHOW [FULL] TABLES` does not show local temporary tables.

Local temporary tables in TiDB are incompatible with MySQL temporary tables in the following aspects:

- TiDB local temporary tables do not support `ALTER TABLE`.
- TiDB local temporary tables ignore the `ENGINE` table option, and always store temporary table data in TiDB memory with a memory limit.
- When `MEMORY` is declared as the storage engine, TiDB local temporary tables are not restricted by the `MEMORY` storage engine.
- When `INNODB` or `MYISAM` is declared as the storage engine, TiDB local temporary tables ignore the system variables specific to the InnoDB temporary tables.
- MySQL does not permit referencing to the same temporary table multiple times in the same SQL statement. TiDB local temporary tables do not have this restriction.
- The system table `information_schema.INNODB_TEMP_TABLE_INFO` that shows temporary tables in MySQL does not exist in TiDB. Currently, TiDB does not have a system table that shows local temporary tables.
- TiDB does not have internal temporary tables. The MySQL system variables for internal temporary tables do not take effect for TiDB.

14.10.13.4 Global temporary tables

The global temporary table is an extension of TiDB. The characteristics are as follows:

- The table definition of a global temporary table is persistent and visible to all sessions.
- The data of a global temporary table is visible only in the current transaction. When the transaction ends, the data is automatically cleared.
- A global temporary table cannot have the same name as an ordinary table.

To create a global temporary table, you can use the `CREATE GLOBAL TEMPORARY TABLE` statement ended with `ON COMMIT DELETE ROWS`. To drop a global temporary table, you can use the `DROP TABLE` or `DROP GLOBAL TEMPORARY TABLE` statement.

14.10.13.4.1 Usage examples of global temporary tables

Note:

- Before you use the temporary table in TiDB, pay attention to the compatibility restrictions with other TiDB features.
- If you have created global temporary tables on a TiDB cluster of v5.3.0 or later, when the cluster is downgraded to a version earlier than v5.3.0, these tables are handled as ordinary tables. In this case, a data error occurs.

Create a global temporary table `users` in session A:

```
CREATE GLOBAL TEMPORARY TABLE users (
    id BIGINT,
    name VARCHAR(100),
    city VARCHAR(50),
    PRIMARY KEY(id)
) ON COMMIT DELETE ROWS;
```

```
Query OK, 0 rows affected (0.01 sec)
```

The data written to `users` is visible to the current transaction:

```
BEGIN;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
INSERT INTO users(id, name, city) VALUES(1001, 'Davis', 'LosAngeles');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
SELECT * FROM users;
```

```
+------+-------+------------+
| id   | name | city       |
+------+-------+------------+
| 1001 | Davis | LosAngeles |
+------+-------+------------+
1 row in set (0.00 sec)
```

After the transaction ends, the data is automatically cleared:

```
COMMIT;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
SELECT * FROM users;
```

```
Empty set (0.00 sec)
```

After `users` is created in session A, session B can also read from and write to the `users` table:

```
SELECT * FROM users;
```

```
Empty set (0.00 sec)
```

> **Note:**
>
> If the transaction is automatically committed, after the SQL statement is executed, the inserted data is automatically cleared and unavailable to subsequent SQL executions. Therefore, you should use non-autocommit transactions to read from and write to global temporary tables.

14.10.13.5 Limit the memory usage of temporary tables

No matter which storage engine is declared as `ENGINE` when you define a table, the data of local temporary tables and global temporary tables is only stored in the memory of TiDB instances. This data is not persisted.

To avoid memory overflow, you can limit the size of each temporary table using the `tidb_tmp_table_max_size` system variable. Once a temporary table is larger than the `tidb_tmp_table_max_size` threshold value, TiDB reports an error. The default value of `tidb_tmp_table_max_size` is `64MB`.

For example, set the maximum size of a temporary table to `256MB`:

```
SET GLOBAL tidb_tmp_table_max_size=268435456;
```

14.10.13.6 Compatibility restrictions with other TiDB features

Local temporary tables and global temporary tables in TiDB are **NOT** compatible with the following TiDB features:

- `AUTO_RANDOM` columns
- `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` table options

- Partitioned tables
- `SPLIT REGION` statements
- `ADMIN CHECK TABLE` and `ADMIN CHECKSUM TABLE` statements
- `FLASHBACK TABLE` and `RECOVER TABLE` statements
- Executing `CREATE TABLE LIKE` statements based on a temporary table
- Stale Read
- Foreign keys
- SQL bindings
- TiFlash replicas
- Creating views on a temporary table
- Placement Rules
- Execution plans involving a temporary table are not cached by `prepare plan cache`.

Local temporary tables in TiDB do **NOT** support the following feature:

- Reading historical data using the `tidb_snapshot` system variable.

14.10.13.7 TiDB migration tool support

Local temporary tables are not exported, backed up or replicated by TiDB migration tools, because these tables are visible only to the current session.

Global temporary tables are exported, backed up, and replicated by TiDB migration tools, because the table definition is globally visible. Note that the data on the tables are not exported.

> **Note:**
>
> - Replicating temporary tables using TiCDC requires TiCDC v5.3.0 or later. Otherwise, the table definition of the downstream table is wrong.
> - Backing up temporary tables using BR requires BR v5.3.0 or later. Otherwise, the table definitions of the backed temporary tables are wrong.
> - The cluster to export, the cluster after data restore, and the downstream cluster of a replication should support global temporary tables. Otherwise, an error is reported.

14.10.13.8 See also

- CREATE TABLE
- CREATE TABLE LIKE
- DROP TABLE

14.10.14 Cached Tables

In v6.0.0, TiDB introduces the cached table feature for frequently accessed but rarely updated small hotspot tables. When this feature is used, the data of an entire table is loaded into the memory of the TiDB server, and TiDB directly gets the table data from the memory without accessing TiKV, which improves the read performance.

This document describes the usage scenarios of cached tables, the examples, and the compatibility restrictions with other TiDB features.

14.10.14.1 Usage scenarios

The cached table feature is suitable for tables with the following characteristics:

- The data volume of the table is small.
- The table is read-only or rarely updated.
- The table is frequently accessed, and you expect a better read performance.

When the data volume of the table is small but the data is frequently accessed, the data is concentrated on a Region in TiKV and makes it a hotspot Region, which affects the performance. Therefore, the typical usage scenarios of cached tables are as follows:

- Configuration tables, from which applications read the configuration information.
- The tables of exchange rates in the financial sector. These tables are updated only once a day but not in real-time.
- Bank branch or network information tables, which are rarely updated.

Take configuration tables as an example. When the application restarts, the configuration information is loaded in all connections, which causes a high read latency. In this case, you can solve this problem by using the cached tables feature.

14.10.14.2 Examples

This section describes the usage of cached tables by examples.

14.10.14.2.1 Set a normal table to a cached table

Suppose that there is a table `users`:

```
CREATE TABLE users (
    id BIGINT,
    name VARCHAR(100),
    PRIMARY KEY(id)
);
```

To set this table to a cached table, use the `ALTER TABLE` statement:

```
ALTER TABLE users CACHE;
```

```
Query OK, 0 rows affected (0.01 sec)
```

14.10.14.2.2 Verify a cached table

To verify a cached table, use the `SHOW CREATE TABLE` statement. If the table is cached, the returned result contains the `CACHED ON` attribute:

```
SHOW CREATE TABLE users;
```

```
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| Table | Create Table
    ↪
    ↪ |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
| users | CREATE TABLE `users` (
  `id` bigint(20) NOT NULL,
  `name` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`) /*T![clustered_index] CLUSTERED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin /* CACHED ON */
    ↪  |
+--
    ↪ -----+-------------------------------------------------------------------------
    ↪
1 row in set (0.00 sec)
```

After reading data from a cached table, TiDB loads the data in memory. You can use the `trace` statement to check whether the data is loaded into memory. When the cache is not loaded, the returned result contains the `regionRequest.SendReqCtx` attribute, which indicates that TiDB reads data from TiKV.

```
TRACE SELECT * FROM users;
```

```
+--
    ↪ ---------------------------------------------+----------------+------------+
    ↪
| operation                                    | startTS        | duration |
+--
    ↪ ---------------------------------------------+----------------+------------+
    ↪
```

```
| trace                                      | 17:47:39.969980 | 827.73µs |
|   ─session.ExecuteStmt                      | 17:47:39.969986 | 413.31µs |
|     ─executor.Compile                       | 17:47:39.969993 | 198.29µs |
|     ─session.runStmt                        | 17:47:39.970221 | 157.252µs |
|       ─TableReaderExecutor.Open             | 17:47:39.970294 | 47.068µs |
|         ─distsql.Select                     | 17:47:39.970312 | 24.729µs |
|           ─regionRequest.SendReqCtx         | 17:47:39.970454 | 189.601µs |
|   ─* executor.UnionScanExec.Next            | 17:47:39.970407 | 353.073µs |
|     ─* executor.TableReaderExecutor.Next    | 17:47:39.970411 | 301.106µs |
|     ─* executor.TableReaderExecutor.Next    | 17:47:39.970746 | 6.57µs |
|   ─* executor.UnionScanExec.Next            | 17:47:39.970772 | 17.589µs |
|     ─* executor.TableReaderExecutor.Next    | 17:47:39.970776 | 6.59µs |
+--
   ↪ ---------------------------------------------+-----------------+-----------+
   ↪
12 rows in set (0.01 sec)
```

After executing `trace` again, the returned result no longer contains the `regionRequest.SendReqCtx` attribute, which indicates that TiDB no longer reads data from TiKV but reads data from the memory instead.

```
+---------------------------------------+-----------------+-----------+
| operation                             | startTS         | duration  |
+---------------------------------------+-----------------+-----------+
| trace                                 | 17:47:40.533888 | 453.547µs |
|   ─session.ExecuteStmt                | 17:47:40.533894 | 402.341µs |
|     ─executor.Compile                 | 17:47:40.533903 | 205.54µs |
|     ─session.runStmt                  | 17:47:40.534141 | 132.084µs |
|       ─TableReaderExecutor.Open       | 17:47:40.534202 | 14.749µs |
|   ─* executor.UnionScanExec.Next      | 17:47:40.534306 | 3.21µs |
|   ─* executor.UnionScanExec.Next      | 17:47:40.534316 | 1.219µs |
+---------------------------------------+-----------------+-----------+
7 rows in set (0.00 sec)
```

Note that the `UnionScan` operator is used to read the cached tables, so you can see `UnionScan` in the execution plan of the cached tables through `explain`:

```
+--
   ↪ ----------------------+---------+-----------+--------------+-----------------
   ↪
| id                     | estRows | task    | access object | operator info
   ↪            |
+--
   ↪ ----------------------+---------+-----------+--------------+-----------------
   ↪
```

```
| UnionScan_5          | 1.00   | root     |              |
   ↪                              |
|  ─TableReader_7      | 1.00   | root     |              | data:
   ↪ TableFullScan_6       |
|    ─TableFullScan_6  | 1.00   | cop[tikv] | table:users | keep order:
   ↪ false, stats:pseudo |
+--
   ↪ -----------------------+---------+-----------+---------------+-------------------
   ↪
3 rows in set (0.00 sec)
```

14.10.14.2.3 Write data to a cached table

Cached tables support data writes. For example, you can insert a record into the `users` table:

```
INSERT INTO users(id, name) VALUES(1001, 'Davis');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
SELECT * FROM users;
```

```
+------+-------+
| id   | name |
+------+-------+
| 1001 | Davis |
+------+-------+
1 row in set (0.00 sec)
```

Note:

When you insert data to a cached table, second-level write latency might occur. The latency is controlled by the global environment variable `tidb_table_cache_lease`. You can decide whether to use the cached table feature by checking whether the latency is acceptable based on your application. For example, in a read-only scenario, you can increase the value of `tidb_table_cache_lease`:

```
set @@global.tidb_table_cache_lease = 10;
```

The write latency of cached tables is high, because the cached table feature is implemented with a complex mechanism that requires a lease to be set for each cache. When there are multiple TiDB instances, one instance does not

know whether the other instances have cached data. If an instance modifies the table data directly, the other instances read the old cache data. To ensure correctness, the cached table implementation uses a lease mechanism to ensure that the data is not modified before the lease expires. That is why the write latency is high.

14.10.14.2.4 Revert a cached table to a normal table

Note:

Executing DDL statements on a cached table will fail. Before executing DDL statements on a cached table, you need to remove the cache attribute first and set the cached table back to a normal table.

```
TRUNCATE TABLE users;
```

```
ERROR 8242 (HY000): 'Truncate Table' is unsupported on cache tables.
```

```
mysql> ALTER TABLE users ADD INDEX k_id(id);
```

```
ERROR 8242 (HY000): 'Alter Table' is unsupported on cache tables.
```

To revert a cached table to a normal table, use `ALTER TABLE t NOCACHE`:

```
ALTER TABLE users NOCACHE;
```

```
Query OK, 0 rows affected (0.00 sec)
```

14.10.14.3 Size limit of cached tables

Cached tables are only suitable for scenarios with small tables, because TiDB loads the data of an entire table into memory, and the cached data becomes invalid after modification and needs to be reloaded.

Currently, the size limit of a cached table is 64 MB in TiDB. If the table data exceeds 64 MB, executing `ALTER TABLE t CACHE` will fail.

14.10.14.4 Compatibility restrictions with other TiDB features

Cached tables **DO NOT** support the following features:

- Performing the `ALTER TABLE t ADD PARTITION` operation on partitioned tables is not supported.
- Performing the `ALTER TABLE t CACHE` operation on temporary tables is not supported.
- Performing the `ALTER TABLE t CACHE` operation on views is not supported.
- Stale Read is not supported.
- Direct DDL operations on a cached table are not supported. You need to set the cached table back to a normal table first by using `ALTER TABLE t NOCACHE` before performing DDL operations.

Cached tables **CANNOT** be used in the following scenarios:

- Setting the system variable `tidb_snapshot` to read historical data.
- During modification, the cached data becomes invalid until the data is reloaded.

14.10.14.5 Compatibility with TiDB migration tools

The cached table is a TiDB extension to MySQL syntax. Only TiDB can recognize the `ALTER TABLE ... CACHE` statement. TiDB migration tools **DO NOT** support cached tables, including Backup & Restore (BR), TiCDC, and Dumpling. These tools treat cached tables as normal tables.

That is to say, when a cached table is backed up and restored, it becomes a normal table. If the downstream cluster is a different TiDB cluster and you want to continue using the cached table feature, you can manually enable cached tables on the downstream cluster by executing `ALTER TABLE ... CACHE` on the downstream table.

14.10.14.6 See also

- ALTER TABLE
- System Variables

14.10.15 Character Set and Collation

14.10.15.1 Character Set and Collation

This document introduces the character sets and collations supported by TiDB.

14.10.15.1.1 Concepts

A character set is a set of symbols and encodings. The default character set in TiDB is utf8mb4, which matches the default in MySQL 8.0 and above.

A collation is a set of rules for comparing characters in a character set, and the sorting order of characters. For example in a binary collation `A` and `a` do not compare as equal:

```
SET NAMES utf8mb4 COLLATE utf8mb4_bin;
SELECT 'A' = 'a';
SET NAMES utf8mb4 COLLATE utf8mb4_general_ci;
SELECT 'A' = 'a';
```

```
SELECT 'A' = 'a';
```

```
+-----------+
| 'A' = 'a' |
+-----------+
|         0 |
+-----------+
1 row in set (0.00 sec)
```

```
SET NAMES utf8mb4 COLLATE utf8mb4_general_ci;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
SELECT 'A' = 'a';
```

```
+-----------+
| 'A' = 'a' |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)
```

TiDB defaults to using a binary collation. This differs from MySQL, which uses a case-insensitive collation by default.

14.10.15.1.2 Character sets and collations supported by TiDB

Currently, TiDB supports the following character sets:

```
SHOW CHARACTER SET;
```

```
+--
    ↪ -------+-----------------------------------+------------------+---------+
    ↪
```

```
| Charset | Description                     | Default collation | Maxlen |
+--
    ↪ -------+---------------------------------+-------------------+--------+
    ↪
| ascii  | US ASCII                        | ascii_bin       |      1 |
| binary | binary                          | binary          |      1 |
| gbk    | Chinese Internal Code Specification | gbk_chinese_ci | 2 |
| latin1 | Latin1                          | latin1_bin      |      1 |
| utf8   | UTF-8 Unicode                   | utf8_bin        |      3 |
| utf8mb4 | UTF-8 Unicode                  | utf8mb4_bin     |      4 |
+--
    ↪ -------+---------------------------------+-------------------+--------+
    ↪
6 rows in set (0.00 sec)
```

TiDB supports the following collations:

```
SHOW COLLATION;
```

```
+--------------------+---------+------+---------+----------+---------+
| Collation          | Charset | Id | Default | Compiled | Sortlen |
+--------------------+---------+------+---------+----------+---------+
| ascii_bin          | ascii   |   65 | Yes     | Yes      |       1 |
| binary             | binary  |   63 | Yes     | Yes      |       1 |
| gbk_bin            | gbk     |   87 |         | Yes      |       1 |
| gbk_chinese_ci     | gbk     |   28 | Yes     | Yes      |       1 |
| latin1_bin         | latin1  |   47 | Yes     | Yes      |       1 |
| utf8_bin           | utf8    |   83 | Yes     | Yes      |       1 |
| utf8_general_ci    | utf8    |   33 |         | Yes      |       1 |
| utf8_unicode_ci    | utf8    |  192 |         | Yes      |       1 |
| utf8mb4_bin        | utf8mb4 |  46 | Yes     | Yes      |       1 |
| utf8mb4_general_ci | utf8mb4 | 45 |      | Yes      |       1 |
| utf8mb4_unicode_ci | utf8mb4 | 224 |     | Yes      |       1 |
+--------------------+---------+------+---------+----------+---------+
11 rows in set (0.00 sec)
```

Warning:

TiDB incorrectly treats latin1 as a subset of utf8. This can lead to unexpected behaviors when you store characters that differ between latin1 and utf8 encodings. It is strongly recommended to the utf8mb4 character set. See TiDB #18955 for more details.

Note:

The default collations in TiDB (binary collations, with the suffix `_bin`) are different than the default collations in MySQL (typically general collations, with the suffix `_general_ci`). This can cause incompatible behavior when specifying an explicit character set but relying on the implicit default collation to be chosen.

You can use the following statement to view the collations (under the new framework for collations) that corresponds to the character set.

```
SHOW COLLATION WHERE Charset = 'utf8mb4';
```

```
+--------------------+---------+------+---------+----------+---------+
| Collation          | Charset | Id   | Default | Compiled | Sortlen |
+--------------------+---------+------+---------+----------+---------+
| utf8mb4_bin        | utf8mb4 | 46   | Yes     | Yes      |       1 |
| utf8mb4_general_ci | utf8mb4 | 45   |         | Yes      |       1 |
| utf8mb4_unicode_ci | utf8mb4 | 224  |         | Yes      |       1 |
+--------------------+---------+------+---------+----------+---------+
3 rows in set (0.00 sec)
```

For details about the TiDB support of the GBK character set, see GBK.

14.10.15.1.3 `utf8` and `utf8mb4` in TiDB

In MySQL, the character set `utf8` is limited to a maximum of three bytes. This is sufficient to store characters in the Basic Multilingual Plane (BMP), but not enough to store characters such as emojis. For this, it is recommended to use the character set `utf8mb4` instead.

By default, TiDB provides the same 3-byte limit on `utf8` to ensure that data created in TiDB can still safely be restored in MySQL. This can be disabled by changing the value of `check-mb4-value-in-utf8` to `FALSE` in your TiDB configuration file.

The following demonstrates the default behavior when inserting a 4-byte emoji character into a table. The `INSERT` statement fails for the `utf8` character set, but succeeds for `utf8mb4`:

```
CREATE TABLE utf8_test (
    -> c char(1) NOT NULL
    -> ) CHARACTER SET utf8;
```

```
Query OK, 0 rows affected (0.09 sec)
```

```
CREATE TABLE utf8m4_test (
    -> c char(1) NOT NULL
    -> ) CHARACTER SET utf8mb4;
```

```
Query OK, 0 rows affected (0.09 sec)
```

```
INSERT INTO utf8_test VALUES ('□');
```

```
ERROR 1366 (HY000): incorrect utf8 value f09f9889(□) for column c
```

```
INSERT INTO utf8m4_test VALUES ('□');
```

```
Query OK, 1 row affected (0.02 sec)
```

```
SELECT char_length(c), length(c), c FROM utf8_test;
```

```
Empty set (0.01 sec)
```

```
SELECT char_length(c), length(c), c FROM utf8m4_test;
```

```
+----------------+-----------+------+
| char_length(c) | length(c) | c |
+----------------+-----------+------+
|              1 |         4 | □    |
+----------------+-----------+------+
1 row in set (0.00 sec)
```

14.10.15.1.4 Character set and collation in different layers

The character set and collation can be set at different layers.

Database character set and collation

Each database has a character set and a collation. You can use the following statements to specify the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

`DATABASE` can be replaced with `SCHEMA` here.

Different databases can use different character sets and collations. Use the `character_set_database` and `collation_database` to see the character set and collation of the current database:

```
CREATE SCHEMA test1 CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

```
Query OK, 0 rows affected (0.09 sec)
```

```
USE test1;
```

```
Database changed
```

```
SELECT @@character_set_database, @@collation_database;
```

```
+--------------------------/----------------------+
| @@character_set_database | @@collation_database |
+--------------------------/----------------------+
| utf8mb4                  | utf8mb4_general_ci   |
+--------------------------/----------------------+
1 row in set (0.00 sec)
```

```
CREATE SCHEMA test2 CHARACTER SET latin1 COLLATE latin1_bin;
```

```
Query OK, 0 rows affected (0.09 sec)
```

```
USE test2;
```

```
Database changed
```

```
SELECT @@character_set_database, @@collation_database;
```

```
+--------------------------/----------------------+
| @@character_set_database | @@collation_database |
+--------------------------/----------------------+
| latin1                   | latin1_bin           |
+--------------------------/----------------------+
1 row in set (0.00 sec)
```

You can also see the two values in `INFORMATION_SCHEMA`:

```
SELECT DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'db_name';
```

Table character set and collation

You can use the following statement to specify the character set and collation for tables:

```
CREATE TABLE tbl_name (column_list)
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]]

ALTER TABLE tbl_name
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]
```

For example:

```
CREATE TABLE t1(a int) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

```
Query OK, 0 rows affected (0.08 sec)
```

If the table character set and collation are not specified, the database character set and collation are used as their default values.

Column character set and collation

You can use the following statement to specify the character set and collation for columns:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]

col_name {ENUM | SET} (val_list)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

If the column character set and collation are not specified, the table character set and collation are used as their default values.

String character sets and collation

Each string corresponds to a character set and a collation. When you use a string, this option is available:

```
[_charset_name]'string' [COLLATE collation_name]
```

Example:

```
SELECT 'string';
SELECT _utf8mb4'string';
SELECT _utf8mb4'string' COLLATE utf8mb4_general_ci;
```

Rules:

- Rule 1: If you specify `CHARACTER SET charset_name` and `COLLATE collation_name` ↪ , then the `charset_name` character set and the `collation_name` collation are used directly.
- Rule 2: If you specify `CHARACTER SET charset_name` but do not specify `COLLATE` ↪ `collation_name`, the `charset_name` character set and the default collation of `charset_name` are used.
- Rule 3: If you specify neither `CHARACTER SET charset_name` nor `COLLATE` ↪ `collation_name`, the character set and collation given by the system variables `character_set_connection` and `collation_connection` are used.

Client connection character set and collation

- The server character set and collation are the values of the `character_set_server` and `collation_server` system variables.
- The character set and collation of the default database are the values of the `character_set_database` and `collation_database` system variables.

You can use `character_set_connection` and `collation_connection` to specify the character set and collation for each connection. The `character_set_client` variable is to set the client character set.

Before returning the result, the `character_set_results` system variable indicates the character set in which the server returns query results to the client, including the metadata of the result.

You can use the following statement to set the character set and collation that is related to the client:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

  `SET NAMES` indicates what character set the client will use to send SQL statements to the server. `SET NAMES utf8mb4` indicates that all the requests from the client use utf8mb4, as well as the results from the server.

  The `SET NAMES 'charset_name'` statement is equivalent to the following statement combination:

```
SET character_set_client = charset_name;
SET character_set_results = charset_name;
SET character_set_connection = charset_name;
```

  `COLLATE` is optional, if absent, the default collation of the `charset_name` is used to set the `collation_connection`.

- `SET CHARACTER SET 'charset_name'`

  Similar to `SET NAMES`, the `SET NAMES 'charset_name'` statement is equivalent to the following statement combination:

```
SET character_set_client = charset_name;
SET character_set_results = charset_name;
SET charset_connection = @@charset_database;
SET collation_connection = @@collation_database;
```

14.10.15.1.5 Selection priorities of character sets and collations

String > Column > Table > Database > Server

14.10.15.1.6 General rules on selecting character sets and collation

- Rule 1: If you specify `CHARACTER SET charset_name` and `COLLATE collation_name` ↪ , then the `charset_name` character set and the `collation_name` collation are used directly.
- Rule 2: If you specify `CHARACTER SET charset_name` and do not specify `COLLATE` ↪ `collation_name`, then the `charset_name` character set and the default collation of `charset_name` are used.
- Rule 3: If you specify neither `CHARACTER SET charset_name` nor `COLLATE` ↪ `collation_name`, the character set and collation with higher optimization levels are used.

14.10.15.1.7 Validity check of characters

If the specified character set is `utf8` or `utf8mb4`, TiDB only supports the valid `utf8` characters. For invalid characters, TiDB reports the `incorrect utf8 value` error. This validity check of characters in TiDB is compatible with MySQL 8.0 but incompatible with MySQL 5.7 or earlier versions.

To disable this error reporting, use `set @@tidb_skip_utf8_check=1;` to skip the character check.

> **Note:**
>
> If the character check is skipped, TiDB might fail to detect illegal UTF-8 characters written by the application, cause decoding errors when `ANALYZE` is executed, and introduce other unknown encoding issues. If your application cannot guarantee the validity of the written string, it is not recommended to skip the character check.

14.10.15.1.8 Collation support framework

The syntax support and semantic support for the collation are influenced by the new_collations_enabled_on_first_bootstrap configuration item. The syntax support and semantic support are different. The former indicates that TiDB can parse and set collations. The latter indicates that TiDB can correctly use collations when comparing strings.

Before v4.0, TiDB provides only the old framework for collations. In this framework, TiDB supports syntactically parsing most of the MySQL collations but semantically takes all collations as binary collations.

Since v4.0, TiDB supports a new framework for collations. In this framework, TiDB semantically parses different collations and strictly follows the collations when comparing strings.

Old framework for collations

Before v4.0, you can specify most of the MySQL collations in TiDB, and these collations are processed according to the default collations, which means that the byte order determines the character order. Different from MySQL, TiDB does not handle the trailing spaces of a character, which causes the following behavior differences:

```
CREATE TABLE t(a varchar(20) charset utf8mb4 collate utf8mb4_general_ci
    ↪ PRIMARY KEY);
```

```
Query OK, 0 rows affected
```

```
INSERT INTO t VALUES ('A');
```

```
Query OK, 1 row affected
```

```
INSERT INTO t VALUES ('a');
```

```
Query OK, 1 row affected
```

In TiDB, the preceding statement is successfully executed. In MySQL, because utf8mb4_general_ci is case-insensitive, the Duplicate entry 'a' error is reported.

```
INSERT INTO t1 VALUES ('a ');
```

```
Query OK, 1 row affected
```

In TiDB, the preceding statement is successfully executed. In MySQL, because comparison is performed after the spaces are filled in, the Duplicate entry 'a ' error is returned.

New framework for collations

Since TiDB v4.0, a complete framework for collations is introduced.

This new framework supports semantically parsing collations and introduces the `new_collations_enabled_on_first_bootstrap` configuration item to decide whether to enable the new framework when a cluster is first initialized. To enable the new framework, set `new_collations_enabled_on_first_bootstrap` to `true`. For details, see new_collations_enabled_on_first_bootstrap. If you initialize the cluster after the configuration item is enabled, you can check whether the new collation is enabled through the `new_collation_enabled` variable in the `mysql.tidb` table:

```
SELECT VARIABLE_VALUE FROM mysql.tidb WHERE VARIABLE_NAME='
    ↪ new_collation_enabled';
```

```
+----------------+
| VARIABLE_VALUE |
+----------------+
| True           |
+----------------+
1 row in set (0.00 sec)
```

Under the new framework, TiDB supports the `utf8_general_ci`, `utf8mb4_general_ci`, `utf8_unicode_ci`, `utf8mb4_unicode_ci`, `gbk_chinese_ci`, and `gbk_bin` collations, which is compatible with MySQL.

When one of `utf8_general_ci`, `utf8mb4_general_ci`, `utf8_unicode_ci`, `utf8mb4_unicode_ci` ↪ , and `gbk_chinese_ci` is used, the string comparison is case-insensitive and accent-insensitive. At the same time, TiDB also corrects the collation's `PADDING` behavior:

```
CREATE TABLE t(a varchar(20) charset utf8mb4 collate utf8mb4_general_ci
    ↪ PRIMARY KEY);
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
INSERT INTO t VALUES ('A');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
INSERT INTO t VALUES ('a');
```

```
ERROR 1062 (23000): Duplicate entry 'a' for key 'PRIMARY' # TiDB is
    ↪ compatible with the case-insensitive collation of MySQL.
```

```
INSERT INTO t VALUES ('a ');
```

```
ERROR 1062 (23000): Duplicate entry 'a ' for key 'PRIMARY' # TiDB modifies
    ↪ the `PADDING` behavior to be compatible with MySQL.
```

Note:

The implementation of padding in TiDB is different from that in MySQL. In MySQL, padding is implemented by filling in spaces. In TiDB, padding is implemented by cutting out the spaces at the end. The two approaches are the same in most cases. The only exception is when the end of the string contains characters that are less than spaces (0x20). For example, the result of `'a' < 'a\t'` in TiDB is `1`, but in MySQL, `'a' < 'a\t'` is equivalent to `'a ' < 'a\t'`, and the result is `0`.

14.10.15.1.9 Coercibility values of collations in expressions

If an expression involves multiple clauses of different collations, you need to infer the collation used in the calculation. The rules are as follows:

- The coercibility value of the explicit `COLLATE` clause is `0`.
- If the collations of two strings are incompatible, the coercibility value of the concatenation of two strings with different collations is `1`.
- The collation of the column, `CAST()`, `CONVERT()`, or `BINARY()` has a coercibility value of `2`.
- The system constant (the string returned by `USER ()` or `VERSION ()`) has a coercibility value of `3`.
- The coercibility value of constants is `4`.
- The coercibility value of numbers or intermediate variables is `5`.
- `NULL` or expressions derived from `NULL` has a coercibility value of `6`.

When inferring collations, TiDB prefers using the collation of expressions with lower coercibility values. If the coercibility values of two clauses are the same, the collation is determined according to the following priority:

binary > utf8mb4_bin > (utf8mb4_general_ci = utf8mb4_unicode_ci) > utf8_bin > (utf8_general_ci = utf8_unicode_ci) > latin1_bin > ascii_bin

TiDB cannot infer the collation and reports an error in the following situations:

- If the collations of two clauses are different and the coercibility value of both clauses is `0`.
- If the collations of two clauses are incompatible and the returned type of expression is `String`.

14.10.15.1.10 COLLATE clause

TiDB supports using the COLLATE clause to specify the collation of an expression. The coercibility value of this expression is 0, which has the highest priority. See the following example:

```
SELECT 'a' = _utf8mb4 'A' collate utf8mb4_general_ci;
```

```
+-----------------------------------------------+
| 'a' = _utf8mb4 'A' collate utf8mb4_general_ci |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

For more details, see Connection Character Sets and Collations.

14.10.15.2 GBK

Starting from v5.4.0, TiDB supports the GBK character set. This document provides the TiDB support and compatibility information of the GBK character set.

Starting from v6.0.0, TiDB enables the new framework for collations by default. The default collation for TiDB GBK character set is gbk_chinese_ci, which is consistent with MySQL.

```
SHOW CHARACTER SET WHERE CHARSET = 'gbk';
+--
    ↪ -------+-------------------------------------+-------------------+--------+
    ↪
| Charset | Description                         | Default collation | Maxlen |
+--
    ↪ -------+-------------------------------------+-------------------+--------+
    ↪
| gbk     | Chinese Internal Code Specification | gbk_chinese_ci | 2 |
+--
    ↪ -------+-------------------------------------+-------------------+--------+
    ↪
1 row in set (0.00 sec)

SHOW COLLATION WHERE CHARSET = 'gbk';
+---------------+---------+------+---------+----------+---------+
| Collation     | Charset | Id | Default | Compiled | Sortlen |
+---------------+---------+------+---------+----------+---------+
| gbk_bin       | gbk     |   87 |         | Yes      |       1 |
| gbk_chinese_ci | gbk  |   28 | Yes     | Yes      |       1 |
+---------------+---------+------+---------+----------+---------+
```

```
2 rows in set (0.00 sec)
```

14.10.15.2.1 MySQL compatibility

This section provides the compatibility information between MySQL and TiDB.

Collations

The default collation of the GBK character set in MySQL is `gbk_chinese_ci`. The default collation for the GBK character set in TiDB depends on the value of the TiDB configuration item `new_collations_enabled_on_first_bootstrap`:

- By default, the TiDB configuration item `new_collations_enabled_on_first_bootstrap` is set to `true`, which means that the new framework for collations is enabled and the default collation for the GBK character set is `gbk_chinese_ci`.
- When the TiDB configuration item `new_collations_enabled_on_first_bootstrap` is set to `false`, the new framework for collations is disabled, and the default collation for the GBK character set is `gbk_bin`.

Additionally, because TiDB converts GBK to `utf8mb4` and then uses a binary collation, the `gbk_bin` collation in TiDB is not the same as the `gbk_bin` collation in MySQL.

Illegal character compatibility

- If the system variables `character_set_client` and `character_set_connection` are not set to `gbk` at the same time, TiDB handles illegal characters in the same way as MySQL.
- If `character_set_client` and `character_set_connection` are both set to `gbk`, TiDB handles illegal characters differently than MySQL.
    - MySQL handles illegal GBK character sets in reading and writing operations differently.
    - TiDB handles illegal GBK character sets in reading and writing operations in the same way. In the SQL strict mode, TiDB reports an error when either reading or writing illegal GBK characters. In the non-strict mode, TiDB replaces illegal GBK characters with `?` when either reading or writing illegal GBK characters.

For example, after `SET NAMES gbk`, if you create a table using the `CREATE TABLE gbk_table(a VARCHAR(32)CHARACTER SET gbk)` statement in MySQL and TiDB respectively and then execute the SQL statements in the following table, you can see the detailed differences.

Database	If the configured SQL mode contains either STRICT_ALL_TABLES or STRICT_TRANS_TABLES	If the configured SQL mode contains neither STRICT_ALL_TABLES nor STRICT_TRANS_TABLES
MySQL	SELECT HEX('一a'); e4b88061INSERT INTO gbk_table ↪ values('一a'); Incorrect Error	SELECT HEX('一a'); e4b88061INSERT ↪ INTO gbk_table VALUES('一a'); ↪ SELECT HEX(a)FROM gbk_table; e4b8
TiDB	SELECT HEX('一a'); Incorrect ErrorINSERT INTO ↪ gbk_table VALUES('一a'); Incorrect Error	SELECT HEX('一a'); e4b83fINSERT INTO ↪ gbk_table VALUES('一a');SELECT ↪ HEX(a)FROM gbk_table; e4b83f

In the above table, the result of `SELECT HEX('a');` in the `utf8mb4` byte set is `e4b88061`.

Other MySQL compatibility

- Currently, TiDB does not support using the `ALTER TABLE` statement to convert other character set types to `gbk` or from `gbk` to other character set types.
- TiDB does not support the use of `_gbk`. For example:

```
sql  CREATE TABLE t(a CHAR(10)CHARSET BINARY); Query OK, 0 rows affected
↪  (0.00 sec)INSERT INTO t VALUES (_gbk'啊'); ERROR 1115 (42000): Unsupported
↪  character introducer: 'gbk'
```

- Currently, for binary characters of the `ENUM` and `SET` types, TiDB deals with them as the `utf8mb4` character set.

14.10.15.2.2 Component compatibility

- Currently, TiFlash does not support the GBK character set.
- TiDB Data Migration (DM) does not support migrating `charset=GBK` tables to TiDB clusters earlier than v5.4.0.
- TiDB Lightning does not support importing `charset=GBK` tables to TiDB clusters earlier than v5.4.0.
- TiCDC versions earlier than v6.1.0 do not support replicating `charset=GBK` tables. No version of TiCDC supports replicating `charset=GBK` tables to TiDB clusters earlier than v6.1.0.
- Backup & Restore (BR) versions earlier than v5.4.0 do not support recovering `charset ↪ =GBK` tables. No version of BR supports recovering `charset=GBK` tables to TiDB clusters earlier than v5.4.0.

14.10.16 Placement Rules in SQL

Placement Rules in SQL is a feature that enables you to specify where data is stored in a TiKV cluster using SQL interfaces. Using this feature, tables and partitions are scheduled to specific regions, data centers, racks, or hosts. This is useful for scenarios including optimizing a high availability strategy with lower cost, ensuring that local replicas of data are available for local stale reads, and adhering to data locality requirements.

> **Note:**
>
> The implementation of *Placement Rules in SQL* relies on the *placement rules feature* of PD. For details, refer to Configure Placement Rules. In the context of Placement Rules in SQL, *placement rules* might refer to *placement policies* attached to other objects, or to rules that are sent from TiDB to PD.

The detailed user scenarios are as follows:

- Merge multiple databases of different applications to reduce the cost on database maintenance
- Increase replica count for important data to improve the application availability and data reliability
- Store new data into NVMe storage and store old data into SSDs to lower the cost on data archiving and storage
- Schedule the leaders of hotspot data to high-performance TiKV instances
- Separate cold data to lower-cost storage mediums to improve cost efficiency
- Support the physical isolation of computing resources between different users, which meets the isolation requirements of different users in a cluster, and the isolation requirements of CPU, I/O, memory, and other resources with different mixed loads

14.10.16.1 Specify placement rules

To specify placement rules, first create a placement policy using `CREATE PLACEMENT POLICY`:

```
CREATE PLACEMENT POLICY myplacementpolicy PRIMARY_REGION="us-east-1"
    ↪ REGIONS="us-east-1,us-west-1";
```

Then attach the policy to a table or partition using either `CREATE TABLE` or `ALTER TABLE`. Then, the placement rules are specified on the table or the partition:

```
CREATE TABLE t1 (a INT) PLACEMENT POLICY=myplacementpolicy;
CREATE TABLE t2 (a INT);
ALTER TABLE t2 PLACEMENT POLICY=myplacementpolicy;
```

A placement policy is not associated with any database schema and has the global scope. Therefore, assigning a placement policy does not require any additional privileges over the `CREATE TABLE` privilege.

To modify a placement policy, you can use `ALTER PLACEMENT POLICY`, and the changes will propagate to all objects assigned with the corresponding policy.

```
ALTER PLACEMENT POLICY myplacementpolicy FOLLOWERS=5;
```

To drop policies that are not attached to any table or partition, you can use `DROP PLACEMENT POLICY`:

```
DROP PLACEMENT POLICY myplacementpolicy;
```

14.10.16.2 View current placement rules

If a table has placement rules attached, you can view the placement rules in the output of `SHOW CREATE TABLE`. To view the definition of the policy available, execute `SHOW CREATE PLACEMENT POLICY`:

```
tidb> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin /*T![placement]
    ↪ PLACEMENT POLICY=`myplacementpolicy` */
1 row in set (0.00 sec)

tidb> SHOW CREATE PLACEMENT POLICY myplacementpolicy\G
*************************** 1. row ***************************
       Policy: myplacementpolicy
Create Policy: CREATE PLACEMENT POLICY myplacementpolicy PRIMARY_REGION="us
    ↪ -east-1" REGIONS="us-east-1,us-west-1"
1 row in set (0.00 sec)
```

You can also view definitions of placement policies using the `INFORMATION_SCHEMA.PLACEMENT_POLICIES` table.

```
tidb> select * from information_schema.placement_policies\G
***************************[ 1. row ]***************************
POLICY_ID          | 1
CATALOG_NAME       | def
POLICY_NAME        | p1
PRIMARY_REGION     | us-east-1
REGIONS            | us-east-1,us-west-1
CONSTRAINTS        |
```

```
LEADER_CONSTRAINTS |
FOLLOWER_CONSTRAINTS |
LEARNER_CONSTRAINTS |
SCHEDULE           |
FOLLOWERS          | 4
LEARNERS           | 0
1 row in set
```

The `information_schema.tables` and `information_schema.partitions` tables also include a column for `tidb_placement_policy_name`, which shows all objects with placement rules attached:

```
SELECT * FROM information_schema.tables WHERE tidb_placement_policy_name IS
   ↪  NOT NULL;
SELECT * FROM information_schema.partitions WHERE
   ↪ tidb_placement_policy_name IS NOT NULL;
```

Rules that are attached to objects are applied *asynchronously*. To view the current scheduling progress of placement, use `SHOW PLACEMENT`.

14.10.16.3 Option reference

Note:

- Placement options depend on labels correctly specified in the configuration of each TiKV node. For example, the `PRIMARY_REGION` option depends on the `region` label in TiKV. To see a summary of all labels available in your TiKV cluster, use the statement `SHOW PLACEMENT LABELS`:

```
sql   mysql> show placement labels; +--------+----------------+ | disk
↪  | Key | Values       |    +--------+----------------+ | disk
↪  | ["ssd"]      |    | region | ["us-east-1"] | | zone  | ["
↪ us-east-1a"] | +--------+----------------+ 3 rows in set
↪ (0.00 sec)
```

- When you use `CREATE PLACEMENT POLICY` to create a placement policy, TiDB does not check whether the labels exist. Instead, TiDB performs the check when you attach the policy to a table.

Option Name	Description
`PRIMARY_REGION`	Raft leaders are placed in stores that have the `region` label that matches the value of this option.
`REGIONS`	Raft followers are placed in stores that have the `region` label that matches the value of this option.
`SCHEDULE`	The strategy used to schedule the placement of followers. The value options are `EVEN` (default) or `MAJORITY_IN_PRIMARY`.
`FOLLOWERS`	The number of followers. For example, `FOLLOWERS=2` means that there will be 3 replicas of the data (2 followers and 1 leader).

In addition to the placement options above, you can also use the advance configurations. For details, see Advance placement options.

Option Name	Description
`CONSTRAINTS`	A list of constraints that apply to all roles. For example, `CONSTRAINTS ↪ ="[+disk ↪ =ssd]"`.
`LEADER_CONSTRAINTS` ↪	A list of constraints that only apply to leader.
`FOLLOWER_CONSTRAINTS` ↪	A list of constraints that only apply to followers.
`LEARNER_CONSTRAINTS` ↪	A list of constraints that only apply to learners.
`LEARNERS`	The number of learners.

14.10.16.4 Examples

14.10.16.4.1 Increase the number of replicas

The default configuration of `max-replicas` is 3. To increase this for a specific set of tables, you can use a placement policy as follows:

```
CREATE PLACEMENT POLICY fivereplicas FOLLOWERS=4;
CREATE TABLE t1 (a INT) PLACEMENT POLICY=fivereplicas;
```

Note that the PD configuration includes the leader and follower count, thus 4 followers + 1 leader equals 5 replicas in total.

To expand on this example, you can also use `PRIMARY_REGION` and `REGIONS` placement options to describe the placement for the followers:

```
CREATE PLACEMENT POLICY eastandwest PRIMARY_REGION="us-east-1" REGIONS="us-
    ↪ east-1,us-east-2,us-west-1" SCHEDULE="MAJORITY_IN_PRIMARY" FOLLOWERS
    ↪ =4;
CREATE TABLE t1 (a INT) PLACEMENT POLICY=eastandwest;
```

The `SCHEDULE` option instructs TiDB on how to balance the followers. The default schedule of `EVEN` ensures a balance of followers in all regions.

To ensure that enough followers are placed in the primary region (`us-east-1`) so that quorum can be achieved, you can use the `MAJORITY_IN_PRIMARY` schedule. This schedule helps provide lower latency transactions at the expense of some availability. If the primary region fails, `MAJORITY_IN_PRIMARY` cannot provide automatic failover.

14.10.16.4.2 Assign placement to a partitioned table

In addition to assigning placement options to tables, you can also assign the options to table partitions. For example:

```
CREATE PLACEMENT POLICY p1 FOLLOWERS=5;
CREATE PLACEMENT POLICY europe PRIMARY_REGION="eu-central-1" REGIONS="eu-
    ↪ central-1,eu-west-1";
CREATE PLACEMENT POLICY northamerica PRIMARY_REGION="us-east-1" REGIONS="us
    ↪ -east-1";

SET tidb_enable_list_partition = 1;
CREATE TABLE t1 (
  country VARCHAR(10) NOT NULL,
  userdata VARCHAR(100) NOT NULL
) PLACEMENT POLICY=p1 PARTITION BY LIST COLUMNS (country) (
  PARTITION pEurope VALUES IN ('DE', 'FR', 'GB') PLACEMENT POLICY=europe,
  PARTITION pNorthAmerica VALUES IN ('US', 'CA', 'MX') PLACEMENT POLICY=
     ↪ northamerica,
  PARTITION pAsia VALUES IN ('CN', 'KR', 'JP')
);
```

If a partition has no attached policies, it tries to apply possibly existing policies on the table. For example, the `pEurope` partition will apply the `europe` policy, but the `pAsia` partition will apply the `p1` policy from table `t1`. If `t1` has no assigned policies, `pAsia` will not apply any policy, too.

You can also alter the placement policies assigned to a specific partition. For example:

```
ALTER TABLE t1 PARTITION pEurope PLACEMENT POLICY=p1;
```

14.10.16.4.3 Set the default placement for a schema

You can directly attach the default placement rules to a database schema. This works similar to setting the default character set or collation for a schema. Your specified placement options apply when no other options are specified. For example:

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us
    ↪ -east-2"; -- Create placement policies

CREATE PLACEMENT POLICY p2 FOLLOWERS=4;

CREATE PLACEMENT POLICY p3 FOLLOWERS=2;

CREATE TABLE t1 (a INT); -- Creates a table t1 with no placement options.

ALTER DATABASE test PLACEMENT POLICY=p2; -- Changes the default placement
    ↪ option, and does not apply to the existing table t1.

CREATE TABLE t2 (a INT); -- Creates a table t2 with the default placement
    ↪ policy p2.

CREATE TABLE t3 (a INT) PLACEMENT POLICY=p1; -- Creates a table t3 without
    ↪ the default policy p2, because this statement has specified another
    ↪ placement rule.

ALTER DATABASE test PLACEMENT POLICY=p3; -- Changes the default policy, and
    ↪  does not apply to existing tables.

CREATE TABLE t4 (a INT); -- Creates a table t4 with the default policy p3.

ALTER PLACEMENT POLICY p3 FOLLOWERS=3; -- The table with policy p3 (t4)
    ↪ will have FOLLOWERS=3.
```

Note that this is different from the inheritance between partitions and tables, where changing the policy of tables will affect their partitions. Tables inherit the policy of schema only when they are created without policies attached, and modifying the policies of schemas does not affect created tables.

14.10.16.4.4 Advanced placement options

The placement options `PRIMARY_REGION`, `REGIONS`, and `SCHEDULE` meet the basic needs of data placement at the loss of some flexibility. For more complex scenarios with the need for higher flexibility, you can also use the advanced placement options of `CONSTRAINTS` and `FOLLOWER_CONSTRAINTS`. You cannot specify the `PRIMARY_REGION`, `REGIONS`, or `SCHEDULE` option with the `CONSTRAINTS` option at the same time. If you specify both at the same time, an error will be returned.

For example, to set constraints that data must reside on a TiKV store where the label `disk` must match a value:

```
CREATE PLACEMENT POLICY storageonnvme CONSTRAINTS="[+disk=nvme]";
CREATE PLACEMENT POLICY storageonssd CONSTRAINTS="[+disk=ssd]";
CREATE PLACEMENT POLICY companystandardpolicy CONSTRAINTS="";

CREATE TABLE t1 (id INT, name VARCHAR(50), purchased DATE)
PLACEMENT POLICY=companystandardpolicy
PARTITION BY RANGE( YEAR(purchased) ) (
  PARTITION p0 VALUES LESS THAN (2000) PLACEMENT POLICY=storageonssd,
  PARTITION p1 VALUES LESS THAN (2005),
  PARTITION p2 VALUES LESS THAN (2010),
  PARTITION p3 VALUES LESS THAN (2015),
  PARTITION p4 VALUES LESS THAN MAXVALUE PLACEMENT POLICY=storageonnvme
);
```

You can either specify constraints in list format (`[+disk=ssd]`) or in dictionary format (`{+disk=ssd: 1,+disk=nvme: 2}`).

In list format, constraints are specified as a list of key-value pairs. The key starts with either a `+` or a `-`. `+disk=ssd` indicates that the label `disk` must be set to `ssd`, and `-disk=nvme` indicates that the label `disk` must not be `nvme`.

In dictionary format, constraints also indicate a number of instances that apply to that rule. For example, `FOLLOWER_CONSTRAINTS="{+region=us-east-1: 1,+region=us-east-2: 1,+region=us-west-1: 1}";` indicates that 1 follower is in us-east-1, 1 follower is in us-east-2 and 1 follower is in us-west-1. For another example, `FOLLOWER_CONSTRAINTS='{"+region=us-east-1,+disk=nvme":1,"+region=us-west-1":1}';` indicates that 1 follower is in us-east-1 with an nvme disk, and 1 follower is in us-west-1.

> **Note:**
>
> Dictionary and list formats are based on the YAML parser, but the YAML syntax might be incorrectly parsed. For example, `"{+disk=ssd:1,+disk=nvme:2}"` is incorrectly parsed as `'{"+disk=ssd:1": null, "+disk=nvme:1": null}'`. But `"{+disk=ssd: 1,+disk=nvme: 1}"` is correctly parsed as `'{"+disk=ssd": 1, "+disk=nvme": 1}'`.

14.10.16.5 Compatibility with tools

Tool Name	Minimum supported version	Description
Backup & Restore (BR)	6.0	Supports importing and exporting placement rules. Refer to BR Compatibility for details.

Tool Name	Minimum supported version	Description
TiDB Lightning	Not compatible yet	An error is reported when TiDB Lightning imports backup data that contains placement policies
TiCDC	6.0	Ignores placement rules, and does not replicate the rules to the downstream

Tool Name	Minimum supported version	Description
TiDB Binlog	6.0	Ignores placement rules, and does not replicate the rules to the downstream

14.10.16.6 Known limitations

The following known limitations are as follows:

- Temporary tables do not support placement options.
- Syntactic sugar rules are permitted for setting `PRIMARY_REGION` and `REGIONS`. In the future, we plan to add varieties for `PRIMARY_RACK`, `PRIMARY_ZONE`, and `PRIMARY_HOST`. See issue #18030.
- In TiDB v6.1.0 and v6.1.1, TiFlash learners are not configurable through Placement Rules syntax.
- Placement rules only ensure that data at rest resides on the correct TiKV store. The rules do not guarantee that data in transit (via either user queries or internal operations) only occurs in a specific region.

14.10.17 System Tables

14.10.17.1 `mysql` Schema

The `mysql` schema contains TiDB system tables. The design is similar to the `mysql` schema in MySQL, where tables such as `mysql.user` can be edited directly. It also contains a number of tables which are extensions to MySQL.

14.10.17.1.1 Grant system tables

These system tables contain grant information about user accounts and their privileges:

- `user`: user accounts, global privileges, and other non-privilege columns
- `db`: database-level privileges
- `tables_priv`: table-level privileges
- `columns_priv`: column-level privileges
- `default_roles`: the default roles for a user
- `global_grants`: dynamic privileges
- `global_priv`: the authentication information based on certificates
- `role_edges`: the relationship between roles

14.10.17.1.2 Server-side help system tables

Currently, the `help_topic` is NULL.

14.10.17.1.3 Statistics system tables

- `stats_buckets`: the buckets of statistics
- `stats_histograms`: the histograms of statistics
- `stats_top_n`: the TopN of statistics
- `stats_meta`: the meta information of tables, such as the total number of rows and updated rows
- `stats_extended`: extended statistics, such as the order correlation between columns
- `stats_feedback`: the query feedback of statistics
- `stats_fm_sketch`: the FMSketch distribution of the histogram of the statistics column
- `stats_meta_history`: the meta information in the historical statistics
- `stats_history`: the other information in the historical statistics
- `analyze_options`: the default `analyze` options for each table
- `column_stats_usage`: the usage of column statistics
- `schema_index_usage`: the usage of indexes
- `analyze_jobs`: the ongoing statistics collection tasks and the history task records within the last 7 days

14.10.17.1.4 Execution plan-related system tables

- `bind_info`: the binding information of execution plans
- `capture_plan_baselines_blacklist`: the blocklist for the automatic binding of the execution plan

14.10.17.1.5 GC worker system tables

- `gc_delete_range`: the KV range to be deleted
- `gc_delete_range_done`: the deleted KV range

14.10.17.1.6 Miscellaneous system tables

- `GLOBAL_VARIABLES`: global system variable table

- `tidb`: to record the version information when TiDB executes `bootstrap`
- `expr_pushdown_blacklist`: the blocklist for expression pushdown
- `opt_rule_blacklist`: the blocklist for logical optimization rules
- `table_cache_meta`: the metadata of cached tables
- `advisory_locks`: information related to Locking functions

14.10.17.2 INFORMATION_SCHEMA

14.10.17.2.1 Information Schema

Information Schema provides an ANSI-standard way of viewing system metadata. TiDB also provides a number of custom `INFORMATION_SCHEMA` tables, in addition to the tables included for MySQL compatibility.

Many `INFORMATION_SCHEMA` tables have a corresponding `SHOW` command. The benefit of querying `INFORMATION_SCHEMA` is that it is possible to join between tables.

Tables for MySQL compatibility

Table Name	Description
CHARACTER_SETS	Provides a list of character sets the server supports.
COLLATIONS	Provides a list of collations that the server supports.
COLLATION_CHARACTER_SET_APPLICABILITY	Explains which collations apply to which character sets.
COLUMNS	Provides a list of columns for all tables.
COLUMN_PRIVILEGES	Not implemented by TiDB. Returns zero rows.
COLUMN_STATISTICS	Not implemented by TiDB. Returns zero rows.
ENGINES	Provides a list of supported storage engines.

Table Name	Description
EVENTS	Not implemented by TiDB. Returns zero rows.
FILES	Not implemented by TiDB. Returns zero rows.
GLOBAL_STATUS	Not implemented by TiDB. Returns zero rows.
GLOBAL_VARIABLES	Not implemented by TiDB. Returns zero rows.
KEY_COLUMN_USAGE	Describes the key constraints of the columns, such as the primary key constraint.
OPTIMIZER_TRACE	Not implemented by TiDB. Returns zero rows.
PARAMETERS	Not implemented by TiDB. Returns zero rows.
PARTITIONS	Provides a list of table partitions.
PLUGINS	Not implemented by TiDB. Returns zero rows.
PROCESSLIST	Provides similar information to the command SHOW PROCESSLIST.
PROFILING	Not implemented by TiDB. Returns zero rows.
REFERENTIAL_CONSTRAINTS	Provides information on FOREIGN KEY constraints.
ROUTINES	Not implemented by TiDB. Returns zero rows.

Table Name	Description
SCHEMATA	Provides similar information to `SHOW DATABASES`.
SCHEMA_PRIVILEGES	Not implemented by TiDB. Returns zero rows.
SESSION_STATUS	Not implemented by TiDB. Returns zero rows.
SESSION_VARIABLES	Provides similar functionality to the command `SHOW SESSION ↪ VARIABLES`
STATISTICS	Provides information on table indexes.
TABLES	Provides a list of tables that the current user has visibility of. Similar to `SHOW TABLES`.
TABLESPACES	Not implemented by TiDB. Returns zero rows.
TABLE_CONSTRAINTS	Provides information on primary keys, unique indexes and foreign keys.
TABLE_PRIVILEGES	Not implemented by TiDB. Returns zero rows.
TRIGGERS	Not implemented by TiDB. Returns zero rows.
USER_PRIVILEGES	Summarizes the privileges associated with the current user.

Table Name	Description
VIEWS	Provides a list of views that the current user has visibility of. Similar to running `SHOW FULL TABLES WHERE table_type = 'VIEW'`

Tables that are TiDB extensions

Table Name	Description
ANALYZE_STATUS	Provides information about tasks to collect statistics.
CLIENT_ERRORS_SUMMARY_BY_HOST	Provides a summary of errors and warnings generated by client requests and returned to clients.
CLIENT_ERRORS_SUMMARY_BY_USER	Provides a summary of errors and warnings generated by clients.
CLIENT_ERRORS_SUMMARY_GLOBAL	Provides a summary of errors and warnings generated by clients.

Table Name	Description
CLUSTER_CONFIG	Provides details about configuration settings for the entire TiDB cluster. This table is not applicable to TiDB Cloud.
CLUSTER_DEADLOCKS	Provides a cluster-level view of the DEADLOCKS table.
CLUSTER_HARDWARE	Provides details on the underlying physical hardware discovered on each TiDB component. This table is not applicable to TiDB Cloud.
CLUSTER_INFO	Provides details on the current cluster topology.

Table Name	Description
CLUSTER_LOAD	Provides current load information for TiDB servers in the cluster. This table is not applicable to TiDB Cloud.
CLUSTER_LOG	Provides a log for the entire TiDB cluster. This table is not applicable to TiDB Cloud.
CLUSTER_PROCESSLIST	Provides a cluster-level view of the PROCESSLIST table.
CLUSTER_SLOW_QUERY	Provides a cluster-level view of the SLOW_QUERY table.
CLUSTER_STATEMENTS_SUMMARY	Provides a cluster-level view of the STATEMENTS_SUMMARY table.

Table Name	Description
CLUSTER_STATEMENTS_SUMMARY_HISTORY	Provides a cluster-level view of the STATEMENTS_SUMMARY_HIST ↪ table.
CLUSTER_TIDB_TRX	Provides a cluster-level view of the TIDB_TRX table.
CLUSTER_SYSTEMINFO	Provides details about kernel parameter configura-tion for servers in the cluster. This table is not applicable to TiDB Cloud.
DATA_LOCK_WAITS	Provides the lock-waiting information on the TiKV server.
DDL_JOBS	Provides similar output to ADMIN ↪ SHOW ↪ DDL ↪ JOBS

Table Name	Description
DEADLOCKS	Provides the information of several deadlock errors that have recently occurred.
INSPECTION_RESULT	Triggers internal diagnostics checks. This table is not applicable to TiDB Cloud.
INSPECTION_RULES	A list of internal diagnostic checks performed. This table is not applicable to TiDB Cloud.
INSPECTION_SUMMARY	A summarized report of important monitoring metrics. This table is not applicable to TiDB Cloud.

Table Name	Description
METRICS_SUMMARY	A summary of metrics extracted from Prometheus. This table is not applicable to TiDB Cloud.
METRICS_SUMMARY_BY_LABEL	See METRICS_SUMMARY table.
METRICS_TABLES	Provides the PromQL definitions for tables in METRICS_SCHEMA . This table is not applicable to TiDB Cloud.
PLACEMENT_POLICIES	Provides information on all placement policies. This table is not applicable to TiDB Cloud.
SEQUENCES	The TiDB implementation of sequences is based on MariaDB.

Table Name	Description
SLOW_QUERY	Provides information on slow queries on the current TiDB server.
STATEMENTS_SUMMARY	Similar to performance_schema statement summary in MySQL.
STATEMENTS_SUMMARY_HISTORY	Similar to performance_schema statement summary history in MySQL.
TABLE_STORAGE_STATS	Provides details about table sizes in storage.
TIDB_HOT_REGIONS	Provides statistics about which regions are hot. This table is not applicable to TiDB Cloud.
TIDB_HOT_REGIONS_HISTORY	Provides history statistics about which Regions are hot.

Table Name	Description
TIDB_INDEXES	Provides index information about TiDB tables.
TIDB_SERVERS_INFO	Provides a list of TiDB servers (namely, tidb-server component)
TIDB_TRX	Provides the information of the transactions that are being executed on the TiDB node.
TIFLASH_REPLICA	Provides details about TiFlash replicas.
TIKV_REGION_PEERS	Provides details about where regions are stored.
TIKV_REGION_STATUS	Provides statistics about regions.
TIKV_STORE_STATUS	Provides basic information about TiKV servers.

14.10.17.2.2 ANALYZE_STATUS

The `ANALYZE_STATUS` table provides information about the running tasks that collect statistics and a limited number of history tasks.

Starting from TiDB v6.1.0, the `ANALYZE_STATUS` table supports showing cluster-level tasks. Even after a TiDB restart, you can still view task records before the restart using this table. Before TiDB v6.1.0, the `ANALYZE_STATUS` table can only show instance-level tasks, and task records are cleared after a TiDB restart.

Starting from TiDB v6.1.0, you can view the history tasks within the last 7 days through the system table `mysql.analyze_jobs`.

```
USE information_schema;
DESC analyze_status;
```

```
+---------------+---------------------+------+-----+---------+-------+
| Field         | Type                | Null | Key | Default | Extra |
+---------------+---------------------+------+-----+---------+-------+
| TABLE_SCHEMA  | varchar(64)         | YES  |     | NULL    |       |
| TABLE_NAME    | varchar(64)         | YES  |     | NULL    |       |
| PARTITION_NAME | varchar(64)        | YES  |     | NULL    |       |
| JOB_INFO      | longtext            | YES  |     | NULL    |       |
| PROCESSED_ROWS | bigint(64) unsigned | YES  |     | NULL    |       |
| START_TIME    | datetime            | YES  |     | NULL    |       |
| END_TIME      | datetime            | YES  |     | NULL    |       |
| STATE         | varchar(64)         | YES  |     | NULL    |       |
| FAIL_REASON   | longtext            | YES  |     | NULL    |       |
| INSTANCE      | varchar(512)        | YES  |     | NULL    |       |
| PROCESS_ID    | bigint(64) unsigned | YES  |     | NULL    |       |
+---------------+---------------------+------+-----+---------+-------+
11 rows in set (0.00 sec)
```

```
SELECT * FROM information_schema.analyze_status;
```

```
+--
   ↪ -----------+-----------+---------------+-------------------------------------
   ↪
| TABLE_SCHEMA | TABLE_NAME | PARTITION_NAME | JOB_INFO
   ↪                                                  | PROCESSED_ROWS |
   ↪ START_TIME      | END_TIME          | STATE    | FAIL_REASON | INSTANCE
   ↪    | PROCESS_ID |
+--
   ↪ -----------+-----------+---------------+-------------------------------------
   ↪
| test         | t          | p1            | analyze table all columns with 256
   ↪  buckets, 500 topn, 1 samplerate |    0 | 2022-05-27 11:30:12 |
   ↪ 2022-05-27 11:30:12 | finished | NULL | 127.0.0.1:4000 |  NULL |
```

```
| test       | t         | p0           | analyze table all columns with 256
   ↪  buckets, 500 topn, 1 samplerate |    0 | 2022-05-27 11:30:12 |
   ↪ 2022-05-27 11:30:12 | finished | NULL | 127.0.0.1:4000 |  NULL |
| test       | t         | p1           | analyze index idx
   ↪                                          |            0 | 2022-05-27
   ↪ 11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪ NULL |
| test       | t         | p0           | analyze index idx
   ↪                                          |            0 | 2022-05-27
   ↪ 11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪ NULL |
| test       | t         | p1           | analyze columns
   ↪                                            |            0 | 2022-05-27
   ↪  11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪  NULL |
| test       | t         | p0           | analyze columns
   ↪                                            |            0 | 2022-05-27
   ↪  11:29:46 | 2022-05-27 11:29:46 | finished | NULL | 127.0.0.1:4000 |
   ↪  NULL |
+--
   ↪ ------------+------------+----------------+-----------------------------------
   ↪
6 rows in set (0.00 sec)
```

Fields in the `ANALYZE_STATUS` table are described as follows:

- `TABLE_SCHEMA`: The name of the database to which the table belongs.
- `TABLE_NAME`: The name of the table.
- `PARTITION_NAME`: The name of the partitioned table.
- `JOB_INFO`: The information of the `ANALYZE` task. If an index is analyzed, this information will include the index name. When `tidb_analyze_version =2`, this information will include configuration items such as sample rate.
- `PROCESSED_ROWS`: The number of rows that have been processed.
- `START_TIME`: The start time of the `ANALYZE` task.
- `END_TIME`: The end time of the `ANALYZE` task.
- `STATE`: The execution status of the `ANALYZE` task. Its value can be `pending`, `running` ↪ ,`finished` or `failed`.
- `FAIL_REASON`: The reason why the task fails. If the execution is successful, the value is `NULL`.
- `INSTANCE`: The TiDB instance that executes the task.
- `PROCESS_ID`: The process ID that executes the task.

14.10.17.2.3 CLIENT_ERRORS_SUMMARY_BY_HOST

The table `CLIENT_ERRORS_SUMMARY_BY_HOST` provides a summary of SQL errors and warnings that have been returned to clients that connect to a TiDB server. These include:

- Malformed SQL statements.
- Division by zero errors.
- The attempt to insert out-of-range or duplicate key values.
- Permission errors.
- A table that does not exist.

These errors are returned to the client via the MySQL server protocol, where applications are expected to take appropriate action. The `information_schema` ↪ `.CLIENT_ERRORS_SUMMARY_BY_HOST` table provides a useful method to inspect errors in the scenario where applications are not correctly handling (or logging) errors returned by the TiDB server.

Because `CLIENT_ERRORS_SUMMARY_BY_HOST` summarizes the errors on a per-remote-host basis, it can be useful to diagnose scenarios where one application server is generating more errors than other servers. Possible scenarios include:

- An outdated MySQL client library.
- An outdated application (possibly this server was missed when rolling out a new deployment).
- Incorrect usage of the "host" portion of user permissions.
- Unreliable network connectivity generating more timeouts or disconnected connections.

The summarized counts can be reset using the statement `FLUSH CLIENT_ERRORS_SUMMARY` ↪ . The summary is local to each TiDB server and is only retained in memory. Summaries will be lost if the TiDB server restarts.

```
USE information_schema;
DESC CLIENT_ERRORS_SUMMARY_BY_HOST;
```

```
+---------------+---------------+------+------+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+------+---------+-------+
| HOST          | varchar(255) | NO |     | NULL   |       |
| ERROR_NUMBER | bigint(64)  | NO  |     | NULL   |       |
| ERROR_MESSAGE | varchar(1024) | NO |  | NULL   |       |
| ERROR_COUNT | bigint(64)   | NO  |     | NULL   |       |
| WARNING_COUNT | bigint(64)  | NO |     | NULL   |       |
| FIRST_SEEN   | timestamp    | YES |     | NULL   |       |
| LAST_SEEN    | timestamp    | YES |     | NULL   |       |
+---------------+---------------+------+------+---------+-------+
7 rows in set (0.00 sec)
```

Field description:

- `HOST`: The remote host of the client.
- `ERROR_NUMBER`: The MySQL-compatible error number that was returned.
- `ERROR_MESSAGE`: The error message which matches the error number (in prepared statement form).
- `ERROR_COUNT`: The number of times this error was returned to the client host.
- `WARNING_COUNT`: The number of times this warning was returned to the client host.
- `FIRST_SEEN`: The first time this error (or warning) was seen from the client host.
- `LAST_SEEN`: The most recent time this error (or warning) was seen from the client host.

The following example shows a warning being generated when the client connects to a local TiDB server. The summary is reset after executing `FLUSH CLIENT_ERRORS_SUMMARY`:

```
SELECT 0/0;
SELECT * FROM CLIENT_ERRORS_SUMMARY_BY_HOST;
FLUSH CLIENT_ERRORS_SUMMARY;
SELECT * FROM CLIENT_ERRORS_SUMMARY_BY_HOST;
```

```
+-----+
| 0/0 |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

+--
    ↪ ---------+--------------+---------------+------------+--------------+---------
    ↪
| HOST      | ERROR_NUMBER | ERROR_MESSAGE | ERROR_COUNT | WARNING_COUNT |
    ↪ FIRST_SEEN | LAST_SEEN       |
+--
    ↪ ---------+--------------+---------------+------------+--------------+---------
    ↪
| 127.0.0.1 |       1365 | Division by 0 |       0 |            1 |
    ↪ 2021-03-18 12:51:54 | 2021-03-18 12:51:54 |
+--
    ↪ ---------+--------------+---------------+------------+--------------+---------
    ↪
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Empty set (0.00 sec)
```

14.10.17.2.4 CLIENT_ERRORS_SUMMARY_BY_USER

The table `CLIENT_ERRORS_SUMMARY_BY_USER` provides a summary of SQL errors and warnings that have been returned to clients that connect to a TiDB server. These include:

- Malformed SQL statements.
- Division by zero errors.
- The attempt to insert out-of-range or duplicate key values.
- Permission errors.
- A table that does not exist.

Client errors are returned to the client via the MySQL server protocol, where applications are expected to take appropriate action. The `information_schema` ↪ `.CLIENT_ERRORS_SUMMARY_BY_USER` table provides an useful method to inspect errors in the scenario where applications are not correctly handling (or logging) errors returned by the TiDB server.

Because `CLIENT_ERRORS_SUMMARY_BY_USER` summarizes the errors on a per-user basis, it can be useful to diagnose scenarios where one user server is generating more errors than other servers. Possible scenarios include:

- Permission errors.
- Missing tables, or relational objects.
- Incorrect SQL syntax, or incompatibilities between the application and the version of TiDB.

The summarized counts can be reset with the statement `FLUSH CLIENT_ERRORS_SUMMARY` ↪ `.` The summary is local to each TiDB server and is only retained in memory. Summaries will be lost if the TiDB server restarts.

```
USE information_schema;
DESC CLIENT_ERRORS_SUMMARY_BY_USER;
```

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| USER         | varchar(64)  | NO   |     | NULL    |       |
| ERROR_NUMBER | bigint(64)   | NO   |     | NULL    |       |
| ERROR_MESSAGE | varchar(1024) | NO |   | NULL    |       |
| ERROR_COUNT  | bigint(64)   | NO   |     | NULL    |       |
| WARNING_COUNT | bigint(64)  | NO   |     | NULL    |       |
| FIRST_SEEN   | timestamp    | YES  |     | NULL    |       |
| LAST_SEEN    | timestamp    | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Field description:

- `USER`: The authenticated user.
- `ERROR_NUMBER`: The MySQL-compatible error number that was returned.
- `ERROR_MESSAGE`: The error message which matches the error number (in prepared statement form).
- `ERROR_COUNT`: The number of times this error was returned to the user.
- `WARNING_COUNT`: The number of times this warning was returned to the user.
- `FIRST_SEEN`: The first time this error (or warning) was sent to the user.
- `LAST_SEEN`: The most recent time this error (or warning) was sent to the user.

The following example shows a warning being generated when the client connects to a local TiDB server. The summary is reset after executing `FLUSH CLIENT_ERRORS_SUMMARY`:

```
SELECT 0/0;
SELECT * FROM CLIENT_ERRORS_SUMMARY_BY_USER;
FLUSH CLIENT_ERRORS_SUMMARY;
SELECT * FROM CLIENT_ERRORS_SUMMARY_BY_USER;
```

```
+-----+
| 0/0 |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

+--
    ↪ ----+--------------+---------------+-------------+---------------+--------------
    ↪
| USER | ERROR_NUMBER | ERROR_MESSAGE | ERROR_COUNT | WARNING_COUNT |
    ↪ FIRST_SEEN | LAST_SEEN      |
+--
    ↪ ----+--------------+---------------+-------------+---------------+--------------
    ↪
| root |       1365 | Division by 0 |       0 |             1 | 2021-03-18
    ↪ 13:05:36 | 2021-03-18 13:05:36 |
+--
    ↪ ----+--------------+---------------+-------------+---------------+--------------
    ↪
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Empty set (0.00 sec)
```

14.10.17.2.5 CLIENT_ERRORS_SUMMARY_GLOBAL

The table `CLIENT_ERRORS_SUMMARY_GLOBAL` provides a global summary of all SQL errors and warnings that have been returned to clients that connect to a TiDB server. These include:

- Malformed SQL statements.
- Division by zero errors.
- The attempt to insert out-of-range of duplicate key values.
- Permission errors.
- A table does not exist.

Client errors are returned to the client via the MySQL server protocol, where applications are expected to take appropriate action. The `information_schema` ↪ `.CLIENT_ERRORS_SUMMARY_GLOBAL` table provides a high-level overview, and is useful in the scenario where applications are not correctly handling (or logging) errors returned by the TiDB server.

The summarized counts can be reset with the statement `FLUSH CLIENT_ERRORS_SUMMARY` ↪ `.` The summary is local to each TiDB server and is only retained in memory. Summaries will be lost if the TiDB server restarts.

```
USE information_schema;
DESC CLIENT_ERRORS_SUMMARY_GLOBAL;
```

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| ERROR_NUMBER  | bigint(64)    | NO   |     | NULL    |       |
| ERROR_MESSAGE | varchar(1024) | NO   |     | NULL    |       |
| ERROR_COUNT   | bigint(64)    | NO   |     | NULL    |       |
| WARNING_COUNT | bigint(64)    | NO   |     | NULL    |       |
| FIRST_SEEN    | timestamp     | YES  |     | NULL    |       |
| LAST_SEEN     | timestamp     | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Field description:

- `ERROR_NUMBER`: The MySQL-compatible error number that was returned.
- `ERROR_MESSAGE`: The error message which matches the error number (in prepared statement form).
- `ERROR_COUNT`: The number of times this error was returned.
- `WARNING_COUNT`: The number of times this warning was returned.
- `FIRST_SEEN`: The first time this error (or warning) was sent.
- `LAST_SEEN`: The most recent time this error (or warning) was sent.

The following example shows a warning being generated when connecting to a local TiDB server. The summary is reset after executing `FLUSH CLIENT_ERRORS_SUMMARY`:

```
SELECT 0/0;
SELECT * FROM CLIENT_ERRORS_SUMMARY_GLOBAL;
FLUSH CLIENT_ERRORS_SUMMARY;
SELECT * FROM CLIENT_ERRORS_SUMMARY_GLOBAL;
```

```
+-----+
| 0/0 |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)

+--
    ↪ ------------+---------------+-------------+---------------+---------------------
    ↪
| ERROR_NUMBER | ERROR_MESSAGE | ERROR_COUNT | WARNING_COUNT | FIRST_SEEN |
    ↪ LAST_SEEN       |
+--
    ↪ ------------+---------------+-------------+---------------+---------------------
    ↪
|         1365 | Division by 0 |           0 |             1 | 2021-03-18 13:10:51
    ↪ | 2021-03-18 13:10:51 |
+--
    ↪ ------------+---------------+-------------+---------------+---------------------
    ↪
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Empty set (0.00 sec)
```

14.10.17.2.6 CHARACTER_SETS

The `CHARACTER_SETS` table provides information about character sets. Currently, TiDB only supports some of the character sets.

```
USE information_schema;
DESC character_sets;
```

```
+--------------------+-------------+------+------+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+------+---------+-------+
```

```
| CHARACTER_SET_NAME | varchar(32) | YES |   | NULL   |       |
| DEFAULT_COLLATE_NAME | varchar(32) | YES | | NULL   |       |
| DESCRIPTION        | varchar(60) | YES |    | NULL   |       |
| MAXLEN             | bigint(3) | YES |     | NULL   |       |
+--------------------+-------------+-----+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
SELECT * FROM `character_sets`;
```

```
+--------------------+----------------------+--------------+--------+
| CHARACTER_SET_NAME | DEFAULT_COLLATE_NAME | DESCRIPTION | MAXLEN |
+--------------------+----------------------+--------------+--------+
| utf8               | utf8_bin             | UTF-8 Unicode |   3 |
| utf8mb4            | utf8mb4_bin          | UTF-8 Unicode |   4 |
| ascii              | ascii_bin            | US ASCII      |   1 |
| latin1             | latin1_bin           | Latin1        |   1 |
| binary             | binary               | binary        |   1 |
+--------------------+----------------------+--------------+--------+
5 rows in set (0.00 sec)
```

The description of columns in the `CHARACTER_SETS` table is as follows:

- `CHARACTER_SET_NAME`: The name of the character set.
- `DEFAULT_COLLATE_NAME` The default collation name of the character set.
- `DESCRIPTION` The description of the character set.
- `MAXLEN` The maximum length required to store a character in this character set.

14.10.17.2.7 CLUSTER_CONFIG

You can use the `CLUSTER_CONFIG` cluster configuration table to get the current configuration of all server components in the cluster. This simplifies the usage over earlier releases of TiDB, where obtaining similar information would require accessing the HTTP API end points of each instance.

```
USE information_schema;
DESC cluster_config;
```

```
+----------+--------------+------+------+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+------+---------+-------+
| TYPE     | varchar(64)  | YES |     | NULL    |       |
| INSTANCE | varchar(64)  | YES |    | NULL    |       |
| KEY      | varchar(256) | YES |    | NULL    |       |
| VALUE    | varchar(128) | YES |    | NULL    |       |
+----------+--------------+------+------+---------+-------+
```

Field description:

- `TYPE`: The instance type. The optional values are `tidb`, `pd`, and `tikv`.
- `INSTANCE`: The service address of the instance.
- `KEY`: The configuration item name.
- `VALUE`: The configuration item value.

The following example shows how to query the `coprocessor` configuration on the TiKV instance using the `CLUSTER_CONFIG` table:

```
SELECT * FROM cluster_config WHERE type='tikv' AND `key` LIKE 'coprocessor%
    ↪ ';
```

```
+------+----------------+-----------------------------------+---------+
| TYPE | INSTANCE       | KEY                               | VALUE   |
+------+----------------+-----------------------------------+---------+
| tikv | 127.0.0.1:20165 | coprocessor.batch-split-limit | 10 |
| tikv | 127.0.0.1:20165 | coprocessor.region-max-keys | 1440000 |
| tikv | 127.0.0.1:20165 | coprocessor.region-max-size | 144MiB |
| tikv | 127.0.0.1:20165 | coprocessor.region-split-keys | 960000 |
| tikv | 127.0.0.1:20165 | coprocessor.region-split-size | 96MiB |
| tikv | 127.0.0.1:20165 | coprocessor.split-region-on-table | false |
+------+----------------+-----------------------------------+---------+
6 rows in set (0.00 sec)
```

14.10.17.2.8 CLUSTER_HARDWARE

The `CLUSTER_HARDWARE` hardware system table provides the hardware information of the server where each instance of the cluster is located.

```
USE information_schema;
DESC cluster_hardware;
```

```
+-------------+--------------+------+------+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+------+---------+-------+
| TYPE        | varchar(64)  | YES |     | NULL    |       |
| INSTANCE    | varchar(64)  | YES |     | NULL    |       |
| DEVICE_TYPE | varchar(64)  | YES |    | NULL    |       |
| DEVICE_NAME | varchar(64)  | YES |    | NULL    |       |
| NAME        | varchar(256) | YES |    | NULL    |       |
| VALUE       | varchar(128) | YES |    | NULL    |       |
+-------------+--------------+------+------+---------+-------+
6 rows in set (0.00 sec)
```

Field description:

- `TYPE`: Corresponds to the `TYPE` field in the `information_schema.cluster_info` table. The optional values are `tidb`, `pd`, and `tikv`.
- `INSTANCE`: Corresponds to the `INSTANCE` field in the `information_schema.cluster_info` cluster information table.
- `DEVICE_TYPE`: Hardware type. Currently, you can query the `cpu`, `memory`, `disk`, and `net` types.
- `DEVICE_NAME`: Hardware name. The value of `DEVICE_NAME` varies with `DEVICE_TYPE`.
    - `cpu`: The hardware name is cpu.
    - `memory`: The hardware name is memory.
    - `disk`: The disk name.
    - `net`: The network card name.
- `NAME`: The different information names of the hardware. For example, cpu has two information names: `cpu-logical-cores` and `cpu-physical-cores`, which respectively mean logical core numbers and physical core numbers.
- `VALUE`: The value of the corresponding hardware information, such as the disk volume and CPU core numbers.

The following example shows how to query the CPU information using the `CLUSTER_HARDWARE` table:

```
SELECT * FROM cluster_hardware WHERE device_type='cpu' AND device_name='cpu
    ↪ ' AND name LIKE '%cores';
```

```
+--
    ↪ ----+----------------+------------+------------+------------------+-------+
    ↪
| TYPE | INSTANCE     | DEVICE_TYPE | DEVICE_NAME | NAME         | VALUE |
+--
    ↪ ----+----------------+------------+------------+------------------+-------+
    ↪
| tidb | 0.0.0.0:4000 | cpu        | cpu        | cpu-logical-cores | 16  |
| tidb | 0.0.0.0:4000 | cpu        | cpu        | cpu-physical-cores | 8  |
| pd   | 127.0.0.1:2379 | cpu      | cpu        | cpu-logical-cores | 16  |
| pd   | 127.0.0.1:2379 | cpu      | cpu        | cpu-physical-cores | 8  |
| tikv | 127.0.0.1:20165 | cpu     | cpu        | cpu-logical-cores | 16  |
| tikv | 127.0.0.1:20165 | cpu     | cpu        | cpu-physical-cores | 8  |
+--
    ↪ ----+----------------+------------+------------+------------------+-------+
    ↪
6 rows in set (0.03 sec)
```

14.10.17.2.9 CLUSTER_INFO

The `CLUSTER_INFO` cluster topology table provides the current topology information of the cluster, the version information of each instance, the Git Hash corresponding to the instance version, the starting time of each instance, and the running time of each instance.

```
USE information_schema;
desc cluster_info;
```

```
+----------------+-------------+------+------+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+------+---------+-------+
| TYPE           | varchar(64) | YES |    | NULL   |       |
| INSTANCE       | varchar(64) | YES |    | NULL   |       |
| STATUS_ADDRESS | varchar(64) | YES | | NULL   |       |
| VERSION        | varchar(64) | YES |    | NULL   |       |
| GIT_HASH       | varchar(64) | YES |    | NULL   |       |
| START_TIME     | varchar(32) | YES |    | NULL   |       |
| UPTIME         | varchar(32) | YES |    | NULL   |       |
| SERVER_ID      | bigint(21) | YES |     | NULL   |       |
+----------------+-------------+------+------+---------+-------+
8 rows in set (0.01 sec)
```

Field description:

- `TYPE`: The instance type. The optional values are `tidb`, `pd`, and `tikv`.
- `INSTANCE`: The instance address, which is a string in the format of `IP:PORT`.
- `STATUS_ADDRESS`: The service address of HTTP API. Some commands in tikv-ctl, pd-ctl, or tidb-ctl might use this API and this address. You can also get more cluster information via this address. Refer to TiDB HTTP API document for details.
- `VERSION`: The semantic version number of the corresponding instance. To be compatible with the MySQL version number, the TiDB version is displayed in the format of `${mysql-version}-${tidb-version}`.
- `GIT_HASH`: The Git Commit Hash when compiling the instance version, which is used to identify whether two instances are of the absolutely consistent version.
- `START_TIME`: The starting time of the corresponding instance.
- `UPTIME`: The uptime of the corresponding instance.
- `SERVER_ID`: The server ID of the corresponding instance.

```
SELECT * FROM cluster_info;
```

```
+--
    ↪ ----+----------------+----------------+-------------+----------------------
    ↪
```

```
| TYPE | INSTANCE     | STATUS_ADDRESS | VERSION    | GIT_HASH
   ↪                                | START_TIME             | UPTIME
   ↪            |
+--
   ↪ ----+----------------+----------------+-------------+----------------------
   ↪
| tidb | 0.0.0.0:4000 | 0.0.0.0:10080 | 4.0.0-beta.2 | 0
   ↪ df3b74f55f8f8fbde39bbd5d471783f49dc10f7 | 2020-07-05T09:25:53-06:00 |
   ↪  26h39m4.352862693s |
| pd   | 127.0.0.1:2379 | 127.0.0.1:2379 | 4.1.0-alpha | 1
   ↪ ad59bcbf36d87082c79a1fffa3b0895234ac862 | 2020-07-05T09:25:47-06:00 |
   ↪  26h39m10.352868103s |
| tikv | 127.0.0.1:20165 | 127.0.0.1:20180 | 4.1.0-alpha |
   ↪ b45e052df8fb5d66aa8b3a77b5c992ddbfbb79df | 2020-07-05T09:25:50-06:00
   ↪ | 26h39m7.352869963s |
+--
   ↪ ----+----------------+----------------+-------------+----------------------
   ↪
3 rows in set (0.00 sec)
```

14.10.17.2.10 CLUSTER_LOAD

The CLUSTER_LOAD cluster load table provides the current load information of the server where each instance of the TiDB cluster is located.

```
USE information_schema;
DESC cluster_load;
```

```
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| TYPE        | varchar(64)  | YES  |     | NULL    |       |
| INSTANCE    | varchar(64)  | YES  |     | NULL    |       |
| DEVICE_TYPE | varchar(64)  | YES  |     | NULL    |       |
| DEVICE_NAME | varchar(64)  | YES  |     | NULL    |       |
| NAME        | varchar(256) | YES  |     | NULL    |       |
| VALUE       | varchar(128) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Field description:

- TYPE: Corresponds to the TYPE field in the information_schema.cluster_info table. The optional values are tidb, pd, and tikv.

- `INSTANCE`: Corresponds to the `INSTANCE` field in the `information_schema.cluster_info` cluster information table.
- `DEVICE_TYPE`: Hardware type. Currently, you can query the `cpu`, `memory`, `disk`, and `net` types.
- `DEVICE_NAME`: Hardware name. The value of `DEVICE_NAME` varies with `DEVICE_TYPE`.
    - `cpu`: The hardware name is cpu.
    - `disk`: The disk name.
    - `net`: The network card name.
    - `memory`: The hardware name is memory.
- `NAME`: Different load types. For example, cpu has three load types: `load1`, `load5`, and `load15`, which respectively mean the average load of cpu within 1 minute, 5 minutes, and 15 minutes.
- `VALUE`: The value of the hardware load. For example, `1min`, `5min`, and `15min` respectively mean the average load of the hardware within 1 minute, 5 minutes, and 15 minutes.

The following example shows how to query the current load information of cpu using the `CLUSTER_LOAD` table:

```
SELECT * FROM cluster_load WHERE device_type='cpu' AND device_name='cpu';
```

```
+------+-----------------+-------------+-------------+--------+-------+
| TYPE | INSTANCE        | DEVICE_TYPE | DEVICE_NAME | NAME | VALUE |
+------+-----------------+-------------+-------------+--------+-------+
| tidb | 0.0.0.0:4000 | cpu         | cpu         | load1 | 0.13 |
| tidb | 0.0.0.0:4000 | cpu         | cpu         | load5 | 0.25 |
| tidb | 0.0.0.0:4000 | cpu         | cpu         | load15 | 0.31 |
| pd   | 127.0.0.1:2379 | cpu       | cpu         | load1 | 0.13 |
| pd   | 127.0.0.1:2379 | cpu       | cpu         | load5 | 0.25 |
| pd   | 127.0.0.1:2379 | cpu       | cpu         | load15 | 0.31 |
| tikv | 127.0.0.1:20165 | cpu      | cpu         | load1 | 0.13 |
| tikv | 127.0.0.1:20165 | cpu      | cpu         | load5 | 0.25 |
| tikv | 127.0.0.1:20165 | cpu      | cpu         | load15 | 0.31 |
+------+-----------------+-------------+-------------+--------+-------+
9 rows in set (1.50 sec)
```

14.10.17.2.11 CLUSTER_LOG

You can query cluster logs on the `CLUSTER_LOG` cluster log table. By pushing down query conditions to each instance, the impact of the query on cluster performance is less than that of the `grep` command.

To get the logs of the TiDB cluster before v4.0, you need to log in to each instance to summarize logs. This cluster log table in 4.0 provides the global and time-ordered log

search result, which makes it easier to track full-link events. For example, by searching logs according to the `region id`, you can query all logs in the life cycle of this Region. Similarly, by searching the full link log through the slow log's `txn id`, you can query the flow and the number of keys scanned by this transaction at each instance.

```
USE information_schema;
DESC cluster_log;
```

```
+---------+-----------------+------+------+---------+-------+
| Field   | Type            | Null | Key | Default | Extra |
+---------+-----------------+------+------+---------+-------+
| TIME    | varchar(32)     | YES |      | NULL    |       |
| TYPE    | varchar(64)     | YES |      | NULL    |       |
| INSTANCE | varchar(64)    | YES |      | NULL    |       |
| LEVEL   | varchar(8)      | YES |      | NULL    |       |
| MESSAGE | var_string(1024) | YES |   | NULL    |       |
+---------+-----------------+------+------+---------+-------+
5 rows in set (0.00 sec)
```

Field description:

- `TIME`: The time to print the log.
- `TYPE`: The instance type. The optional values are `tidb`, `pd`, and `tikv`.
- `INSTANCE`: The service address of the instance.
- `LEVEL`: The log level.
- `MESSAGE`: The log content.

Note:

- All fields of the cluster log table are pushed down to the corresponding instance for execution. To reduce the overhead of using the cluster log table, you must specify the keywords used for the search, the time range, and as many conditions as possible. For example, `select * from cluster_log where message like '%ddl %' and time > '2020-05-18 20:40:00' and time<'2020-05-18 21:40:00' and type='tidb'`.
- The `message` field supports the `like` and `regexp` regular expressions, and the corresponding pattern is encoded as `regexp`. Specifying multiple `message` conditions is equivalent to the `pipeline` form of the `grep` command. For example, executing the `select * from cluster_log where message like 'coprocessor%' and message regexp '.*slow.*' and time > '2020-05-18 20:40:00' and time <'2020-05-18 21:40:00'` statement is equivalent to executing

`grep 'coprocessor' xxx.log | grep -E '.*slow.*'` on all cluster instances.

The following example shows how to query the execution process of a DDL statement using the `CLUSTER_LOG` table:

```
SELECT time,instance,left(message,150) FROM cluster_log WHERE message LIKE
    ↪ '%ddl%job%ID.80%' AND type='tidb' AND time > '2020-05-18 20:40:00'
    ↪ AND time < '2020-05-18 21:40:00'
```

```
+--
    ↪ ----------------------+---------------+-----------------------------------
    ↪
| time                    | instance      | left(message,150)
    ↪
    ↪ |
+--
    ↪ ----------------------+---------------+-----------------------------------
    ↪
| 2020/05/18 21:37:54.784 | 127.0.0.1:4002 | [ddl_worker.go:261] ["[ddl] add
    ↪  DDL jobs"] ["batch count"=1] [jobs="ID:80, Type:create table, State:
    ↪ none, SchemaState:none, SchemaID:1, TableID:79, Ro |
| 2020/05/18 21:37:54.784 | 127.0.0.1:4002 | [ddl.go:477] ["[ddl] start DDL
    ↪ job"] [job="ID:80, Type:create table, State:none, SchemaState:none,
    ↪ SchemaID:1, TableID:79, RowCount:0, ArgLen:1, start |
| 2020/05/18 21:37:55.327 | 127.0.0.1:4000 | [ddl_worker.go:568] ["[ddl] run
    ↪  DDL job"] [worker="worker 1, tp general"] [job="ID:80, Type:create
    ↪ table, State:none, SchemaState:none, SchemaID:1, Ta |
| 2020/05/18 21:37:55.381 | 127.0.0.1:4000 | [ddl_worker.go:763] ["[ddl]
    ↪ wait latest schema version changed"] [worker="worker 1, tp general"]
    ↪ [ver=70] ["take time"=50.809848ms] [job="ID:80, Type: |
| 2020/05/18 21:37:55.382 | 127.0.0.1:4000 | [ddl_worker.go:359] ["[ddl]
    ↪ finish DDL job"] [worker="worker 1, tp general"] [job="ID:80, Type:
    ↪ create table, State:synced, SchemaState:public, SchemaI |
| 2020/05/18 21:37:55.786 | 127.0.0.1:4002 | [ddl.go:509] ["[ddl] DDL job is
    ↪  finished"] [jobID=80]
    ↪
    ↪ |
+-----------------------+---------------+-----------------------------------
    ↪
```

The query results above show the process of executing a DDL statement:

1. The request with a DDL JOB ID of `80` is sent to the `127.0.0.1:4002` TiDB instance.

2. The `127.0.0.1:4000` TiDB instance processes this DDL request, which indicates that the `127.0.0.1:4000` instance is the DDL owner at that time.
3. The request with a DDL JOB ID of `80` has been processed.

14.10.17.2.12 CLUSTER_SYSTEMINFO

You can use the `CLUSTER_SYSTEMINFO` kernel parameter table to query the kernel configuration information of the server where all instances of the cluster are located. Currently, you can query the information of the `sysctl` system.

```
USE information_schema;
DESC cluster_systeminfo;
```

```
+-------------+--------------+------+------+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+------+---------+-------+
| TYPE        | varchar(64)  | YES  |     | NULL    |       |
| INSTANCE    | varchar(64)  | YES  |     | NULL    |       |
| SYSTEM_TYPE | varchar(64)  | YES  |    | NULL    |       |
| SYSTEM_NAME | varchar(64)  | YES  |    | NULL    |       |
| NAME        | varchar(256) | YES  |    | NULL    |       |
| VALUE       | varchar(128) | YES  |    | NULL    |       |
+-------------+--------------+------+------+---------+-------+
6 rows in set (0.00 sec)
```

Field description:

- `TYPE`: Corresponds to the `TYPE` field in the `information_schema.cluster_info` table. The optional values are `tidb`, `pd`, and `tikv`.
- `INSTANCE`: Corresponds to the `INSTANCE` field in the `information_schema.cluster_info` cluster information table.
- `SYSTEM_TYPE`: The system type. Currently, you can query the `system` system type.
- `SYSTEM_NAME`: The system name. Currently, you can query the `sysctl` system name.
- `NAME`: The configuration name corresponding to `sysctl`.
- `VALUE`: The value of the configuration item corresponding to `sysctl`.

The following example shows how to query the kernel version of all servers in the cluster using the `CLUSTER_SYSTEMINFO` system information table.

```
SELECT * FROM cluster_systeminfo WHERE name LIKE '%kernel.osrelease%'
```

```
+--
    ↪ ----+------------------+------------+------------+-----------------+--------
    ↪
| TYPE | INSTANCE       | SYSTEM_TYPE | SYSTEM_NAME | NAME        | VALUE
    ↪                  |
```

```
+--
   ↪ ----+------------------+------------+------------+-----------------+-------
   ↪
| tidb | 172.16.5.40:4008 | system  | sysctl     | kernel.osrelease |
   ↪ 3.10.0-862.14.4.el7.x86_64 |
| pd   | 172.16.5.40:20379 | system  | sysctl     | kernel.osrelease |
   ↪ 3.10.0-862.14.4.el7.x86_64 |
| tikv | 172.16.5.40:21150 | system | sysctl     | kernel.osrelease |
   ↪ 3.10.0-862.14.4.el7.x86_64 |
+--
   ↪ ----+------------------+------------+------------+-----------------+-------
   ↪
```

14.10.17.2.13 COLLATIONS

The `COLLATIONS` table provides a list of collations that correspond to character sets in the `CHARACTER_SETS` table. Currently, this table is included only for compatibility with MySQL.

```
USE information_schema;
DESC collations;
```

```
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| COLLATION_NAME     | varchar(32) | YES |    | NULL    |       |
| CHARACTER_SET_NAME | varchar(32) | YES | | NULL    |       |
| ID                 | bigint(11) | YES |     | NULL    |       |
| IS_DEFAULT         | varchar(3) | YES |     | NULL    |       |
| IS_COMPILED        | varchar(3) | YES |     | NULL    |       |
| SORTLEN            | bigint(3)  | YES |     | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
SELECT * FROM collations WHERE character_set_name='utf8mb4';
```

```
+--
   ↪ -----------------+--------------------+-----+-----------+-------------+------
   ↪
| COLLATION_NAME   | CHARACTER_SET_NAME | ID | IS_DEFAULT | IS_COMPILED |
   ↪ SORTLEN |
+--
   ↪ -----------------+--------------------+-----+-----------+-------------+------
   ↪
```

```
| utf8mb4_bin      | utf8mb4          |   46 | Yes      | Yes        |       1
    ↪ |
| utf8mb4_general_ci | utf8mb4        |   45 |          | Yes        |       1
    ↪ |
| utf8mb4_unicode_ci | utf8mb4        |  224 |          | Yes        |       1
    ↪ |
+--
    ↪ ------------------+--------------------+------+-----------+-------------+------
    ↪
3 rows in set (0.001 sec)
```

The description of columns in the `COLLATIONS` table is as follows:

- `COLLATION_NAME`: The name of the collation.
- `CHARACTER_SET_NAME`: The name of the character set which the collation belongs to.
- `ID`: The ID of the collation.
- `IS_DEFAULT`: Whether this collation is the default collation of the character set it belongs to.
- `IS_COMPILED`: Whether the character set is compiled into the server.
- `SORTLEN`: The minimum length of memory allocated when the collation sorts characters.

14.10.17.2.14 COLLATION_CHARACTER_SET_APPLICABILITY

The `COLLATION_CHARACTER_SET_APPLICABILITY` table maps collations to the applicable character set name. Similar to the `COLLATIONS` table, it is included only for compatibility with MySQL.

```
USE information_schema;
DESC collation_character_set_applicability;
```

```
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| COLLATION_NAME     | varchar(32) | NO   |     | NULL    |       |
| CHARACTER_SET_NAME | varchar(32) | NO   |     | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
SELECT * FROM collation_character_set_applicability WHERE
    ↪ character_set_name='utf8mb4';
```

```
+---------------+--------------------+
| COLLATION_NAME | CHARACTER_SET_NAME |
+---------------+--------------------+
```

```
| utf8mb4_bin  | utf8mb4          |
+--------------+------------------+
1 row in set (0.00 sec)
```

The description of columns in the `COLLATION_CHARACTER_SET_APPLICABILITY` table is as follows:

- `COLLATION_NAME`: The name of the collation.
- `CHARACTER_SET_NAME`: The name of the character set which the collation belongs to.

14.10.17.2.15 COLUMNS

The `COLUMNS` table provides detailed information about columns in tables.

```
USE information_schema;
DESC columns;
```

```
+--
    ↪ ----------------------+--------------+------+------+---------+-------+
    ↪
| Field                  | Type         | Null | Key | Default | Extra |
+--
    ↪ ----------------------+--------------+------+------+---------+-------+
    ↪
| TABLE_CATALOG          | varchar(512) | YES |     | NULL   |       |
| TABLE_SCHEMA           | varchar(64)  | YES |     | NULL   |       |
| TABLE_NAME             | varchar(64)  | YES |     | NULL   |       |
| COLUMN_NAME            | varchar(64)  | YES |     | NULL   |       |
| ORDINAL_POSITION       | bigint(64)   | YES |     | NULL   |       |
| COLUMN_DEFAULT         | text         | YES |     | NULL   |       |
| IS_NULLABLE            | varchar(3)   | YES |     | NULL   |       |
| DATA_TYPE              | varchar(64)  | YES |     | NULL   |       |
| CHARACTER_MAXIMUM_LENGTH | bigint(21) | YES |   | NULL   |       |
| CHARACTER_OCTET_LENGTH | bigint(21)  | YES |     | NULL   |       |
| NUMERIC_PRECISION      | bigint(21)   | YES |     | NULL   |       |
| NUMERIC_SCALE          | bigint(21)   | YES |     | NULL   |       |
| DATETIME_PRECISION     | bigint(21)   | YES |     | NULL   |       |
| CHARACTER_SET_NAME     | varchar(32)  | YES |     | NULL   |       |
| COLLATION_NAME         | varchar(32)  | YES |     | NULL   |       |
| COLUMN_TYPE            | text         | YES |     | NULL   |       |
| COLUMN_KEY             | varchar(3)   | YES |     | NULL   |       |
| EXTRA                  | varchar(30)  | YES |     | NULL   |       |
| PRIVILEGES             | varchar(80)  | YES |     | NULL   |       |
| COLUMN_COMMENT         | varchar(1024) | YES |   | NULL   |       |
| GENERATION_EXPRESSION  | text         | NO  |     | NULL   |       |
```

```
+--
    ↪ ----------------------+--------------+------+-----+---------+-------+
    ↪
21 rows in set (0.00 sec)
```

```
CREATE TABLE test.t1 (a int);
SELECT * FROM columns WHERE table_schema='test' AND TABLE_NAME='t1'\G
```

```
*************************** 1. row ***************************
          TABLE_CATALOG: def
           TABLE_SCHEMA: test
             TABLE_NAME: t1
            COLUMN_NAME: a
       ORDINAL_POSITION: 1
         COLUMN_DEFAULT: NULL
            IS_NULLABLE: YES
              DATA_TYPE: int
CHARACTER_MAXIMUM_LENGTH: NULL
  CHARACTER_OCTET_LENGTH: NULL
       NUMERIC_PRECISION: 11
           NUMERIC_SCALE: 0
      DATETIME_PRECISION: NULL
      CHARACTER_SET_NAME: NULL
          COLLATION_NAME: NULL
             COLUMN_TYPE: int(11)
              COLUMN_KEY:
                   EXTRA:
              PRIVILEGES: select,insert,update,references
          COLUMN_COMMENT:
   GENERATION_EXPRESSION:
1 row in set (0.02 sec)
```

The description of columns in the `COLUMNS` table is as follows:

- `TABLE_CATALOG`: The name of the catalog to which the table with the column belongs. The value is always `def`.
- `TABLE_SCHEMA`: The name of the schema in which the table with the column is located.
- `TABLE_NAME`: The name of the table with the column.
- `COLUMN_NAME`: The name of the column.
- `ORDINAL_POSITION`: The position of the column in the table.
- `COLUMN_DEFAULT`: The default value of the column. If the explicit default value is `NULL`, or if the column definition does not include the `default` clause, this value is `NULL`.
- `IS_NULLABLE`: Whether the column is nullable. If the column can store null values, this value is `YES`; otherwise, it is `NO`.

- `DATA_TYPE`: The type of data in the column.
- `CHARACTER_MAXIMUM_LENGTH`: For string columns, the maximum length in characters.
- `CHARACTER_OCTET_LENGTH`: For string columns, the maximum length in bytes.
- `NUMERIC_PRECISION`: The numeric precision of a number-type column.
- `NUMERIC_SCALE`: The numeric scale of a number-type column.
- `DATETIME_PRECISION`: For time-type columns, the fractional seconds precision.
- `CHARACTER_SET_NAME`: The name of the character set of a string column.
- `COLLATION_NAME`: The name of the collation of a string column.
- `COLUMN_TYPE`: The column type.
- `COLUMN_KEY`: Whether this column is indexed. This field might have the following values:
    - Empty: This column is not indexed, or this column is indexed and is the second column in a multi-column non-unique index.
    - `PRI`: This column is the primary key or one of multiple primary keys.
    - `UNI`: This column is the first column of the unique index.
    - `MUL`: The column is the first column of a non-unique index, in which a given value is allowed to occur for multiple times.
- `EXTRA`: Any additional information of the given column.
- `PRIVILEGES`: The privilege that the current user has on this column. Currently, this value is fixed in TiDB, and is always `select,insert,update,references`.
- `COLUMN_COMMENT`: Comments contained in the column definition.
- `GENERATION_EXPRESSION`: For generated columns, this value displays the expression used to calculate the column value. For non-generated columns, the value is empty.

The corresponding `SHOW` statement is as follows:

```
SHOW COLUMNS FROM t1 FROM test;
```

```
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

14.10.17.2.16 DATA_LOCK_WAITS

The `DATA_LOCK_WAITS` table shows the ongoing pessimistic locks waiting on all TiKV nodes in the cluster.

```
USE information_schema;
DESC data_lock_waits;
```

```
+--
    ↪ ---------------------+---------------------+------+------+---------+-------+
    ↪
| Field                | Type                | Null | Key | Default | Extra |
+--
    ↪ ---------------------+---------------------+------+------+---------+-------+
    ↪
| KEY                  | text                | NO   |     | NULL    |       |
| KEY_INFO             | text                | YES  |     | NULL    |       |
| TRX_ID               | bigint(21) unsigned | NO   |     | NULL    |       |
| CURRENT_HOLDING_TRX_ID | bigint(21) unsigned | NO | | NULL    |       |
| SQL_DIGEST           | varchar(64)         | YES  |     | NULL    |       |
| SQL_DIGEST_TEXT      | text                | YES  |     | NULL    |       |
+--
    ↪ ---------------------+---------------------+------+------+---------+-------+
    ↪
```

The meaning of each column field in the `DATA_LOCK_WAITS` table is as follows:

- `KEY`: The key that is waiting for the lock and in the hexadecimal form.
- `KEY_INFO`: The detailed information of `KEY`. See the KEY_INFO section.
- `TRX_ID`: The ID of the transaction that is waiting for the lock. This ID is also the `start_ts` of the transaction.
- `CURRENT_HOLDING_TRX_ID`: The ID of the transaction that currently holds the lock. This ID is also the `start_ts` of the transaction.
- `SQL_DIGEST`: The digest of the SQL statement that is currently blocked in the lock-waiting transaction.
- `SQL_DIGEST_TEXT`: The normalized SQL statement (the SQL statement without arguments and formats) that is currently blocked in the lock-waiting transaction. It corresponds to `SQL_DIGEST`.

Warning:

- Only the users with the PROCESS privilege can query this table.
- Currently, the table can only record the **pessimistic lock waiting** information. If an optimistic transaction (such as an autocommit transaction) is blocked by a pessimistic lock, this table will not display the corresponding lock waiting information.
- The information in the `DATA_LOCK_WAITS` table is obtained in real time from all TiKV nodes during the query. Currently, even if a query has the `WHERE` condition, the information collection is still performed on all

TiKV nodes. If your cluster is large and the load is high, querying this table might cause potential risk of performance jitter. Therefore, use it according to your actual situation.
- Information from different TiKV nodes is NOT guaranteed to be snapshots of the same time.
- The information (SQL digest) in the `SQL_DIGEST` column is the hash value calculated from the normalized SQL statement. The information in the `SQL_DIGEST_TEXT` column is internally queried from statements summary tables, so it is possible that the corresponding statement cannot be found internally. For the detailed description of SQL digests and the statements summary tables, see Statement Summary Tables.

`KEY_INFO`

The `KEY_INFO` column shows the detailed information of the `KEY` column. The information is shown in the JSON format. The description of each field is as follows:

- `"db_id"`: The ID of the schema to which the key belongs.
- `"db_name"`: The name of the schema to which the key belongs.
- `"table_id"`: The ID of the table to which the key belongs.
- `"table_name"`: The name of the table to which the key belongs.
- `"partition_id"`: The ID of the partition where the key is located.
- `"partition_name"`: The name of the partition where the key is located.
- `"handle_type"`: The handle type of the row key (that is, the key that stores a row of data). The possible values are as follows:
  - `"int"`: The handle type is int, which means that the handle is the row ID.
  - `"common"`: The handle type is not int64. This type is shown in the non-int primary key when clustered index is enabled.
  - `"unknown"`: The handle type is currently not supported.
- `"handle_value"`: The handle value.
- `"index_id"`: The index ID to which the index key (the key that stores the index) belongs.
- `"index_name"`: The name of the index to which the index key belongs.
- `"index_values"`: The index value in the index key.

In the above fields, if the information of a field is not applicable or currently unavailable, the field is omitted in the query result. For example, the row key information does not contain `index_id`, `index_name`, and `index_values`; the index key does not contain `handle_type` and `handle_value`; non-partitioned tables do not display `partition_id` and `partition_name` ↪ ; the key information in the deleted table cannot obtain schema information such as `table_name`, `db_id`, `db_name`, and `index_name`, and it is unable to distinguish whether the table is a partitioned table.

Note:

If a key comes from a table with partitioning enabled, and the information of the schema to which the key belongs cannot be queried due to some reasons (for example, the table to which the key belongs has been deleted) during the query, the ID of the partition to which the key belongs might be appear in the `table_id` field. This is because TiDB encodes the keys of different partitions in the same way as it encodes the keys of several independent tables. Therefore, when the schema information is missing, TiDB cannot confirm whether the key belongs to an unpartitioned table or to one partition of a table.

Example

```
select * from information_schema.data_lock_waits\G
```

```
*************************** 1. row ***************************
                KEY: 7480000000000000355F728000000000000001
           KEY_INFO: {"db_id":1,"db_name":"test","table_id":53,"table_name
              ↪ ":"t","handle_type":"int","handle_value":"1"}
             TRX_ID: 426790594290122753
CURRENT_HOLDING_TRX_ID: 426790590082449409
          SQL_DIGEST: 38
              ↪ b03afa5debbdf0326a014dbe5012a62c51957f1982b3093e748460f8b00821
              ↪
      SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ?
1 row in set (0.01 sec)
```

The above query result shows that the transaction of the ID 426790594290122753 is trying to obtain the pessimistic lock on the key "7480000000000000355F728000000000000001" when executing a statement that has digest "38b03afa5debbdf0326a014dbe5012a62c51957f1982b3093e74
↪ " and is in the form of `update `t` set `v` = `v` + ? where `id` = ?`, but the lock on this key was held by the transaction of the ID 426790590082449409.

14.10.17.2.17 DDL_JOBS

The `DDL_JOBS` table provides an `INFORMATION_SCHEMA` interface to the `ADMIN SHOW ↪ DDL JOBS` command. It provides both the current status and a short history of DDL operations across the TiDB cluster.

```
USE information_schema;
DESC ddl_jobs;
```

```
+--------------+------------+------+-----+---------+-------+
| Field        | Type       | Null | Key | Default | Extra |
+--------------+------------+------+-----+---------+-------+
| JOB_ID       | bigint(21) | YES  |     | NULL    |       |
| DB_NAME      | varchar(64) | YES |     | NULL    |       |
| TABLE_NAME   | varchar(64) | YES |     | NULL    |       |
| JOB_TYPE     | varchar(64) | YES |     | NULL    |       |
| SCHEMA_STATE | varchar(64) | YES |     | NULL    |       |
| SCHEMA_ID    | bigint(21) | YES  |     | NULL    |       |
| TABLE_ID     | bigint(21) | YES  |     | NULL    |       |
| ROW_COUNT    | bigint(21) | YES  |     | NULL    |       |
| START_TIME   | datetime   | YES  |     | NULL    |       |
| END_TIME     | datetime   | YES  |     | NULL    |       |
| STATE        | varchar(64) | YES |     | NULL    |       |
| QUERY        | varchar(64) | YES |     | NULL    |       |
+--------------+------------+------+-----+---------+-------+
12 rows in set (0.00 sec)
```

```
SELECT * FROM ddl_jobs LIMIT 3\G
```

```
*************************** 1. row ***************************
     JOB_ID: 44
    DB_NAME: mysql
  TABLE_NAME: opt_rule_blacklist
    JOB_TYPE: create table
SCHEMA_STATE: public
   SCHEMA_ID: 3
    TABLE_ID: 43
   ROW_COUNT: 0
  START_TIME: 2020-07-06 15:24:27
    END_TIME: 2020-07-06 15:24:27
       STATE: synced
       QUERY: CREATE TABLE IF NOT EXISTS mysql.opt_rule_blacklist (
        name char(100) NOT NULL
    );
*************************** 2. row ***************************
     JOB_ID: 42
    DB_NAME: mysql
  TABLE_NAME: expr_pushdown_blacklist
    JOB_TYPE: create table
SCHEMA_STATE: public
   SCHEMA_ID: 3
    TABLE_ID: 41
   ROW_COUNT: 0
```

```
  START_TIME: 2020-07-06 15:24:27
    END_TIME: 2020-07-06 15:24:27
       STATE: synced
       QUERY: CREATE TABLE IF NOT EXISTS mysql.expr_pushdown_blacklist (
        name char(100) NOT NULL,
        store_type char(100) NOT NULL DEFAULT 'tikv,tiflash,tidb',
        reason varchar(200)
    );
*************************** 3. row ***************************
      JOB_ID: 40
     DB_NAME: mysql
  TABLE_NAME: stats_top_n
    JOB_TYPE: create table
SCHEMA_STATE: public
   SCHEMA_ID: 3
    TABLE_ID: 39
   ROW_COUNT: 0
  START_TIME: 2020-07-06 15:24:26
    END_TIME: 2020-07-06 15:24:27
       STATE: synced
       QUERY: CREATE TABLE if not exists mysql.stats_top_n (
        table_id bigint(64) NOT NULL,
        is_index tinyint(2) NOT NULL,
        hist_id bigint(64) NOT NULL,
        value longblob,
        count bigint(64) UNSIGNED NOT NULL,
        index tbl(table_id, is_index, hist_id)
    );
3 rows in set (0.01 sec)
```

14.10.17.2.18 DEADLOCKS

The DEADLOCKS table shows the information of the several deadlock errors that have occurred recently on the current TiDB node.

```
USE information_schema;
DESC deadlocks;
```

```
+--
    ↪ ----------------------+--------------------+------+------+---------+-------+
    ↪
| Field                | Type               | Null | Key | Default | Extra |
+--
    ↪ ----------------------+--------------------+------+------+---------+-------+
    ↪
```

```
| DEADLOCK_ID          | bigint(21)          | NO   |     | NULL    |       |
| OCCUR_TIME           | timestamp(6)        | YES  |     | NULL    |       |
| RETRYABLE            | tinyint(1)          | NO   |     | NULL    |       |
| TRY_LOCK_TRX_ID      | bigint(21) unsigned | NO   |     | NULL    |       |
| CURRENT_SQL_DIGEST   | varchar(64)         | YES  |     | NULL    |       |
| CURRENT_SQL_DIGEST_TEXT | text             | YES  |     | NULL    |       |
| KEY                  | text                | YES  |     | NULL    |       |
| KEY_INFO             | text                | YES  |     | NULL    |       |
| TRX_HOLDING_LOCK     | bigint(21) unsigned | NO   |     | NULL    |       |
+--
    ↪ ---------------------+---------------------+------+------+---------+-------+
    ↪
```

The `DEADLOCKS` table uses multiple rows to show the same deadlock event, and each row displays the information about one of the transactions involved in the deadlock event. If the TiDB node records multiple deadlock errors, each error is distinguished using the `DEADLOCK_ID` column. The same `DEADLOCK_ID` indicates the same deadlock event. Note that `DEADLOCK_ID` **does not guarantee global uniqueness and will not be persisted**. It only shows the same deadlock event in the same result set.

The meaning of each column field in the `DEADLOCKS` table is as follows:

- `DEADLOCK_ID`: The ID of the deadlock event. When multiple deadlock errors exist in the table, you can use this column to distinguish rows that belong to different deadlock errors.
- `OCCUR_TIME`: The time when the deadlock error occurs.
- `RETRYABLE`: Whether the deadlock error can be retried. For the description of retryable deadlock errors, see the Retryable deadlock errors section.
- `TRY_LOCK_TRX_ID`: The ID of the transaction that tries to acquire lock. This ID is also the `start_ts` of the transaction.
- `CURRENT_SQL_DIGEST`: The digest of the SQL statement currently being executed in the lock-acquiring transaction.
- `CURRENT_SQL_DIGEST_TEXT`: The normalized form of the SQL statement that is currently being executed in the lock-acquiring transaction.
- `KEY`: The blocked key that the transaction tries to lock. The value of this field is displayed in the form of hexadecimal string.
- `KEY_INFO`: The detailed information of `KEY`. See the KEY_INFO section.
- `TRX_HOLDING_LOCK`: The ID of the transaction that currently holds the lock on the key and causes blocking. This ID is also the `start_ts` of the transaction.

To adjust the maximum number of deadlock events that can be recorded in the `DEADLOCKS` table, adjust the `pessimistic-txn.deadlock-history-capacity` configuration in the TiDB configuration file. By default, the information of the recent 10 deadlock events is recorded in the table.

Warning:

- Only users with the PROCESS privilege can query this table.
- The information (SQL digest) in the `CURRENT_SQL_DIGEST` column is the hash value calculated from the normalized SQL statement. The information in the `CURRENT_SQL_DIGEST_TEXT` column is internally queried from statements summary tables, so it is possible that the corresponding statement cannot be found internally. For the detailed description of SQL digests and the statements summary tables, see Statement Summary Tables.

`KEY_INFO`

The `KEY_INFO` column shows the detailed information of the `KEY` column. The information is shown in the JSON format. The description of each field is as follows:

- `"db_id"`: The ID of the schema to which the key belongs.
- `"db_name"`: The name of the schema to which the key belongs.
- `"table_id"`: The ID of the table to which the key belongs.
- `"table_name"`: The name of the table to which the key belongs.
- `"partition_id"`: The ID of the partition where the key is located.
- `"partition_name"`: The name of the partition where the key is located.
- `"handle_type"`: The handle type of the row key (that is, the key that stores a row of data). The possible values are as follows:
  - `"int"`: The handle type is int, which means that the handle is the row ID.
  - `"common"`: The handle type is not int64. This type is shown in the non-int primary key when clustered index is enabled.
  - `"unknown"`: The handle type is currently not supported.
- `"handle_value"`: The handle value.
- `"index_id"`: The index ID to which the index key (the key that stores the index) belongs.
- `"index_name"`: The name of the index to which the index key belongs.
- `"index_values"`: The index value in the index key.

In the above fields, if the information of a field is not applicable or currently unavailable, the field is omitted in the query result. For example, the row key information does not contain `index_id`, `index_name`, and `index_values`; the index key does not contain `handle_type` and `handle_value`; non-partitioned tables do not display `partition_id` and `partition_name` ↪ ; the key information in the deleted table cannot obtain schema information such as `table_name`, `db_id`, `db_name`, and `index_name`, and it is unable to distinguish whether the table is a partitioned table.

> **Note:**
>
> If a key comes from a table with partitioning enabled, and the information of the schema to which the key belongs cannot be queried due to some reasons (for example, the table to which the key belongs has been deleted) during the query, the ID of the partition to which the key belongs might be appear in the `table_id` field. This is because TiDB encodes the keys of different partitions in the same way as it encodes the keys of several independent tables. Therefore, when the schema information is missing, TiDB cannot confirm whether the key belongs to an unpartitioned table or to one partition of a table.

Retryable deadlock errors

> **Note:**
>
> The `DEADLOCKS` table does not collect the information of retryable deadlock errors by default. If you want the table to collect the retryable deadlock error information, you can adjust the value of `pessimistic-txn.deadlock-history-collect-retryable` in the TiDB configuration file.

When transaction A is blocked by a lock already held by transaction B, and transaction B is directly or indirectly blocked by the lock held by the current transaction A, a deadlock error will occur. In this deadlock, there might be two cases:

- Case 1: Transaction B might be (directly or indirectly) blocked by a lock generated by a statement that has been executed after transaction A starts and before transaction A gets blocked.
- Case 2: Transaction B might also be blocked by the statement currently being executed in transaction A.

In case 1, TiDB will report a deadlock error to the client of transaction A and terminate the transaction.

In case 2, the statement currently being executed in transaction A will be automatically retried in TiDB. For example, suppose that transaction A executes the following statement:

```
update t set v = v + 1 where id = 1 or id = 2;
```

Transaction B executes the following two statements successively.

```
update t set v = 4 where id = 2;
update t set v = 2 where id = 1;
```

Then if transaction A locks the two rows with `id = 1` and `id = 2`, and the two transactions run in the following sequence:

1. Transaction A locks the row with `id = 1`.
2. Transaction B executes the first statement and locks the row with `id = 2`.
3. Transaction B executes the second statement and tries to lock the row with `id = 1`, which is blocked by transaction A.
4. Transaction A tries to lock the row with `id = 2` and is blocked by transaction B, which forms a deadlock.

For this case, because the statement of transaction A that blocks other transactions is also the statement currently being executed, the pessimistic lock on the current statement can be resolved (so that transaction B can continue to run), and the current statement can be retried. TiDB uses the key hash internally to determine whether this is the case.

When a retryable deadlock occurs, the internal automatic retry will not cause a transaction error, so it is transparent to the client. However, if this situation occurs frequently, the performance might be affected. When this occurs, you can see `single statement deadlock, retry statement` in the TiDB log.

Example 1

Assume that the table definition and the initial data are as follows:

```
create table t (id int primary key, v int);
insert into t values (1, 10), (2, 20);
```

Two transactions are executed in the following order:

Transaction 1	Transaction 2	Description
`update t set v = 11 where id = 1;`		
	`update t set v = 21 where id = 2;`	
`update t set v = 12 where id = 2;`		Transaction 1 gets bl
	`update t set v = 22 where id = 1;`	Transaction 2 reports

Next, transaction 2 reports a deadlock error. At this time, query the `DEADLOCKS` table:

```
select * from information_schema.deadlocks;
```

The expected output is as follows:

```
+--
    ↪ ----------+----------------------------+----------+--------------------+------
```

```
    ↪
| DEADLOCK_ID | OCCUR_TIME            | RETRYABLE | TRY_LOCK_TRX_ID |
    ↪ CURRENT_SQL_DIGEST                                          |
    ↪ CURRENT_SQL_DIGEST_TEXT            | KEY
    ↪                                  | KEY_INFO
    ↪
    ↪ | TRX_HOLDING_LOCK |
+--
    ↪ ----------+----------------------------+----------+-------------------+------
    ↪
|           1 | 2021-08-05 11:09:03.230341 |    0 | 426812829645406216 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000002 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"2"} |
    ↪ 426812829645406217 |
|           1 | 2021-08-05 11:09:03.230341 |    0 | 426812829645406217 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000001 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"1"} |
    ↪ 426812829645406216 |
+--
    ↪ ----------+----------------------------+----------+-------------------+------
    ↪
```

Two rows of data are generated in the `DEADLOCKS` table. The `DEADLOCK_ID` field of both rows is `1`, which means that the information in both rows belongs to the same deadlock error. The first row shows that on the key of `"74800000000000000355F728000000000000002"` ↪ , the transaction of the ID `"426812829645406216"` is blocked by the transaction of the ID `"426812829645406217"`. The second row shows that on the key of `"7480000000000000355` ↪ `F728000000000000001"`, the transaction of the ID `"426812829645406217"` is blocked by the transaction of the ID `426812829645406216`, which constitutes mutual blocking and forms a deadlock.

Example 2

Assume that you query the `DEADLOCKS` table and get the following result:

```
+--
    ↪ ----------+----------------------------+----------+-------------------+------
    ↪
| DEADLOCK_ID | OCCUR_TIME            | RETRYABLE | TRY_LOCK_TRX_ID |
    ↪ CURRENT_SQL_DIGEST                                          |
    ↪ CURRENT_SQL_DIGEST_TEXT            | KEY
    ↪                                  | KEY_INFO
    ↪
```

```
    ↪ | TRX_HOLDING_LOCK |
+--
    ↪ ----------+----------------------------+----------+--------------------+------
    ↪
|         1 | 2021-08-05 11:09:03.230341 |    0 | 426812829645406216 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000002 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"2"} |
    ↪ 426812829645406217 |
|         1 | 2021-08-05 11:09:03.230341 |    0 | 426812829645406217 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000001 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"1"} |
    ↪ 426812829645406216 |
|         2 | 2021-08-05 11:09:21.252154 |    0 | 426812832017809412 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000002 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"2"} |
    ↪ 426812832017809413 |
|         2 | 2021-08-05 11:09:21.252154 |    0 | 426812832017809413 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000003 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"3"} |
    ↪ 426812832017809414 |
|         2 | 2021-08-05 11:09:21.252154 |    0 | 426812832017809414 |
    ↪ 22230766411edb40f27a68dadefc63c6c6970d5827f1e5e22fc97be2c4d8350d |
    ↪ update `t` set `v` = ? where `id` = ? ; | 7480000000000000355
    ↪ F728000000000000001 | {"db_id":1,"db_name":"test","table_id":53,"
    ↪ table_name":"t","handle_type":"int","handle_value":"1"} |
    ↪ 426812832017809412 |
+--
    ↪ ----------+----------------------------+----------+--------------------+------
    ↪
```

The `DEADLOCK_ID` column in the above query result shows that the first two rows together represent the information of a deadlock error, and the two transactions that wait for each other form the deadlock. The next three rows together represent the information of another deadlock error, and the three transactions that wait in a cycle form the deadlock.

CLUSTER_DEADLOCKS

The `CLUSTER_DEADLOCKS` table returns information about the recent deadlock errors on

each TiDB node in the entire cluster, which is the combined information of the DEADLOCKS table on each node. CLUSTER_DEADLOCKS also includes an additional INSTANCE column to display the IP address and port of the node to distinguish between different TiDB nodes.

Note that, because DEADLOCK_ID does not guarantee global uniqueness, in the query result of the CLUSTER_DEADLOCKS table, you need to use the INSTANCE and DEADLOCK_ID together to distinguish the information of different deadlock errors in the result set.

```
USE information_schema;
DESC cluster_deadlocks;
```

```
+--
    ↪ ----------------------+--------------------+------+------+---------+-------+
    ↪
| Field                | Type               | Null | Key | Default | Extra |
+--
    ↪ ----------------------+--------------------+------+------+---------+-------+
    ↪
| INSTANCE             | varchar(64)        | YES |      | NULL    |       |
| DEADLOCK_ID          | bigint(21)         | NO  |      | NULL    |       |
| OCCUR_TIME           | timestamp(6)       | YES |      | NULL    |       |
| RETRYABLE            | tinyint(1)         | NO  |      | NULL    |       |
| TRY_LOCK_TRX_ID      | bigint(21) unsigned | NO |    | NULL    |       |
| CURRENT_SQL_DIGEST   | varchar(64)        | YES |      | NULL    |       |
| CURRENT_SQL_DIGEST_TEXT | text            | YES |      | NULL    |       |
| KEY                  | text               | YES |      | NULL    |       |
| KEY_INFO             | text               | YES |      | NULL    |       |
| TRX_HOLDING_LOCK     | bigint(21) unsigned | NO |    | NULL    |       |
+--
    ↪ ----------------------+--------------------+------+------+---------+-------+
    ↪
```

14.10.17.2.19 ENGINES

The ENGINES table provides information about storage engines. For compatibility, TiDB will always describe InnoDB as the only supported engine. In addition, other column values in the ENGINES table are also fixed values.

```
USE information_schema;
DESC engines;
```

```
+-------------+-------------+------+------+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+------+---------+-------+
| ENGINE      | varchar(64) | YES |     | NULL    |       |
| SUPPORT     | varchar(8) | YES |     | NULL    |       |
```

```
| COMMENT      | varchar(80) | YES |     | NULL    |         |
| TRANSACTIONS | varchar(3)  | YES |     | NULL    |         |
| XA           | varchar(3)  | YES |     | NULL    |         |
| SAVEPOINTS   | varchar(3)  | YES |     | NULL    |         |
+--------------+-------------+-----+-----+---------+---------+
6 rows in set (0.00 sec)
```

```
SELECT * FROM engines;
```

```
+--------+---------+----------------------------------------------------------------+-------
    ↪
| ENGINE | SUPPORT | COMMENT                                                        |
    ↪ TRANSACTIONS | XA | SAVEPOINTS |
+--------+---------+----------------------------------------------------------------+-------
    ↪
| InnoDB | DEFAULT | Supports transactions, row-level locking, and foreign
    ↪ keys | YES | YES | YES    |
+--------+---------+----------------------------------------------------------------+-------
    ↪
1 row in set (0.01 sec)
```

The description of columns in the `ENGINES` table is as follows:

- `ENGINES`: The name of the storage engine.
- `SUPPORT`: The level of support that the server has on the storage engine. In TiDB, the value is always `DEFAULT`.
- `COMMENT`: The brief comment on the storage engine.
- `TRANSACTIONS`: Whether the storage engine supports transactions.
- `XA`: Whether the storage engine supports XA transactions.
- `SAVEPOINTS`: Whether the storage engine supports `savepoints`.

14.10.17.2.20 INSPECTION_RESULT

TiDB has some built-in diagnostic rules for detecting faults and hidden issues in the system.

The `INSPECTION_RESULT` diagnostic feature can help you quickly find problems and reduce your repetitive manual work. You can use the `select * from information_schema ↪ .inspection_result` statement to trigger the internal diagnostics.

The structure of the `information_schema.inspection_result` diagnostic result table `information_schema.inspection_result` is as follows:

```
USE information_schema;
DESC inspection_result;
```

```
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| RULE           | varchar(64) | YES  |     | NULL    |       |
| ITEM           | varchar(64) | YES  |     | NULL    |       |
| TYPE           | varchar(64) | YES  |     | NULL    |       |
| INSTANCE       | varchar(64) | YES  |     | NULL    |       |
| STATUS_ADDRESS | varchar(64) | YES  |     | NULL    |       |
| VALUE          | varchar(64) | YES  |     | NULL    |       |
| REFERENCE      | varchar(64) | YES  |     | NULL    |       |
| SEVERITY       | varchar(64) | YES  |     | NULL    |       |
| DETAILS        | varchar(256) | YES |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
9 rows in set (0.00 sec)
```

Field description:

- `RULE`: The name of the diagnostic rule. Currently, the following rules are available:
    - `config`: Checks whether the configuration is consistent and proper. If the same configuration is inconsistent on different instances, a `warning` diagnostic result is generated.
    - `version`: The consistency check of version. If the same version is inconsistent on different instances, a `warning` diagnostic result is generated.
    - `node-load`: Checks the server load. If the current system load is too high, the corresponding `warning` diagnostic result is generated.
    - `critical-error`: Each module of the system defines critical errors. If a critical error exceeds the threshold within the corresponding time period, a warning diagnostic result is generated.
    - `threshold-check`: The diagnostic system checks the thresholds of key metrics. If a threshold is exceeded, the corresponding diagnostic information is generated.
- `ITEM`: Each rule diagnoses different items. This field indicates the specific diagnostic items corresponding to each rule.
- `TYPE`: The instance type of the diagnostics. The optional values are `tidb`, `pd`, and `tikv`.
- `INSTANCE`: The specific address of the diagnosed instance.
- `STATUS_ADDRESS`: The HTTP API service address of the instance.
- `VALUE`: The value of a specific diagnostic item.
- `REFERENCE`: The reference value (threshold value) for this diagnostic item. If `VALUE` exceeds the threshold, the corresponding diagnostic information is generated.
- `SEVERITY`: The severity level. The optional values are `warning` and `critical`.
- `DETAILS`: Diagnostic details, which might also contain SQL statement(s) or document links for further diagnostics.

Diagnostics example

Diagnose issues currently existing in the cluster.

```
SELECT * FROM information_schema.inspection_result\G
```

```
***************************[ 1. row ]***************************
RULE     | config
ITEM     | log.slow-threshold
TYPE     | tidb
INSTANCE | 172.16.5.40:4000
VALUE    | 0
REFERENCE | not 0
SEVERITY | warning
DETAILS  | slow-threshold = 0 will record every query to slow log, it may
   ↪ affect performance
***************************[ 2. row ]***************************
RULE     | version
ITEM     | git_hash
TYPE     | tidb
INSTANCE |
VALUE    | inconsistent
REFERENCE | consistent
SEVERITY | critical
DETAILS  | the cluster has 2 different tidb version, execute the sql to see
   ↪ more detail: select * from information_schema.cluster_info where type
   ↪ ='tidb'
***************************[ 3. row ]***************************
RULE     | threshold-check
ITEM     | storage-write-duration
TYPE     | tikv
INSTANCE | 172.16.5.40:23151
VALUE    | 130.417
REFERENCE | < 0.100
SEVERITY | warning
DETAILS  | max duration of 172.16.5.40:23151 tikv storage-write-duration was
   ↪  too slow
***************************[ 4. row ]***************************
RULE     | threshold-check
ITEM     | rocksdb-write-duration
TYPE     | tikv
INSTANCE | 172.16.5.40:20151
VALUE    | 108.105
REFERENCE | < 0.100
SEVERITY | warning
DETAILS  | max duration of 172.16.5.40:20151 tikv rocksdb-write-duration was
```

```
    ↪  too slow
```

The following issues can be detected from the diagnostic result above:

- The first row indicates that TiDB's `log.slow-threshold` value is configured to `0`, which might affect performance.
- The second row indicates that two different TiDB versions exist in the cluster.
- The third and fourth rows indicate that the TiKV write delay is too long. The expected delay is no more than 0.1 second, while the actual delay is far longer than expected.

You can also diagnose issues existing within a specified range, such as from "2020-03-26 00:03:00" to "2020-03-26 00:08:00". To specify the time range, use the SQL Hint of `/*+ time_range()*/`. See the following query example:

```
select /*+ time_range("2020-03-26 00:03:00", "2020-03-26 00:08:00") */ *
    ↪ from information_schema.inspection_result\G
```

```
***************************[ 1. row ]***************************
RULE     | critical-error
ITEM     | server-down
TYPE     | tidb
INSTANCE | 172.16.5.40:4009
VALUE    |
REFERENCE |
SEVERITY | critical
DETAILS  | tidb 172.16.5.40:4009 restarted at time '2020/03/26 00:05:45.670'
***************************[ 2. row ]***************************
RULE     | threshold-check
ITEM     | get-token-duration
TYPE     | tidb
INSTANCE | 172.16.5.40:10089
VALUE    | 0.234
REFERENCE | < 0.001
SEVERITY | warning
DETAILS  | max duration of 172.16.5.40:10089 tidb get-token-duration is too
    ↪ slow
```

The following issues can be detected from the diagnostic result above:

- The first row indicates that the `172.16.5.40:4009` TiDB instance is restarted at `2020/03/26 00:05:45.670`.
- The second row indicates that the maximum `get-token-duration` time of the `172.16.5.40:10089` TiDB instance is 0.234s, but the expected time is less than 0.001s.

You can also specify conditions, for example, to query the `critical` level diagnostic results:

```
select * from information_schema.inspection_result where severity='critical
    ↪ ';
```

Query only the diagnostic result of the `critical-error` rule:

```
select * from information_schema.inspection_result where rule='critical-
    ↪ error';
```

Diagnostic rules

The diagnostic module contains a series of rules. These rules compare the results with the thresholds after querying the existing monitoring tables and cluster information tables. If the results exceed the thresholds, the diagnostics of `warning` or `critical` is generated and the corresponding information is provided in the `details` column.

You can query the existing diagnostic rules by querying the `inspection_rules` system table:

```
select * from information_schema.inspection_rules where type='inspection';
```

```
+-----------------+------------+---------+
| NAME            | TYPE       | COMMENT |
+-----------------+------------+---------+
| config          | inspection |         |
| version         | inspection |         |
| node-load       | inspection |         |
| critical-error  | inspection |         |
| threshold-check | inspection |         |
+-----------------+------------+---------+
```

`config` diagnostic rule

In the `config` diagnostic rule, the following two diagnostic rules are executed by querying the `CLUSTER_CONFIG` system table:

- Check whether the configuration values of the same component are consistent. Not all configuration items has this consistency check. The allowlist of consistency check is as follows:

```
// The allowlist of the TiDB configuration consistency check
port
status.status-port
host
path
advertise-address
```

```
status.status-port
log.file.filename
log.slow-query-file
tmp-storage-path

// The allowlist of the PD configuration consistency check
advertise-client-urls
advertise-peer-urls
client-urls
data-dir
log-file
log.file.filename
metric.job
name
peer-urls

// The allowlist of the TiKV configuration consistency check
server.addr
server.advertise-addr
server.status-addr
log-file
raftstore.raftdb-path
storage.data-dir
storage.block-cache.capacity
```

- Check whether the values of the following configuration items are as expected.

Component	Configuration item	Expected value
TiDB	log.slow-threshold	larger than `0`
TiKV	raftstore.sync-log	`true`

`version` diagnostic rule

The `version` diagnostic rule checks whether the version hash of the same component is consistent by querying the `CLUSTER_INFO` system table. See the following example:

```
SELECT * FROM information_schema.inspection_result WHERE rule='version'\G
```

```
***************************[ 1. row ]***************************
RULE     | version
ITEM     | git_hash
TYPE     | tidb
INSTANCE |
VALUE    | inconsistent
REFERENCE | consistent
```

```
SEVERITY | critical
DETAILS  | the cluster has 2 different tidb versions, execute the sql to see
    ↪  more detail: SELECT * FROM information_schema.cluster_info WHERE
    ↪ type='tidb'
```

`critical-error` diagnostic rule

In `critical-error` diagnostic rule, the following two diagnostic rules are executed:

- Detect whether the cluster has the following errors by querying the related monitoring system tables in the metrics schema:

Component	Error name	Monitoring table	Error description
TiDB	panic-count	tidb_panic_count_total_count	Panic occurs in TiDB.
TiDB	binlog-error	tidb_binlog_error_total_count	An error occurs when TiDB writes binlog.
TiKV	critical-error	tikv_critical_error_total_coun	The critical error of TiKV.

Component	Error name	Monitoring table	Error description
TiKV	scheduler-is-busy	tikv_scheduler_is_busy_total_count	The TiKV scheduler is too busy, which makes TiKV temporarily unavailable.
TiKV	coprocessor-is-busy	tikv_coprocessor_is_busy_total_count	The TiKV Coprocessor is too busy.
TiKV	channel-is-full	tikv_channel_full_total_count	The “channel full” error occurs in TiKV.
TiKV	tikv_engine_write_stall	tikv_engine_write_stall	The “stall” error occurs in TiKV.

- Check whether any component is restarted by querying the `metrics_schema.up` monitoring table and the `CLUSTER_LOG` system table.

`threshold-check` diagnostic rule

The `threshold-check` diagnostic rule checks whether the following metrics in the cluster exceed the threshold by querying the related monitoring system tables in the metrics schema:

Component	Monitoring metric	Monitoring table	Expected value	Description
TiDB	tso-duration	pd_tso_wait_duration	< 50ms	The wait duration of getting the TSO of transaction.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiDB	get-token-duration	tidb_get_token_duration	< 1ms	Query the time it takes to get the token. The related TiDB configuration item is `token-limit`.
TiDB	load-schema-duration	tidb_load_schema_duration	< 1s	The time it takes for TiDB to update the schema metadata.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiKV	scheduler cmd-duration	tikv_scheduler_command_duration	< 0.1s	The time it takes for TiKV to execute the KV cmd request.
TiKV	handle-snapshot-duration	tikv_handle_snapshot_duration	< 30s	The time it takes for TiKV to handle the snapshot.
TiKV	storage-write-duration	tikv_storage_async_request_duration	< 0.1s	The write latency of TiKV.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiKV	storage-snapshot-duration	tikv_storage_async_request_duration	< 50ms	The time it takes for TiKV to get the snapshot.
TiKV	rocksdb-write-duration	tikv_engine_write_duration	< 100ms	The write latency of TiKV RocksDB.
TiKV	rocksdb-get-duration	tikv_engine_max_get_duration	< 50ms	The read latency of TiKV RocksDB.
TiKV	rocksdb-seek-duration	tikv_engine_max_seek_duration	< 50ms	The latency of TiKV RocksDB to execute `seek`.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiKV	scheduler-pending-cmd-coun	tikv_scheduler_pen	< 1000	The number of commands stalled in TiKV.
TiKV	index-block-cache-hit	tikv_block_index_cache_hit	> 0.95	The hit rate of index block cache in TiKV.
TiKV	filter-block-cache-hit	tikv_block_filter_cache_hit	> 0.95	The hit rate of filter block cache in TiKV.
TiKV	data-block-cache-hit	tikv_block_data_cache_hit	> 0.80	The hit rate of data block cache in TiKV.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiKV	leader-score-balance	pd_scheduler_store_s	< 0.05	Checks whether the leader score of each TiKV instance is balanced. The expected difference between instances is less than 5%.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiKV	region-score-balance	pd_scheduler_store_status	< 0.05	Checks whether the Region score of each TiKV instance is balanced. The expected difference between instances is less than 5%.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiKV	store-available-balance	pd_scheduler_store_status	< 0.2	Checks whether the available storage of each TiKV instance is balanced. The expected difference between instances is less than 20%.

Component	Monitoring metric	Monitoring table	Expected value	Description
TiKV	region-count	pd_scheduler_store_status	< 20000	Checks the number of Regions on each TiKV instance. The expected number of Regions in a single instance is less than 20,000.

Component	Monitoring metric	Monitoring table	Expected value	Description
PD	region-health	pd_region_health	< 100	Detects the number of Regions that are in the process of scheduling in the cluster. The expected number is less than 100 in total.

In addition, this rule also checks whether the CPU usage of the following threads in a TiKV instance is too high:

- scheduler-worker-cpu
- coprocessor-normal-cpu
- coprocessor-high-cpu
- coprocessor-low-cpu
- grpc-cpu
- raftstore-cpu
- apply-cpu

- storage-readpool-normal-cpu
- storage-readpool-high-cpu
- storage-readpool-low-cpu
- split-check-cpu

The built-in diagnostic rules are constantly being improved. If you have more diagnostic rules, welcome to create a PR or an issue in the `tidb` repository.

14.10.17.2.21 INSPECTION_RULES

The `INSPECTION_RULES` table provides information about which diagnostic tests are run in an inspection result. See inspection result for example usage.

```
USE information_schema;
DESC inspection_rules;
```

```
+---------+-------------+------+------+---------+-------+
| Field   | Type        | Null | Key  | Default | Extra |
+---------+-------------+------+------+---------+-------+
| NAME    | varchar(64) | YES  |      | NULL    |       |
| TYPE    | varchar(64) | YES  |      | NULL    |       |
| COMMENT | varchar(256)| YES  |      | NULL    |       |
+---------+-------------+------+------+---------+-------+
3 rows in set (0.00 sec)
```

```
SELECT * FROM inspection_rules;
```

```
+-----------------+------------+---------+
| NAME            | TYPE       | COMMENT |
+-----------------+------------+---------+
| config          | inspection |         |
| version         | inspection |         |
| node-load       | inspection |         |
| critical-error  | inspection |         |
| threshold-check | inspection |         |
| ddl             | summary    |         |
| gc              | summary    |         |
| pd              | summary    |         |
| query-summary   | summary    |         |
| raftstore       | summary    |         |
| read-link       | summary    |         |
| rocksdb         | summary    |         |
| stats           | summary    |         |
| wait-events     | summary    |         |
| write-link      | summary    |         |
```

```
+-----------------+------------+---------+
15 rows in set (0.00 sec)
```

14.10.17.2.22 INSPECTION_SUMMARY

In some scenarios, you might need to pay attention only to the monitoring summary of specific links or modules. For example, the number of threads for Coprocessor in the thread pool is configured as 8. If the CPU usage of Coprocessor reaches 750%, you can determine that a risk exists and Coprocessor might become a bottleneck in advance. However, some monitoring metrics vary greatly due to different user workloads, so it is difficult to define specific thresholds. It is important to troubleshoot issues in this scenario, so TiDB provides the `inspection_summary` table for link summary.

The structure of the `information_schema.inspection_summary` inspection summary table is as follows:

```
USE information_schema;
DESC inspection_summary;
```

```
+-------------+-------------+------+------+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+------+---------+-------+
| RULE        | varchar(64) | YES |     | NULL   |       |
| INSTANCE    | varchar(64) | YES |     | NULL   |       |
| METRICS_NAME | varchar(64) | YES |   | NULL   |       |
| LABEL       | varchar(64) | YES |     | NULL   |       |
| QUANTILE    | double      | YES |     | NULL   |       |
| AVG_VALUE   | double(22,6) | YES |    | NULL   |       |
| MIN_VALUE   | double(22,6) | YES |    | NULL   |       |
| MAX_VALUE   | double(22,6) | YES |    | NULL   |       |
| COMMENT     | varchar(256) | YES |    | NULL   |       |
+-------------+-------------+------+------+---------+-------+
9 rows in set (0.00 sec)
```

Field description:

- `RULE`: Summary rules. Because new rules are being added continuously, you can execute the `select * from inspection_rules where type='summary'` statement to query the latest rule list.
- `INSTANCE`: The monitored instance.
- `METRICS_NAME`: The monitoring metrics name.
- `QUANTILE`: Takes effect on monitoring tables that contain `QUANTILE`. You can specify multiple percentiles by pushing down predicates. For example, you can execute `select * from inspection_summary where rule='ddl' and quantile in (0.80, 0.90, 0.99, 0.999)` to summarize the DDL-related monitoring

metrics and query the P80/P90/P99/P999 results. `AVG_VALUE`, `MIN_VALUE`, and `MAX_VALUE` respectively indicate the average value, minimum value, and maximum value of the aggregation.

- `COMMENT`: The comment about the corresponding monitoring metric.

> **Note:**
>
> Because summarizing all results causes overhead, it is recommended to display the specific `rule` in the SQL predicate to reduce overhead. For example, executing `select * from inspection_summary where rule in ('read-link', 'ddl')` summarizes the read link and DDL-related monitoring metrics.

Usage example:

Both the diagnostic result table and the diagnostic monitoring summary table can specify the diagnostic time range using `hint`. `select /*+ time_range('2020-03-07 12:00:00','2020-03-07 13:00:00')*/* from inspection_summary` is the monitoring summary for the `2020-03-07 12:00:00` to `2020-03-07 13:00:00` period. Like the monitoring summary table, you can use the `inspection_summary` table to quickly find the monitoring items with large differences by comparing the data of two different periods.

The following example compares the monitoring metrics of read links in two time periods:

- `(2020-01-16 16:00:54.933, 2020-01-16 16:10:54.933)`
- `(2020-01-16 16:10:54.933, 2020-01-16 16:20:54.933)`

```
SELECT
  t1.avg_value / t2.avg_value AS ratio,
  t1.*,
  t2.*
FROM
  (
    SELECT
      /*+ time_range("2020-01-16 16:00:54.933", "2020-01-16 16:10:54.933")
          ↪ */ *
    FROM information_schema.inspection_summary WHERE rule='read-link'
  ) t1
  JOIN
  (
    SELECT
      /*+ time_range("2020-01-16 16:10:54.933", "2020-01-16 16:20:54.933")
          ↪ */ *
```

```
    FROM information_schema.inspection_summary WHERE rule='read-link'
  ) t2
  ON t1.metrics_name = t2.metrics_name
  and t1.instance = t2.instance
  and t1.label = t2.label
ORDER BY
  ratio DESC;
```

14.10.17.2.23 KEY_COLUMN_USAGE

The KEY_COLUMN_USAGE table describes the key constraints of the columns, such as the primary key constraint.

```
USE information_schema;
DESC key_column_usage;
```

```
+-------------------------------+-------------+------+-----+---------+-------+
   ↪
| Field                         | Type        | Null | Key | Default | Extra |
+-------------------------------+-------------+------+-----+---------+-------+
   ↪
| CONSTRAINT_CATALOG            | varchar(512) | NO |     | NULL    |       |
| CONSTRAINT_SCHEMA             | varchar(64) | NO |     | NULL    |       |
| CONSTRAINT_NAME               | varchar(64) | NO |     | NULL    |       |
| TABLE_CATALOG                 | varchar(512) | NO |     | NULL    |       |
| TABLE_SCHEMA                  | varchar(64) | NO |     | NULL    |       |
| TABLE_NAME                    | varchar(64) | NO |     | NULL    |       |
| COLUMN_NAME                   | varchar(64) | NO |     | NULL    |       |
| ORDINAL_POSITION              | bigint(10) | NO  |     | NULL    |       |
| POSITION_IN_UNIQUE_CONSTRAINT | bigint(10) | YES |  | NULL    |       |
| REFERENCED_TABLE_SCHEMA       | varchar(64) | YES |     | NULL    |       |
| REFERENCED_TABLE_NAME         | varchar(64) | YES |     | NULL    |       |
| REFERENCED_COLUMN_NAME        | varchar(64) | YES |     | NULL    |       |
+-------------------------------+-------------+------+-----+---------+-------+
   ↪
12 rows in set (0.00 sec)
```

```
SELECT * FROM key_column_usage WHERE table_schema='mysql' and table_name='
   ↪ user';
```

```
*************************** 1. row ***************************
           CONSTRAINT_CATALOG: def
            CONSTRAINT_SCHEMA: mysql
              CONSTRAINT_NAME: PRIMARY
                TABLE_CATALOG: def
```

```
               TABLE_SCHEMA: mysql
                 TABLE_NAME: user
                COLUMN_NAME: Host
           ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: NULL
     REFERENCED_TABLE_SCHEMA: NULL
       REFERENCED_TABLE_NAME: NULL
      REFERENCED_COLUMN_NAME: NULL
*************************** 2. row ***************************
          CONSTRAINT_CATALOG: def
           CONSTRAINT_SCHEMA: mysql
             CONSTRAINT_NAME: PRIMARY
               TABLE_CATALOG: def
                TABLE_SCHEMA: mysql
                  TABLE_NAME: user
                 COLUMN_NAME: User
            ORDINAL_POSITION: 2
POSITION_IN_UNIQUE_CONSTRAINT: NULL
     REFERENCED_TABLE_SCHEMA: NULL
       REFERENCED_TABLE_NAME: NULL
      REFERENCED_COLUMN_NAME: NULL
2 rows in set (0.00 sec)
```

The description of columns in the `KEY_COLUMN_USAGE` table is as follows:

- `CONSTRAINT_CATALOG`: The name of the catalog to which the constraint belongs. The value is always `def`.
- `CONSTRAINT_SCHEMA`: The name of the schema to which the constraint belongs.
- `CONSTRAINT_NAME`: The name of the constraint.
- `TABLE_CATALOG`: The name of the catalog to which the table belongs. The value is always `def`.
- `TABLE_SCHEMA`: The name of the schema to which the table belongs.
- `TABLE_NAME`: The name of the table with constraints.
- `COLUMN_NAME`: The name of the column with constraints.
- `ORDINAL_POSITION`: The position of the column in the constraint, rather than in the table. The position number starts from `1`.
- `POSITION_IN_UNIQUE_CONSTRAINT`: The unique constraint and the primary key constraint are empty. For foreign key constraints, this column is the position of the referenced table's key.
- `REFERENCED_TABLE_SCHEMA`: The name of the schema referenced by the constraint. Currently in TiDB, the value of this column in all constraints is `nil`, except for the foreign key constraint.
- `REFERENCED_TABLE_NAME`: The name of the table referenced by the constraint. Currently in TiDB, the value of this column in all constraints is `nil`, except for the foreign key constraint.

- `REFERENCED_COLUMN_NAME`: The name of the column referenced by the constraint. Currently in TiDB, the value of this column in all constraints is `nil`, except for the foreign key constraint.

14.10.17.2.24 METRICS_SUMMARY

The TiDB cluster has many monitoring metrics. To make it easy to detect abnormal monitoring metrics, TiDB 4.0 introduces the following two monitoring summary tables:

- `information_schema.metrics_summary`
- `information_schema.metrics_summary_by_label`

The two tables summarize all monitoring data for you to check each monitoring metric efficiently. Compared with `information_schema.metrics_summary`, the `information_schema.metrics_summary_by_label` table has an additional `label` column and performs differentiated statistics according to different labels.

```
USE information_schema;
DESC metrics_summary;
```

```
+--------------+--------------+------+------+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+------+---------+-------+
| METRICS_NAME | varchar(64) | YES |   | NULL   |       |
| QUANTILE     | double       | YES |      | NULL   |       |
| SUM_VALUE    | double(22,6) | YES |    | NULL   |       |
| AVG_VALUE    | double(22,6) | YES |    | NULL   |       |
| MIN_VALUE    | double(22,6) | YES |    | NULL   |       |
| MAX_VALUE    | double(22,6) | YES |    | NULL   |       |
| COMMENT      | varchar(256) | YES |    | NULL   |       |
+--------------+--------------+------+------+---------+-------+
7 rows in set (0.00 sec)
```

Field description:

- `METRICS_NAME`: The monitoring table name.
- `QUANTILE`: The percentile. You can specify `QUANTILE` using SQL statements. For example:
  - `select * from metrics_summary where quantile=0.99` specifies viewing the data of the 0.99 percentile.
  - `select * from metrics_summary where quantile in (0.80, 0.90, 0.99, 0.999)` specifies viewing the data of the 0.8, 0.90, 0.99, 0.999 percentiles at the same time.

- `SUM_VALUE`, `AVG_VALUE`, `MIN_VALUE`, and `MAX_VALUE` respectively mean the sum, the average value, the minimum value, and the maximum value.
- `COMMENT`: The comment for the corresponding monitoring table.

For example:

To query the three groups of monitoring items with the highest average time consumption in the TiDB cluster within the time range of `'2020-03-08 13:23:00', '2020-03-08 13:33: 00'`, you can directly query the `information_schema.metrics_summary` table and use the `/*+ time_range()*/` hint to specify the time range. The SQL statement is as follows:

```
SELECT /*+ time_range('2020-03-08 13:23:00','2020-03-08 13:33:00') */ *
FROM information_schema.metrics_summary
WHERE metrics_name LIKE 'tidb%duration'
 AND avg_value > 0
 AND quantile = 0.99
ORDER BY avg_value DESC
LIMIT 3\G
```

```
***************************[ 1. row ]***************************
METRICS_NAME | tidb_get_token_duration
QUANTILE    | 0.99
SUM_VALUE   | 8.972509
AVG_VALUE   | 0.996945
MIN_VALUE   | 0.996515
MAX_VALUE   | 0.997458
COMMENT     |  The quantile of Duration (us) for getting token, it should be
    ↪ small until concurrency limit is reached(second)
***************************[ 2. row ]***************************
METRICS_NAME | tidb_query_duration
QUANTILE    | 0.99
SUM_VALUE   | 0.269079
AVG_VALUE   | 0.007272
MIN_VALUE   | 0.000667
MAX_VALUE   | 0.01554
COMMENT     | The quantile of TiDB query durations(second)
***************************[ 3. row ]***************************
METRICS_NAME | tidb_kv_request_duration
QUANTILE    | 0.99
SUM_VALUE   | 0.170232
AVG_VALUE   | 0.004601
MIN_VALUE   | 0.000975
MAX_VALUE   | 0.013
COMMENT     | The quantile of kv requests durations by store
```

Similarly, the following example queries the `metrics_summary_by_label` monitoring summary table:

```
SELECT /*+ time_range('2020-03-08 13:23:00','2020-03-08 13:33:00') */ *
FROM information_schema.metrics_summary_by_label
WHERE metrics_name LIKE 'tidb%duration'
 AND avg_value > 0
 AND quantile = 0.99
ORDER BY avg_value DESC
LIMIT 10\G
```

```
***************************[ 1. row ]***************************
INSTANCE    | 172.16.5.40:10089
METRICS_NAME | tidb_get_token_duration
LABEL       |
QUANTILE    | 0.99
SUM_VALUE   | 8.972509
AVG_VALUE   | 0.996945
MIN_VALUE   | 0.996515
MAX_VALUE   | 0.997458
COMMENT     |  The quantile of Duration (us) for getting token, it should be
   ↪ small until concurrency limit is reached(second)
***************************[ 2. row ]***************************
INSTANCE    | 172.16.5.40:10089
METRICS_NAME | tidb_query_duration
LABEL       | Select
QUANTILE    | 0.99
SUM_VALUE   | 0.072083
AVG_VALUE   | 0.008009
MIN_VALUE   | 0.007905
MAX_VALUE   | 0.008241
COMMENT     | The quantile of TiDB query durations(second)
***************************[ 3. row ]***************************
INSTANCE    | 172.16.5.40:10089
METRICS_NAME | tidb_query_duration
LABEL       | Rollback
QUANTILE    | 0.99
SUM_VALUE   | 0.072083
AVG_VALUE   | 0.008009
MIN_VALUE   | 0.007905
MAX_VALUE   | 0.008241
COMMENT     | The quantile of TiDB query durations(second)
```

The second and third rows of the query results above indicate that the `Select` and `Rollback` statements on `tidb_query_duration` have a long average execution time.

In addition to the example above, you can use the monitoring summary table to quickly find the module with the largest change from the monitoring data by comparing the full link monitoring items of the two time periods, and quickly locate the bottleneck. The following example compares all monitoring items in two periods (where t1 is the baseline) and sorts these items according to the greatest difference:

- Period t1: `("2020-03-03 17:08:00", "2020-03-03 17:11:00")`
- Period t2: `("2020-03-03 17:18:00", "2020-03-03 17:21:00")`

The monitoring items of the two time periods are joined according to `METRICS_NAME` and sorted according to the difference value. `TIME_RANGE` is the hint that specifies the query time.

```
SELECT GREATEST(t1.avg_value,t2.avg_value)/LEAST(t1.avg_value,
        t2.avg_value) AS ratio,
        t1.metrics_name,
        t1.avg_value as t1_avg_value,
        t2.avg_value as t2_avg_value,
        t2.comment
FROM
    (SELECT /*+ time_range("2020-03-03 17:08:00", "2020-03-03 17:11:00")*/
       ↪ *
    FROM information_schema.metrics_summary ) t1
JOIN
    (SELECT /*+ time_range("2020-03-03 17:18:00", "2020-03-03 17:21:00")*/
       ↪ *
    FROM information_schema.metrics_summary ) t2
    ON t1.metrics_name = t2.metrics_name
ORDER BY ratio DESC LIMIT 10;
```

```
+--
   ↪ -------------+-------------------------------------------+---------------+-----
   ↪
| ratio        | metrics_name                              | t1_avg_value |
   ↪ t2_avg_value   | comment
   ↪
   ↪ |
+--
   ↪ -------------+-------------------------------------------+---------------+-----
   ↪
| 5865.59537065 | tidb_slow_query_cop_process_total_time |  0.016333 |
   ↪ 95.804724 | The total time of TiDB slow query statistics with slow
   ↪ query total cop process time(second) |
| 3648.74109023 | tidb_distsql_partial_scan_key_total_num | 10865.666667 |
   ↪ 39646004.4394 | The total num of distsql partial scan key numbers
   ↪                            |
```

```
|  267.002351165 | tidb_slow_query_cop_wait_total_time |    0.003333 |
   ↪       0.890008 | The total time of TiDB slow query statistics with
   ↪ slow query total cop wait time(second) |
|  192.43267836 | tikv_cop_total_response_total_size | 2515333.66667 |
   ↪ 484032394.445 |
   ↪
   ↪ |
|  192.43267836 | tikv_cop_total_response_size_per_seconds | 41922.227778 |
   ↪ 8067206.57408 |
   ↪
   ↪ |
|  152.780296296 | tidb_distsql_scan_key_total_num   |   5304.333333 |
   ↪ 810397.618317 | The total num of distsql scan numbers
   ↪                                                |
|  126.042290167 | tidb_distsql_execution_total_time |     0.421622 |
   ↪ 53.142143 | The total time of distsql execution(second)
   ↪                                          |
|  105.164020657 | tikv_cop_scan_details             |    134.450733 |
   ↪ 14139.379665 |
   ↪
   ↪ |
|  105.164020657 | tikv_cop_scan_details_total       |   8067.043981 |
   ↪ 848362.77991 |
   ↪
   ↪ |
|  101.635495394 | tikv_cop_scan_keys_num            |   1070.875   |
   ↪ 108838.91113 |
   ↪
   ↪ |
+--
   ↪ --------------+-------------------------------------------+----------------+-----
   ↪
```

From the query result above, you can get the following information:

- `tib_slow_query_cop_process_total_time` (the time consumption of `cop process` in TiDB slow queries) in the period t2 is 5,865 times higher than that in period t1.
- `tidb_distsql_partial_scan_key_total_num` (the number of keys to scan requested by TiDB's `distsql`) in period t2 is 3,648 times higher than that in period t1. During period t2, `tidb_slow_query_cop_wait_total_time` (the waiting time of Coprocessor requesting to queue up in the TiDB slow query) is 267 times higher than that in period t1.
- `tikv_cop_total_response_size` (the size of the TiKV Coprocessor request result) in period t2 is 192 times higher than that in period t1.

- `tikv_cop_scan_details` in period t2 (the scan requested by the TiKV Coprocessor) is 105 times higher than that in period t1.

From the result above, you can see that the Coprocessor requests in period t2 are much more than those in period t1. This causes TiKV Coprocessor to be overloaded, and the `cop task` has to wait. It might be that some large queries appear in period t2 that bring more load.

In fact, during the entire time period from t1 to t2, the `go-ycsb` pressure test is running. Then 20 `tpch` queries are running during period t2. So it is the `tpch` queries that cause many Coprocessor requests.

14.10.17.2.25 METRICS_TABLES

The `METRICS_TABLES` table provides the PromQL (Prometheus Query Language) definition for each of the views in the metrics_schema database.

```
USE information_schema;
DESC metrics_tables;
```

```
+------------+--------------+------+------+---------+-------+
| Field      | Type         | Null | Key  | Default | Extra |
+------------+--------------+------+------+---------+-------+
| TABLE_NAME | varchar(64)  | YES  |      | NULL    |       |
| PROMQL     | varchar(64)  | YES  |      | NULL    |       |
| LABELS     | varchar(64)  | YES  |      | NULL    |       |
| QUANTILE   | double       | YES  |      | NULL    |       |
| COMMENT    | varchar(256) | YES  |      | NULL    |       |
+------------+--------------+------+------+---------+-------+
```

Field description:

- `TABLE_NAME`: Corresponds to the table name in `metrics_schema`.
- `PROMQL`: The working principle of the monitoring table is to map SQL statements to `PromQL` and convert Prometheus results into SQL query results. This field is the expression template of `PromQL`. When you query the data of the monitoring table, the query conditions are used to rewrite the variables in this template to generate the final query expression.
- `LABELS`: The label for the monitoring item. Each label corresponds to a column in the monitoring table. If the SQL statement contains the filter of the corresponding column, the corresponding `PromQL` changes accordingly.
- `QUANTILE`: The percentile. For monitoring data of the histogram type, a default percentile is specified. If the value of this field is `0`, it means that the monitoring item corresponding to the monitoring table is not a histogram.
- `COMMENT`: The comment about the monitoring table.

```
SELECT * FROM metrics_tables LIMIT 5\G
```

```
*************************** 1. row ***************************
TABLE_NAME: abnormal_stores
    PROMQL: sum(pd_cluster_status{ type=~"store_disconnected_count|
        ↪ store_unhealth_count|store_low_space_count|store_down_count|
        ↪ store_offline_count|store_tombstone_count"})
    LABELS: instance,type
  QUANTILE: 0
   COMMENT:
*************************** 2. row ***************************
TABLE_NAME: etcd_disk_wal_fsync_rate
    PROMQL: delta(etcd_disk_wal_fsync_duration_seconds_count{
        ↪ $LABEL_CONDITIONS}[$RANGE_DURATION])
    LABELS: instance
  QUANTILE: 0
   COMMENT: The rate of writing WAL into the persistent storage
*************************** 3. row ***************************
TABLE_NAME: etcd_wal_fsync_duration
    PROMQL: histogram_quantile($QUANTILE, sum(rate(
        ↪ etcd_disk_wal_fsync_duration_seconds_bucket{$LABEL_CONDITIONS}[
        ↪ $RANGE_DURATION])) by (le,instance))
    LABELS: instance
  QUANTILE: 0.99
   COMMENT: The quantile time consumed of writing WAL into the persistent
       ↪ storage
*************************** 4. row ***************************
TABLE_NAME: etcd_wal_fsync_total_count
    PROMQL: sum(increase(etcd_disk_wal_fsync_duration_seconds_count{
        ↪ $LABEL_CONDITIONS}[$RANGE_DURATION])) by (instance)
    LABELS: instance
  QUANTILE: 0
   COMMENT: The total count of writing WAL into the persistent storage
*************************** 5. row ***************************
TABLE_NAME: etcd_wal_fsync_total_time
    PROMQL: sum(increase(etcd_disk_wal_fsync_duration_seconds_sum{
        ↪ $LABEL_CONDITIONS}[$RANGE_DURATION])) by (instance)
    LABELS: instance
  QUANTILE: 0
   COMMENT: The total time of writing WAL into the persistent storage
5 rows in set (0.00 sec)
```

14.10.17.2.26 PARTITIONS

The PARTITIONS table provides information about partitioned tables.

```
USE information_schema;
DESC partitions;
```

```
+-------------------------------+-------------+------+------+---------+-------+
   ↪
| Field                         | Type        | Null | Key | Default | Extra |
+-------------------------------+-------------+------+------+---------+-------+
   ↪
| TABLE_CATALOG                 | varchar(512) | YES |    | NULL    |       |
| TABLE_SCHEMA                  | varchar(64) | YES |     | NULL    |       |
| TABLE_NAME                    | varchar(64) | YES |     | NULL    |       |
| PARTITION_NAME                | varchar(64) | YES |     | NULL    |       |
| SUBPARTITION_NAME             | varchar(64) | YES |     | NULL    |       |
| PARTITION_ORDINAL_POSITION | bigint(21) | YES |     | NULL    |       |
| SUBPARTITION_ORDINAL_POSITION | bigint(21) | YES |  | NULL    |       |
| PARTITION_METHOD              | varchar(18) | YES |     | NULL    |       |
| SUBPARTITION_METHOD           | varchar(12) | YES |     | NULL    |       |
| PARTITION_EXPRESSION          | longblob    | YES |     | NULL    |       |
| SUBPARTITION_EXPRESSION       | longblob    | YES |     | NULL    |       |
| PARTITION_DESCRIPTION         | longblob    | YES |     | NULL    |       |
| TABLE_ROWS                    | bigint(21)  | YES |     | NULL    |       |
| AVG_ROW_LENGTH                | bigint(21)  | YES |     | NULL    |       |
| DATA_LENGTH                   | bigint(21)  | YES |     | NULL    |       |
| MAX_DATA_LENGTH               | bigint(21)  | YES |     | NULL    |       |
| INDEX_LENGTH                  | bigint(21)  | YES |     | NULL    |       |
| DATA_FREE                     | bigint(21)  | YES |     | NULL    |       |
| CREATE_TIME                   | datetime    | YES |     | NULL    |       |
| UPDATE_TIME                   | datetime    | YES |     | NULL    |       |
| CHECK_TIME                    | datetime    | YES |     | NULL    |       |
| CHECKSUM                      | bigint(21)  | YES |     | NULL    |       |
| PARTITION_COMMENT             | varchar(80) | YES |     | NULL    |       |
| NODEGROUP                     | varchar(12) | YES |     | NULL    |       |
| TABLESPACE_NAME               | varchar(64) | YES |     | NULL    |       |
+-------------------------------+-------------+------+------+---------+-------+
   ↪
25 rows in set (0.00 sec)
```

```
CREATE TABLE test.t1 (id INT NOT NULL PRIMARY KEY) PARTITION BY HASH (id)
   ↪ PARTITIONS 2;
SELECT * FROM partitions WHERE table_schema='test' AND table_name='t1'\G
```

```
*************************** 1. row ***************************
               TABLE_CATALOG: def
```

```
               TABLE_SCHEMA: test
                 TABLE_NAME: t1
              PARTITION_NAME: p0
           SUBPARTITION_NAME: NULL
   PARTITION_ORDINAL_POSITION: 1
SUBPARTITION_ORDINAL_POSITION: NULL
            PARTITION_METHOD: HASH
         SUBPARTITION_METHOD: NULL
        PARTITION_EXPRESSION: `id`
     SUBPARTITION_EXPRESSION: NULL
       PARTITION_DESCRIPTION:
                  TABLE_ROWS: 0
             AVG_ROW_LENGTH: 0
                DATA_LENGTH: 0
            MAX_DATA_LENGTH: 0
               INDEX_LENGTH: 0
                  DATA_FREE: 0
                CREATE_TIME: 2020-07-06 16:35:28
                UPDATE_TIME: NULL
                 CHECK_TIME: NULL
                   CHECKSUM: NULL
          PARTITION_COMMENT:
                  NODEGROUP: NULL
             TABLESPACE_NAME: NULL
*************************** 2. row ***************************
              TABLE_CATALOG: def
               TABLE_SCHEMA: test
                 TABLE_NAME: t1
              PARTITION_NAME: p1
           SUBPARTITION_NAME: NULL
   PARTITION_ORDINAL_POSITION: 2
SUBPARTITION_ORDINAL_POSITION: NULL
            PARTITION_METHOD: HASH
         SUBPARTITION_METHOD: NULL
        PARTITION_EXPRESSION: `id`
     SUBPARTITION_EXPRESSION: NULL
       PARTITION_DESCRIPTION:
                  TABLE_ROWS: 0
             AVG_ROW_LENGTH: 0
                DATA_LENGTH: 0
            MAX_DATA_LENGTH: 0
               INDEX_LENGTH: 0
                  DATA_FREE: 0
                CREATE_TIME: 2020-07-06 16:35:28
                UPDATE_TIME: NULL
```

```
                CHECK_TIME: NULL
                  CHECKSUM: NULL
           PARTITION_COMMENT:
                 NODEGROUP: NULL
            TABLESPACE_NAME: NULL
2 rows in set (0.00 sec)
```

14.10.17.2.27 PLACEMENT_POLICIES

The PLACEMENT_POLICIES table provides information on all placement policies. For details, refer to Placement Rules in SQL.

```
USE information_schema;
DESC placement_policies;
```

```
+----------------------+---------------+------+-----+---------+-------+
| Field                | Type          | Null | Key | Default | Extra |
+----------------------+---------------+------+-----+---------+-------+
| POLICY_ID            | bigint(64)    | NO   |     | <null> |       |
| CATALOG_NAME         | varchar(512)  | NO   |     | <null> |       |
| POLICY_NAME          | varchar(64)   | NO   |     | <null> |       |
| PRIMARY_REGION       | varchar(1024) | YES  |     | <null> |       |
| REGIONS              | varchar(1024) | YES  |     | <null> |       |
| CONSTRAINTS          | varchar(1024) | YES  |     | <null> |       |
| LEADER_CONSTRAINTS   | varchar(1024) | YES  |     | <null> |       |
| FOLLOWER_CONSTRAINTS | varchar(1024) | YES  |     | <null> |       |
| LEARNER_CONSTRAINTS  | varchar(1024) | YES  |     | <null> |       |
| SCHEDULE             | varchar(20)   | YES  |     | <null> |       |
| FOLLOWERS            | bigint(64)    | YES  |     | <null> |       |
| LEARNERS             | bigint(64)    | YES  |     | <null> |       |
+----------------------+---------------+------+-----+---------+-------+
12 rows in set (0.00 sec)
```

Examples

The PLACEMENT_POLICIES table only shows all placement policies. To view the canonical version of placement rules (including all placement policies and objects assigned placement policies), use the statement SHOW PLACEMENT instead:

```
CREATE TABLE t1 (a INT);
CREATE PLACEMENT POLICY p1 primary_region="us-east-1" regions="us-east-1";
CREATE TABLE t3 (a INT) PLACEMENT POLICY=p1;
SHOW PLACEMENT; -- Shows all information, including table t3.
SELECT * FROM information_schema.placement_policies; -- Only shows
    ↪ placement policies, excluding t3.
```

```
Query OK, 0 rows affected (0.09 sec)

Query OK, 0 rows affected (0.11 sec)

Query OK, 0 rows affected (0.08 sec)

+--
   ↪ -------------+-----------------------------------------------+-----------------
   ↪
| Target      | Placement                                   | Scheduling_State
   ↪ |
+--
   ↪ -------------+-----------------------------------------------+-----------------
   ↪
| POLICY p1   | PRIMARY_REGION="us-east-1" REGIONS="us-east-1" | NULL     |
| TABLE test.t3 | PRIMARY_REGION="us-east-1" REGIONS="us-east-1" | PENDING
   ↪ |
+--
   ↪ -------------+-----------------------------------------------+-----------------
   ↪
2 rows in set (0.00 sec)

+--
   ↪ ---------+--------------+-------------+----------------+-----------+------------
   ↪
| POLICY_ID | CATALOG_NAME | POLICY_NAME | PRIMARY_REGION | REGIONS |
   ↪ CONSTRAINTS | LEADER_CONSTRAINTS | FOLLOWER_CONSTRAINTS |
   ↪ LEARNER_CONSTRAINTS | SCHEDULE | FOLLOWERS | LEARNERS |
+--
   ↪ ---------+--------------+-------------+----------------+-----------+------------
   ↪
| 1        | def         | p1         | us-east-1    | us-east-1 |          |
   ↪                  |                    |                  |         | 2
   ↪        | 0       |
+--
   ↪ ---------+--------------+-------------+----------------+-----------+------------
   ↪
1 rows in set (0.00 sec)
```

14.10.17.2.28 PROCESSLIST

`PROCESSLIST`, just like `SHOW PROCESSLIST`, is used to view the requests that are being handled.

The `PROCESSLIST` table has additional columns not present in `SHOW PROCESSLIST`:

- A `DIGEST` column to show the digest of the SQL statement.
- A `MEM` column to show the memory used by the request that is being processed, in bytes.
- A `DISK` column to show the disk usage in bytes.
- A `TxnStart` column to show the start time of the transaction

```
USE information_schema;
DESC processlist;
```

```
+----------+---------------------+------+------+---------+-------+
| Field    | Type                | Null | Key | Default | Extra |
+----------+---------------------+------+------+---------+-------+
| ID       | bigint(21) unsigned | NO |    | 0       |       |
| USER     | varchar(16)         | NO   |     |         |       |
| HOST     | varchar(64)         | NO   |     |         |       |
| DB       | varchar(64)         | YES  |     | NULL    |       |
| COMMAND | varchar(16)          | NO   |     |         |       |
| TIME     | int(7)              | NO   |     | 0       |       |
| STATE    | varchar(7)          | YES  |     | NULL    |       |
| INFO     | longtext            | YES  |     | NULL    |       |
| DIGEST   | varchar(64)         | YES  |     |         |       |
| MEM      | bigint(21) unsigned | YES  |   | NULL    |       |
| DISK     | bigint(21) unsigned | YES  |   | NULL    |       |
| TxnStart | varchar(64)         | NO   |     |         |       |
+----------+---------------------+------+------+---------+-------+
12 rows in set (0.00 sec)
```

```
SELECT * FROM processlist\G
```

```
*************************** 1. row ***************************
     ID: 16
   USER: root
   HOST: 127.0.0.1
     DB: information_schema
 COMMAND: Query
   TIME: 0
  STATE: autocommit
   INFO: SELECT * FROM processlist
    MEM: 0
TxnStart:
1 row in set (0.00 sec)
```

Fields in the PROCESSLIST table are described as follows:

- ID: The ID of the user connection.
- USER: The name of the user who is executing PROCESS.
- HOST: The address that the user is connecting to.
- DB: The name of the currently connected default database.
- COMMAND: The command type that PROCESS is executing.
- TIME: The current execution duration of PROCESS, in seconds.
- STATE: The current connection state.
- INFO: The requested statement that is being processed.
- DIGEST: The digest of the SQL statement.
- MEM: The memory used by the request that is being processed, in bytes.
- DISK: The disk usage in bytes.
- TxnStart: The start time of the transaction.

CLUSTER_PROCESSLIST

CLUSTER_PROCESSLIST is the cluster system table corresponding to PROCESSLIST. It is used to query the PROCESSLIST information of all TiDB nodes in the cluster. The table schema of CLUSTER_PROCESSLIST has one more column than PROCESSLIST, the INSTANCE column, which stores the address of the TiDB node this row of data is from.

```
SELECT * FROM information_schema.cluster_processlist;
```

```
+--
   ↪ --------------+-----+------+----------+------+---------+------+------------+---
   ↪
| INSTANCE      | ID  | USER | HOST     | DB   | COMMAND | TIME | STATE    | INFO
   ↪                                              | MEM | TxnStart
   ↪                           |
+--
   ↪ --------------+-----+------+----------+------+---------+------+------------+---
   ↪
| 10.0.1.22:10080 | 150 | u1 | 10.0.1.1 | test | Query | 0 | autocommit |
   ↪ select count(*) from usertable              | 372 | 05-28
   ↪ 03:54:21.230(416976223923077223) |
| 10.0.1.22:10080 | 138 | root | 10.0.1.1 | test | Query | 0 | autocommit |
   ↪ SELECT * FROM information_schema.cluster_processlist | 0 | 05-28
   ↪ 03:54:21.230(416976223923077220) |
| 10.0.1.22:10080 | 151 | u1 | 10.0.1.1 | test | Query | 0 | autocommit |
   ↪ select count(*) from usertable              | 372 | 05-28
   ↪ 03:54:21.230(416976223923077224) |
| 10.0.1.21:10080 | 15 | u2 | 10.0.1.1 | test | Query | 0 | autocommit |
   ↪ select max(field0) from usertable           | 496 | 05-28
   ↪ 03:54:21.230(416976223923077222) |
```

```
| 10.0.1.21:10080 | 14 | u2 | 10.0.1.1 | test | Query | 0 | autocommit |
    ↪ select max(field0) from usertable           | 496 | 05-28
    ↪ 03:54:21.230(416976223923077225) |
+--
    ↪ --------------+-----+------+---------+------+--------+------+-----------+---
    ↪
```

14.10.17.2.29 REFERENTIAL_CONSTRAINTS

The `REFERENTIAL_CONSTRAINTS` table provides information about `FOREIGN KEY` relationships between tables. Note that TiDB currently does not enforce `FOREIGN KEY` constraints, or perform actions such as `ON DELETE CASCADE`.

```
USE information_schema;
DESC referential_constraints;
```

```
+--
    ↪ ------------------------+-------------+------+------+---------+-------+
    ↪
| Field                  | Type        | Null | Key | Default | Extra |
+--
    ↪ ------------------------+-------------+------+------+---------+-------+
    ↪
| CONSTRAINT_CATALOG     | varchar(512) | NO |     | NULL   |       |
| CONSTRAINT_SCHEMA      | varchar(64) | NO |      | NULL   |       |
| CONSTRAINT_NAME        | varchar(64) | NO |      | NULL   |       |
| UNIQUE_CONSTRAINT_CATALOG | varchar(512) | NO | | NULL   |       |
| UNIQUE_CONSTRAINT_SCHEMA | varchar(64) | NO |   | NULL   |       |
| UNIQUE_CONSTRAINT_NAME | varchar(64) | YES |    | NULL   |       |
| MATCH_OPTION           | varchar(64) | NO |      | NULL   |       |
| UPDATE_RULE            | varchar(64) | NO |      | NULL   |       |
| DELETE_RULE            | varchar(64) | NO |      | NULL   |       |
| TABLE_NAME             | varchar(64) | NO |      | NULL   |       |
| REFERENCED_TABLE_NAME  | varchar(64) | NO |      | NULL   |       |
+--
    ↪ ------------------------+-------------+------+------+---------+-------+
    ↪
11 rows in set (0.00 sec)
```

```
CREATE TABLE test.parent (
 id INT NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (id)
);

CREATE TABLE test.child (
```

```
 id INT NOT NULL AUTO_INCREMENT,
 name varchar(255) NOT NULL,
 parent_id INT DEFAULT NULL,
 PRIMARY KEY (id),
 CONSTRAINT fk_parent FOREIGN KEY (parent_id) REFERENCES parent (id) ON
    ↪ UPDATE CASCADE ON DELETE RESTRICT
);

SELECT * FROM referential_constraints\G
```

```
*************************** 1. row ***************************
       CONSTRAINT_CATALOG: def
        CONSTRAINT_SCHEMA: test
          CONSTRAINT_NAME: fk_parent
UNIQUE_CONSTRAINT_CATALOG: def
 UNIQUE_CONSTRAINT_SCHEMA: test
   UNIQUE_CONSTRAINT_NAME: PRIMARY
             MATCH_OPTION: NONE
              UPDATE_RULE: CASCADE
              DELETE_RULE: RESTRICT
               TABLE_NAME: child
    REFERENCED_TABLE_NAME: parent
1 row in set (0.00 sec)
```

14.10.17.2.30 SCHEMATA

The SCHEMATA table provides information about databases. The table data is equivalent to the result of the SHOW DATABASES statement.

```
USE information_schema;
desc SCHEMATA;
```

```
+----------------------------+-------------+------+------+---------+-------+
   ↪
| Field                      | Type        | Null | Key | Default | Extra |
+----------------------------+-------------+------+------+---------+-------+
   ↪
| CATALOG_NAME               | varchar(512) | YES |    | NULL   |       |
| SCHEMA_NAME                | varchar(64) | YES |     | NULL   |       |
| DEFAULT_CHARACTER_SET_NAME | varchar(64) | YES | | NULL   |       |
| DEFAULT_COLLATION_NAME | varchar(32) | YES |     | NULL   |       |
| SQL_PATH                   | varchar(512) | YES |    | NULL   |       |
+----------------------------+-------------+------+------+---------+-------+
   ↪
5 rows in set (0.00 sec)
```

```
SELECT * FROM SCHEMATA;
```

```
+--------------+--------------------+----------------------------+------------------------
   ↪
| CATALOG_NAME | SCHEMA_NAME        | DEFAULT_CHARACTER_SET_NAME |
   ↪ DEFAULT_COLLATION_NAME | SQL_PATH |
+--------------+--------------------+----------------------------+------------------------
   ↪
| def          | INFORMATION_SCHEMA | utf8mb4                    | utf8mb4_bin
   ↪           | NULL     |
| def          | METRICS_SCHEMA     | utf8mb4                    | utf8mb4_bin
   ↪           | NULL     |
| def          | mysql              | utf8mb4                    | utf8mb4_bin
   ↪           | NULL     |
| def          | PERFORMANCE_SCHEMA | utf8mb4                    | utf8mb4_bin
   ↪           | NULL     |
| def          | test               | utf8mb4                    | utf8mb4_bin
   ↪           | NULL     |
+--------------+--------------------+----------------------------+------------------------
   ↪
5 rows in set (0.00 sec)
```

Fields in the `SCHEMATA` table are described as follows:

- `CATALOG_NAME`: The catalog to which the database belongs.
- `SCHEMA_NAME`: The database name.
- `DEFAULT_CHARACTER_SET_NAME`: The default character set of the database.
- `DEFAULT_COLLATION_NAME`: The default collation of the database.
- `SQL_PATH`: The value of this item is always `NULL`.

14.10.17.2.31 SEQUENCES

The `SEQUENCES` table provides information about sequences. The sequences feature is modeled on a similar feature in MariaDB.

```
USE information_schema;
DESC sequences;
```

```
+-----------------+-------------+------+------+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+------+---------+-------+
| TABLE_CATALOG | varchar(512) | NO |     | NULL   |       |
| SEQUENCE_SCHEMA | varchar(64) | NO |    | NULL   |       |
| SEQUENCE_NAME | varchar(64) | NO |     | NULL   |       |
```

```
| CACHE          | tinyint(4)  | NO   |     | NULL    |       |
| CACHE_VALUE    | bigint(21)  | YES  |     | NULL    |       |
| CYCLE          | tinyint(4)  | NO   |     | NULL    |       |
| INCREMENT      | bigint(21)  | NO   |     | NULL    |       |
| MAX_VALUE      | bigint(21)  | YES  |     | NULL    |       |
| MIN_VALUE      | bigint(21)  | YES  |     | NULL    |       |
| START          | bigint(21)  | YES  |     | NULL    |       |
| COMMENT        | varchar(64) | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
11 rows in set (0.00 sec)
```

```
CREATE SEQUENCE test.seq;
SELECT nextval(test.seq);
SELECT * FROM sequences\G
```

```
+-------------------+
| nextval(test.seq) |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.01 sec)

*************************** 1. row ***************************
  TABLE_CATALOG: def
SEQUENCE_SCHEMA: test
  SEQUENCE_NAME: seq
          CACHE: 1
    CACHE_VALUE: 1000
          CYCLE: 0
      INCREMENT: 1
      MAX_VALUE: 9223372036854775806
      MIN_VALUE: 1
          START: 1
        COMMENT:
1 row in set (0.00 sec)
```

14.10.17.2.32 SESSION_VARIABLES

The `SESSION_VARIABLES` table provides information about session variables. The table data is similar to the result of the `SHOW SESSION VARIABLES` statement.

```
USE information_schema;
DESC session_variables;
```

```
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| VARIABLE_NAME  | varchar(64)   | YES  |     | NULL    |       |
| VARIABLE_VALUE | varchar(1024) | YES  |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
SELECT * FROM session_variables ORDER BY variable_name LIMIT 10;
```

```
+-----------------------------------+-----------------+
| VARIABLE_NAME                     | VARIABLE_VALUE  |
+-----------------------------------+-----------------+
| allow_auto_random_explicit_insert | off             |
| auto_increment_increment          | 1               |
| auto_increment_offset             | 1               |
| autocommit                        | 1               |
| automatic_sp_privileges           | 1               |
| avoid_temporal_upgrade            | 0               |
| back_log                          | 80              |
| basedir                           | /usr/local/mysql |
| big_tables                        | 0               |
| bind_address                      | *               |
+-----------------------------------+-----------------+
10 rows in set (0.00 sec)
```

The description of columns in the `SESSION_VARIABLES` table is as follows:

- `VARIABLE_NAME`: The name of the session-level variable in the database.
- `VARIABLE_VALUE`: The value of the session-level variable in the database.

14.10.17.2.33 SLOW_QUERY

The `SLOW_QUERY` table provides the slow query information of the current node, which is the parsing result of the TiDB slow log file. The column names in the table are corresponding to the field names in the slow log.

For how to use this table to identify problematic statements and improve query performance, see Slow Query Log Document.

```
USE information_schema;
DESC slow_query;
```

```
+--------------------------+---------------------+------+------+---------+-------+
    ↪
```

```
| Field                    | Type               | Null | Key | Default | Extra |
+--------------------------+--------------------+------+-----+---------+-------+
   ↪
| Time                     | timestamp(6)       | YES |     | NULL    |       |
| Txn_start_ts             | bigint(20) unsigned | YES |    | NULL    |       |
| User                     | varchar(64)        | YES |     | NULL    |       |
| Host                     | varchar(64)        | YES |     | NULL    |       |
| Conn_ID                  | bigint(20) unsigned | YES |    | NULL    |       |
| Query_time               | double             | YES |     | NULL    |       |
| Parse_time               | double             | YES |     | NULL    |       |
| Compile_time             | double             | YES |     | NULL    |       |
| Rewrite_time             | double             | YES |     | NULL    |       |
| Preproc_subqueries       | bigint(20) unsigned | YES |    | NULL    |       |
| Preproc_subqueries_time | double             | YES |     | NULL    |       |
| Optimize_time            | double             | YES |     | NULL    |       |
| Wait_TS                  | double             | YES |     | NULL    |       |
| Prewrite_time            | double             | YES |     | NULL    |       |
| Wait_prewrite_binlog_time | double            | YES |     | NULL    |       |
| Commit_time              | double             | YES |     | NULL    |       |
| Get_commit_ts_time       | double             | YES |     | NULL    |       |
| Commit_backoff_time      | double             | YES |     | NULL    |       |
| Backoff_types            | varchar(64)        | YES |     | NULL    |       |
| Resolve_lock_time        | double             | YES |     | NULL    |       |
| Local_latch_wait_time    | double             | YES |     | NULL    |       |
| Write_keys               | bigint(22)         | YES |     | NULL    |       |
| Write_size               | bigint(22)         | YES |     | NULL    |       |
| Prewrite_region          | bigint(22)         | YES |     | NULL    |       |
| Txn_retry                | bigint(22)         | YES |     | NULL    |       |
| Cop_time                 | double             | YES |     | NULL    |       |
| Process_time             | double             | YES |     | NULL    |       |
| Wait_time                | double             | YES |     | NULL    |       |
| Backoff_time             | double             | YES |     | NULL    |       |
| LockKeys_time            | double             | YES |     | NULL    |       |
| Request_count            | bigint(20) unsigned | YES |    | NULL    |       |
| Total_keys               | bigint(20) unsigned | YES |    | NULL    |       |
| Process_keys             | bigint(20) unsigned | YES |    | NULL    |       |
| DB                       | varchar(64)        | YES |     | NULL    |       |
| Index_names              | varchar(100)       | YES |     | NULL    |       |
| Is_internal              | tinyint(1)         | YES |     | NULL    |       |
| Digest                   | varchar(64)        | YES |     | NULL    |       |
| Stats                    | varchar(512)       | YES |     | NULL    |       |
| Cop_proc_avg             | double             | YES |     | NULL    |       |
| Cop_proc_p90             | double             | YES |     | NULL    |       |
| Cop_proc_max             | double             | YES |     | NULL    |       |
| Cop_proc_addr            | varchar(64)        | YES |     | NULL    |       |
```

```
| Cop_wait_avg           | double            | YES |     | NULL   |       |
| Cop_wait_p90           | double            | YES |     | NULL   |       |
| Cop_wait_max           | double            | YES |     | NULL   |       |
| Cop_wait_addr          | varchar(64)       | YES |     | NULL   |       |
| Mem_max                | bigint(20)        | YES |     | NULL   |       |
| Disk_max               | bigint(20)        | YES |     | NULL   |       |
| Succ                   | tinyint(1)        | YES |     | NULL   |       |
| Plan_from_cache        | tinyint(1)        | YES |     | NULL   |       |
| Plan                   | longblob          | YES |     | NULL   |       |
| Plan_digest            | varchar(128)      | YES |     | NULL   |       |
| Prev_stmt              | longblob          | YES |     | NULL   |       |
| Query                  | longblob          | YES |     | NULL   |       |
+------------------------+-------------------+-----+-----+--------+-------+
    ↪
54 rows in set (0.00 sec)
```

CLUSTER_SLOW_QUERY table

The `CLUSTER_SLOW_QUERY` table provides the slow query information of all nodes in the cluster, which is the parsing result of the TiDB slow log files. You can use the `CLUSTER_SLOW_QUERY` table the way you do with `SLOW_QUERY`. The table schema of the `CLUSTER_SLOW_QUERY` table differs from that of the `SLOW_QUERY` table in that an `INSTANCE` column is added to `CLUSTER_SLOW_QUERY`. The `INSTANCE` column represents the TiDB node address of the row information on the slow query.

For how to use this table to identify problematic statements and improve query performance, see Slow Query Log Document.

```
desc cluster_slow_query;
```

```
+--
    ↪ ------------------------+--------------------+------+------+---------+-------+
    ↪
| Field                  | Type               | Null | Key | Default | Extra |
+--
    ↪ ------------------------+--------------------+------+------+---------+-------+
    ↪
| INSTANCE               | varchar(64)        | YES |     | NULL   |       |
| Time                   | timestamp(6)       | YES |     | NULL   |       |
| Txn_start_ts           | bigint(20) unsigned | YES |    | NULL   |       |
| User                   | varchar(64)        | YES |     | NULL   |       |
| Host                   | varchar(64)        | YES |     | NULL   |       |
| Conn_ID                | bigint(20) unsigned | YES |    | NULL   |       |
| Query_time             | double             | YES |     | NULL   |       |
| Parse_time             | double             | YES |     | NULL   |       |
| Compile_time           | double             | YES |     | NULL   |       |
```

```
| Rewrite_time           | double              | YES |     | NULL |       |
| Preproc_subqueries     | bigint(20) unsigned | YES |     | NULL |       |
| Preproc_subqueries_time | double             | YES |     | NULL |       |
| Optimize_time          | double              | YES |     | NULL |       |
| Wait_TS                | double              | YES |     | NULL |       |
| Prewrite_time          | double              | YES |     | NULL |       |
| Wait_prewrite_binlog_time | double           | YES |     | NULL |       |
| Commit_time            | double              | YES |     | NULL |       |
| Get_commit_ts_time     | double              | YES |     | NULL |       |
| Commit_backoff_time    | double              | YES |     | NULL |       |
| Backoff_types          | varchar(64)         | YES |     | NULL |       |
| Resolve_lock_time      | double              | YES |     | NULL |       |
| Local_latch_wait_time  | double              | YES |     | NULL |       |
| Write_keys             | bigint(22)          | YES |     | NULL |       |
| Write_size             | bigint(22)          | YES |     | NULL |       |
| Prewrite_region        | bigint(22)          | YES |     | NULL |       |
| Txn_retry              | bigint(22)          | YES |     | NULL |       |
| Cop_time               | double              | YES |     | NULL |       |
| Process_time           | double              | YES |     | NULL |       |
| Wait_time              | double              | YES |     | NULL |       |
| Backoff_time           | double              | YES |     | NULL |       |
| LockKeys_time          | double              | YES |     | NULL |       |
| Request_count          | bigint(20) unsigned | YES |     | NULL |       |
| Total_keys             | bigint(20) unsigned | YES |     | NULL |       |
| Process_keys           | bigint(20) unsigned | YES |     | NULL |       |
| DB                     | varchar(64)         | YES |     | NULL |       |
| Index_names            | varchar(100)        | YES |     | NULL |       |
| Is_internal            | tinyint(1)          | YES |     | NULL |       |
| Digest                 | varchar(64)         | YES |     | NULL |       |
| Stats                  | varchar(512)        | YES |     | NULL |       |
| Cop_proc_avg           | double              | YES |     | NULL |       |
| Cop_proc_p90           | double              | YES |     | NULL |       |
| Cop_proc_max           | double              | YES |     | NULL |       |
| Cop_proc_addr          | varchar(64)         | YES |     | NULL |       |
| Cop_wait_avg           | double              | YES |     | NULL |       |
| Cop_wait_p90           | double              | YES |     | NULL |       |
| Cop_wait_max           | double              | YES |     | NULL |       |
| Cop_wait_addr          | varchar(64)         | YES |     | NULL |       |
| Mem_max                | bigint(20)          | YES |     | NULL |       |
| Disk_max               | bigint(20)          | YES |     | NULL |       |
| Succ                   | tinyint(1)          | YES |     | NULL |       |
| Plan_from_cache        | tinyint(1)          | YES |     | NULL |       |
| Plan                   | longblob            | YES |     | NULL |       |
| Plan_digest            | varchar(128)        | YES |     | NULL |       |
| Prev_stmt              | longblob            | YES |     | NULL |       |
```

```
| Query                    | longblob          | YES |      | NULL   |       |
+--
   ↪ -------------------------+---------------------+------+------+---------+-------+
   ↪
55 rows in set (0.00 sec)
```

When the cluster system table is queried, TiDB does not obtain data from all nodes, but pushes down the related calculation to other nodes. The execution plan is as follows:

```
desc SELECT count(*) FROM cluster_slow_query WHERE user = 'u1';
```

```
+-------------------------+----------+-----------+-------------------------+----------
   ↪
| id                      | estRows | task     | access object          |
   ↪ operator info                                    |
+-------------------------+----------+-----------+-------------------------+----------
   ↪
| StreamAgg_20            | 1.00     | root     |                        | funcs:
   ↪ count(Column#53)->Column#51                |
| ─TableReader_21         | 1.00     | root     |                        | data:
   ↪ StreamAgg_9                                |
|   ─StreamAgg_9          | 1.00     | cop[tidb] |                        | funcs
   ↪ :count(1)->Column#53                       |
|     ─Selection_19       | 10.00    | cop[tidb] |                        | eq(
   ↪ information_schema.cluster_slow_query.user, "u1") |
|       ─TableFullScan_18 | 10000.00 | cop[tidb] | table:CLUSTER_SLOW_QUERY
   ↪  | keep order:false, stats:pseudo        |
+-------------------------+----------+-----------+-------------------------+----------
   ↪
```

In the above execution plan, the `user = u1` condition is pushed down to other (`cop`) TiDB nodes, and the aggregate operator is also pushed down (the `StreamAgg` operator in the graph).

Currently, because statistics of the system tables are not collected, sometimes some aggregation operators cannot be pushed down, which results in slow execution. In this case, you can manually specify the SQL HINT to push down the aggregation operators. For example:

```
SELECT /*+ AGG_TO_COP() */ count(*) FROM cluster_slow_query GROUP BY user;
```

14.10.17.2.34 STATISTICS

The `STATISTICS` table provides information about table indexes.

```
USE information_schema;
DESC statistics;
```

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| TABLE_CATALOG | varchar(512) | YES |  | NULL   |       |
| TABLE_SCHEMA | varchar(64) | YES |    | NULL   |       |
| TABLE_NAME   | varchar(64) | YES |     | NULL   |       |
| NON_UNIQUE   | varchar(1)  | YES |     | NULL   |       |
| INDEX_SCHEMA | varchar(64) | YES |    | NULL   |       |
| INDEX_NAME   | varchar(64) | YES |     | NULL   |       |
| SEQ_IN_INDEX | bigint(2)   | YES |     | NULL   |       |
| COLUMN_NAME | varchar(21) | YES |     | NULL   |       |
| COLLATION    | varchar(1)  | YES |     | NULL   |       |
| CARDINALITY | bigint(21)  | YES |     | NULL   |       |
| SUB_PART     | bigint(3)   | YES |     | NULL   |       |
| PACKED       | varchar(10) | YES |     | NULL   |       |
| NULLABLE     | varchar(3)  | YES |     | NULL   |       |
| INDEX_TYPE   | varchar(16) | YES |     | NULL   |       |
| COMMENT      | varchar(16) | YES |     | NULL   |       |
| INDEX_COMMENT | varchar(1024) | YES | | NULL   |       |
| IS_VISIBLE   | varchar(3)  | YES |     | NULL   |       |
| Expression   | varchar(64) | YES |     | NULL   |       |
+--------------+--------------+------+-----+---------+-------+
18 rows in set (0.00 sec)
```

Fields in the `STATISTICS` table are described as follows:

- `TABLE_CATALOG`: The name of the catalog to which the table containing the index belongs. This value is always `def`.
- `TABLE_SCHEMA`: The name of the database to which the table containing the index belongs.
- `TABLE_NAME`: The name of the table containing the index.
- `NON_UNIQUE`: If the index must not contain duplicate values, the value is `0`; if duplicate values are allowed in the index, the value is `1`.
- `INDEX_SCHEMA`: The name of the database to which the index belongs.
- `INDEX_NAME`: The name of the index. If the index is the primary key, then the value is always `PRIMARY`.
- `SEQ_IN_INDEX`: The column number in the index, starting from `1`.
- `COLUMN_NAME`: The column name. See the description of the `Expression` column.
- `COLLATION`: The sorting method of the columns in the index. The value can be `A` (ascending order), `D` (descending order) or `NULL` (unsorted).
- `CARDINALITY`: TiDB does not use this field. The field value is always `0`.
- `SUB_PART`: The prefix of the index. If only part of the prefix of the column is indexed, the value is the number of indexed characters; if the entire column is indexed, the value is `NULL`.

- `PACKED`: TiDB does not use this field. This value is always `NULL`.
- `NULLABLE`: If the column might contain a `NULL` value, the value is `YES`; if not, the value is `''`.
- `INDEX_TYPE`: The type of the index.
- `COMMENT`: Other information related to the index.
- `INDEX_COMMENT`: Any comment with comment attribute provided for the index when creating the index.
- `IS_VISIBLE`: Whether the optimizer can use this index.
- `Expression` For the index key of the non-expression part, this value is `NULL`; for the index key of the expression part, this value is the expression itself. Refer to Expression Index.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name'

SHOW INDEX
  FROM tbl_name
  FROM db_name
```

14.10.17.2.35 TABLES

The `TABLES` table provides information about tables in databases:

```
USE information_schema;
DESC tables;
```

```
+--
    ↪ ------------------------+--------------+------+------+---------+-------+
    ↪
| Field                  | Type         | Null | Key | Default | Extra |
+--
    ↪ ------------------------+--------------+------+------+---------+-------+
    ↪
| TABLE_CATALOG          | varchar(512) | YES |     | NULL    |       |
| TABLE_SCHEMA           | varchar(64)  | YES |     | NULL    |       |
| TABLE_NAME             | varchar(64)  | YES |     | NULL    |       |
| TABLE_TYPE             | varchar(64)  | YES |     | NULL    |       |
| ENGINE                 | varchar(64)  | YES |     | NULL    |       |
| VERSION                | bigint(21)   | YES |     | NULL    |       |
| ROW_FORMAT             | varchar(10)  | YES |     | NULL    |       |
| TABLE_ROWS             | bigint(21)   | YES |     | NULL    |       |
| AVG_ROW_LENGTH         | bigint(21)   | YES |     | NULL    |       |
```

```
| DATA_LENGTH            | bigint(21)  | YES |     | NULL    |       |
| MAX_DATA_LENGTH        | bigint(21)  | YES |     | NULL    |       |
| INDEX_LENGTH           | bigint(21)  | YES |     | NULL    |       |
| DATA_FREE              | bigint(21)  | YES |     | NULL    |       |
| AUTO_INCREMENT         | bigint(21)  | YES |     | NULL    |       |
| CREATE_TIME            | datetime    | YES |     | NULL    |       |
| UPDATE_TIME            | datetime    | YES |     | NULL    |       |
| CHECK_TIME             | datetime    | YES |     | NULL    |       |
| TABLE_COLLATION        | varchar(32) | NO  |     | utf8_bin |      |
| CHECKSUM               | bigint(21)  | YES |     | NULL    |       |
| CREATE_OPTIONS         | varchar(255) | YES |    | NULL    |       |
| TABLE_COMMENT          | varchar(2048) | YES |   | NULL    |       |
| TIDB_TABLE_ID          | bigint(21)  | YES |     | NULL    |       |
| TIDB_ROW_ID_SHARDING_INFO | varchar(255) | YES | | NULL    |       |
+--
    ↪ ------------------------+--------------+------+------+---------+-------+
    ↪
23 rows in set (0.00 sec)
```

```
SELECT * FROM tables WHERE table_schema='mysql' AND table_name='user'\G
```

```
*************************** 1. row ***************************
           TABLE_CATALOG: def
            TABLE_SCHEMA: mysql
              TABLE_NAME: user
              TABLE_TYPE: BASE TABLE
                  ENGINE: InnoDB
                 VERSION: 10
              ROW_FORMAT: Compact
              TABLE_ROWS: 0
          AVG_ROW_LENGTH: 0
             DATA_LENGTH: 0
         MAX_DATA_LENGTH: 0
            INDEX_LENGTH: 0
               DATA_FREE: 0
          AUTO_INCREMENT: NULL
             CREATE_TIME: 2020-07-05 09:25:51
             UPDATE_TIME: NULL
              CHECK_TIME: NULL
         TABLE_COLLATION: utf8mb4_bin
                CHECKSUM: NULL
          CREATE_OPTIONS:
           TABLE_COMMENT:
           TIDB_TABLE_ID: 5
TIDB_ROW_ID_SHARDING_INFO: NULL
```

```
1 row in set (0.00 sec)
```

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']

SHOW TABLES
  FROM db_name
  [LIKE 'wild']
```

The description of columns in the `TABLES` table is as follows:

- `TABLE_CATALOG`: The name of the catalog which the table belongs to. The value is always `def`.
- `TABLE_SCHEMA`: The name of the schema which the table belongs to.
- `TABLE_NAME`: The name of the table.
- `TABLE_TYPE`: The type of the table.
- `ENGINE`: The type of the storage engine. The value is currently `InnoDB`.
- `VERSION`: Version. The value is `10` by default.
- `ROW_FORMAT`: The row format. The value is currently `Compact`.
- `TABLE_ROWS`: The number of rows in the table in statistics.
- `AVG_ROW_LENGTH`: The average row length of the table. `AVG_ROW_LENGTH` = `DATA_LENGTH` / `TABLE_ROWS`.
- `DATA_LENGTH`: Data length. `DATA_LENGTH` = `TABLE_ROWS` * the sum of storage lengths of the columns in the tuple. The replicas of TiKV are not taken into account.
- `MAX_DATA_LENGTH`: The maximum data length. The value is currently `0`, which means the data length has no upper limit.
- `INDEX_LENGTH`: The index length. `INDEX_LENGTH` = `TABLE_ROWS` * the sum of lengths of the columns in the index tuple. The replicas of TiKV are not taken into account.
- `DATA_FREE`: Data fragment. The value is currently `0`.
- `AUTO_INCREMENT`: The current step of the auto- increment primary key.
- `CREATE_TIME`: The time at which the table is created.
- `UPDATE_TIME`: The time at which the table is updated.
- `CHECK_TIME`: The time at which the table is checked.
- `TABLE_COLLATION`: The collation of strings in the table.
- `CHECKSUM`: Checksum.
- `CREATE_OPTIONS`: Creates options.
- `TABLE_COMMENT`: The comments and notes of the table.

Most of the information in the table is the same as MySQL. Only two columns are newly defined by TiDB:

- `TIDB_TABLE_ID`: to indicate the internal ID of a table. This ID is unique in a TiDB cluster.

- `TIDB_ROW_ID_SHARDING_INFO`: to indicate the sharding type of a table. The possible values are as follows:
    - `"NOT_SHARDED"`: the table is not sharded.
    - `"NOT_SHARDED(PK_IS_HANDLE)"`: the table that defines an integer Primary Key as its row id is not sharded.
    - `"PK_AUTO_RANDOM_BITS={bit_number}"`: the table that defines an integer Primary Key as its row id is sharded because the Primary Key is assigned with `AUTO_RANDOM` attribute.
    - `"SHARD_BITS={bit_number}"`: the table is sharded using `SHARD_ROW_ID_BITS={bit_number}`.
    - NULL: the table is a system table or view, and thus cannot be sharded.

14.10.17.2.36 TABLE_CONSTRAINTS

The `TABLE_CONSTRAINTS` table describes which tables have constraints.

```
USE information_schema;
DESC table_constraints;
```

```
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| CONSTRAINT_CATALOG | varchar(512) | YES | | NULL   |       |
| CONSTRAINT_SCHEMA | varchar(64) | YES |   | NULL   |       |
| CONSTRAINT_NAME  | varchar(64) | YES |    | NULL   |       |
| TABLE_SCHEMA     | varchar(64) | YES |    | NULL   |       |
| TABLE_NAME       | varchar(64) | YES |    | NULL   |       |
| CONSTRAINT_TYPE  | varchar(64) | YES |    | NULL   |       |
+--------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
SELECT * FROM table_constraints WHERE constraint_type='UNIQUE';
```

```
+--
    ↪ -----------------+-------------------+-----------------------+-------------
    ↪
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME  | TABLE_SCHEMA
    ↪     | TABLE_NAME                    | CONSTRAINT_TYPE |
+--
    ↪ -----------------+-------------------+-----------------------+-------------
    ↪
| def                | mysql             | name                  | mysql
    ↪            | help_topic                    | UNIQUE          |
| def                | mysql             | tbl                   | mysql
    ↪            | stats_meta                    | UNIQUE          |
```

```
| def              | mysql            | tbl                   | mysql
   ↪            | stats_histograms                | UNIQUE        |
| def              | mysql            | tbl                   | mysql
   ↪            | stats_buckets                   | UNIQUE        |
| def              | mysql            | delete_range_index    | mysql
   ↪            | gc_delete_range                 | UNIQUE        |
| def              | mysql            | delete_range_done_index | mysql
   ↪         | gc_delete_range_done            | UNIQUE        |
| def              | PERFORMANCE_SCHEMA | SCHEMA_NAME        |
   ↪ PERFORMANCE_SCHEMA | events_statements_summary_by_digest | UNIQUE |
+--
   ↪ ------------------+--------------------+-------------------------+--------------
   ↪
7 rows in set (0.01 sec)
```

Fields in the `TABLE_CONSTRAINTS` table are described as follows:

- `CONSTRAINT_CATALOG`: The name of the catalog to which the constraint belongs. This value is always `def`.
- `CONSTRAINT_SCHEMA`: The name of the database to which the constraint belongs.
- `CONSTRAINT_NAME`: The name of the constraint.
- `TABLE_NAME`: The name of the table.
- `CONSTRAINT_TYPE`: The type of the constraint. The value can be `UNIQUE`, `PRIMARY` ↪ `KEY` or `FOREIGN KEY`. The `UNIQUE` and `PRIMARY KEY` information is similar to the execution result of the `SHOW INDEX` statement.

14.10.17.2.37 TABLE_STORAGE_STATS

The `TABLE_STORAGE_STATS` table provides information about table sizes as stored by the storage engine (TiKV).

```
USE information_schema;
DESC table_storage_stats;
```

```
+--------------------+-------------+------+------+---------+-------+
| Field            | Type       | Null | Key | Default | Extra |
+--------------------+-------------+------+------+---------+-------+
| TABLE_SCHEMA     | varchar(64) | YES |    | NULL   |       |
| TABLE_NAME       | varchar(64) | YES |    | NULL   |       |
| TABLE_ID         | bigint(21) | YES |     | NULL   |       |
| PEER_COUNT       | bigint(21) | YES |     | NULL   |       |
| REGION_COUNT     | bigint(21) | YES |     | NULL   |       |
| EMPTY_REGION_COUNT | bigint(21) | YES |  | NULL   |       |
| TABLE_SIZE       | bigint(64) | YES |     | NULL   |       |
| TABLE_KEYS       | bigint(64) | YES |     | NULL   |       |
```

```
+-------------------+-------------+------+------+---------+-------+
8 rows in set (0.00 sec)
```

```
CREATE TABLE test.t1 (id INT);
INSERT INTO test.t1 VALUES (1);
SELECT * FROM table_storage_stats WHERE table_schema = 'test' AND
    ↪ table_name = 't1'\G
```

```
*************************** 1. row ***************************
     TABLE_SCHEMA: test
       TABLE_NAME: t1
         TABLE_ID: 56
       PEER_COUNT: 1
     REGION_COUNT: 1
EMPTY_REGION_COUNT: 1
       TABLE_SIZE: 1
       TABLE_KEYS: 0
1 row in set (0.00 sec)
```

Fields in the `TABLE_STORAGE_STATS` table are described as follows:

- `TABLE_SCHEMA`: The name of the schema to which the table belongs.
- `TABLE_NAME`: The name of the table.
- `TABLE_ID`: The ID of the table.
- `PEER_COUNT`: The number of replicas of the table.
- `REGION_COUNT`: The number of Regions.
- `EMPTY_REGION_COUNT`: The number of Regions that do not contain data in this table.
- `TABLE_SIZE`: The total size of the table, in the unit of MiB.
- `TABLE_KEYS`: The total number of records in the table.

14.10.17.2.38 TIDB_HOT_REGIONS

The `TIDB_HOT_REGIONS` table provides information about the current hot Regions. For information about history hot Regions, see `[TIDB_HOT_REGIONS_HISTORY](#
↪ tidb_hot_regions_history)`.

```
USE information_schema;
DESC tidb_hot_regions;
```

```
+---------------+-------------+------+------+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+------+---------+-------+
| TABLE_ID      | bigint(21)  | YES |     | NULL    |       |
| INDEX_ID      | bigint(21)  | YES |     | NULL    |       |
| DB_NAME       | varchar(64) | YES |    | NULL    |       |
```

```
| TABLE_NAME   | varchar(64) | YES |    | NULL   |       |
| INDEX_NAME   | varchar(64) | YES |    | NULL   |       |
| REGION_ID    | bigint(21) | YES |     | NULL   |       |
| TYPE         | varchar(64) | YES |    | NULL   |       |
| MAX_HOT_DEGREE | bigint(21) | YES |   | NULL   |       |
| REGION_COUNT | bigint(21) | YES |     | NULL   |       |
| FLOW_BYTES   | bigint(21) | YES |     | NULL   |       |
+---------------+------------+------+------+---------+-------+
10 rows in set (0.00 sec)
```

The description of columns in the `TIDB_HOT_REGIONS` table is as follows:

- `TABLE_ID`: The ID of the table in which the hot Region is located.
- `INDEX_ID`: The ID of the index in which the hot Region is located.
- `DB_NAME`: The database name of the object in which the hot Region is located.
- `TABLE_NAME`: The name of the table in which the hot Region is located.
- `INDEX_NAME`: The name of the index in which the hot Region is located.
- `REGION_ID`: The ID of the hot Region.
- `TYPE`: The type of the hot Region.
- `MAX_HOT_DEGREE`: The maximum hot degree of the Region.
- `REGION_COUNT`: The number of hot Regions in the instance.
- `FLOW_BYTES`: The number of bytes written and read in the Region.

14.10.17.2.39 TIDB_HOT_REGIONS_HISTORY

The `TIDB_HOT_REGIONS_HISTORY` table provides information about history hot Regions that are periodically recorded locally by PD.

You can specify the record interval by configuring `hot-regions-write-interval`. The default value is 10 minutes. You can specify the period for reserving history information about hot Regions by configuring `hot-regions-reserved-days`. The default value is 7 days. See PD configuration file description for details.

```
USE information_schema;
DESC tidb_hot_regions_history;
```

```
+-------------+--------------+------+------+---------+-------+
| Field       | Type         | Null | Key  | Default | Extra |
+-------------+--------------+------+------+---------+-------+
| UPDATE_TIME | timestamp(6) | YES  |      | NULL    |       |
| DB_NAME     | varchar(64)  | YES  |      | NULL    |       |
| TABLE_NAME  | varchar(64)  | YES  |      | NULL    |       |
| TABLE_ID    | bigint(21)   | YES  |      | NULL    |       |
| INDEX_NAME  | varchar(64)  | YES  |      | NULL    |       |
| INDEX_ID    | bigint(21)   | YES  |      | NULL    |       |
| REGION_ID   | bigint(21)   | YES  |      | NULL    |       |
```

```
| STORE_ID   | bigint(21) | YES |     | NULL   |       |
| PEER_ID    | bigint(21) | YES |     | NULL   |       |
| IS_LEARNER | tinyint(1) | NO  |     | 0      |       |
| IS_LEADER  | tinyint(1) | NO  |     | 0      |       |
| TYPE       | varchar(64) | YES |    | NULL   |       |
| HOT_DEGREE | bigint(21) | YES |     | NULL   |       |
| FLOW_BYTES | double     | YES |     | NULL   |       |
| KEY_RATE   | double     | YES |     | NULL   |       |
| QUERY_RATE | double     | YES |     | NULL   |       |
+------------+------------+-----+-----+--------+-------+
16 rows in set (0.00 sec)
```

The fields in the `TIDB_HOT_REGIONS_HISTORY` table are described as follows:

- UPDATE_TIME: The update time of the hot Region.
- DB_NAME: The database name of the object in which the hot Region is located.
- TABLE_ID: The ID of the table in which the hot Region is located.
- TABLE_NAME: The name of the table in which the hot Region is located.
- INDEX_NAME: The name of the index in which the hot Region is located.
- INDEX_ID: The ID of the index in which the hot Region is located.
- REGION_ID: The ID of the hot Region.
- STORE_ID: The ID of the store in which the hot Region is located.
- PEER_ID: The ID of the Peer corresponding to the hot Region.
- IS_LEARNER: Whether the PEER is the LEARNER.
- IS_LEADER: Whether the PEER is the LEADER.
- TYPE: The type of the hot Region.
- HOT_DEGREE: The hot degree of the hot Region.
- FLOW_BYTES: The number of bytes written and read in the Region.
- KEY_RATE: The number of keys written and read in the Region.
- QUERY_RATE: The number of queries written and read in the Region.

Note:

`UPDATE_TIME`, `REGION_ID`, `STORE_ID`, `PEER_ID`, `IS_LEARNER`, `IS_LEADER` and `TYPE` fields are pushed down to the PD servers for execution. To reduce the overhead of using the table, you must specify the time range for the search, and specify as many conditions as possible. For example,

```
select * from tidb_hot_regions_history where store_id = 11
↪ and update_time > '2020-05-18 20:40:00' and update_time <
↪ '2020-05-18 21:40:00' and type='write'.
```

Common user scenarios

- Query hot Regions within a specific period of time. Replace `update_time` with your actual time.

```
SELECT * FROM INFORMATION_SCHEMA.TIDB_HOT_REGIONS_HISTORY WHERE
    ↪ update_time >'2021-08-18 21:40:00' and update_time <'2021-09-19
    ↪ 00:00:00';
```

> **Note:**
>
> `UPDATE_TIME` also supports Unix timestamps. For example, `update_time >TIMESTAMP('2021-08-18 21:40:00')` or `update_time > FROM_UNIXTIME(1629294000.000)`.

- Query hot Regions in a table within a specific period of time. Replace `update_time` and `table_name` with your actual values.

```
SELECT * FROM INFORMATION_SCHEMA.TIDB_HOT_REGIONS_HISTORY WHERE
    ↪ update_time >'2021-08-18 21:40:00' and update_time <'2021-09-19
    ↪ 00:00:00' and TABLE_NAME = 'table_name';
```

- Query the distribution of hot Regions within a specific period of time. Replace `update_time` and `table_name` with your actual values.

```
SELECT count(region_id) cnt, store_id FROM INFORMATION_SCHEMA.
    ↪ TIDB_HOT_REGIONS_HISTORY WHERE update_time >'2021-08-18 21:40:00
    ↪ ' and update_time <'2021-09-19 00:00:00' and table_name = '
    ↪ table_name' GROUP BY STORE_ID ORDER BY cnt DESC;
```

- Query the distribution of hot Leader Regions within a specific period of time. Replace `update_time` and `table_name` with your actual values.

```
SELECT count(region_id) cnt, store_id FROM INFORMATION_SCHEMA.
    ↪ TIDB_HOT_REGIONS_HISTORY WHERE update_time >'2021-08-18 21:40:00
    ↪ ' and update_time <'2021-09-19 00:00:00' and table_name = '
    ↪ table_name' and is_leader=1 GROUP BY STORE_ID ORDER BY cnt DESC;
```

- Query the distribution of hot Index Regions within a specific period of time. Replace `update_time` and `table_name` with your actual values.

```
SELECT count(region_id) cnt, index_name, store_id FROM
    ↪ INFORMATION_SCHEMA.TIDB_HOT_REGIONS_HISTORY WHERE update_time >'
    ↪ 2021-08-18 21:40:00' and update_time <'2021-09-19 00:00:00' and
    ↪ table_name = 'table_name' group by index_name, store_id order by
    ↪  index_name,cnt desc;
```

- Query the distribution of hot Index Leader Regions within a specific period of time. Replace `update_time` and `table_name` with your actual values.

```
SELECT count(region_id) cnt, index_name, store_id FROM
    ↪ INFORMATION_SCHEMA.TIDB_HOT_REGIONS_HISTORY WHERE update_time >'
    ↪ 2021-08-18 21:40:00' and update_time <'2022-09-19 00:00:00' and
    ↪ table_name = 'table_name' and is_leader=1 group by index_name,
    ↪ store_id order by index_name,cnt desc;
```

14.10.17.2.40 TIDB_INDEXES

The `TIDB_INDEXES` table provides the INDEX information of all tables.

```
USE information_schema;
DESC tidb_indexes;
```

```
+---------------+--------------+------+------+---------+-------+
| Field         | Type         | Null | Key  | Default | Extra |
+---------------+--------------+------+------+---------+-------+
| TABLE_SCHEMA  | varchar(64)  | YES  |      | NULL    |       |
| TABLE_NAME    | varchar(64)  | YES  |      | NULL    |       |
| NON_UNIQUE    | bigint(21)   | YES  |      | NULL    |       |
| KEY_NAME      | varchar(64)  | YES  |      | NULL    |       |
| SEQ_IN_INDEX  | bigint(21)   | YES  |      | NULL    |       |
| COLUMN_NAME   | varchar(64)  | YES  |      | NULL    |       |
| SUB_PART      | bigint(21)   | YES  |      | NULL    |       |
| INDEX_COMMENT | varchar(2048) | YES |  | NULL    |       |
| Expression    | varchar(64)  | YES  |      | NULL    |       |
| INDEX_ID      | bigint(21)   | YES  |      | NULL    |       |
| IS_VISIBLE    | varchar(64)  | YES  |      | NULL    |       |
| CLUSTERED     | varchar(64)  | YES  |      | NULL    |       |
+---------------+--------------+------+------+---------+-------+
12 rows in set (0.00 sec)
```

`INDEX_ID` is the unique ID that TiDB allocates for each index. It can be used to do a join operation with `INDEX_ID` obtained from another table or API.

For example, you can obtain `TABLE_ID` and `INDEX_ID` that are involved in some slow query in the `SLOW_QUERY` table and then obtain the specific index information using the following SQL statements:

```
SELECT
 tidb_indexes.*
FROM
 tidb_indexes,
 tables
WHERE
```

```
 tidb_indexes.table_schema = tables.table_schema
AND tidb_indexes.table_name = tidb_indexes.table_name
AND tables.tidb_table_id = ?
AND index_id = ?
```

Fields in the TIDB_INDEXES table are described as follows:

- TABLE_SCHEMA: The name of the schema to which the index belongs.
- TABLE_NAME: The name of the table to which the index belongs.
- NON_UNIQUE: If the index is unique, the value is 0; otherwise, the value is 1.
- KEY_NAME: The index name. If the index is the primary key, the name is PRIMARY.
- SEQ_IN_INDEX: The sequential number of columns in the index, which starts from 1.
- COLUMN_NAME: The name of the column where the index is located.
- SUB_PART: The prefix length of the index. If the the column is partly indexed, the SUB_PART value is the count of the indexed characters; otherwise, the value is NULL.
- INDEX_COMMENT: The comment of the index, which is made when the index is created.
- INDEX_ID: The index ID.
- IS_VISIBLE: Whether the index is visible.
- CLUSTERED: Whether it is a clustered index.

14.10.17.2.41 TIDB_SERVERS_INFO

The TIDB_SERVERS_INFO table provides information about TiDB servers in the TiDB Cluster (namely, tidb-server processes).

```
USE information_schema;
DESC tidb_servers_info;
```

```
+---------------+-------------+------+------+---------+-------+
| Field         | Type        | Null | Key  | Default | Extra |
+---------------+-------------+------+------+---------+-------+
| DDL_ID        | varchar(64) | YES  |      | NULL    |       |
| IP            | varchar(64) | YES  |      | NULL    |       |
| PORT          | bigint(21)  | YES  |      | NULL    |       |
| STATUS_PORT   | bigint(21)  | YES  |      | NULL    |       |
| LEASE         | varchar(64) | YES  |      | NULL    |       |
| VERSION       | varchar(64) | YES  |      | NULL    |       |
| GIT_HASH      | varchar(64) | YES  |      | NULL    |       |
| BINLOG_STATUS | varchar(64) | YES  |      | NULL    |       |
+---------------+-------------+------+------+---------+-------+
8 rows in set (0.00 sec)
```

```
SELECT * FROM tidb_servers_info\G
```

```
*************************** 1. row ***************************
      DDL_ID: 771c169d-0a3a-48ea-a93c-a4d6751d3674
          IP: 0.0.0.0
        PORT: 4000
  STATUS_PORT: 10080
       LEASE: 45s
     VERSION: 5.7.25-TiDB-v4.0.0-beta.2-720-g0df3b74f5
    GIT_HASH: 0df3b74f55f8f8fbde39bbd5d471783f49dc10f7
BINLOG_STATUS: Off
1 row in set (0.00 sec)
```

14.10.17.2.42 TIDB_TRX

The TIDB_TRX table provides information about the transactions currently being executed on the TiDB node.

```
USE information_schema;
DESC tidb_trx;
```

```
+--
   ↪ ----------------------+-----------------------------------------------------------
   ↪
| Field                | Type
   ↪                                                  | Null | Key |
   ↪ Default | Extra |
+--
   ↪ ----------------------+-----------------------------------------------------------
   ↪
| ID                   | bigint(21) unsigned
   ↪                                        | NO   | PRI | NULL    |       |
| START_TIME           | timestamp(6)
   ↪                                             | YES |      | NULL    |
   ↪      |
| CURRENT_SQL_DIGEST   | varchar(64)
   ↪                                              | YES |      | NULL    |
   ↪      |
| CURRENT_SQL_DIGEST_TEXT | text
   ↪                                                  | YES |      | NULL    |
   ↪       |
| STATE                | enum('Idle','Running','LockWaiting','Committing','
   ↪ RollingBack') | YES | | NULL |    |
| WAITING_START_TIME   | timestamp(6)
   ↪                                             | YES |      | NULL    |
   ↪      |
```

```
| MEM_BUFFER_KEYS      | bigint(64)
   ↪                                                | YES |      | NULL   |
   ↪       |
| MEM_BUFFER_BYTES     | bigint(64)
   ↪                                                | YES |      | NULL   |
   ↪       |
| SESSION_ID           | bigint(21) unsigned
   ↪                                          | YES |      | NULL   |       |
| USER                 | varchar(16)
   ↪                                             | YES |      | NULL   |
   ↪       |
| DB                   | varchar(64)
   ↪                                             | YES |      | NULL   |
   ↪       |
| ALL_SQL_DIGESTS      | text
   ↪                                                   | YES |      | NULL
   ↪    |       |
+--
   ↪ ----------------------+------------------------------------------------------------
   ↪
```

The meaning of each column field in the `TIDB_TRX` table is as follows:

- `ID`: The transaction ID, which is the `start_ts` (start timestamp) of the transaction.
- `START_TIME`: The start time of the transaction, which is the physical time corresponding to the `start_ts` of the transaction.
- `CURRENT_SQL_DIGEST`: The digest of the SQL statement currently being executed in the transaction.
- `CURRENT_SQL_DIGEST_TEXT`: The normalized form of the SQL statement currently being executed by the transaction, that is, the SQL statement without arguments and format. It corresponds to `CURRENT_SQL_DIGEST`.
- `STATE`: The current state of the transaction. The possible values include:
    - `Idle`: The transaction is in an idle state, that is, it is waiting for the user to input a query.
    - `Running`: The transaction is executing a query.
    - `LockWaiting`: The transaction is waiting for the pessimistic lock to be acquired. Note that the transaction enters this state at the beginning of the pessimistic locking operation, no matter whether it is blocked by other transactions or not.
    - `Committing`: The transaction is in the process of commit.
    - `RollingBack`: The transaction is being rolled back.
- `WAITING_START_TIME`: When the value of `STATE` is `LockWaiting`, this column shows the start time of the waiting.
- `MEM_BUFFER_KEYS`: The number of keys written into the memory buffer by the current transaction.

- `MEM_BUFFER_BYTES`: The total number of key-value bytes written into the memory buffer by the current transaction.
- `SESSION_ID`: The ID of the session to which this transaction belongs.
- `USER`: The name of the user who performs the transaction.
- `DB`: The current default database name of the session in which the transaction is executed.
- `ALL_SQL_DIGESTS`: The digest list of statements that have been executed by the transaction. The list is shown as a string array in JSON format. Each transaction records at most the first 50 statements. Using the `TIDB_DECODE_SQL_DIGESTS` function, you can convert the information in this column into a list of corresponding normalized SQL statements.

Note:

- Only users with the PROCESS privilege can obtain the complete information in this table. Users without the PROCESS privilege can only query information of the transactions performed by the current user.
- The information (SQL digest) in the `CURRENT_SQL_DIGEST` and `ALL_SQL_DIGESTS` columns is the hash value calculated from the normalized SQL statement. The information in the `CURRENT_SQL_DIGEST_TEXT` column and the result returned from the `TIDB_DECODE_SQL_DIGESTS` function are internally queried from the statements summary tables, so it is possible that the corresponding statement cannot be found internally. For the detailed description of SQL digests and the statements summary tables, see Statement Summary Tables.
- The `TIDB_DECODE_SQL_DIGESTS` function call has a high overhead. If the function is called to query historical SQL statements for a large number of transactions, the query might take a long time. If the cluster is large with many concurrent transactions, avoid directly using this function on the `ALL_SQL_DIGEST` column while querying the full table of `TIDB_TRX`. This means to avoid an SQL statement like `select *, ↪ tidb_decode_sql_digests(all_sql_digests)from tidb_trx`.
- Currently the `TIDB_TRX` table does not support showing information of TiDB internal transactions.

Example

```
select * from information_schema.tidb_trx\G
```

```
*************************** 1. row ***************************
                 ID: 426789913200689153
         START_TIME: 2021-08-04 10:51:54.883000
```

```
    CURRENT_SQL_DIGEST: NULL
CURRENT_SQL_DIGEST_TEXT: NULL
                STATE: Idle
    WAITING_START_TIME: NULL
       MEM_BUFFER_KEYS: 1
      MEM_BUFFER_BYTES: 29
            SESSION_ID: 7
                  USER: root
                    DB: test
       ALL_SQL_DIGESTS: ["
          ↪ e6f07d43b5c21db0fbb9a31feac2dc599787763393dd5acbfad80e247eb02ad5
          ↪ ","04
          ↪ fa858fa491c62d194faec2ab427261cc7998b3f1ccf8f6844febca504cb5e9
          ↪ ","
          ↪ b83710fa8ab7df8504920e8569e48654f621cf828afbe7527fd003b79f48da9e
          ↪ "]
*************************** 2. row ***************************
                  ID: 426789921471332353
          START_TIME: 2021-08-04 10:52:26.433000
    CURRENT_SQL_DIGEST: 38
       ↪ b03afa5debbdf0326a014dbe5012a62c51957f1982b3093e748460f8b00821
CURRENT_SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ?
                STATE: LockWaiting
    WAITING_START_TIME: 2021-08-04 10:52:35.106568
       MEM_BUFFER_KEYS: 0
      MEM_BUFFER_BYTES: 0
            SESSION_ID: 9
                  USER: root
                    DB: test
       ALL_SQL_DIGESTS: ["
          ↪ e6f07d43b5c21db0fbb9a31feac2dc599787763393dd5acbfad80e247eb02ad5
          ↪ ","38
          ↪ b03afa5debbdf0326a014dbe5012a62c51957f1982b3093e748460f8b00821
          ↪ "]
2 rows in set (0.01 sec)
```

From the query result of this example, you can see that: the current node has two on-going transactions. One transaction is in the idle state (`STATE` is `Idle` and `CURRENT_SQL_DIGEST` is `NULL`), and this transaction has executed 3 statements (there are three records in the `ALL_SQL_DIGESTS` list, which are the digests of the three SQL statements that have been executed). Another transaction is executing a statement and waiting for the lock (`STATE` is `LockWaiting` and `WAITING_START_TIME` shows the start time of the waiting lock). The transaction has executed 2 statements, and the statement currently being executed is in the form of `"update `t` set `v` = `v` + ? where `id` = ?"`.

```
select id, all_sql_digests, tidb_decode_sql_digests(all_sql_digests) as
    ↪ all_sqls from information_schema.tidb_trx\G
```

```
*************************** 1. row ***************************
            id: 426789913200689153
all_sql_digests: ["
    ↪ e6f07d43b5c21db0fbb9a31feac2dc599787763393dd5acbfad80e247eb02ad5","04
    ↪ fa858fa491c62d194faec2ab427261cc7998b3f1ccf8f6844febca504cb5e9","
    ↪ b83710fa8ab7df8504920e8569e48654f621cf828afbe7527fd003b79f48da9e"]
       all_sqls: ["begin","insert into `t` values ( ... )","update `t` set `
          ↪ v` = `v` + ?"]
*************************** 2. row ***************************
            id: 426789921471332353
all_sql_digests: ["
    ↪ e6f07d43b5c21db0fbb9a31feac2dc599787763393dd5acbfad80e247eb02ad5","38
    ↪ b03afa5debbdf0326a014dbe5012a62c51957f1982b3093e748460f8b00821"]
       all_sqls: ["begin","update `t` set `v` = `v` + ? where `id` = ?"]
```

This query calls the TIDB_DECODE_SQL_DIGESTS function on the ALL_SQL_DIGESTS column of the TIDB_TRX table, and converts the SQL digest array into an array of normalized SQL statement through the system internal query. This helps you visually obtain the information of the statements that have been historically executed by the transaction. However, note that the above query scans the entire table of TIDB_TRX and calls the TIDB_DECODE_SQL_DIGESTS function for each row. Calling the TIDB_DECODE_SQL_DIGESTS function has a high overhead. Therefore, if many concurrent transactions exist in the cluster, try to avoid this type of query.

CLUSTER_TIDB_TRX

The TIDB_TRX table only provides information about the transactions that are being executed on a single TiDB node. If you want to view the information of the transactions that are being executed on all TiDB nodes in the entire cluster, you need to query the CLUSTER_TIDB_TRX table. Compared with the query result of the TIDB_TRX table, the query result of the CLUSTER_TIDB_TRX table includes an extra INSTANCE field. The INSTANCE field displays the IP address and port of each node in the cluster, which is used to distinguish the TiDB nodes where the transactions are located.

```
USE information_schema;
DESC cluster_tidb_trx;
```

```
mysql> desc cluster_tidb_trx;
+--
    ↪ ----------------------+-------------------------------------------------------
    ↪
| Field                 | Type
    ↪                                                    | Null | Key |
    ↪ Default | Extra |
```

```
+--
   ↪ ---------------------+-----------------------------------------------------
   ↪
| INSTANCE             | varchar(64)
   ↪                                                  | YES |      | NULL   |
   ↪       |
| ID                   | bigint(21) unsigned
   ↪                                          | NO   | PRI | NULL   |       |
| START_TIME           | timestamp(6)
   ↪                                               | YES |      | NULL   |
   ↪       |
| CURRENT_SQL_DIGEST   | varchar(64)
   ↪                                                | YES |      | NULL   |
   ↪       |
| CURRENT_SQL_DIGEST_TEXT | text
   ↪                                                    | YES |      | NULL   |
   ↪        |
| STATE                | enum('Idle','Running','LockWaiting','Committing','
   ↪ RollingBack') | YES | | NULL |    |
| WAITING_START_TIME   | timestamp(6)
   ↪                                               | YES |      | NULL   |
   ↪       |
| MEM_BUFFER_KEYS      | bigint(64)
   ↪                                                 | YES |      | NULL   |
   ↪       |
| MEM_BUFFER_BYTES     | bigint(64)
   ↪                                                 | YES |      | NULL   |
   ↪       |
| SESSION_ID           | bigint(21) unsigned
   ↪                                          | YES |      | NULL   |       |
| USER                 | varchar(16)
   ↪                                               | YES |      | NULL   |
   ↪       |
| DB                   | varchar(64)
   ↪                                               | YES |      | NULL   |
   ↪       |
| ALL_SQL_DIGESTS      | text
   ↪                                                      | YES |      | NULL
   ↪    |       |
+--
   ↪ ---------------------+-----------------------------------------------------
   ↪
```

14.10.17.2.43 TIFLASH_REPLICA

The TIFLASH_REPLICA table provides information about TiFlash replicas available.

```
USE information_schema;
DESC tiflash_replica;
```

```
+----------------+------------+------+------+---------+-------+
| Field          | Type       | Null | Key | Default | Extra |
+----------------+------------+------+------+---------+-------+
| TABLE_SCHEMA   | varchar(64) | YES |    | NULL    |       |
| TABLE_NAME     | varchar(64) | YES |    | NULL    |       |
| TABLE_ID       | bigint(21) | YES |     | NULL    |       |
| REPLICA_COUNT | bigint(64) | YES |     | NULL    |       |
| LOCATION_LABELS | varchar(64) | YES | | NULL    |       |
| AVAILABLE      | tinyint(1) | YES |     | NULL    |       |
| PROGRESS       | double     | YES |     | NULL    |       |
+----------------+------------+------+------+---------+-------+
7 rows in set (0.01 sec)
```

14.10.17.2.44 TIKV_REGION_PEERS

The TIKV_REGION_PEERS table shows detailed information of a single Region node in TiKV, such as whether it is a learner or leader.

```
USE information_schema;
DESC tikv_region_peers;
```

```
+--------------+------------+------+------+---------+-------+
| Field        | Type       | Null | Key | Default | Extra |
+--------------+------------+------+------+---------+-------+
| REGION_ID    | bigint(21) | YES |     | NULL    |       |
| PEER_ID      | bigint(21) | YES |     | NULL    |       |
| STORE_ID     | bigint(21) | YES |     | NULL    |       |
| IS_LEARNER | tinyint(1) | NO  |     | 0       |       |
| IS_LEADER  | tinyint(1) | NO  |     | 0       |       |
| STATUS       | varchar(10) | YES |    | 0       |       |
| DOWN_SECONDS | bigint(21) | YES |    | 0       |       |
+--------------+------------+------+------+---------+-------+
7 rows in set (0.01 sec)
```

For example, you can query the specific TiKV addresses for the top 3 Regions with the maximum value of WRITTEN_BYTES using the following SQL statement:

```
SELECT
 address,
 tikv.address,
 region.region_id
```

```
FROM
 tikv_store_status tikv,
 tikv_region_peers peer,
 (SELECT * FROM tikv_region_status ORDER BY written_bytes DESC LIMIT 3)
    ↪ region
WHERE
  region.region_id = peer.region_id
 AND peer.is_leader = 1
 AND peer.store_id = tikv.store_id;
```

Fields in the `TIKV_REGION_PEERS` table are described as follows:

- REGION_ID: The Region ID.
- PEER_ID: The ID of the Region peer.
- STORE_ID: The ID of the TiKV store where the Region is located.
- IS_LEARNER: Whether the peer is learner.
- IS_LEADER: Whether the peer is leader.
- STATUS: The statuses of a peer:
  - PENDING: Temporarily unavailable.
  - DOWN: Offline and converted. This peer no longer provides service.
  - NORMAL: Running normally.
- DOWN_SECONDS: The duration of being offline, in seconds.

14.10.17.2.45 TIKV_REGION_STATUS

The `TIKV_REGION_STATUS` table shows some basic information of TiKV Regions via PD's API, like the Region ID, starting and ending key-values, and read and write traffic.

```
USE information_schema;
DESC tikv_region_status;
```

```
+--
   ↪ ------------------------+-------------+------+------+---------+-------+
   ↪
| Field                  | Type       | Null | Key | Default | Extra |
+--
   ↪ ------------------------+-------------+------+------+---------+-------+
   ↪
| REGION_ID              | bigint(21) | YES |     | NULL   |       |
| START_KEY              | text       | YES |     | NULL   |       |
| END_KEY                | text       | YES |     | NULL   |       |
| TABLE_ID               | bigint(21) | YES |     | NULL   |       |
| DB_NAME                | varchar(64) | YES |    | NULL   |       |
| TABLE_NAME             | varchar(64) | YES |    | NULL   |       |
```

```
| IS_INDEX                | tinyint(1)  | NO  |     | 0       |       |
| INDEX_ID                | bigint(21)  | YES |     | NULL    |       |
| INDEX_NAME              | varchar(64) | YES |     | NULL    |       |
| EPOCH_CONF_VER          | bigint(21)  | YES |     | NULL    |       |
| EPOCH_VERSION           | bigint(21)  | YES |     | NULL    |       |
| WRITTEN_BYTES           | bigint(21)  | YES |     | NULL    |       |
| READ_BYTES              | bigint(21)  | YES |     | NULL    |       |
| APPROXIMATE_SIZE        | bigint(21)  | YES |     | NULL    |       |
| APPROXIMATE_KEYS        | bigint(21)  | YES |     | NULL    |       |
| REPLICATIONSTATUS_STATE | varchar(64) | YES |     | NULL    |       |
| REPLICATIONSTATUS_STATEID | bigint(21) | YES | | NULL    |       |
+--
    ↪ ------------------------+-------------+------+------+---------+-------+
    ↪
17 rows in set (0.00 sec)
```

The descriptions of the columns in the `TIKV_REGION_STATUS` table are as follows:

- `REGION_ID`: The ID of the Region.
- `START_KEY`: The value of the start key of the Region.
- `END_KEY`: The value of the end key of the Region.
- `TABLE_ID`: The ID of the table to which the Region belongs.
- `DB_NAME`: The name of the database to which `TABLE_ID` belongs.
- `TABLE_NAME`: The name of the table to which the Region belongs.
- `IS_INDEX`: Whether the Region data is an index. 0 means that it is not an index, while 1 means that it is an index. If the current Region contains both table data and index data, there will be multiple rows of records, and `IS_INDEX` is 0 and 1 respectively.
- `INDEX_ID`: The ID of the index to which the Region belongs. If `IS_INDEX` is 0, the value of this column is NULL.
- `INDEX_NAME`: The name of the index to which the Region belongs. If `IS_INDEX` is 0, the value of this column is NULL.
- `EPOCH_CONF_VER`: The version number of the Region configuration. The version number increases when a peer is added or removed.
- `EPOCH_VERSION`: The current version number of the Region. The version number increases when the Region is split or merged.
- `WRITTEN_BYTES`: The amount of data (bytes) written to the Region.
- `READ_BYTES`: The amount of data (bytes) that has been read from the Region.
- `APPROXIMATE_SIZE`: The approximate data size (MB) of the Region.
- `APPROXIMATE_KEYS`: The approximate number of keys in the Region.
- `REPLICATIONSTATUS_STATE`: The current replication status of the Region. The status might be `UNKNOWN`, `SIMPLE_MAJORITY`, or `INTEGRITY_OVER_LABEL`.
- `REPLICATIONSTATUS_STATEID`: The identifier corresponding to `REPLICATIONSTATUS_STATE` ↪ .

Also, you can implement the `top confver`, `top read` and `top write` operations in pd-

ctl via the `ORDER BY X LIMIT Y` operation on the `EPOCH_CONF_VER`, `WRITTEN_BYTES` and `READ_BYTES` columns.

You can query the top 3 Regions with the most write data using the following SQL statement:

```
SELECT * FROM tikv_region_status ORDER BY written_bytes DESC LIMIT 3;
```

14.10.17.2.46 TIKV_STORE_STATUS

The `TIKV_STORE_STATUS` table shows some basic information of TiKV nodes via PD's API, like the ID allocated in the cluster, address and port, and status, capacity, and the number of Region leaders of the current node.

```
USE information_schema;
DESC tikv_store_status;
```

```
+------------------+-------------+------+------+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+------+---------+-------+
| STORE_ID         | bigint(21)  | YES |     | NULL    |       |
| ADDRESS          | varchar(64) | YES |    | NULL    |       |
| STORE_STATE      | bigint(21)  | YES |     | NULL    |       |
| STORE_STATE_NAME | varchar(64) | YES |  | NULL    |       |
| LABEL            | json        | YES |     | NULL    |       |
| VERSION          | varchar(64) | YES |    | NULL    |       |
| CAPACITY         | varchar(64) | YES |    | NULL    |       |
| AVAILABLE        | varchar(64) | YES |    | NULL    |       |
| LEADER_COUNT     | bigint(21)  | YES |     | NULL    |       |
| LEADER_WEIGHT    | double      | YES |     | NULL    |       |
| LEADER_SCORE     | double      | YES |     | NULL    |       |
| LEADER_SIZE      | bigint(21)  | YES |     | NULL    |       |
| REGION_COUNT     | bigint(21)  | YES |     | NULL    |       |
| REGION_WEIGHT    | double      | YES |     | NULL    |       |
| REGION_SCORE     | double      | YES |     | NULL    |       |
| REGION_SIZE      | bigint(21)  | YES |     | NULL    |       |
| START_TS         | datetime    | YES |     | NULL    |       |
| LAST_HEARTBEAT_TS | datetime | YES |    | NULL    |       |
| UPTIME           | varchar(64) | YES |    | NULL    |       |
+------------------+-------------+------+------+---------+-------+
19 rows in set (0.00 sec)
```

The descriptions of the columns in the `TIKV_STORE_STATUS` table are as follows:

- `STORE_ID`: The ID of the Store.
- `ADDRESS`: The address of the Store.

- `STORE_STATE`: The identifier of the Store state, which corresponds to `STORE_STATE_NAME` ↪ .
- `STORE_STATE_NAME`: The name of the Store state. The name is `Up`, `Offline`, or `Tombstone`.
- `LABEL`: The label set for the Store.
- `VERSION`: The version number of the Store.
- `CAPACITY`: The storage capacity of the Store.
- `AVAILABLE`: The remaining storage space of the Store.
- `LEADER_COUNT`: The number of leaders on the Store.
- `LEADER_WEIGHT`: The leader weight of the Store.
- `LEADER_SCORE`: The leader score of the Store.
- `LEADER_SIZE`: The approximate total data size (MB) of all leaders on the Store.
- `REGION_COUNT`: The number of Regions on the Store.
- `REGION_WEIGHT`: The Region weight of the Store.
- `REGION_SCORE`: The Region score of the Store.
- `REGION_SIZE`: The approximate total data size (MB) of all Regions on the Store.
- `START_TS`: The timestamp when the Store is started.
- `LAST_HEARTBEAT_TS`: The timestamp of the last heartbeat sent by the Store.
- `UPTIME`: The total time since the Store starts.

14.10.17.2.47 USER_PRIVILEGES

The `USER_PRIVILEGES` table provides information about global privileges. This information comes from the `mysql.user` system table:

```
USE information_schema;
DESC user_privileges;
```

```
+----------------+-------------+------+------+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+------+---------+-------+
| GRANTEE        | varchar(81) | YES |     | NULL    |       |
| TABLE_CATALOG | varchar(512) | YES |   | NULL    |       |
| PRIVILEGE_TYPE | varchar(64) | YES |   | NULL    |       |
| IS_GRANTABLE | varchar(3) | YES |     | NULL    |       |
+----------------+-------------+------+------+---------+-------+
4 rows in set (0.00 sec)
```

```
SELECT * FROM user_privileges;
```

```
+------------+---------------+-------------------------+--------------+
| GRANTEE    | TABLE_CATALOG | PRIVILEGE_TYPE          | IS_GRANTABLE |
+------------+---------------+-------------------------+--------------+
| 'root'@'%' | def           | Select                  | YES          |
| 'root'@'%' | def           | Insert                  | YES          |
```

```
| 'root'@'%' | def       | Update                | YES        |
| 'root'@'%' | def       | Delete                | YES        |
| 'root'@'%' | def       | Create                | YES        |
| 'root'@'%' | def       | Drop                  | YES        |
| 'root'@'%' | def       | Process               | YES        |
| 'root'@'%' | def       | References            | YES        |
| 'root'@'%' | def       | Alter                 | YES        |
| 'root'@'%' | def       | Show Databases        | YES        |
| 'root'@'%' | def       | Super                 | YES        |
| 'root'@'%' | def       | Execute               | YES        |
| 'root'@'%' | def       | Index                 | YES        |
| 'root'@'%' | def       | Create User           | YES        |
| 'root'@'%' | def       | Trigger               | YES        |
| 'root'@'%' | def       | Create View           | YES        |
| 'root'@'%' | def       | Show View             | YES        |
| 'root'@'%' | def       | Create Role           | YES        |
| 'root'@'%' | def       | Drop Role             | YES        |
| 'root'@'%' | def       | CREATE TEMPORARY TABLES | YES      |
| 'root'@'%' | def       | LOCK TABLES           | YES        |
| 'root'@'%' | def       | CREATE ROUTINE        | YES        |
| 'root'@'%' | def       | ALTER ROUTINE         | YES        |
| 'root'@'%' | def       | EVENT                 | YES        |
| 'root'@'%' | def       | SHUTDOWN              | YES        |
| 'root'@'%' | def       | RELOAD                | YES        |
| 'root'@'%' | def       | FILE                  | YES        |
| 'root'@'%' | def       | CONFIG                | YES        |
+------------+---------------+-------------------------+--------------+
28 rows in set (0.00 sec)
```

Fields in the `USER_PRIVILEGES` table are described as follows:

- `GRANTEE`: The name of the granted user, which is in the format of `'user_name'@'host_name'`.
- `TABLE_CATALOG`: The name of the catalog to which the table belongs. This value is always `def`.
- `PRIVILEGE_TYPE`: The privilege type to be granted. Only one privilege type is shown in each row.
- `IS_GRANTABLE`: If you have the `GRANT OPTION` privilege, the value is `YES`; otherwise, the value is `NO`.

14.10.17.2.48 VIEWS

The `VIEWS` table provides information about SQL views.

```
USE information_schema;
DESC views;
```

```
+---------------------+-------------+------+-----+---------+-------+
| Field               | Type        | Null | Key | Default | Extra |
+---------------------+-------------+------+-----+---------+-------+
| TABLE_CATALOG       | varchar(512) | NO |     | NULL    |       |
| TABLE_SCHEMA        | varchar(64) | NO |     | NULL    |       |
| TABLE_NAME          | varchar(64) | NO |     | NULL    |       |
| VIEW_DEFINITION     | longblob    | NO |     | NULL    |       |
| CHECK_OPTION        | varchar(8) | NO  |     | NULL    |       |
| IS_UPDATABLE        | varchar(3) | NO  |     | NULL    |       |
| DEFINER             | varchar(77) | NO |     | NULL    |       |
| SECURITY_TYPE       | varchar(7) | NO  |     | NULL    |       |
| CHARACTER_SET_CLIENT | varchar(32) | NO |   | NULL    |       |
| COLLATION_CONNECTION | varchar(32) | NO |   | NULL    |       |
+---------------------+-------------+------+-----+---------+-------+
10 rows in set (0.00 sec)
```

```
CREATE VIEW test.v1 AS SELECT 1;
SELECT * FROM views\G
```

```
*************************** 1. row ***************************
       TABLE_CATALOG: def
        TABLE_SCHEMA: test
          TABLE_NAME: v1
     VIEW_DEFINITION: SELECT 1
        CHECK_OPTION: CASCADED
        IS_UPDATABLE: NO
             DEFINER: root@127.0.0.1
       SECURITY_TYPE: DEFINER
CHARACTER_SET_CLIENT: utf8mb4
COLLATION_CONNECTION: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

Fields in the `VIEWS` table are described as follows:

- `TABLE_CATALOG`: The name of the catalog to which the view belongs. This value is always `def`.
- `TABLE_SCHEMA`: The name of the schema to which the view belongs.
- `TABLE_NAME`: The view name.
- `VIEW_DEFINITION`: The definition of view, which is made by the `SELECT` statement when the view is created.
- `CHECK_OPTION`: The `CHECK_OPTION` value. The value options are `NONE`, `CASCADE`, and `LOCAL`.
- `IS_UPDATABLE`: Whether `UPDATE`/`INSERT`/`DELETE` is applicable to the view. In TiDB, the value is always `NO`.

- DEFINER: The name of the user who creates the view, which is in the format of 'user_name'@'host_name'.
- SECURITY_TYPE: The value of SQL SECURITY. The value options are DEFINER and INVOKER.
- CHARACTER_SET_CLIENT: The value of the character_set_client session variable when the view is created.
- COLLATION_CONNECTION: The value of the collation_connection session variable when the view is created.

14.10.17.3 Metrics Schema

The METRICS_SCHEMA is a set of views on top of TiDB metrics that are stored in Prometheus. The source of the PromQL (Prometheus Query Language) for each of the tables is available in INFORMATION_SCHEMA.METRICS_TABLES.

```
USE metrics_schema;
SELECT * FROM uptime;
SELECT * FROM information_schema.metrics_tables WHERE table_name='uptime'\G
```

```
+--
   ↪ ------------------------+----------------+-----------+--------------------+
   ↪
| time                       | instance       | job       | value              |
+--
   ↪ ------------------------+----------------+-----------+--------------------+
   ↪
| 2020-07-06 15:26:26.203000 | 127.0.0.1:10080 | tidb | 123.60300016403198 |
| 2020-07-06 15:27:26.203000 | 127.0.0.1:10080 | tidb | 183.60300016403198 |
| 2020-07-06 15:26:26.203000 | 127.0.0.1:20180 | tikv | 123.60300016403198 |
| 2020-07-06 15:27:26.203000 | 127.0.0.1:20180 | tikv | 183.60300016403198 |
| 2020-07-06 15:26:26.203000 | 127.0.0.1:2379 | pd   | 123.60300016403198 |
| 2020-07-06 15:27:26.203000 | 127.0.0.1:2379 | pd   | 183.60300016403198 |
| 2020-07-06 15:26:26.203000 | 127.0.0.1:9090 | prometheus |
   ↪ 123.72300004959106 |
| 2020-07-06 15:27:26.203000 | 127.0.0.1:9090 | prometheus |
   ↪ 183.72300004959106 |
+--
   ↪ ------------------------+----------------+-----------+--------------------+
   ↪
8 rows in set (0.00 sec)

*************************** 1. row ***************************
TABLE_NAME: uptime
    PROMQL: (time() - process_start_time_seconds{$LABEL_CONDITIONS})
    LABELS: instance,job
  QUANTILE: 0
```

```
   COMMENT: TiDB uptime since last restart(second)
1 row in set (0.00 sec)
```

```
SHOW TABLES;
```

```
+-----------------------------------------------+
| Tables_in_metrics_schema                      |
+-----------------------------------------------+
| abnormal_stores                               |
| etcd_disk_wal_fsync_rate                      |
| etcd_wal_fsync_duration                       |
| etcd_wal_fsync_total_count                    |
| etcd_wal_fsync_total_time                     |
| go_gc_count                                   |
| go_gc_cpu_usage                               |
| go_gc_duration                                |
| go_heap_mem_usage                             |
| go_threads                                    |
| goroutines_count                              |
| node_cpu_usage                                |
| node_disk_available_size                      |
| node_disk_io_util                             |
| node_disk_iops                                |
| node_disk_read_latency                        |
| node_disk_size                                |
..
| tikv_storage_async_request_total_time         |
| tikv_storage_async_requests                   |
| tikv_storage_async_requests_total_count       |
| tikv_storage_command_ops                      |
| tikv_store_size                               |
| tikv_thread_cpu                               |
| tikv_thread_nonvoluntary_context_switches     |
| tikv_thread_voluntary_context_switches        |
| tikv_threads_io                               |
| tikv_threads_state                            |
| tikv_total_keys                               |
| tikv_wal_sync_duration                        |
| tikv_wal_sync_max_duration                    |
| tikv_worker_handled_tasks                     |
| tikv_worker_handled_tasks_total_num           |
| tikv_worker_pending_tasks                     |
| tikv_worker_pending_tasks_total_num           |
| tikv_write_stall_avg_duration                 |
| tikv_write_stall_max_duration                 |
```

```
| tikv_write_stall_reason                          |
| up                                               |
| uptime                                           |
+--------------------------------------------------+
626 rows in set (0.00 sec)
```

The METRICS_SCHEMA is used as a data source for monitoring-related summary tables such as (metrics_summary, metrics_summary_by_label and inspection_summary.

14.10.17.3.1 Additional Examples

Taking the tidb_query_duration monitoring table in metrics_schema as an example, this section illustrates how to use this monitoring table and how it works. The working principles of other monitoring tables are similar to tidb_query_duration.

Query the information related to the tidb_query_duration table on information_schema ↪ .metrics_tables:

```
SELECT * FROM information_schema.metrics_tables WHERE table_name='
    ↪ tidb_query_duration';
```

```
+--
    ↪ ------------------+-------------------------------------------------------------
    ↪
| TABLE_NAME        | PROMQL
    ↪
    ↪ | LABELS          | QUANTILE | COMMENT                                  |
+--
    ↪ ------------------+-------------------------------------------------------------
    ↪
| tidb_query_duration | histogram_quantile($QUANTILE, sum(rate(
    ↪ tidb_server_handle_query_duration_seconds_bucket{$LABEL_CONDITIONS}[
    ↪ $RANGE_DURATION])) by (le,sql_type,instance)) | instance,sql_type |
    ↪ 0.9 | The quantile of TiDB query durations(second) |
+--
    ↪ ------------------+-------------------------------------------------------------
    ↪
```

Field description:

- TABLE_NAME: Corresponds to the table name in metrics_schema . In this example, the table name is tidb_query_duration.
- PROMQL: The working principle of the monitoring table is to first map SQL statements to PromQL, then to request data from Prometheus, and to convert Prometheus results into SQL query results. This field is the expression template of PromQL. When you query the data of the monitoring table, the query conditions are used to rewrite the variables in this template to generate the final query expression.

- `LABELS`: The label for the monitoring item. `tidb_query_duration` has two labels: `instance` and `sql_type`.
- `QUANTILE`: The percentile. For monitoring data of the histogram type, a default percentile is specified. If the value of this field is `0`, it means that the monitoring item corresponding to the monitoring table is not a histogram.
- `COMMENT`: Explanations for the monitoring table. You can see that the `tidb_query_duration` table is used to query the percentile time of the TiDB query execution, such as the query time of P999/P99/P90. The unit is second.

To query the schema of the `tidb_query_duration` table, execute the following statement:

```
SHOW CREATE TABLE metrics_schema.tidb_query_duration;
```

```
+--
    ↪ ------------------+-----------------------------------------------------------
    ↪
| Table              | Create Table
    ↪
    ↪ |
+--
    ↪ ------------------+-----------------------------------------------------------
    ↪
| tidb_query_duration | CREATE TABLE `tidb_query_duration` (
    ↪                                                                    |
|                    |   `time` datetime unsigned DEFAULT CURRENT_TIMESTAMP,
    ↪                                                  |
|                    |   `instance` varchar(512) DEFAULT NULL,
    ↪                                                                    |
|                    |   `sql_type` varchar(512) DEFAULT NULL,
    ↪                                                                    |
|                    |   `quantile` double unsigned DEFAULT '0.9',
    ↪                                                                  |
|                    |   `value` double unsigned DEFAULT NULL
    ↪                                                                     |
|                    | ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=
    ↪ utf8mb4_bin COMMENT='The quantile of TiDB query durations(second)' |
+--
    ↪ ------------------+-----------------------------------------------------------
    ↪
```

- `time`: The time of the monitoring item.
- `instance` and `sql_type`: The labels of the `tidb_query_duration` monitoring item. `instance` means the monitoring address. `sql_type` means the type of the executed SQL statement.

- `quantile`: The percentile. The monitoring item of the histogram type has this column, which indicates the percentile time of the query. For example, `quantile = 0.9` means to query the time of P90.
- `value`: The value of the monitoring item.

The following statement queries the P99 time within the range of [2020-03-25 23:40:00 ↪ , 2020-03-25 23:42:00].

```
SELECT * FROM metrics_schema.tidb_query_duration WHERE value is not null
    ↪ AND time>='2020-03-25 23:40:00' AND time <= '2020-03-25 23:42:00' AND
    ↪  quantile=0.99;
```

```
+--
    ↪ ------------------+------------------+----------+----------+----------------+
    ↪
| time                | instance         | sql_type | quantile | value          |
+--
    ↪ ------------------+------------------+----------+----------+----------------+
    ↪
| 2020-03-25 23:40:00 | 172.16.5.40:10089 | Insert | 0.99 | 0.509929485256
    ↪ |
| 2020-03-25 23:41:00 | 172.16.5.40:10089 | Insert | 0.99 | 0.494690793986
    ↪ |
| 2020-03-25 23:42:00 | 172.16.5.40:10089 | Insert | 0.99 | 0.493460506934
    ↪ |
| 2020-03-25 23:40:00 | 172.16.5.40:10089 | Select | 0.99 | 0.152058493415
    ↪ |
| 2020-03-25 23:41:00 | 172.16.5.40:10089 | Select | 0.99 | 0.152193879678
    ↪ |
| 2020-03-25 23:42:00 | 172.16.5.40:10089 | Select | 0.99 | 0.140498483232
    ↪ |
| 2020-03-25 23:40:00 | 172.16.5.40:10089 | internal | 0.99 | 0.47104  |
| 2020-03-25 23:41:00 | 172.16.5.40:10089 | internal | 0.99 | 0.11776  |
| 2020-03-25 23:42:00 | 172.16.5.40:10089 | internal | 0.99 | 0.11776  |
+--
    ↪ ------------------+------------------+----------+----------+----------------+
    ↪
```

The first row of the query result above means that at the time of 2020-03-25 23:40:00, on the TiDB instance `172.16.5.40:10089`, the P99 execution time of the `Insert` type statement is 0.509929485256 seconds. The meanings of other rows are similar. Other values of the `sql_type` column are described as follows:

- `Select`: The `select` type statement is executed.

- `internal`: The internal SQL statement of TiDB, which is used to update the statistical information and get the global variables.

To view the execution plan of the statement above, execute the following statement:

```
DESC SELECT * FROM metrics_schema.tidb_query_duration WHERE value is not
    ↪ null AND time>='2020-03-25 23:40:00' AND time <= '2020-03-25 23:42:00
    ↪ ' AND quantile=0.99;
```

```
+--
    ↪ ---------------+----------+------+---------------------------+----------------
    ↪
| id              | estRows | task | access object         | operator info
    ↪
    ↪ |
+--
    ↪ ---------------+----------+------+---------------------------+----------------
    ↪
| Selection_5     | 8000.00 | root |                       | not(isnull(Column
    ↪ #5))
    ↪
    ↪ |
|  MemTableScan_6 | 10000.00 | root | table:tidb_query_duration | PromQL:
    ↪ histogram_quantile(0.99, sum(rate(
    ↪ tidb_server_handle_query_duration_seconds_bucket{}[60s])) by (le,
    ↪ sql_type,instance)), start_time:2020-03-25 23:40:00, end_time
    ↪ :2020-03-25 23:42:00, step:1m0s |
+--
    ↪ ---------------+----------+------+---------------------------+----------------
    ↪
```

From the result above, you can see that `PromQL`, `start_time`, `end_time`, and `step` are in the execution plan. During the execution process, TiDB calls the `query_range` HTTP API of Prometheus to query the monitoring data.

You might find that in the range of [2020-03-25 23:40:00, 2020-03-25 23:42:00], each label only has three time values. In the execution plan, the value of `step` is 1 minute, which means that the interval of these values is 1 minute. `step` is determined by the following two session variables:

- `tidb_metric_query_step`: The query resolution step width. To get the `query_range` data from Prometheus, you need to specify `start_time`, `end_time`, and `step`. `step` uses the value of this variable.
- `tidb_metric_query_range_duration`: When the monitoring data is queried, the value of the `$ RANGE_DURATION` field in `PROMQL` is replaced with the value of this variable. The default value is 60 seconds.

To view the values of monitoring items with different granularities, you can modify the two session variables above before querying the monitoring table. For example:

1. Modify the values of the two session variables and set the time granularity to 30 seconds.

> **Note:**
> The minimum granularity supported by Prometheus is 30 seconds.

```
set @@tidb_metric_query_step=30;
set @@tidb_metric_query_range_duration=30;
```

2. Query the `tidb_query_duration` monitoring item as follows. From the result, you can see that within the 3-minute time range, each label has 6 time values, and the interval between each value is 30 seconds.

```
select * from metrics_schema.tidb_query_duration where value is not
    ↪ null and time>='2020-03-25 23:40:00' and time <= '2020-03-25
    ↪ 23:42:00' and quantile=0.99;
```

```
+--
    ↪ ------------------+------------------+---------+---------+------------
    ↪
| time               | instance        | sql_type | quantile | value
    ↪       |
+--
    ↪ ------------------+------------------+---------+---------+------------
    ↪
| 2020-03-25 23:40:00 | 172.16.5.40:10089 | Insert | 0.99 |
    ↪ 0.483285651924 |
| 2020-03-25 23:40:30 | 172.16.5.40:10089 | Insert | 0.99 |
    ↪ 0.484151462113 |
| 2020-03-25 23:41:00 | 172.16.5.40:10089 | Insert | 0.99 | 0.504576
    ↪     |
| 2020-03-25 23:41:30 | 172.16.5.40:10089 | Insert | 0.99 |
    ↪ 0.493577384561 |
| 2020-03-25 23:42:00 | 172.16.5.40:10089 | Insert | 0.99 |
    ↪ 0.49482474311 |
| 2020-03-25 23:40:00 | 172.16.5.40:10089 | Select | 0.99 |
    ↪ 0.189253402185 |
| 2020-03-25 23:40:30 | 172.16.5.40:10089 | Select | 0.99 |
    ↪ 0.184224951851 |
| 2020-03-25 23:41:00 | 172.16.5.40:10089 | Select | 0.99 |
    ↪ 0.151673410553 |
```

```
| 2020-03-25 23:41:30 | 172.16.5.40:10089 | Select | 0.99 |
    ↪ 0.127953838989 |
| 2020-03-25 23:42:00 | 172.16.5.40:10089 | Select | 0.99 |
    ↪ 0.127455434547 |
| 2020-03-25 23:40:00 | 172.16.5.40:10089 | internal | 0.99 | 0.0624
    ↪     |
| 2020-03-25 23:40:30 | 172.16.5.40:10089 | internal | 0.99 | 0.12416
    ↪    |
| 2020-03-25 23:41:00 | 172.16.5.40:10089 | internal | 0.99 | 0.0304
    ↪     |
| 2020-03-25 23:41:30 | 172.16.5.40:10089 | internal | 0.99 | 0.06272
    ↪    |
| 2020-03-25 23:42:00 | 172.16.5.40:10089 | internal | 0.99 |
    ↪ 0.0629333333333 |
+--
    ↪ ------------------+------------------+----------+----------+------------
    ↪
```

3. View the execution plan. From the result, you can also see that the values of `PromQL` and `step` in the execution plan have been changed to 30 seconds.

```
desc select * from metrics_schema.tidb_query_duration where value is
    ↪ not null and time>='2020-03-25 23:40:00' and time <= '2020-03-25
    ↪  23:42:00' and quantile=0.99;
```

```
+--
    ↪ ----------------+----------+------+--------------------------+------------
    ↪
| id              | estRows | task | access object         | operator
    ↪ info
    ↪
    ↪ |
+--
    ↪ ----------------+----------+------+--------------------------+------------
    ↪
| Selection_5     | 8000.00 | root |                       | not(isnull(
    ↪ Column#5))
    ↪
    ↪ |
|  ─MemTableScan_6 | 10000.00 | root | table:tidb_query_duration |
    ↪ PromQL:histogram_quantile(0.99, sum(rate(
    ↪ tidb_server_handle_query_duration_seconds_bucket{}[30s])) by (le
    ↪ ,sql_type,instance)), start_time:2020-03-25 23:40:00, end_time
    ↪ :2020-03-25 23:42:00, step:30s |
```

```
+--
    ↪ ---------------+----------+------+--------------------------+-----------
    ↪
```

14.11 UI

14.11.1 TiDB Dashboard

14.11.1.1 TiDB Dashboard Introduction

TiDB Dashboard is a Web UI for monitoring, diagnosing, and managing the TiDB cluster, which is available since v4.0. It is built into the PD component and does not require an independent deployment.

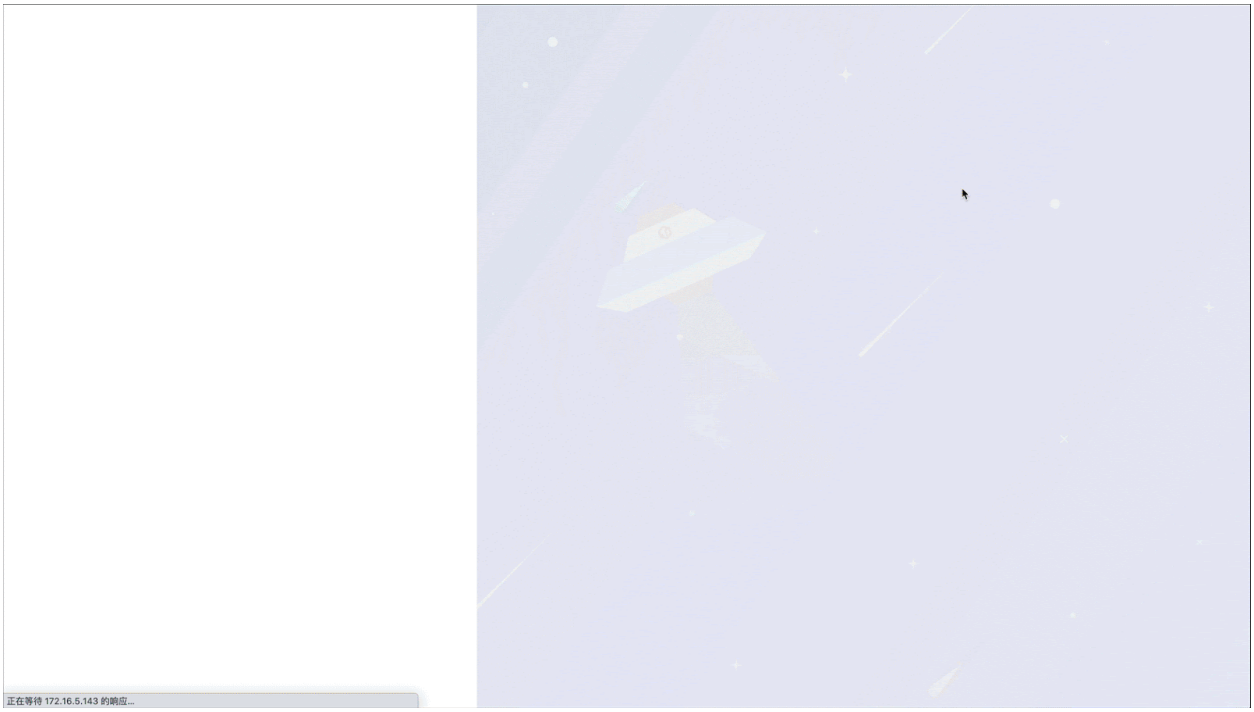


Figure 492: TiDB Dashboard interface

TiDB Dashboard is open-sourced on GitHub.

This document introduces the main features of TiDB Dashboard. You can click links in the following sections to learn more details.

14.11.1.1.1 Show the overall running status of the TiDB cluster

You can use TiDB Dashboard to learn the TiDB cluster's queries per second (QPS), execution time, the types of SQL statements that consume the most resources, and other

overview information.

See TiDB Dashboard Overview for details.

14.11.1.1.2 Show the running status of components and hosts

You can use TiDB Dashboard to view the running status of TiDB, TiKV, PD, TiFlash components in the entire cluster and the running status of the host on which these components are located.

See TiDB Dashboard Cluster Information Page for details.

14.11.1.1.3 Show distribution and trends of read and write traffic

The Key Visualizer feature of TiDB Dashboard visually shows the change of read and write traffic over time in the entire cluster in the form of heatmap. You can use this feature to timely discover changes of application modes or locate hotspot issues with uneven performance.

See Key Visualizer Page for details.

14.11.1.1.4 Show a list of execution information of all SQL statements

The execution information of all SQL statements is listed on the SQL Statements page. You can use this page to learn the execution time and total executions at all stages, which helps you analyze and locate the SQL queries that consume the most resources and improve the overall cluster performance.

See SQL Statements Page of TiDB Dashboard for details.

14.11.1.1.5 Learn the detailed execution information of slow queries

The Slow Queries page of TiDB Dashboard shows a list of all SQL statements that take a long time to execute, including the SQL texts and execution information. This page helps you locate the cause of slow queries or performance jitter.

See Slow Queries Page for details.

14.11.1.1.6 Diagnose common cluster problems and generate reports

The diagnostic feature of TiDB Dashboard automatically determines whether some common risks (such as inconsistent configurations) or problems exist in the cluster, generates reports, and gives operation suggestions, or compares the status of each cluster metric in different time ranges for you to analyze possible problems.

See TiDB Dashboard Cluster Diagnostics Page for details.

14.11.1.1.7 Query logs of all components

On the Search Logs page of TiDB Dashboard, you can quickly search logs of all running instances in the cluster by keywords, time range, and other conditions, package these logs, and download them to your local machine.

See Search Logs Page for details.

14.11.1.1.8 Collect profiling data for each instance

This is an advanced debugging feature that lets you profile each instance online and analyze various internal operations an instance performed during the profiling data collection period and the proportion of the operation execution time in this period without third-party tools.

See Profile Instances Page for details.

14.11.1.2 Maintain

14.11.1.2.1 Deploy TiDB Dashboard

The TiDB Dashboard UI is built into the PD component for v4.0 or higher versions, and no additional deployment is required. Simply deploy a standard TiDB cluster, and TiDB Dashboard will be there.

See the following documents to learn how to deploy a standard TiDB cluster:

- Quick Start Guide for the TiDB Database Platform
- Deploy TiDB in Production Environment
- Kubernetes environment deployment

> **Note:**
>
> You cannot deploy TiDB Dashboard in a TiDB cluster earlier than v4.0.

Deployment with multiple PD instances

When multiple PD instances are deployed in the cluster, only one of these instances serves the TiDB Dashboard.

When PD instances are running for the first time, they automatically negotiate with each other to choose one instance to serve the TiDB Dashboard. TiDB Dashboard will not run on other PD instances. The TiDB Dashboard service will always be provided by the chosen PD instance no matter PD instances are restarted or new PD instances are joined. However, there will be a re-negotiation when the PD instance that serves TiDB Dashboard is removed from the cluster (scaled-in). The negotiation process does not need user intervention.

When you access a PD instance that does not serve TiDB Dashboard, the browser will be redirected automatically to guide you to access the PD instance that serves the TiDB Dashboard, so that you can access the service normally. This process is illustrated in the image below.

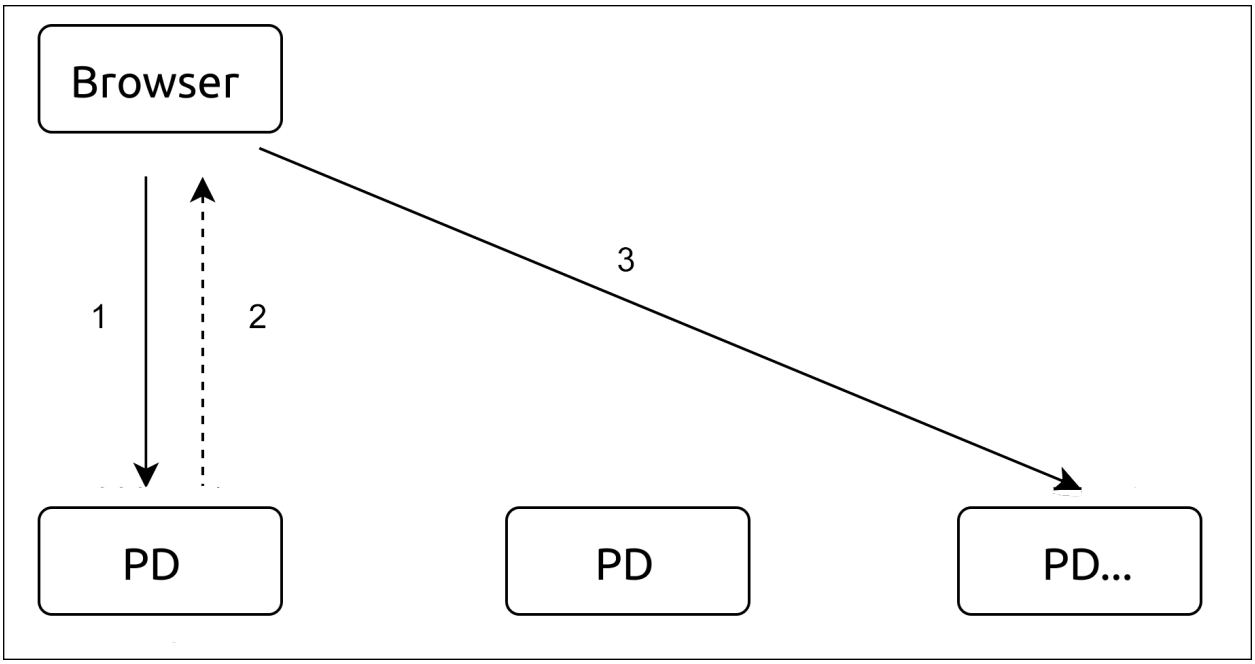


Figure 493: Process Schematic

Note:

The PD instance that serves TiDB Dashboard might not be a PD leader.

Check the PD instance that actually serves TiDB Dashboard

For a running cluster deployed using TiUP, you can use the `tiup cluster display` command to see which PD instance serves TiDB Dashboard. Replace `CLUSTER_NAME` with the cluster name.

```
tiup cluster display CLUSTER_NAME --dashboard
```

A sample output is as follows:

```
http://192.168.0.123:2379/dashboard/
```

Note:

This feature is available only in the later version of the `tiup cluster` deployment tool (v1.0.3 or later).

Upgrade TiUP Cluster

```
tiup update --self
tiup update cluster --force
```

Switch to another PD instance to serve TiDB Dashboard

For a running cluster deployed using TiUP, you can use the `tiup ctl:<cluster-version> pd` command to change the PD instance that serves TiDB Dashboard, or re-specify a PD instance to serve TiDB Dashboard when it is disabled:

```
tiup ctl:<cluster-version> pd -u http://127.0.0.1:2379 config set dashboard
    -address http://9.9.9.9:2379
```

In the command above:

- Replace `127.0.0.1:2379` with the IP and port of any PD instance.
- Replace `9.9.9.9:2379` with the IP and port of the new PD instance that you desire to run the TiDB Dashboard service.

You can use the `tiup cluster display` command to see whether the modification is taking effect (replace `CLUSTER_NAME` with the cluster name):

```
tiup cluster display CLUSTER_NAME --dashboard
```

Warning:

If you change the instance to run TiDB Dashboard, the local data stored in the previous TiDB Dashboard instance will be lost, including the Key Visualize history and search history.

Disable TiDB Dashboard

For a running cluster deployed using TiUP, use the `tiup ctl:<cluster-version> pd` command to disable TiDB Dashboard on all PD instances (replace `127.0.0.1:2379` with the IP and port of any PD instance):

```
tiup ctl:<cluster-version> pd -u http://127.0.0.1:2379 config set dashboard
    ↪ -address none
```

After disabling TiDB Dashboard, checking which PD instance provides the TiDB Dashboard service will fail:

```
Error: TiDB Dashboard is disabled
```

Visiting the TiDB Dashboard address of any PD instance via the browser will also fail:

```
Dashboard is not started.
```

Re-enable TiDB Dashboard

For a running cluster deployed using TiUP, use the `tiup ctl:<cluster-version> pd` ↪ command to request PD to renegotiate an instance to run TiDB Dashboard (replace `127.0.0.1:2379` with the IP and port of any PD instance):

```
tiup ctl:<cluster-version> pd -u http://127.0.0.1:2379 config set dashboard
    ↪ -address auto
```

After executing the command above, you can use the `tiup cluster display` command to view the TiDB Dashboard instance address automatically negotiated by PD (replace `CLUSTER_NAME` with the cluster name):

```
tiup cluster display CLUSTER_NAME --dashboard
```

You can also re-enable TiDB Dashboard by manually specifying the PD instance that serves TiDB Dashboard. See Switch to another PD instance to serve TiDB Dashboard.

> **Warning:**
>
> If the newly enabled TiDB Dashboard instance is different with the previous instance that served the TiDB Dashboard, the local data stored in the previous TiDB Dashboard instance will be lost, including Key Visualize history and search history.

What's next

- To learn how to access and log into the TiDB Dashboard UI, see Access TiDB Dashboard.
- To learn how to enhance the security of TiDB Dashboard, such as configuring a firewall, see Secure TiDB Dashboard.

14.11.1.2.2 Use TiDB Dashboard behind a Reverse Proxy

You can use a reverse proxy to safely expose the TiDB Dashboard service from the internal network to the external.

Procedures

Step 1: Get the actual TiDB Dashboard address

When multiple PD instances are deployed in the cluster, only one of the PD instances actually runs TiDB Dashboard. Therefore, you need to ensure that the upstream of the reverse proxy points to the correct address. For details of this mechanism, see Deployment with multiple PD instances.

When you use the TiUP tool for deployment, execute the following command to get the actual TiDB Dashboard address (replace `CLUSTER_NAME` with your cluster name):

```
tiup cluster display CLUSTER_NAME --dashboard
```

The output is the actual TiDB Dashboard address. A sample is as follows:

```
http://192.168.0.123:2379/dashboard/
```

Note:

This feature is available only in the later version of the `tiup cluster` deployment tool (v1.0.3 or later).

Upgrade TiUP Cluster

```
tiup update --self
tiup update cluster --force
```

Step 2: Configure the reverse proxy

Use HAProxy

When you use HAProxy as the reverse proxy, take the following steps:

1. Use reverse proxy for TiDB Dashboard on the `8033` port (for example). In the HAProxy configuration file, add the following configuration:

```
frontend tidb_dashboard_front
  bind *:8033
  use_backend tidb_dashboard_back if { path /dashboard } or { path_beg
     ↪ /dashboard/ }
```

```
backend tidb_dashboard_back
  mode http
  server tidb_dashboard 192.168.0.123:2379
```

Replace `192.168.0.123:2379` with IP and port of the actual address of the TiDB Dashboard obtained in Step 1.

> **Warning:**
>
> You must retain the `if` part in the `use_backend` directive to ensure that services **only in this path** are behind reverse proxy; otherwise, security risks might be introduced. See Secure TiDB Dashboard.

2. Restart HAProxy for the configuration to take effect.
3. Test whether the reverse proxy is effective: access the `/dashboard/` address on the `8033` port of the machine where HAProxy is located (such as `http://example.com :8033/dashboard/`) to access TiDB Dashboard.

Use NGINX

When you use NGINX as the reverse proxy, take the following steps:

1. Use reverse proxy for TiDB Dashboard on the `8033` port (for example). In the NGINX configuration file, add the following configuration:

```
server {
    listen 8033;
    location /dashboard/ {
    proxy_pass http://192.168.0.123:2379/dashboard/;
    }
}
```

Replace `http://192.168.0.123:2379/dashboard/` with the actual address of the TiDB Dashboard obtained in Step 1.

> **Warning:**
>
> You must keep the `/dashboard/` path in the `proxy_pass` directive to ensure that only the services under this path are reverse proxied. Otherwise, security risks will be introduced. See Secure TiDB Dashboard.

2. Reload NGINX for the configuration to take effect.

```
sudo nginx -s reload
```

3. Test whether the reverse proxy is effective: access the `/dashboard/` address on the `8033` port of the machine where NGINX is located (such as `http://example.com:8033/dashboard/`) to access TiDB Dashboard.

Customize path prefix

TiDB Dashboard provides services by default in the `/dashboard/` path, such as `http://example.com:8033/dashboard/`, which is the case even for reverse proxies. To configure the reverse proxy to provide the TiDB Dashboard service with a non-default path, such as `http://example.com:8033/foo/` or `http://example.com:8033/`, take the following steps.

Step 1: Modify PD configuration to specify the path prefix of TiDB Dashboard service

Modify the `public-path-prefix` configuration item in the `[dashboard]` category of the PD configuration to specify the path prefix of the TiDB Dashboard service. After this item is modified, restart the PD instance for the modification to take effect.

For example, if the cluster is deployed using TiUP and you want the service to run on `http://example.com:8033/foo/`, you can specify the following configuration:

```
server_configs:
  pd:
    dashboard.public-path-prefix: /foo
```

Modify configuration when deploying a new cluster using TiUP

If you are deploying a new cluster, you can add the configuration above to the `topology.yaml` TiUP topology file and deploy the cluster. For specific instruction, see TiUP deployment document.

Modify configuration of a deployed cluster using TiUP

For a deployed cluster:

1. Open the configuration file of the cluster in the edit mode (replace `CLUSTER_NAME` with the cluster name).

   ```
   tiup cluster edit-config CLUSTER_NAME
   ```

2. Modify or add configuration items under the `pd` configuration of `server_configs`. If no `server_configs` exists, add it at the top level:

   ```
   monitored:
     ...
   server_configs:
     tidb: ...
     tikv: ...
     pd:
       dashboard.public-path-prefix: /foo
     ...
   ```

The configuration file after the modification is similar to the following file:

```
server_configs:
  pd:
    dashboard.public-path-prefix: /foo
  global:
    user: tidb
    ...
```

Or

```
monitored:
  ...
server_configs:
  tidb: ...
  tikv: ...
  pd:
    dashboard.public-path-prefix: /foo
```

3. Perform a rolling restart to all PD instances for the modified configuration to take effect (replace `CLUSTER_NAME` with your cluster name):

```
shell   tiup cluster reload CLUSTER_NAME -R pd
```

See Common TiUP Operations - Modify the configuration for details.

If you want that the TiDB Dashboard service is run in the root path (such as `http://example.com:8033/`), use the following configuration:

```
server_configs:
  pd:
    dashboard.public-path-prefix: /
```

Warning:

After the modified and customized path prefix takes effect, you cannot directly access TiDB Dashboard. You can only access TiDB Dashboard through a reverse proxy that matches the path prefix.

Step 2: Modify the reverse proxy configuration

Use HAProxy

Taking `http://example.com:8033/foo/` as an example, the corresponding HAProxy configuration is as follows:

```
frontend tidb_dashboard_front
  bind *:8033
  use_backend tidb_dashboard_back if { path /foo } or { path_beg /foo/ }

backend tidb_dashboard_back
  mode http
  http-request set-path %[path,regsub(^/foo/?,/dashboard/)]
  server tidb_dashboard 192.168.0.123:2379
```

Replace `192.168.0.123:2379` with IP and port of the actual address of the TiDB Dashboard obtained in Step 1.

> **Warning:**
>
> You must retain the `if` part in the `use_backend` directive to ensure that services **only in this path** are behind reverse proxy; otherwise, security risks might be introduced. See Secure TiDB Dashboard.

If you want that the TiDB Dashboard service is run in the root path (such as `http://example.com:8033/`), use the following configuration:

```
frontend tidb_dashboard_front
  bind *:8033
  use_backend tidb_dashboard_back
backend tidb_dashboard_back
  mode http
  http-request set-path /dashboard%[path]
  server tidb_dashboard 192.168.0.123:2379
```

Modify the configuration and restart HAProxy for the modified configuration to take effect.

Use NGINX

Taking `http://example.com:8033/foo/` as an example, the corresponding NGINX configuration is as follows:

```
server {
  listen 8033;
  location /foo/ {
    proxy_pass http://192.168.0.123:2379/dashboard/;
  }
}
```

Replace `http://192.168.0.123:2379/dashboard/` with the actual address of the TiDB Dashboard obtained in Step 1.

> **Warning:**
>
> You must retain the `/dashboard/` path in the `proxy_pass` directive to ensure that services **only in this path** are behind reverse proxy; otherwise, security risks might be introduced. See Secure TiDB Dashboard.

If you want that the TiDB Dashboard service is run in the root path (such as `http://example.com:8033/`), use the following configuration:

```
server {
  listen 8033;
  location / {
    proxy_pass http://192.168.0.123:2379/dashboard/;
  }
}
```

Modify the configuration and restart NGINX for the modified configuration to take effect.

```
sudo nginx -s reload
```

What's next

To learn how to enhance the security of TiDB Dashboard, such as configuring a firewall, see Secure TiDB Dashboard.

14.11.1.2.3 TiDB Dashboard User Management

TiDB Dashboard uses the same user privilege system and sign-in authentication as TiDB. You can control and manage TiDB SQL users to limit their access to TiDB Dashboard. This document describes the least privileges required for TiDB SQL users to access TiDB Dashboard and exemplifies how to create a least-privileged SQL user and how to authorize via RBAC.

For details about how to control and manage TiDB SQL users, see TiDB User Account Management.

Required privileges

- To access TiDB Dashboard when Security Enhanced Mode (SEM) is not enabled on the connected TiDB server, the SQL user should have **all** the following privileges:
    - PROCESS

    - SHOW DATABASES
    - CONFIG
    - DASHBOARD_CLIENT

- To access TiDB Dashboard when Security Enhanced Mode (SEM) is enabled on the connected TiDB server, the SQL user should have **all** the following privileges:

    - PROCESS
    - SHOW DATABASES
    - CONFIG
    - DASHBOARD_CLIENT
    - RESTRICTED_TABLES_ADMIN
    - RESTRICTED_STATUS_ADMIN
    - RESTRICTED_VARIABLES_ADMIN

- To modify the configurations on the interface after signing in to TiDB Dashboard, the SQL user must also have the following privilege:

    - SYSTEM_VARIABLES_ADMIN

> **Note:**
>
> Users with high privileges such as `ALL PRIVILEGES` or `SUPER` can sign in to TiDB Dashboard as well. Therefore, to comply with the least privilege principle, it is highly recommended that you create users with the required privileges only to prevent unintended operations. See Privilege Management for more information on these privileges.

If an SQL user does not meet the preceding privilege requirements, the user fails to sign in to TiDB Dashboard, as shown below.

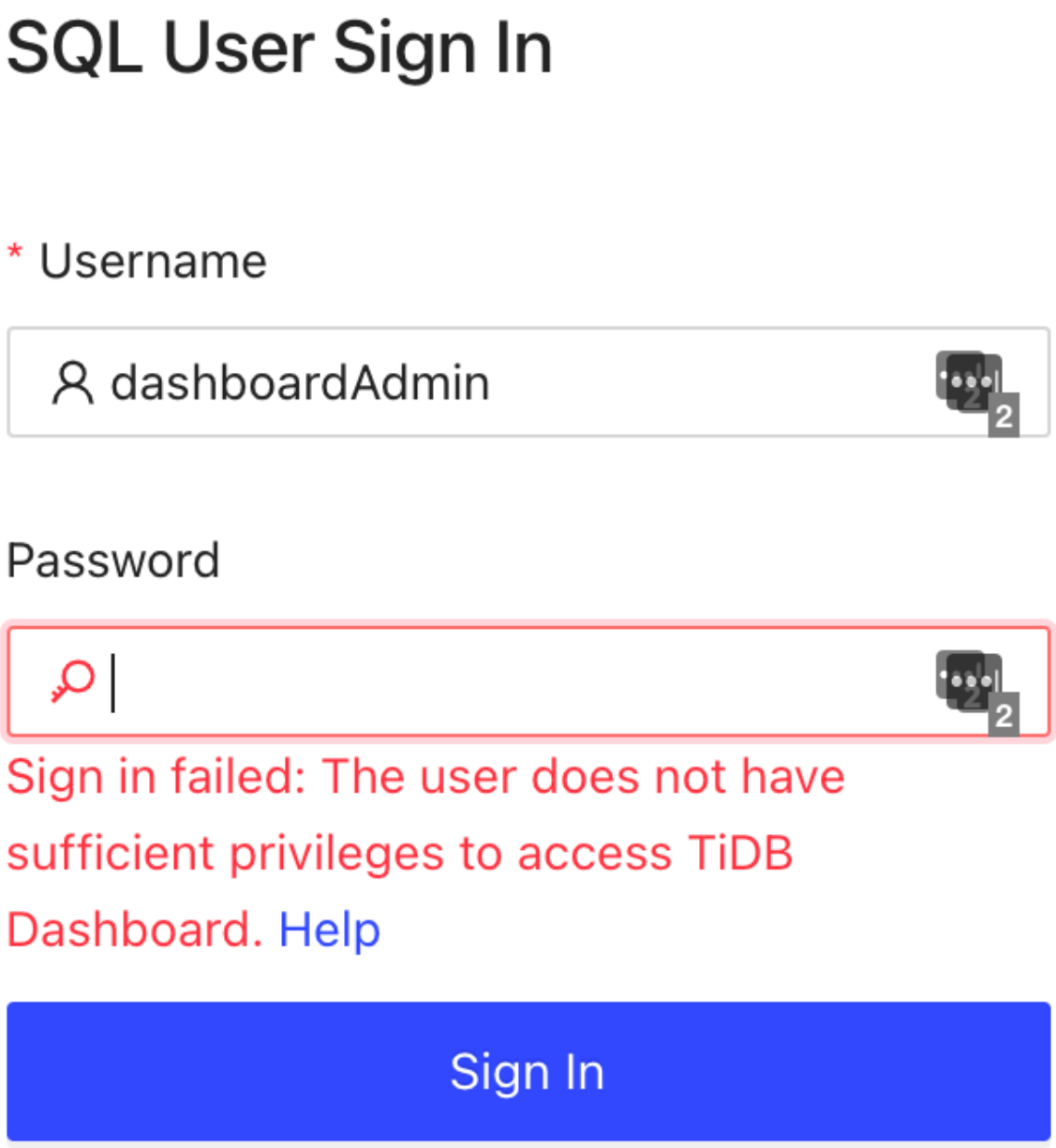


Figure 494: insufficient-privileges

Example: Create a least-privileged SQL user to access TiDB Dashboard

- When Security Enhanced Mode (SEM) is not enabled on the connected TiDB server, to create an SQL user `dashboardAdmin` that can sign in to TiDB Dashboard, execute the following SQL statements:

```
CREATE USER 'dashboardAdmin'@'%' IDENTIFIED BY '<YOUR_PASSWORD>';
```

```
GRANT PROCESS, CONFIG ON *.* TO 'dashboardAdmin'@'%';
GRANT SHOW DATABASES ON *.* TO 'dashboardAdmin'@'%';
GRANT DASHBOARD_CLIENT ON *.* TO 'dashboardAdmin'@'%';

-- To modify the configuration items on the interface after signing
   ↪ in to TiDB Dashboard, the user-defined SQL user must be granted
   ↪  with the following privilege.
GRANT SYSTEM_VARIABLES_ADMIN ON *.* TO 'dashboardAdmin'@'%';
```

- When Security Enhanced Mode (SEM) is enabled on the connected TiDB server, disable SEM first and execute the following SQL statements to create an SQL user `dashboardAdmin` that can sign in to TiDB Dashboard. After creating the user, enable SEM again:

```
CREATE USER 'dashboardAdmin'@'%' IDENTIFIED BY '<YOUR_PASSWORD>';
GRANT PROCESS, CONFIG ON *.* TO 'dashboardAdmin'@'%';
GRANT SHOW DATABASES ON *.* TO 'dashboardAdmin'@'%';
GRANT DASHBOARD_CLIENT ON *.* TO 'dashboardAdmin'@'%';
GRANT RESTRICTED_STATUS_ADMIN ON *.* TO 'dashboardAdmin'@'%';
GRANT RESTRICTED_TABLES_ADMIN ON *.* TO 'dashboardAdmin'@'%';
GRANT RESTRICTED_VARIABLES_ADMIN ON *.* TO 'dashboardAdmin'@'%';

-- To modify the configuration items on the interface after signing
   ↪ in to TiDB Dashboard, the user-defined SQL user must be granted
   ↪  with the following privilege.
GRANT SYSTEM_VARIABLES_ADMIN ON *.* TO 'dashboardAdmin'@'%';
```

Example: Authorize SQL user to access TiDB Dashboard via RBAC

The following example demonstrates how to create a role and a user to access TiDB Dashboard through the role-based access control (RBAC) mechanism.

1. Create a `dashboard_access` role that meets the privilege requirements of TiDB Dashboard:

```
CREATE ROLE 'dashboard_access';
GRANT PROCESS, CONFIG ON *.* TO 'dashboard_access'@'%';
GRANT SHOW DATABASES ON *.* TO 'dashboard_access'@'%';
GRANT DASHBOARD_CLIENT ON *.* TO 'dashboard_access'@'%';
GRANT SYSTEM_VARIABLES_ADMIN ON *.* TO 'dashboard_access'@'%';
```

2. Grant the `dashboard_access` role to other users and set `dashboard_access` as the default role:

```
CREATE USER 'dashboardAdmin'@'%' IDENTIFIED BY '<YOUR_PASSWORD>';
GRANT 'dashboard_access' TO 'dashboardAdmin'@'%';
-- You need to set dashboard_access as the default role to the user
SET DEFAULT ROLE dashboard_access to 'dashboardAdmin'@'%';
```

After the above steps, you can use the `dashboardAdmin` user to sign in to TiDB Dashboard.

Sign in to TiDB Dashboard

After creating an SQL user that meets the privilege requirements of TiDB Dashboard, you can use this user to Sign in to TiDB Dashboard.

14.11.1.2.4 Secure TiDB Dashboard

Although you need to sign into TiDB Dashboard before accessing it, TiDB Dashboard is designed to be accessed by trusted user entities by default. When you want to provide TiDB Dashboard to external network users or untrusted users for access, take the following measures to avoid security vulnerabilities.

Enhance security of TiDB users

Set a strong password for the TiDB `root` user

The account system of TiDB Dashboard is consistent with that of TiDB SQL users. By default, TiDB's `root` user has no password, so accessing TiDB Dashboard does not require password authentication, which will give the malicious visitor high privileges, including executing privileged SQL statements.

It is recommended that you set a strong password for the TiDB `root` user. See TiDB User Account Management for details. Alternatively, you can disable the TiDB `root` user.

Create a least-privileged user for TiDB Dashboard

The account system of TiDB Dashboard is consistent with that of TiDB SQL. Users accessing TiDB Dashboard are authenticated and authorized based on TiDB SQL user's privileges. Therefore, TiDB Dashboard requires limited privileges, or merely the read-only privilege. You can configure users to access TiDB Dashboard based on the principle of least privilege, thus avoiding access of high-privileged users.

It is recommended that you create a least-privileged SQL user to access and sign in to TiDB Dashboard. This avoids access of high-privileged users and improves security. See TiDB Dashboard User Management for details.

Use a firewall to block untrusted access

TiDB Dashboard provides services through the PD client port, which defaults to http://IP:2379/dashboard/. Although TiDB Dashboard requires identity authentication, other privileged interfaces (such as http://IP:2379/pd/api/v1/members) in PD carried on the PD

client port do not require identity authentication and can perform privileged operations. Therefore, exposing the PD client port directly to the external network is extremely risky.

It is recommended that you take the following measures:

- Use a firewall to prohibit a component from accessing **any** client port of the PD component via the external network or untrusted network.

> **Note:**
> TiDB, TiKV, and other components need to communicate with the PD component through the PD client port, so do not block access to the internal network between components. Otherwise, the cluster will become unavailable.

- See Use TiDB Dashboard behind a Reverse Proxy to learn how to configure the reverse proxy to safely provide the TiDB Dashboard service on another port to the external network.

How to open access to TiDB Dashboard port when deploying multiple PD instances

> **Warning:**
> This section describes an unsafe access solution, which is for the test environment only. **DO NOT** use this solution in the production environment.

In the test environment, you might need to configure the firewall to open the TiDB Dashboard port for external access.

When multiple PD instances are deployed, only one of the PD instances actually runs TiDB Dashboard, and browser redirection occurs when you access other PD instances. Therefore, you need to ensure that the firewall is configured with the correct IP address. For details of this mechanism, see Deployment with multiple PD instances.

When using the TiUP deployment tool, you can view the address of the PD instance that actually runs TiDB Dashboard by running the following command (replace `CLUSTER_NAME` with the cluster name):

```
tiup cluster display CLUSTER_NAME --dashboard
```

The output is the actual TiDB Dashboard address.

Note:

This feature is available only in the later version of the `tiup cluster` deployment tool (v1.0.3 or later).

Upgrade TiUP Cluster

```
tiup update --self
tiup update cluster --force
```

The following is a sample output:

```
http://192.168.0.123:2379/dashboard/
```

In this example, the firewall needs to be configured with inbound access for the `2379` port of the `192.168.0.123` open IP, and the TiDB Dashboard is accessed via http://192.168.0.123:2379/dashboard/.

Reverse proxy only for TiDB Dashboard

As mentioned in Use a firewall to block untrusted access, the services provided under the PD client port include not only TiDB Dashboard (located at http://IP:2379/dashboard/), but also other privileged interfaces in PD (such as http://IP:2379/pd/api/v1/members). Therefore, when using a reverse proxy to provide TiDB Dashboard to the external network, ensure that the services **ONLY** with the `/dashboard` prefix are provided (**NOT** all services under the port) to avoid that the external network can access the privileged interface in PD through the reverse proxy.

It is recommended that you see Use TiDB Dashboard behind a Reverse Proxy to learn a safe and recommended reverse proxy configuration.

Enable TLS for reverse proxy

To further enhance the security of the transport layer, you can enable TLS for reverse proxy, and even introduce mTLS to authenticate user certificates.

See Configuring HTTPS servers and HAProxy SSL Termination for more details.

Other recommended safety measures

- Enable TLS Authentication and Encrypt the Stored Data
- Enable TLS Between TiDB Clients and Servers

14.11.1.3 Access TiDB Dashboard

To access TiDB Dashboard, visit http://127.0.0.1:2379/dashboard via your browser. Replace `127.0.0.1:2379` with the actual PD instance address and port.

If multiple PD instances are deployed in your cluster and you can directly access **every** PD instance and port, you can simply replace `127.0.0.1:2379` in the http://127.0.0.1:2379/dashboard/ address with **any** PD instance address and port.

> **Note:**
>
> If a firewall or reverse proxy is configured and you cannot directly access every PD instance, you might not be able to access TiDB Dashboard. Usually, this is because the firewall or reverse proxy is not correctly configured. See Use TiDB Dashboard behind Reverse Proxy and Secure TiDB Dashboard to learn correctly configure the firewall or reverse proxy when multiple PD instances are deployed.

14.11.1.3.1 Browser compatibility

You can use TiDB Dashboard in the following common desktop browsers of a relatively newer version:

- Chrome >= 77
- Firefox >= 68
- Edge >= 17

> **Note:**
>
> If you use the browsers above of earlier versions or other browsers to access TiDB Dashboard, some functions might not work properly.

14.11.1.3.2 Sign in

After accessing TiDB Dashboard, you will be directed to the user login interface, as shown in the image below.

- You can sign in to TiDB Dashboard using the TiDB `root` account.
- If you have created a User-defined SQL User, you can sign in using this account and the corresponding password.

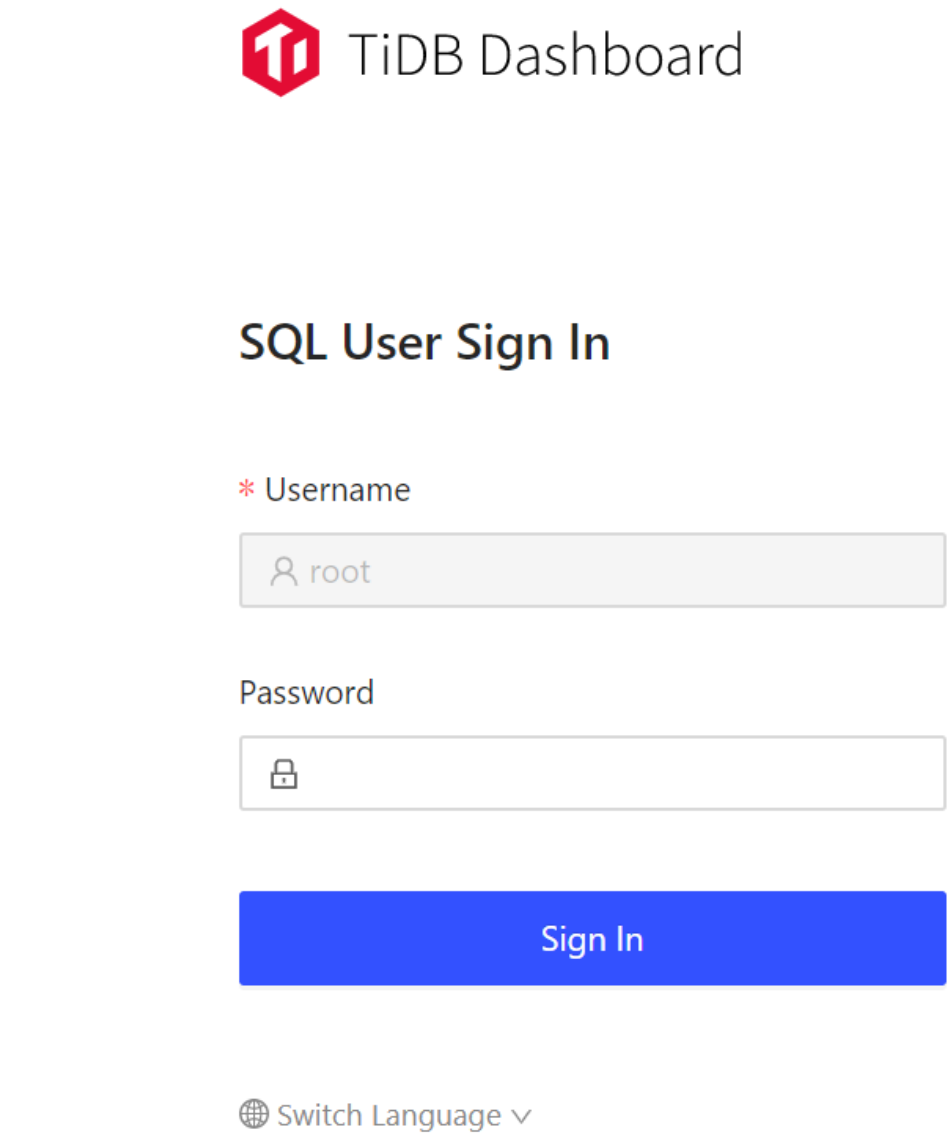


Figure 495: Login interface

If one of the following situations exists, the login might fail:

- TiDB `root` user does not exist.
- PD is not started or cannot be accessed.
- TiDB is not started or cannot be accessed.
- Wrong `root` password.

Once you have signed in, the session remains valid within the next 24 hours. To learn how to sign out, refer to the Logout section.

14.11.1.3.3 Switch language

The following languages are supported in TiDB Dashboard:

- English
- Chinese (simplified)

In the **SQL User Sign In** page, you can click the **Switch Language** drop-down list to switch the interface language.

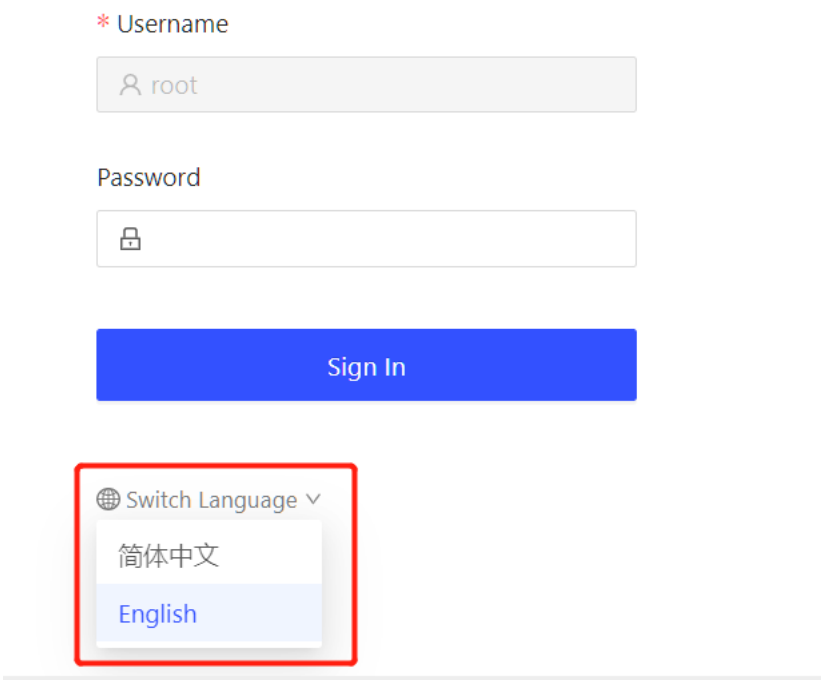


Figure 496: Switch language

14.11.1.3.4 Logout

Once you have logged in, click the login user name in the left navigation bar to switch to the user page. Click the **Logout** button on the user page to log out the current user. After logging out, you need to re-enter your username and password.

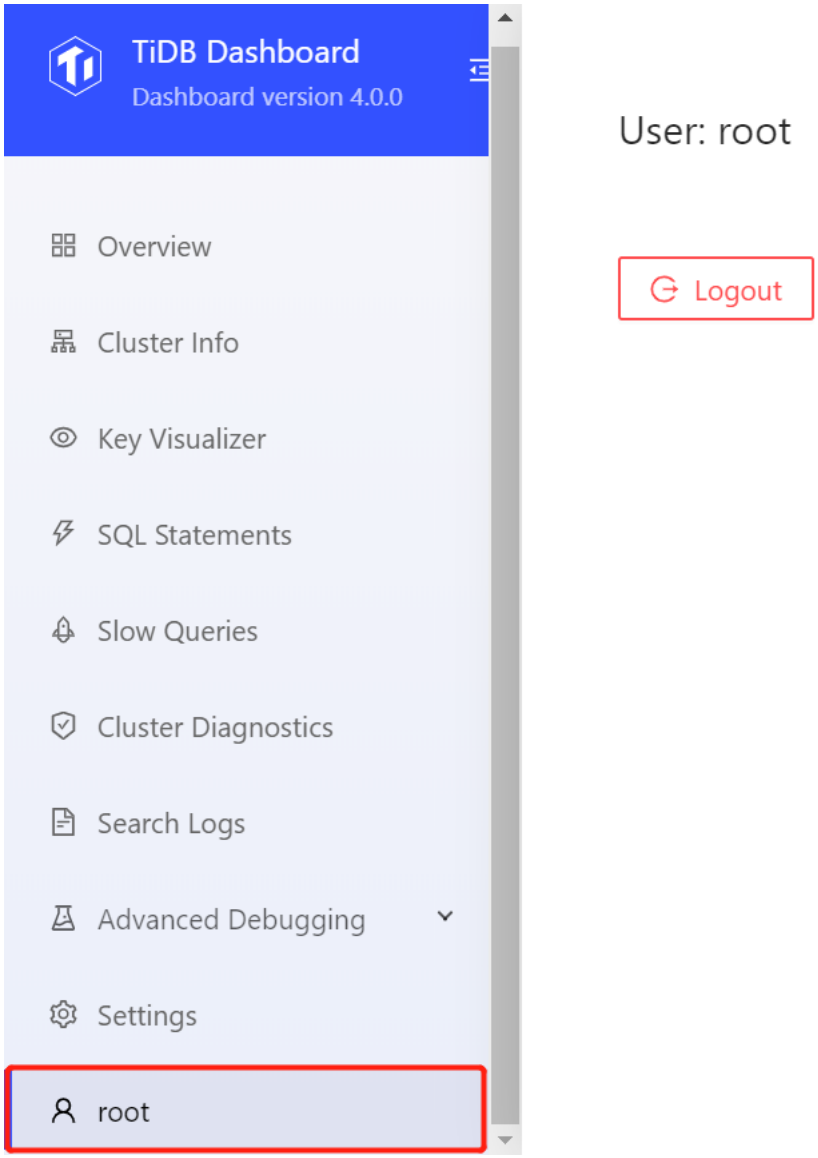


Figure 497: Logout

14.11.1.4 Overview Page

This page shows the overview of the entire TiDB cluster, including the following information:

- Queries per second (QPS) of the entire cluster.
- The query latency of the entire cluster.
- The SQL statements that have accumulated the longest execution time over the recent period.
- The slow queries whose execution time over the recent period exceeds the threshold.
- The node count and status of each instance.
- Monitor and alert messages.

14.11.1.4.1 Access the page

After logging into TiDB Dashboard, the overview page is entered by default, or you can click **Overview** on the left navigation menu to enter this page:



Figure 498: Enter overview page

14.11.1.4.2 QPS

This area shows the number of successful and failed queries per second for the entire cluster over the recent hour:

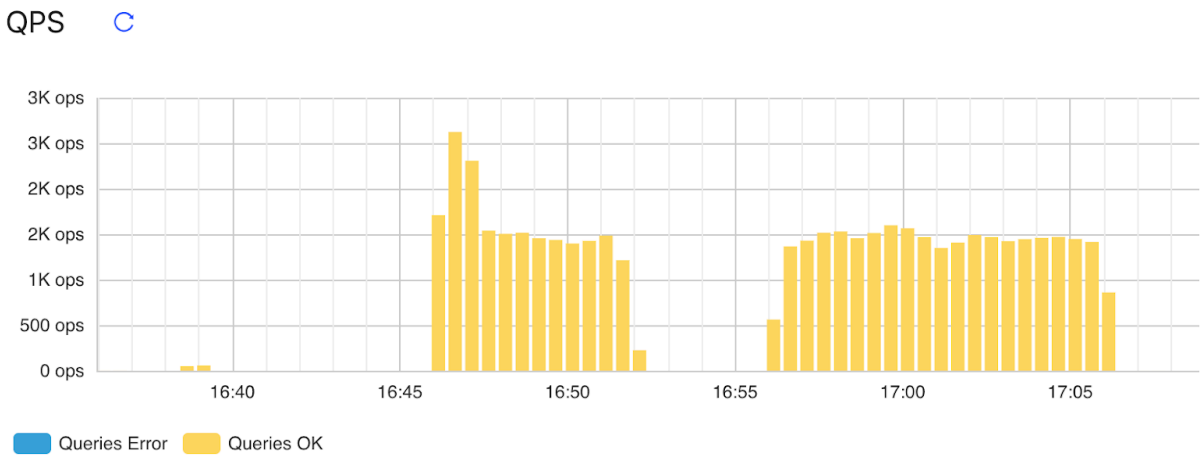


Figure 499: QPS

> **Note:**
>
> This feature is available only in the cluster where the Prometheus monitoring component is deployed. If Prometheus is not deployed, an error will be displayed.

14.11.1.4.3 Latency

This area shows the latency of 99.9%, 99%, and 90% of queries in the entire cluster over the recent one hour:

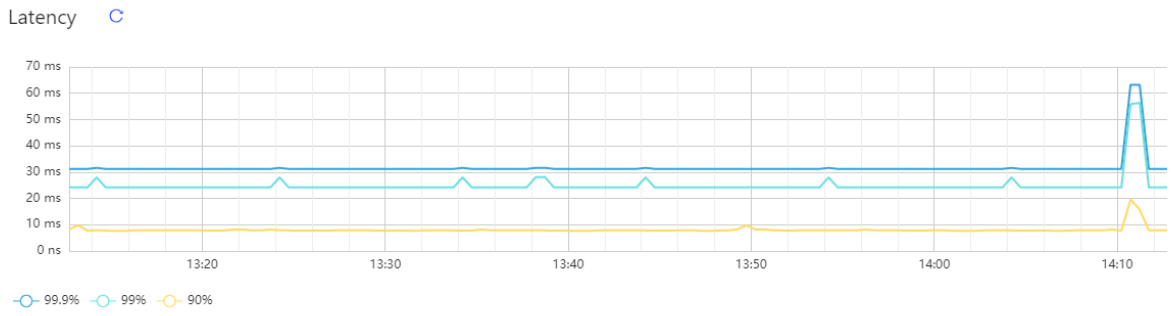


Figure 500: Latency

> **Note:**
>
> This feature is available only on the cluster where the Prometheus monitoring component is deployed. If Prometheus is not deployed, an error will be displayed.

14.11.1.4.4 Top SQL statements

This area shows the ten types of SQL statements that have accumulated the longest execution time in the entire cluster over the recent period. SQL statements with different query parameters but of the same structure are classified into the same SQL type and displayed in the same row:



Figure 501: Top SQL

The information shown in this area is consistent with the more detailed SQL Statements Page. You can click the **Top SQL Statements** heading to view the complete list. For details of the columns in this table, see SQL Statements Page.

> **Note:**
>
> This feature is available only on the cluster where SQL Statements feature is enabled.

14.11.1.4.5 Recent slow queries

By default, this area shows the latest 10 slow queries in the entire cluster over the recent 30 minutes:



Figure 502: Recent slow queries

By default, the SQL query that is executed longer than 300 milliseconds is counted as a slow query and displayed on the table. You can change this threshold by modifying the tidb_slow_log_threshold variable or the instance.tidb_slow_log_threshold TiDB parameter.

The content displayed in this area is consistent with the more detailed Slow Queries Page. You can click the **Recent Slow Queries** title to view the complete list. For details of the columns in this table, see this Slow Queries Page.

> **Note:**
>
> This feature is available only in the cluster with slow query logs enabled. By default, slow query logs are enabled in the cluster deployed using TiUP.

14.11.1.4.6 Instances

This area summarizes the total number of instances and abnormal instances of TiDB, TiKV, PD, and TiFlash in the entire cluster:

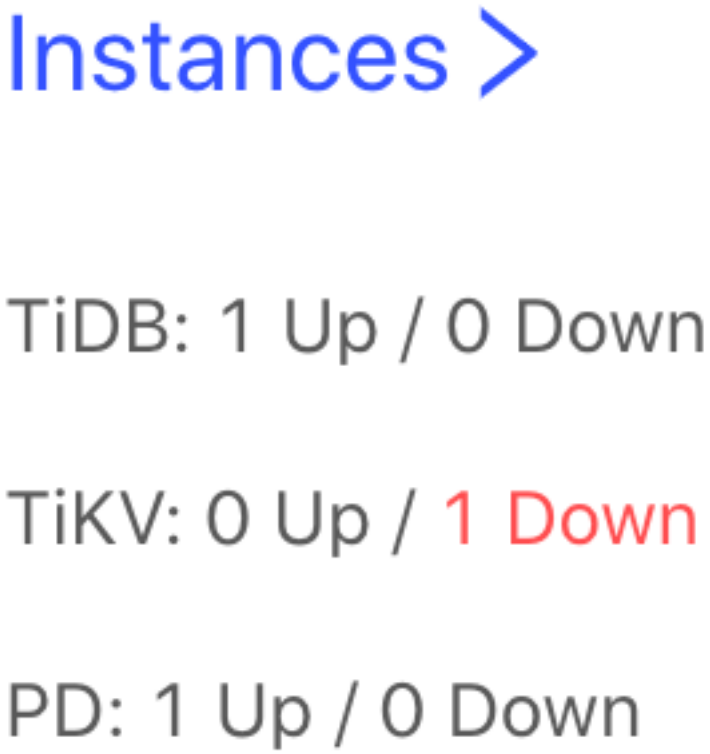


Figure 503: Instances

The statuses in the image above are described as follows:

- Up: The instance is running properly (including the offline storage instance).

- Down: The instance is running abnormally, such as network disconnection and process crash.

Click the **Instance** title to enter the Cluster Info Page that shows the detailed running status of each instance.

14.11.1.4.7 Monitor and alert

This area provides links for you to view detailed monitor and alert:

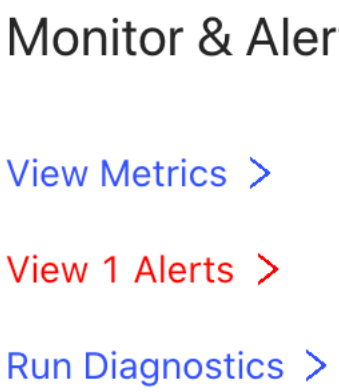


Figure 504: Monitor and alert

- **View Metrics**: Click this link to jump to the Grafana dashboard where you can view detailed monitoring information of the cluster. For details of each monitoring metric in the Grafana dashboard, see monitoring metrics.
- **View Alerts**: Click this link to jump to the AlertManager page where you can view detailed alert information of the cluster. If alerts exist in the cluster, the number of alerts is directly shown in the link text.
- **Run Diagnostics**: Click this link to jump to the more detailed cluster diagnostics page.

> **Note:**
>
> The **View Metrics** link is available only in the cluster where the Grafana node is deployed. The **View Alerts** link is available only in the cluster where the AlertManager node is deployed.

14.11.1.5 TiDB Dashboard Cluster Information Page

On the cluster information page, you can view the running status of TiDB, TiKV, PD, TiFlash components in the entire cluster and the running status of the host on which these components are located.

14.11.1.5.1 Access the page

You can use one of the following two methods to access the cluster information page:

- After logging into TiDB Dashboard, click **Cluster Info** on the left navigation menu:

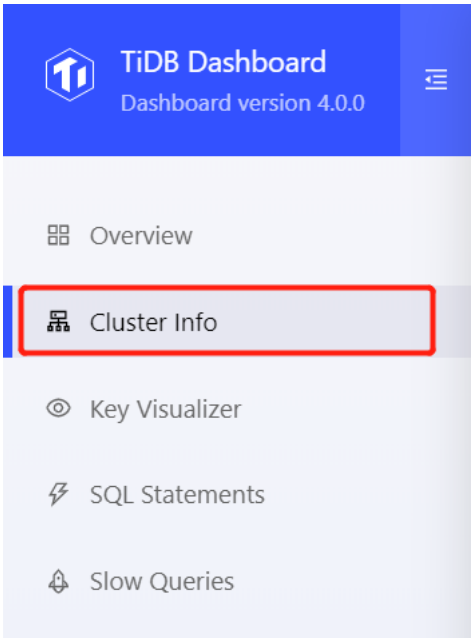


Figure 505: Access cluster information page

- Visit http://127.0.0.1:2379/dashboard/#/cluster_info/instance in your browser. Replace `127.0.0.1:2379` with the actual PD instance address and port.

14.11.1.5.2 Instance list

Click **Instances** to view the list of instances:

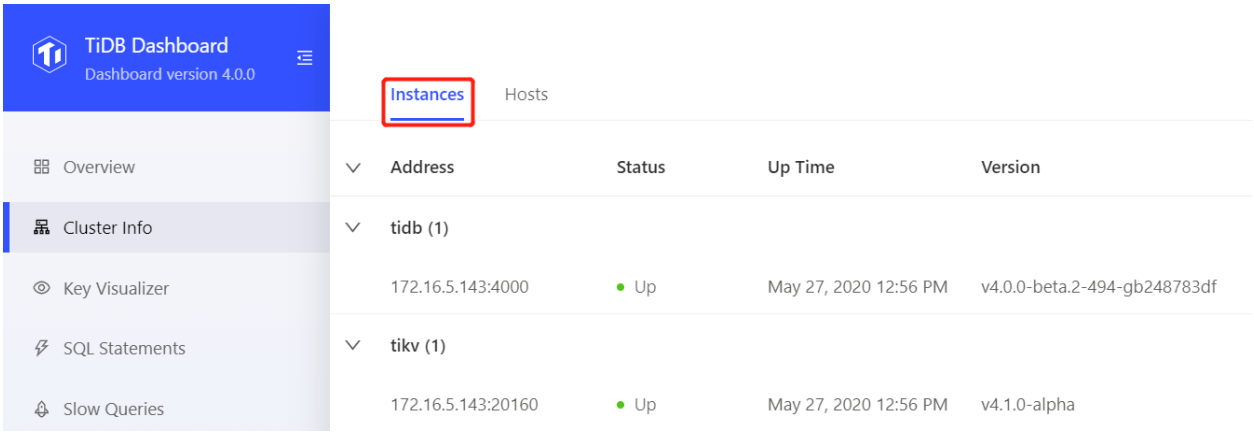


Figure 506: Instance list

This instance list shows the overview information of all instances of TiDB, TiKV, PD, and TiFlash components in the cluster.

The list includes the following information:

- Address: The instance address.
- Status: The running status of the instance.
- Up Time: The start time of the instance.
- Version: The instance version number.
- Deployment directory: The directory in which the instance binary file is located.
- Git Hash: The Git Hash value corresponding to the instance binary file.

Instance status

An instance can run in one of the following statuses:

- Up: The instance is running normally.
- Down or Unreachable: The instance is not running or a network problem exists on the corresponding host.
- Tombstone: The data on the instance has been completely migrated out and the scaling-in is complete. This status exists only on TiKV or TiFlash instances.
- Leaving: The data on the instance is being migrated out and the scaling-in is in process. This status exists only on TiKV or TiFlash instances.
- Unknown: The running state of the instance is unknown.

Note:

- `Leaving` in TiDB Dashboard, `Offline` returned by PD API, and `Pending Offline` in TiUP indicate the same status.
- Some columns in the table can be displayed only when the instance is up.

Instance status derives from the PD scheduling information. For more details, see Information collection.

14.11.1.5.3 Host list

Click **Hosts** to view the list of hosts:

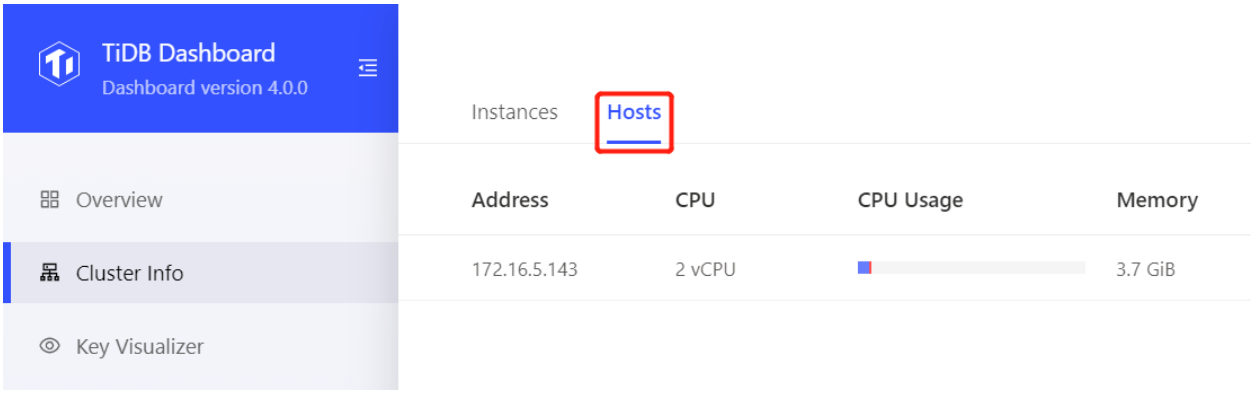


Figure 507: Host list

This host list shows the running status of hosts that correspond to all instances of TiDB, TiKV, PD, and TiFlash components in the cluster.

The list includes the following information:

- Address: The Host IP address.
- CPU: The number of logical cores of the host CPU.
- CPU Usage: The user-mode and kernel-mode CPU usage in the current 1 second.
- Memory: The total physical memory size of the host.
- Memory Usage: The current memory usage of the host.
- Disk: The file system of the disk on the host on which the instance is running and the mounting path of this disk.
- Disk Usage: The space usage of the disk on the host on which the instance is running.

Note:

The host list information is provided by each instance process, so when all instances on the host are down, the host information is not displayed.

14.11.1.6 TiDB Dashboard Top SQL Page

With Top SQL, you can monitor and visually explore the CPU overhead of each SQL statement in your database in real-time, which helps you optimize and resolve database performance issues. Top SQL continuously collects and stores CPU load data summarized by SQL statements at any seconds from all TiDB and TiKV instances. The collected data can be stored for up to 30 days. Top SQL presents you with visual charts and tables to quickly pinpoint which SQL statements are contributing the high CPU load of a TiDB or TiKV instance over a certain period of time.

Top SQL provides the following features:

- Visualize the top 5 types of SQL statements with the highest CPU overhead through charts and tables.
- Display detailed execution information such as queries per second, average latency, and query plan.
- Collect all SQL statements that are executed, including those that are still running.
- Allow viewing data of a specific TiDB and TiKV instance.

14.11.1.6.1 Recommended scenarios

Top SQL is suitable for analyzing performance issues. The following are some typical Top SQL scenarios:

- You discovered that an individual TiKV instance in the cluster has a very high CPU usage through the Grafana charts. You want to know which SQL statements cause the CPU hotspots so that you can optimize them and better leverage all of your distributed resources.
- You discovered that the cluster has a very high CPU usage overall and queries are slow. You want to quickly figure out which SQL statements are currently consuming the most CPU resources so that you can optimize them.
- The CPU usage of the cluster has drastically changed and you want to know the major cause.
- Analyze the most resource-intensive SQL statements in the cluster and optimize them to reduce hardware costs.

Top SQL cannot be used to pinpoint non-performance issues, such as incorrect data or abnormal crashes.

The Top SQL feature is still in an early stage and is being continuously enhanced. Here are some scenarios that are **not supported** at the moment:

- Analyzing the overhead of SQL statements outside of Top 5 (for example, when multiple business workloads are mixed).
- Analyzing the overhead of Top N SQL statements by various dimensions such as users and databases.
- Analyzing database performance issues that are not caused by high CPU load, such as transaction lock conflicts.

14.11.1.6.2 Access the page

You can access the Top SQL page using either of the following methods:

- After logging into TiDB Dashboard, click **Top SQL** on the left navigation bar.

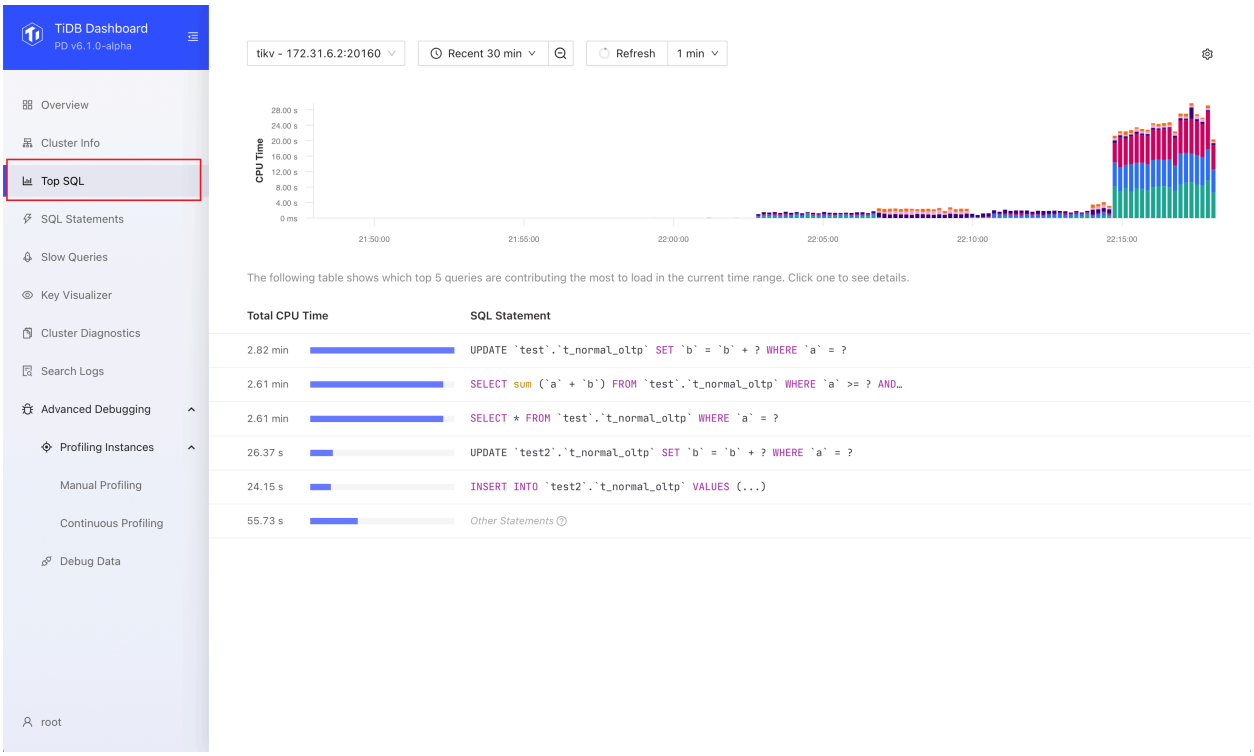


Figure 508: Top SQL

- Visit http://127.0.0.1:2379/dashboard/#/topsql in your browser. Replace `127.0.0.1:2379` ↪ with the actual PD instance address and port.

14.11.1.6.3 Enable Top SQL

> **Note:**
>
> To use Top SQL, your cluster should be deployed or upgraded with a recent version of TiUP (v1.9.0 or above) or TiDB Operator (v1.3.0 or above). If your cluster was upgraded using an earlier version of TiUP or TiDB Operator, see FAQ for instructions.

Top SQL is not enabled by default as it has a slight impact on cluster performance (within 3% on average) when enabled. You can enable Top SQL by the following steps:

1. Visit the Top SQL page.
2. Click **Open Settings**. On the right side of the **Settings** area, switch on **Enable Feature**.
3. Click **Save**.

After enabling the feature, wait up to 1 minute for Top SQL to load the data. Then you can see the CPU load details.

In addition to the UI, you can also enable the Top SQL feature by setting the TiDB system variable `tidb_enable_top_sql`:

```
SET GLOBAL tidb_enable_top_sql = 1;
```

14.11.1.6.4 Use Top SQL

The following are the common steps to use Top SQL.

1. Visit the Top SQL page.
2. Select a particular TiDB or TiKV instance that you want to observe the load.

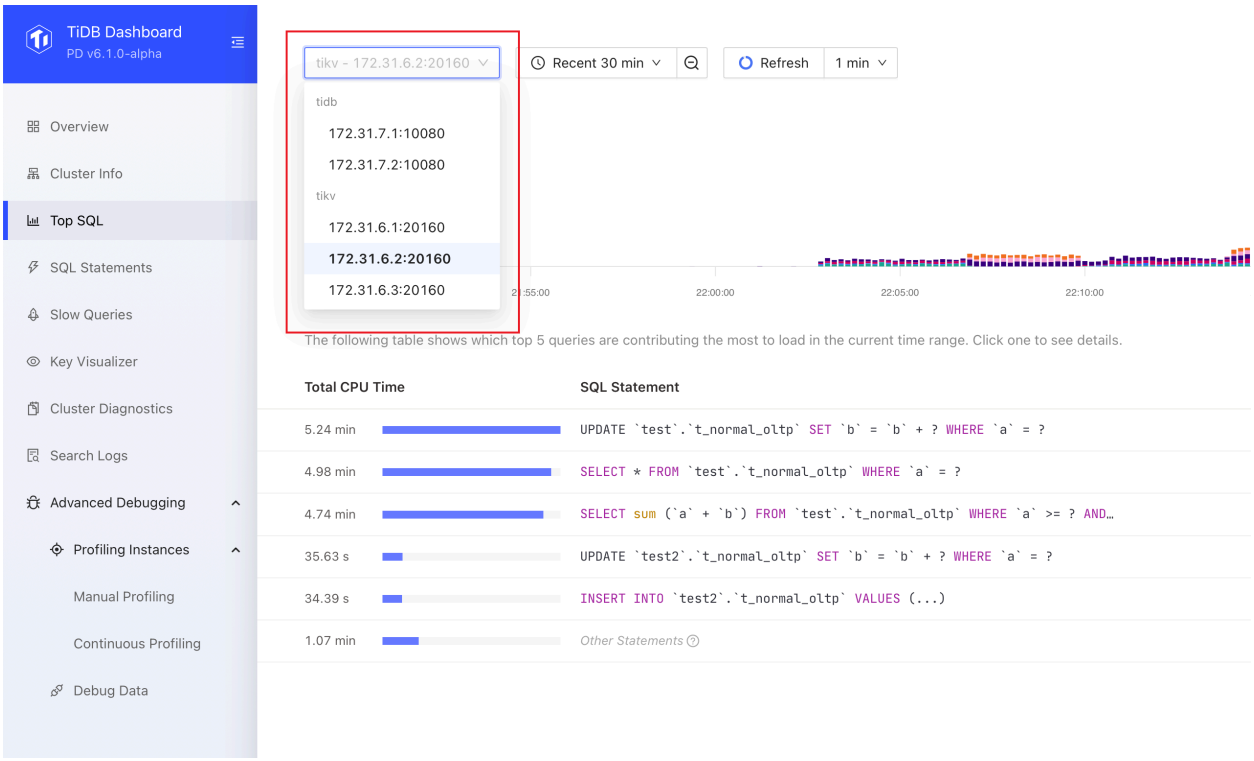


Figure 509: Select Instance

If you are unsure of which TiDB or TiKV instance to observe, you can select an arbitrary instance. Also, when the cluster CPU load is extremely unbalanced, you can first use Grafana charts to determine the specific instance you want to observe.

3. Observe the charts and tables presented by Top SQL.

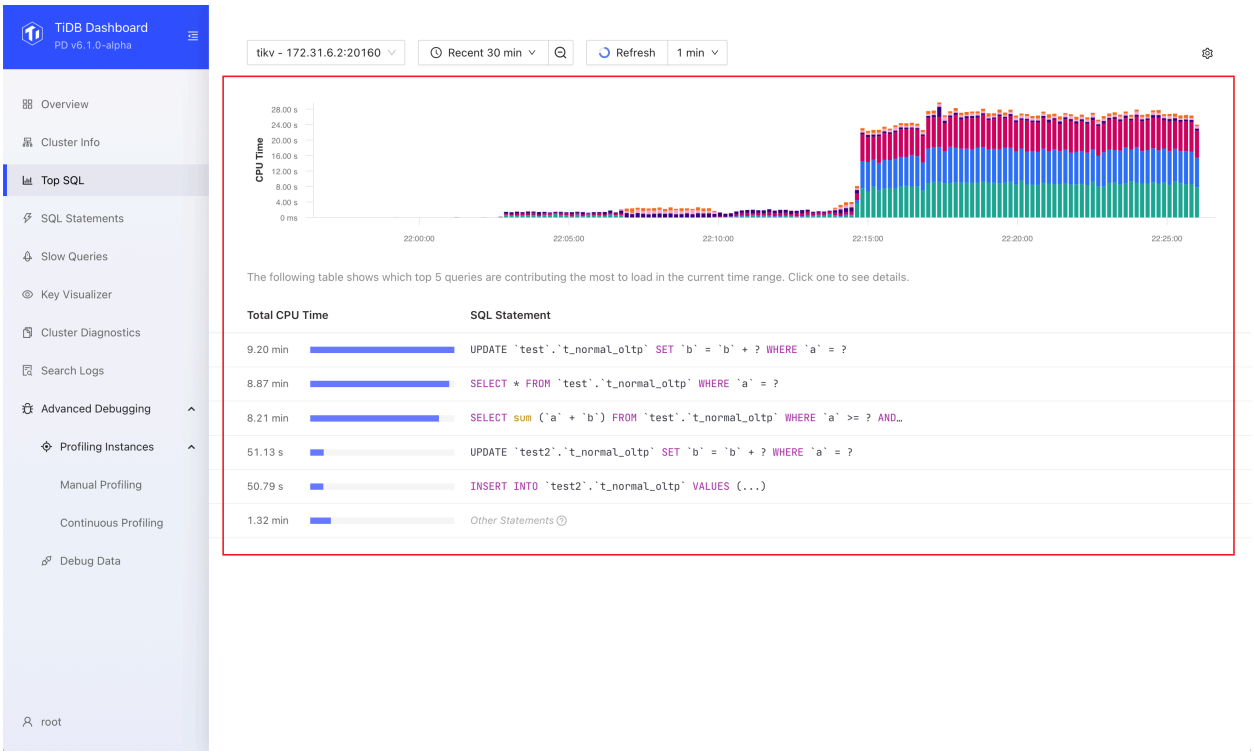


Figure 510: Chart and Table

The size of the bars in the bar chart represents the size of CPU resources consumed by the SQL statement at that moment. Different colors distinguish different types of SQL statements. In most cases, you only need to focus on the SQL statements that have a higher CPU resource overhead in the corresponding time range in the chart.

4. Click a SQL statement in the table to show more information. You can see detailed execution metrics of different plans of that statement, such as Call/sec (average queries per second) and Scan Indexes/sec (average number of index rows scanned per second).

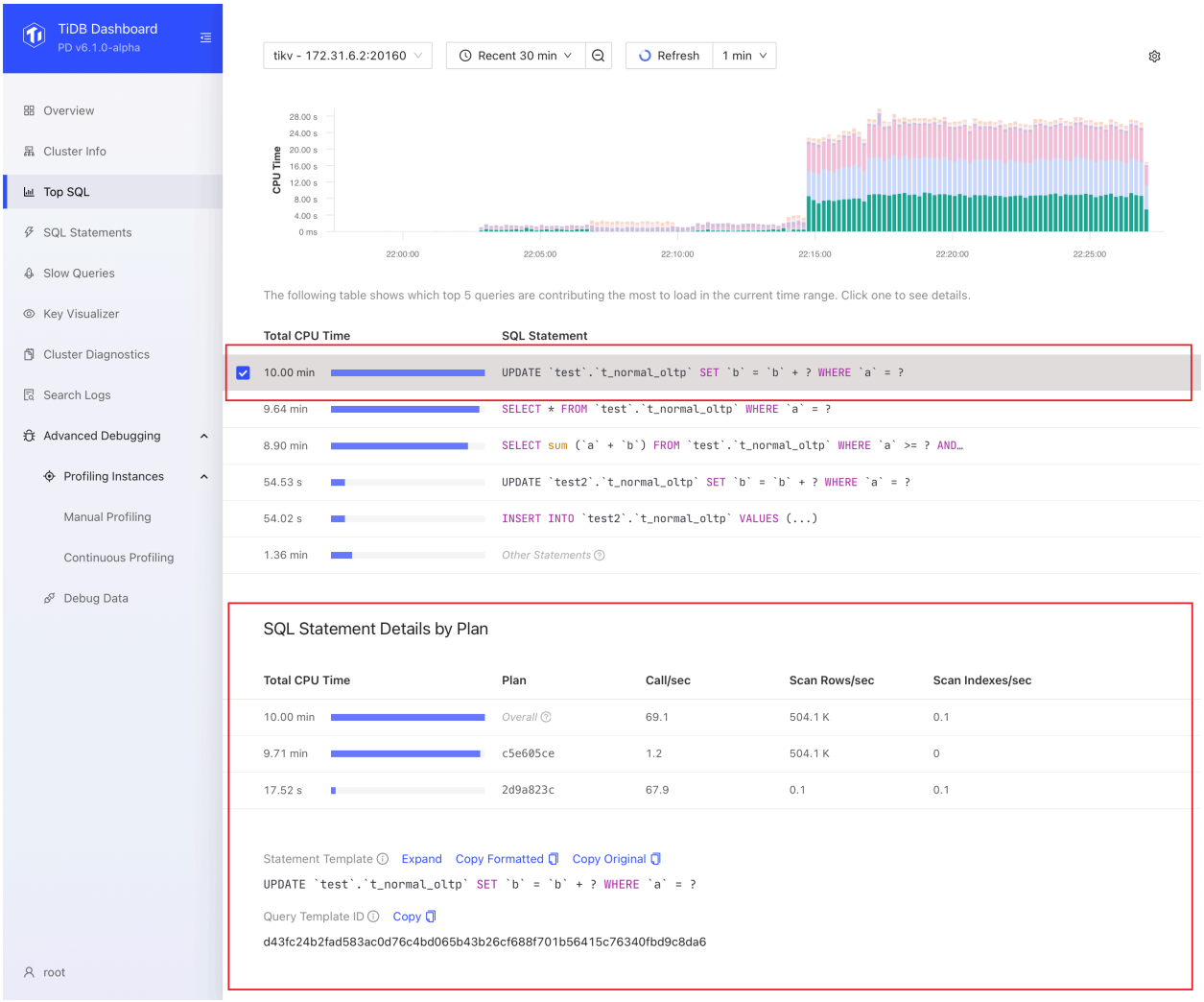


Figure 511: Details

5. Based on these initial clues, you can further explore the SQL Statement or Slow Queries page to find the root cause of high CPU consumption or large data scans of the SQL statement.

Additionally, you can configure Top SQL as follows:

- You can adjust the time range in the time picker or select a time range in the chart to get a more precise and detailed look at the problem. A smaller time range can provide more detailed data, with precision of up to 1 second.



Figure 512: Change time range

- If the chart is out of date, you can click the **Refresh** button or select Auto Refresh options from the **Refresh** drop-down list.

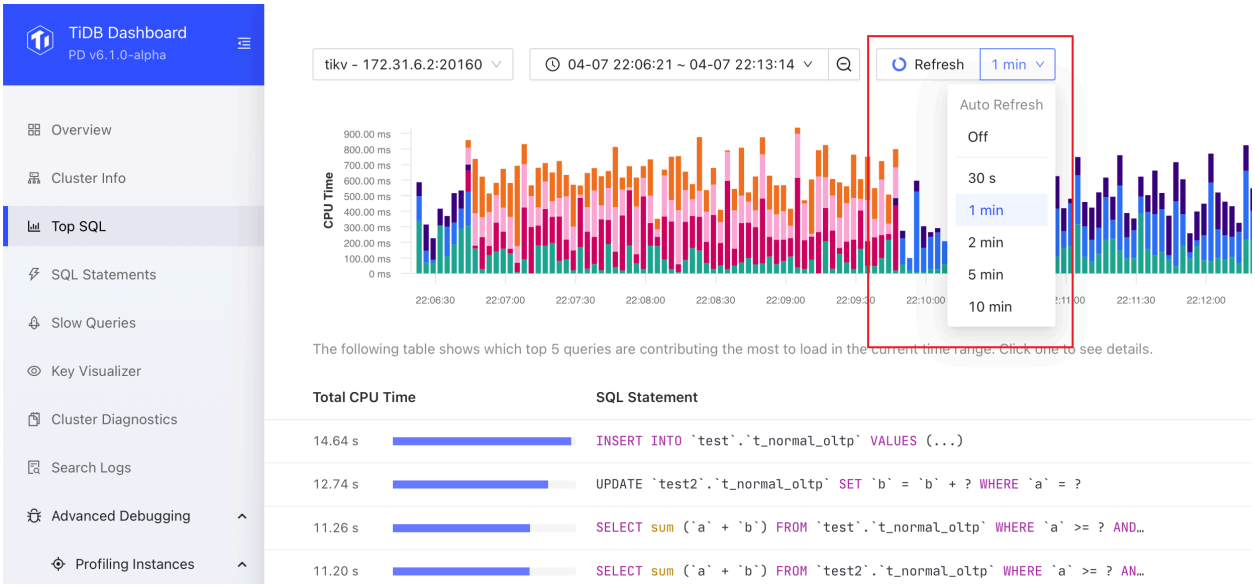


Figure 513: Refresh

14.11.1.6.5 Disable Top SQL

You can disable this feature by following these steps:

1. Visit Top SQL page.
2. Click the gear icon in the upper right corner to open the settings screen and switch off **Enable Feature**.
3. Click **Save**.
4. In the popped-up dialog box, click **Disable**.

In addition to the UI, you can also disable the Top SQL feature by setting the TiDB system variable `tidb_enable_top_sql`:

```
SET GLOBAL tidb_enable_top_sql = 0;
```

14.11.1.6.6 Frequently asked questions

1. Top SQL cannot be enabled and the UI displays "required component NgMonitoring is not started".

See TiDB Dashboard FAQ.

2. Will performance be affected after enabling Top SQL?

This feature has a slight impact on cluster performance. According to our benchmark, the average performance impact is usually less than 3% when the feature is enabled.

3. What is the status of this feature?

It is now a generally available (GA) feature and can be used in production environments.

4. What is the meaning of "Other Statements"?

"Other Statement" counts the total CPU overhead of all non-Top 5 statements. With this information, you can learn the CPU overhead contributed by the Top 5 statements compared with the overall.

5. What is the relationship between the CPU overhead displayed by Top SQL and the actual CPU usage of the process?

Their correlation is strong but they are not exactly the same thing. For example, the cost of writing multiple replicas is not counted in the TiKV CPU overhead displayed by Top SQL. In general, SQL statements with higher CPU usage result in higher CPU overhead displayed in Top SQL.

6. What is the meaning of the Y-axis of the Top SQL chart?

It represents the size of CPU resources consumed. The more resources consumed by a SQL statement, the higher the value is. In most cases, you do not need to care about the meaning or unit of the specific value.

7. Does Top SQL collect running (unfinished) SQL statements?

Yes. The bars displayed in the Top SQL chart at each moment indicate the CPU overhead of all running SQL statements at that moment.

14.11.1.7 Key Visualizer Page

The Key Visualizer page of TiDB Dashboard is used to analyze the usage of TiDB and troubleshoot traffic hotspots. This page visually shows the traffic of the TiDB cluster over a period of time.

14.11.1.7.1 Access Key Visualizer page

You can use one of the following two methods to access the Key Visualizer page:

- After logging into TiDB Dashboard, click **Key Visualizer** on the left navigation menu:

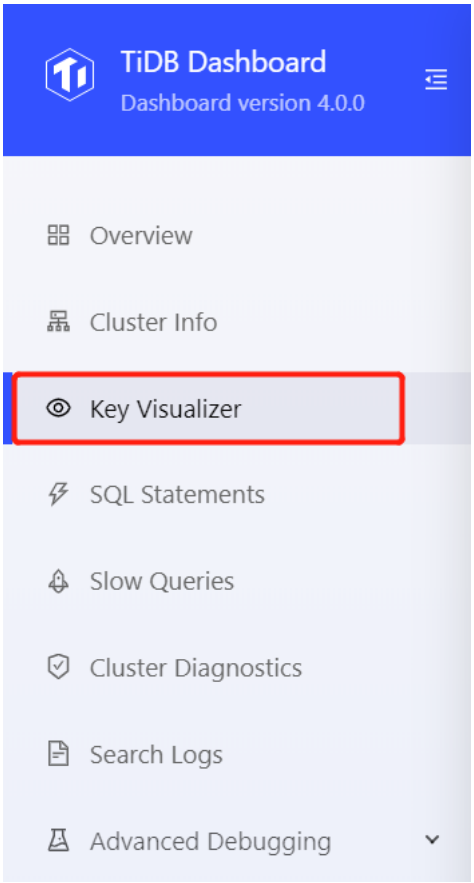


Figure 514: Access Key Visualizer

- Visit http://127.0.0.1:2379/dashboard/#/keyviz in your browser. Replace `127.0.0.1:2379` ↪ with the actual PD instance address and port.

14.11.1.7.2 Interface demonstration

The following image is a demonstration of the Key Visualizer page:

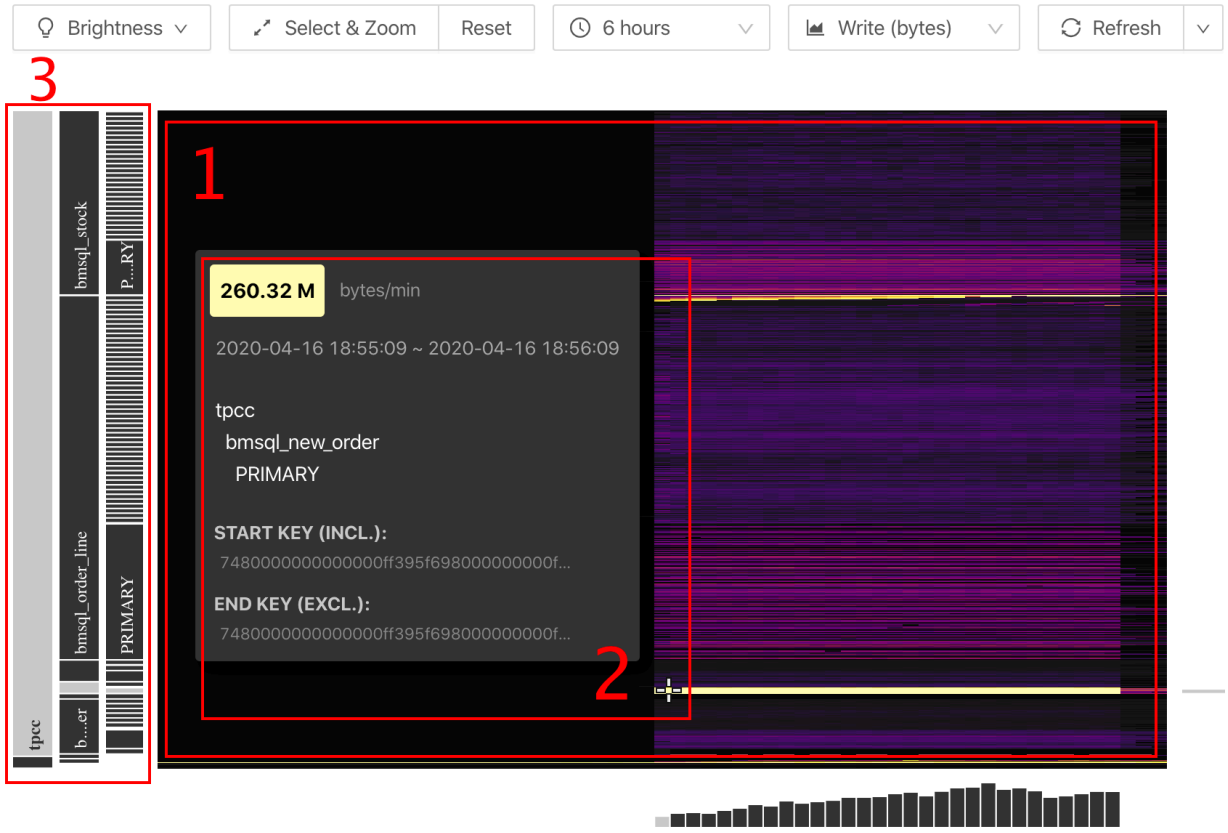


Figure 515: Key Visualizer page

From the interface above, you can see the following objects:

- A large heatmap that shows changes of the overall traffic over time.
- The detailed information of a certain coordinate point.
- Information of tables, indexes, and so on (on the left side of the heatmap).

14.11.1.7.3 Basic concepts

This section introduces the basic concepts that relate to Key Visualizer.

Region

In a TiDB cluster, the stored data is distributed among TiKV instances. Logically, TiKV is a huge and orderly key-value map. The whole key-value space is divided into many segments and each segment consists of a series of adjacent keys. Such segment is called a `Region`.

For detailed introduction of Region, refer to TiDB Internal (I) - Data Storage.

Hotspot

When you use the TiDB database, the hotspot issue is typical, where high traffic is concentrated on a small range of data. Because consecutive data ranges are often processed on the same TiKV instance, the TiKV instance on which the hotspot occurs becomes the performance bottleneck of the whole application. The hotspot issue often occurs in the following scenarios:

- Write adjacent data into a table with the `AUTO_INCREMENT` primary key, which causes a hotspot issue on this table.
- Write adjacent time data into the time index of a table, which causes a hotspot issue on the table index.

For more details about hotspot, refer to Highly Concurrent Write Best Practices

Heatmap

The heatmap is the core part of Key Visualizer, which shows the change of a metric over time. The X-axis of the heatmap indicates the time. The Y-axis of the heatmap indicates the consecutive Regions based on key ranges that cover all schemas and tables of the TiDB cluster.

Colder colors in the heatmap indicate lower read and write traffic of Regions in that period of time. Hotter (brighter) colors indicate higher traffic.

Region compression

A TiDB cluster might have up to hundreds of thousands of Regions. It is difficult to display so many Regions on screen. Therefore, on each heatmap, these Regions are compressed into 1,500 consecutive ranges, each range called a Bucket. In the heatmap, because hotter instances need more attention, Key Visualizer tends to compress a large number of Regions with lower traffic into one Bucket, and displays the Region with higher traffic also in one Bucket.

14.11.1.7.4 Use Key Visualizer

This section introduces how to use Key Visualizer.

Settings

To use the Key Visualizer page for the first time, you need to manually enable this feature on the **Settings** page. Follow the page guide and click **Open Settings** to open the settings page:

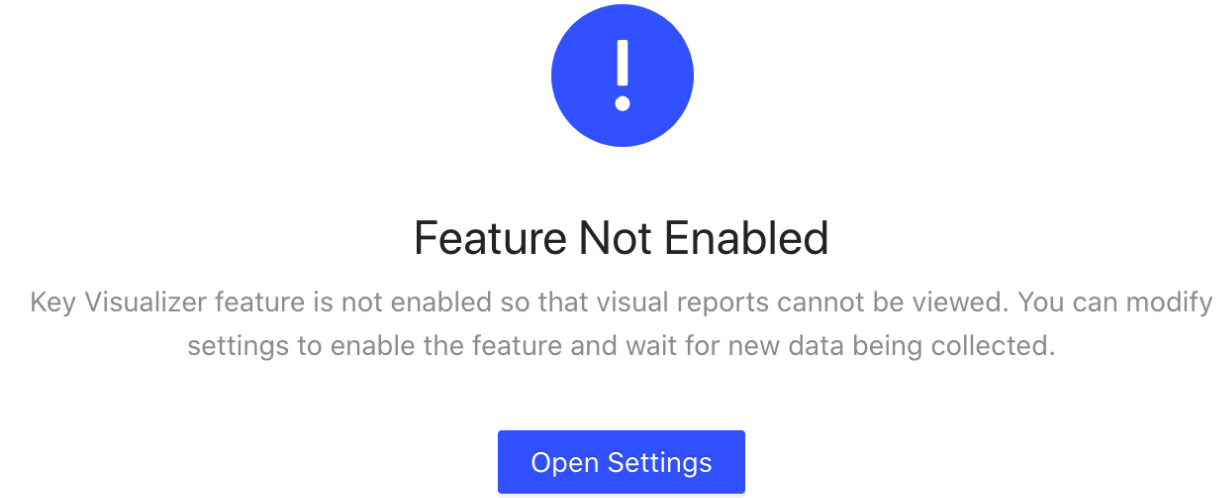


Figure 516: Feature disabled

After this feature is enabled, you can open the settings page by clicking the **Settings** icon in the upper right corner:

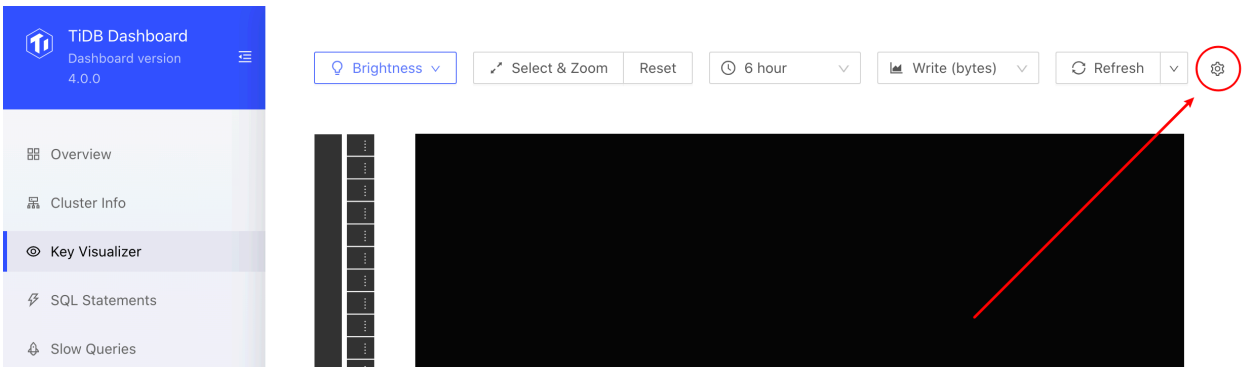


Figure 517: Settings icon

The settings page is shown as follows:

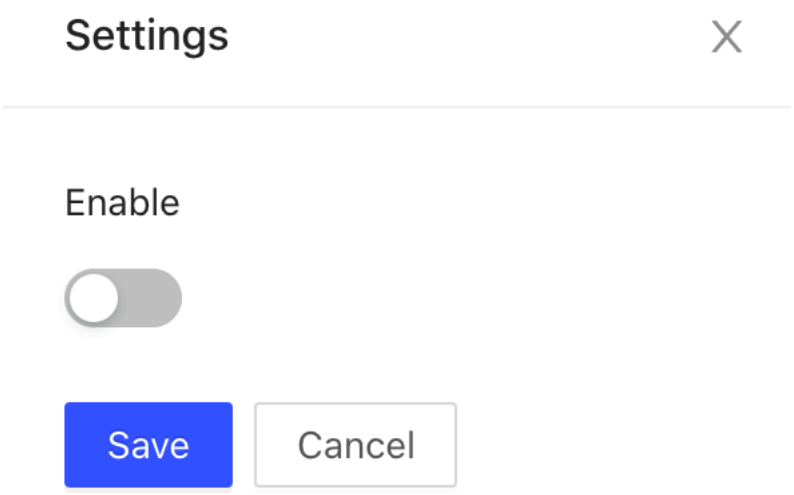


Figure 518: Settings page

Set whether to start data collection through the switch, and click **Save** to take effect. After enabling the feature, you can see that the toolbar is available:

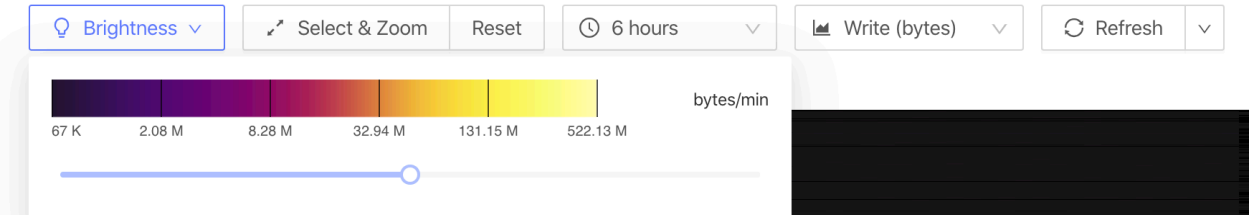


Figure 519: Toolbar

After this feature is enabled, data collection is going on at the backend. You can see the heatmap shortly.

Observe a certain period of time or Region range

When you open Key Visualizer, the heatmap of the entire database over the recent six hours is displayed by default. In this heatmap, the closer to the right side (current time), the shorter the time interval corresponding to each column of Buckets. If you want to observe a specific time period or a specific Region range, you can zoom in to get more details. The specific instructions are as follows:

1. Scroll up or down in the heatmap.
2. Click and drag one of the following buttons to select the range.

- Click the **Select & Zoom** button. Then click and drag this button to select the area to zoom in.

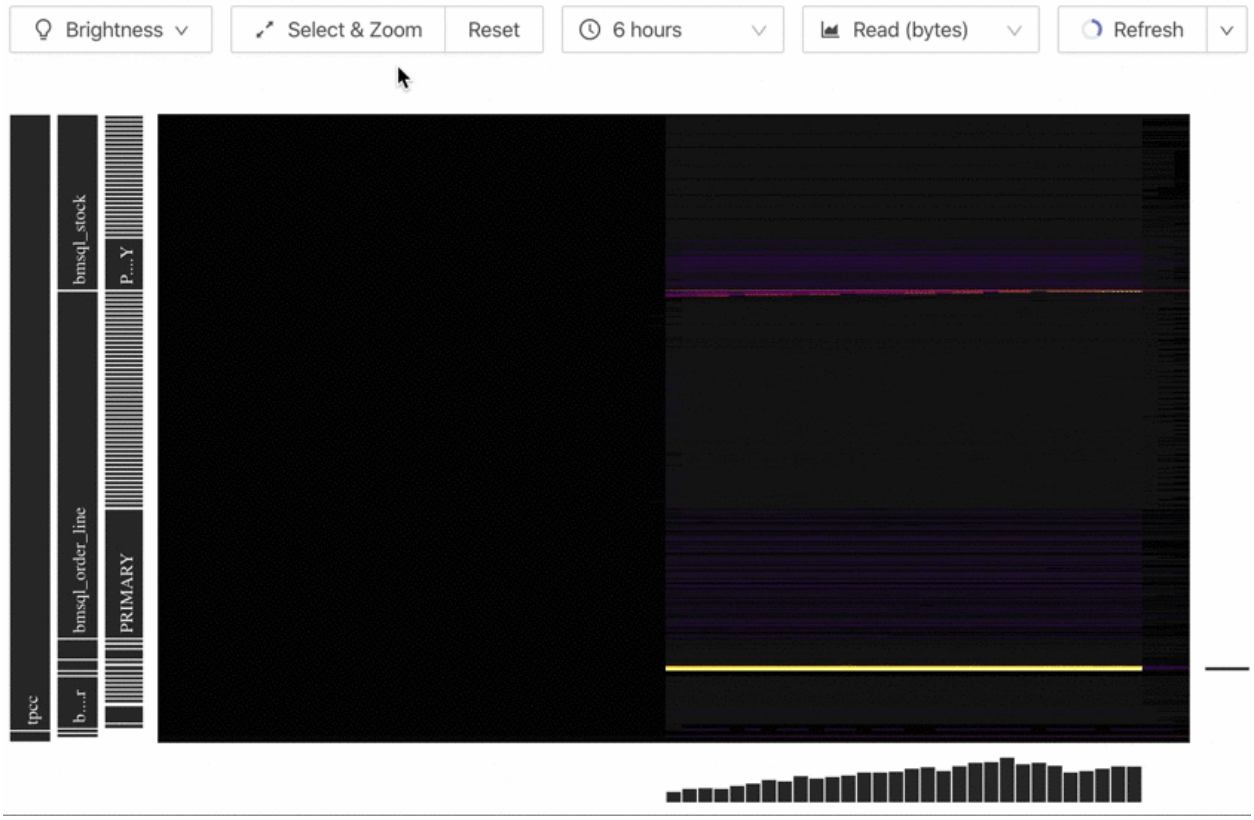


Figure 520: Selection box

- Click the **Reset** button to reset the Region range to the entire database.
- Click the **time selection box** (at the position of `6 hours` on the interface above) and select the observation time period again.

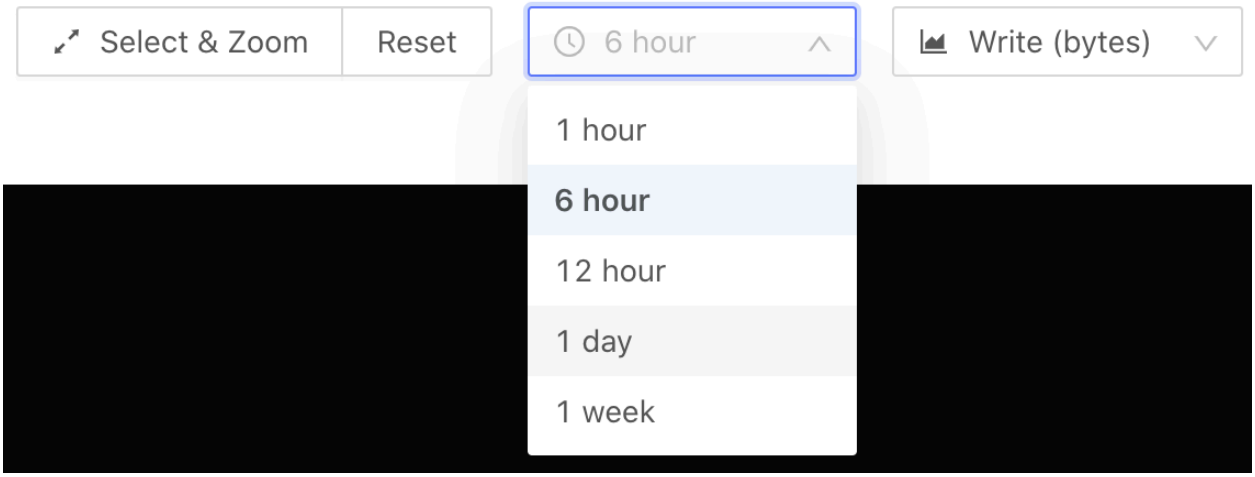


Figure 521: Select time

Note:

If you follow step 2 in the instruction above, the heatmap is redrawn, which might be very different from the original heatmap. This difference is normal because if you observe in more detail, the granularity of the Region compression has changed, or the basis of `hot` in a specific range has changed.

Adjust brightness

The heatmap uses colors of different brightnesses to indicate the Bucket traffic. Colder colors in the heatmap indicate lower read and write traffic of the Region in that period of time. Hotter (brighter) colors indicate higher traffic. If the color is too cold or too hot, it is difficult to observe in details. In this situation, you can click the **Brightness** button and then use the slider to adjust the brightness of the page.

Note:

When Key Visualizer displays the heatmap of an area, it defines the basis of being cold and hot according to the traffic of this area. When the traffic in the entire area is relatively even, even if the overall traffic is low in value, you might still see a large bright-colored area. Remember to include the value into your analysis.

Select metrics

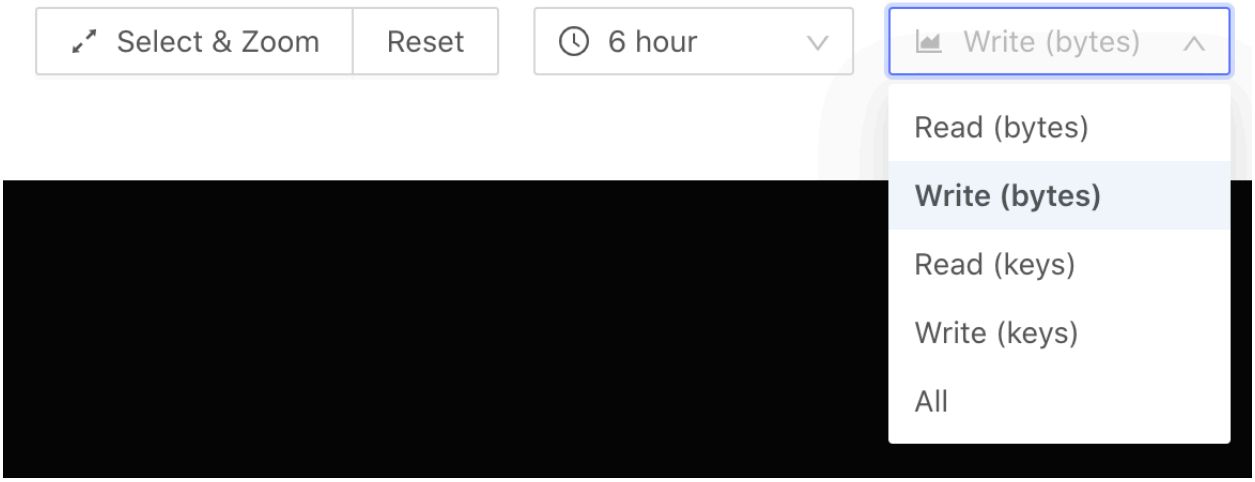


Figure 522: Select metrics

You can view a metric you are interested in by selecting this metric in the **metrics selection box** (at the `Write (bytes)` position in the interface above):

- `Read (bytes)`: Read traffic.
- `Write (bytes)`: Write traffic.
- `Read (keys)`: The number of read rows.
- `Write (keys)`: The number of written rows.
- `All`: The sum of read and write traffic.

Refresh and automatic refresh

To regain a heatmap based on the current time, click the **Refresh** button. If you need to observe the traffic distribution of the database in real time, click the down arrow on the right side of the **Refresh** button and select a fixed time interval for the heatmap to automatically refresh at this interval.

> **Note:**
>
> If you adjust the time range or Region range, the automatic refresh is disabled.

See Bucket details

You can hover your mouse over the Bucket you are interested in to view the detailed information of this Region range. The image below is an example of this information:

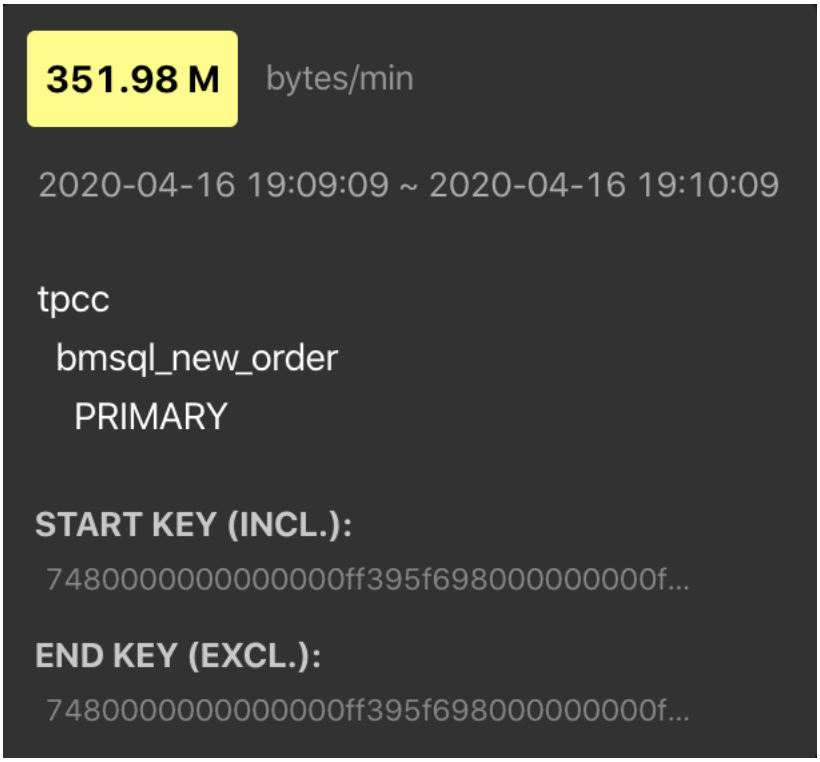


Figure 523: Bucket details

If you want to copy this information, click a Bucket. Then, the page with relevant details is temporarily pinned. Click on the information, and you have copied it to the clipboard.

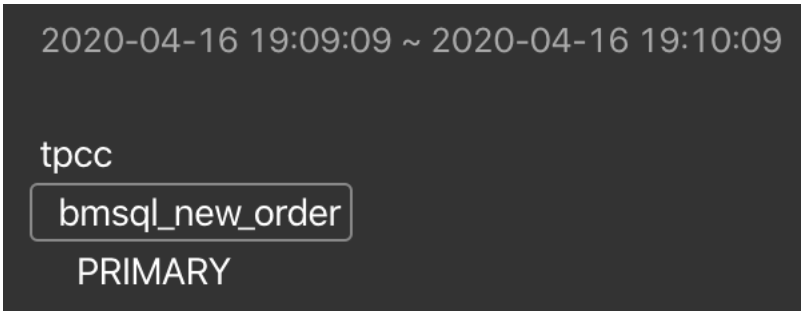


Figure 524: Copy Bucket details

14.11.1.7.5 Common heatmap types

This section shows and interprets four common types of heatmap in Key Visualizer.

Evenly distributed workload

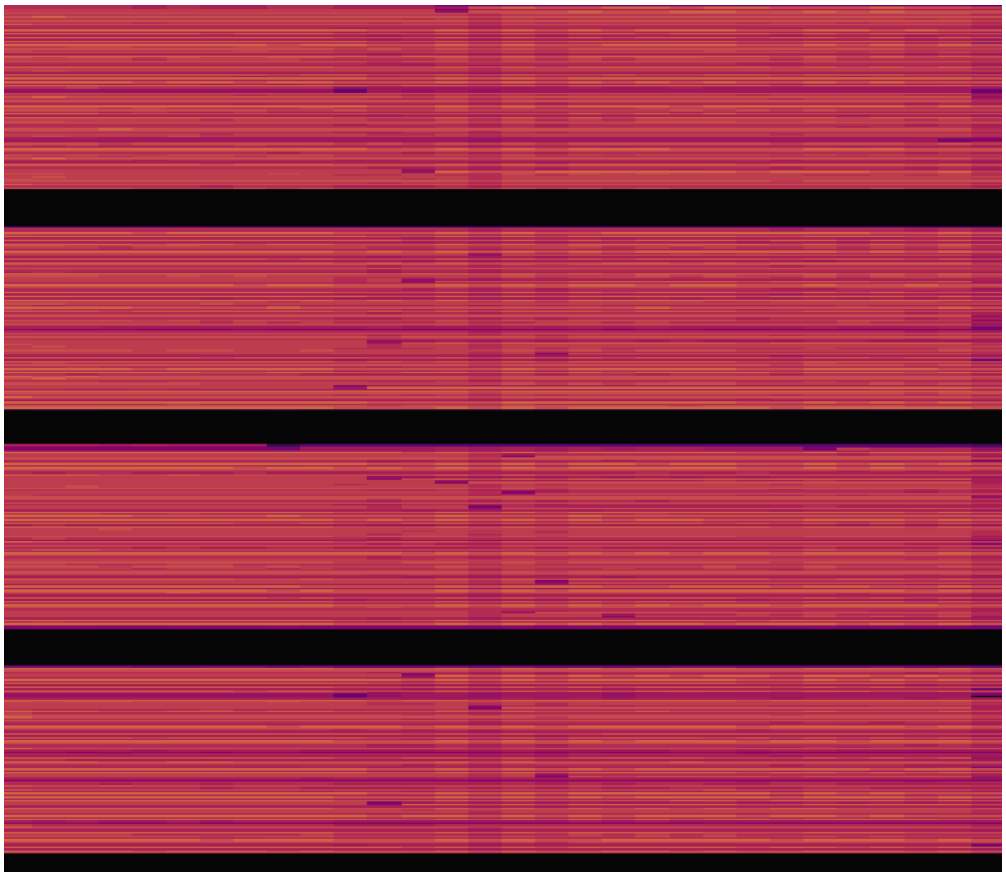


Figure 525: Balanced

In the heatmap above, bright and dark colors are a fine-grained mix. This indicates that reads or writes are evenly distributed over time and among key ranges. The workload is evenly distributed to all nodes, which is ideal for a distributed database.

Periodically reads and writes

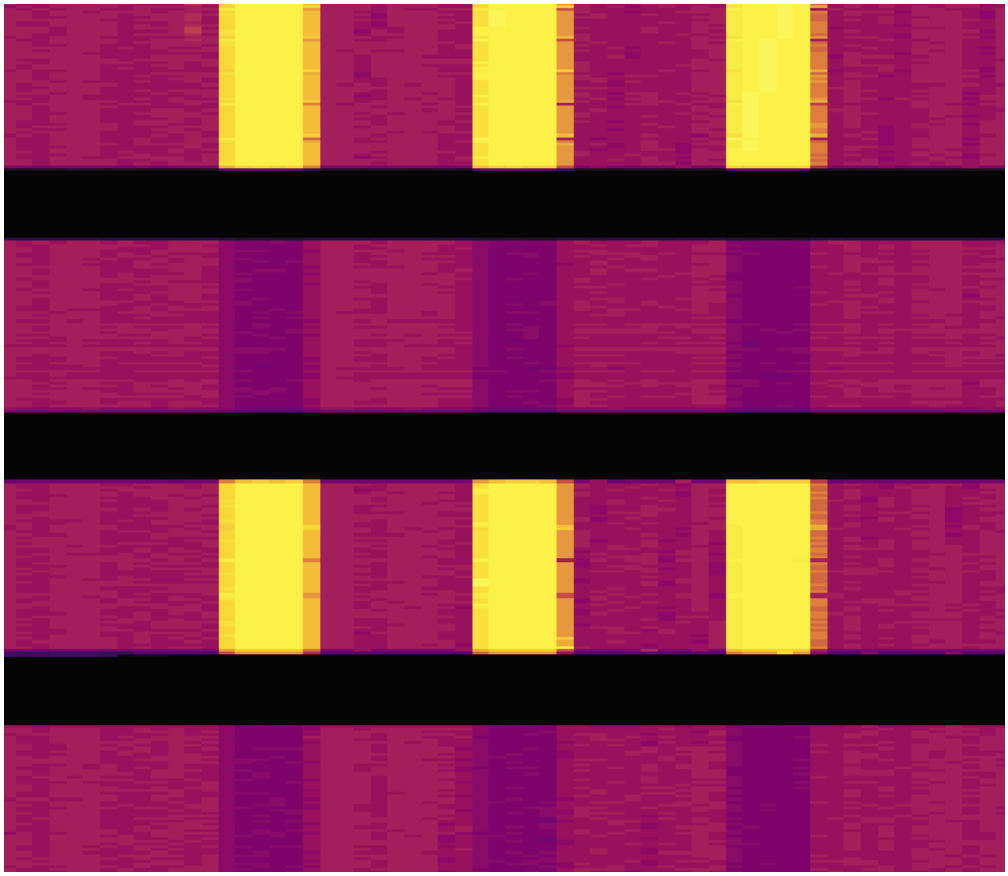


Figure 526: Periodically

In the heatmap above, there is an alternating brightness and darkness along the X-axis (time) but the brightness is relatively even along the Y-axis (Region). This indicates that the reads and writes change periodically, which might occur in scenarios of periodically scheduled tasks. For example, the big data platform periodically extracts data from TiDB every day. In this kind of scenarios, pay attention to whether the resources are sufficient during peak usage.

Concentrated reads or writes

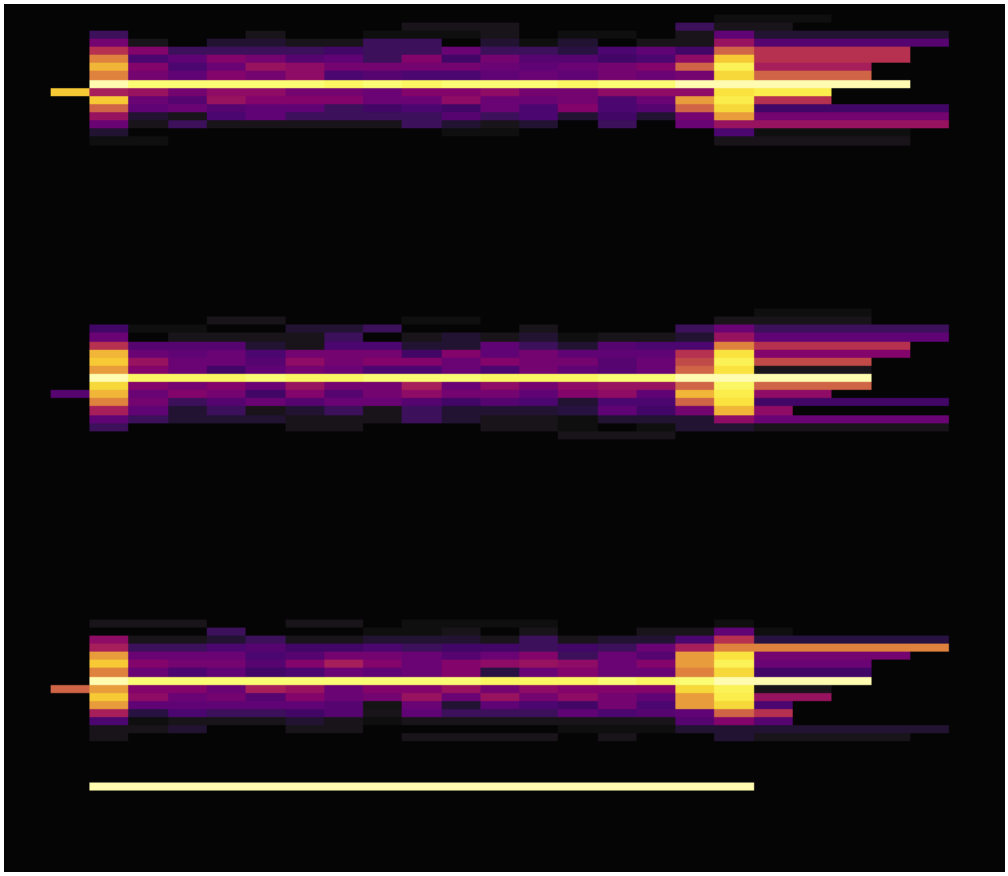


Figure 527: Concentrated

In the heatmap above, you can see several bright lines. Along the Y-axis, the fringes around the bright lines are dark, which indicates that the Regions corresponding to bright lines have high read and write traffic. You can observe whether the traffic distribution is expected by your application. For example, when all services are associated with the user table, the overall traffic of the user table can be high, so it is reasonable to show bright lines in the heatmap.

In addition, the height of the bright lines (the thickness along the Y-axis) is important. Because TiKV has its own Region-based hotspot balancing mechanism, the more Regions involved in the hotspot, the better it is for balancing traffic across all TiKV instances. The thicker and more bright lines indicate that hotspots are more scattered, and TiKV is better used. The thinner and fewer bright lines indicate that hotspots are more concentrated, and the hotspot issue is more obvious in TiKV, which might requires manual intervention.

Sequential reads or writes

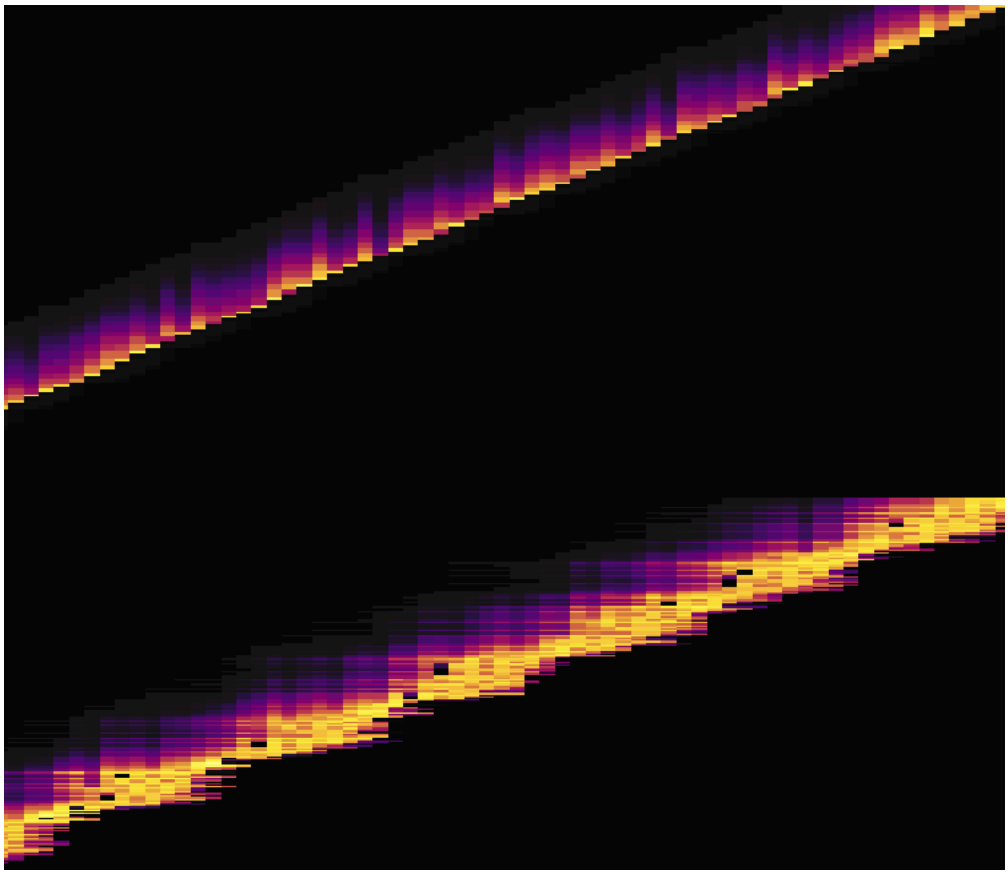


Figure 528: Sequential

In the heatmap above, you can see a bright line. This means that the data reads or writes are sequential. Typical scenarios of sequential data reads or writes are importing data or scanning tables and indexes. For example, you continuously write data to tables with auto-increment IDs.

Regions in the bright areas are the hotspots of read and write traffic, which often become the performance bottleneck of the entire cluster. In this situation, you might need to readjust the primary key for the application. By doing this, you scatter Regions much as possible to spread the pressure across multiple Regions. You can also schedule application tasks during the low-peak period.

Note:

In this section, only the common types of heatmap are shown. Key Visualizer actually displays the heatmap of all schemas and tables in the entire cluster, so you might see different types of heatmap in different areas, or mixed results of multiple heatmap types. Use the heatmap based on the actual situation.

14.11.1.7.6 Address hotspot issues

TiDB has some built-in features to mitigate the common hotspot issue. Refer to Highly Concurrent Write Best Practices for details.

14.11.1.8 TiDB Dashboard Metrics Relation Graph

TiDB Dashboard metrics relation graph is a feature introduced in v4.0.7. This feature presents a relation graph of the monitoring data of each internal process's duration in a TiDB cluster. The aim is to help you quickly understand the duration of each process and their relations.

14.11.1.8.1 Access graph

Log into TiDB Dashboard, click **Cluster Diagnostics** on the left navigation menu, and you can see the page of generating the metrics relation graph.

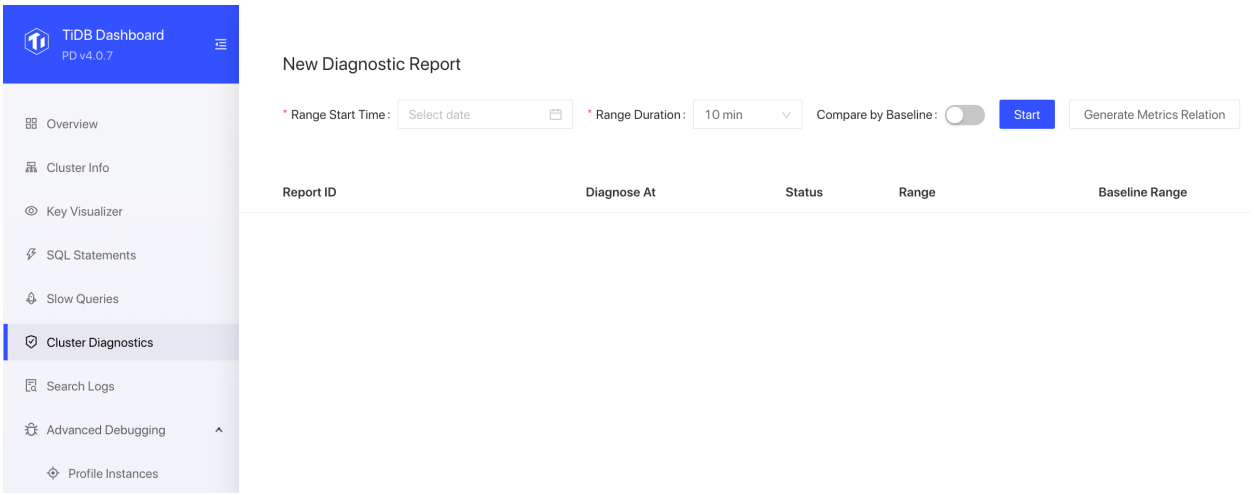


Figure 529: Metrics relation graph homepage

After setting **Range Start Time** and **Range Duration**, click the **Generate Metrics Relation** button and you will enter the page of metrics relation graph.

14.11.1.8.2 Understand graph

The following image is an example of the metrics relation graph. This graph illustrates the proportion of each monitoring metric's duration to the total query duration in a TiDB cluster within 5 minutes after 2020-07-29 16:36:00. The graph also illustrates the relations of each monitoring metric.

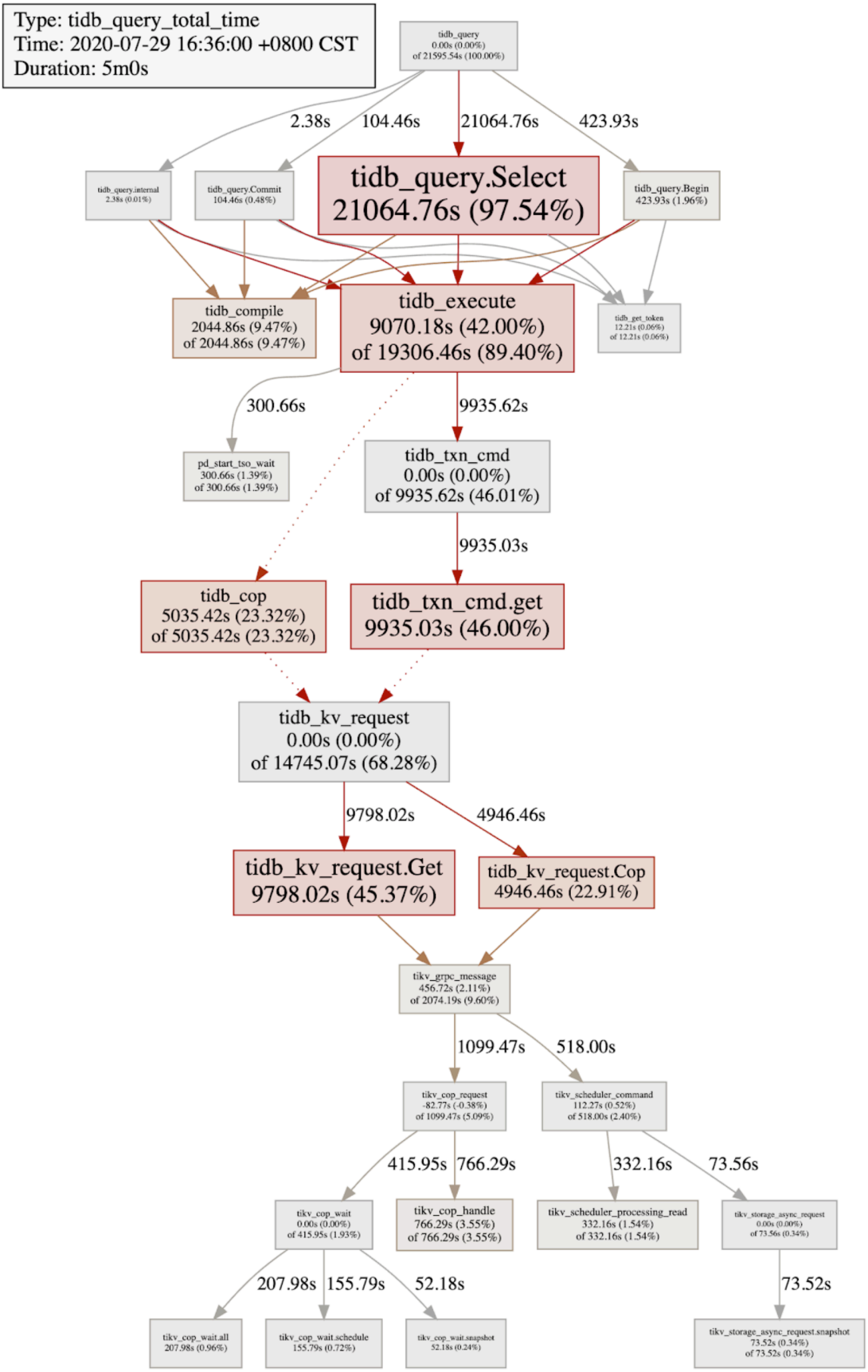


Figure 530: Metrics relation graph example

For example, the node meaning of the `tidb_execute` monitoring metric is as follows:

- The total duration of the `tidb_execute` monitoring metric is 19306.46 seconds, which accounts for 89.4% of the total query duration.
- The duration of the `tidb_execute` node itself is 9070.18 seconds, which accounts for 42% of the total query duration.
- Hover your mouse over the box area, and you can see the detailed information of the metric, including the total duration, the average duration, and the average P99 (99th percentile) duration.

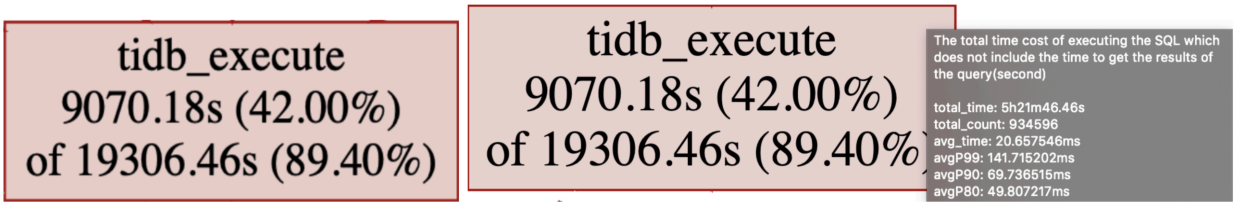


Figure 531: tidb_execute node example

Node information

Each box area represents a monitoring metric and provides the following information:

- The name of the monitoring metric
- The total duration of the monitoring metric
- The proportion of the metric's total duration to the total query duration

The total duration of the metric node = the duration of the metric node itself + the duration of its child nodes. Therefore, the metric graph of some nodes displays the proportion of the node itself's duration to the total duration, such as the graph of `tidb_execute`.

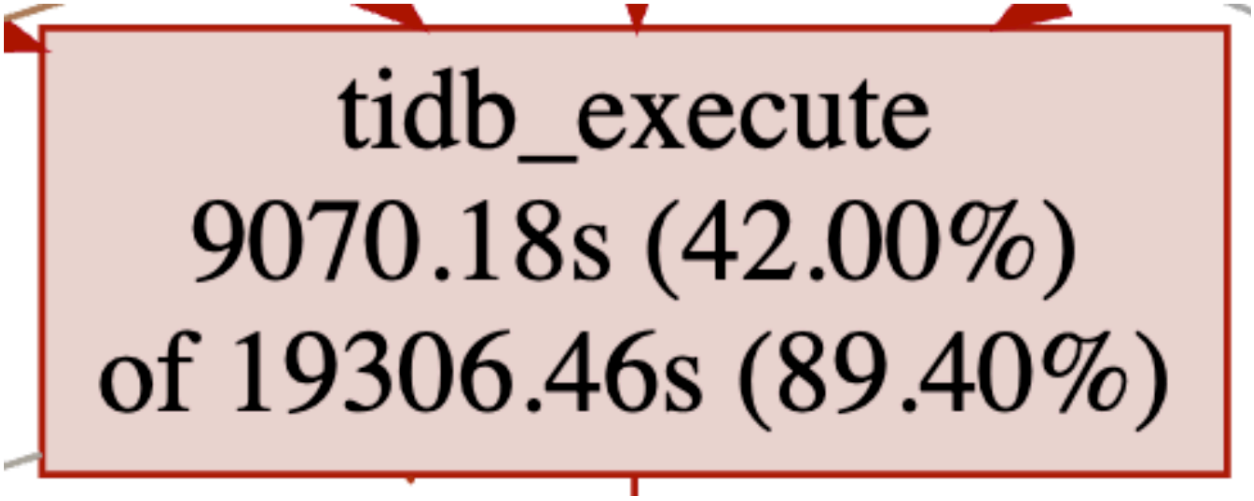


Figure 532: tidb_execute node example1

- `tidb_execute` is the name of the monitoring metric, which represents the execution duration of a SQL query in the TiDB execution engine.
- `19306.46s` represents that total duration of the `tidb_execute` metric is 19306.46 seconds. `89.40%` represents that 19306.46 seconds account for 89.40% of the total time consumed for all SQL queries (including user SQL queries and TiDB's internal SQL queries). The total query duration is the total duration of `tidb_query`.
- `9070.18s` represents that the total execution duration of the `tidb_execute` node itself is 9070.18 seconds, and the rest is the time consumed by its child nodes. `42.00%` represents that 9070.18 seconds account for 42.00% of the total query duration of all queries.

Hover your mouse over the box area and you can see more details of the `tidb_execute` metric node:

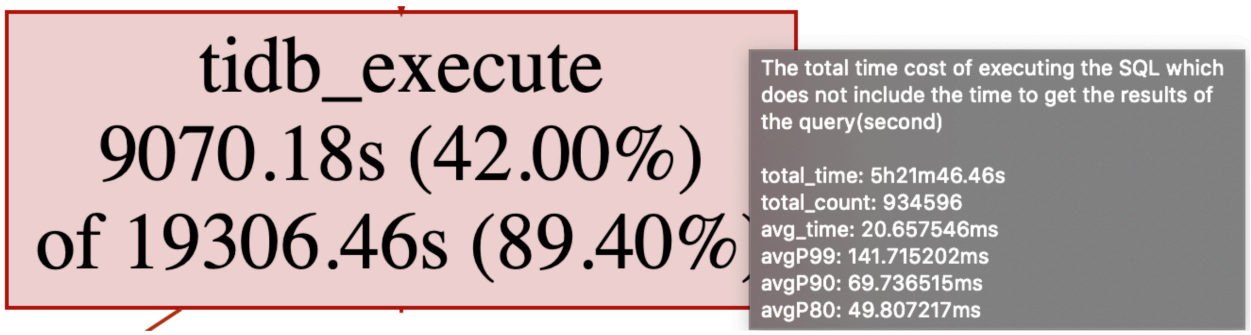


Figure 533: tidb_execute node example2

The text information displayed in the image above is the description of the metric node, including the total duration, the total times, the average duration, and the average duration

P99, P90, and P80.

The parent-child relations between nodes

Taking the `tidb_execute` metric node as an example, this section introduces a metric's child nodes.

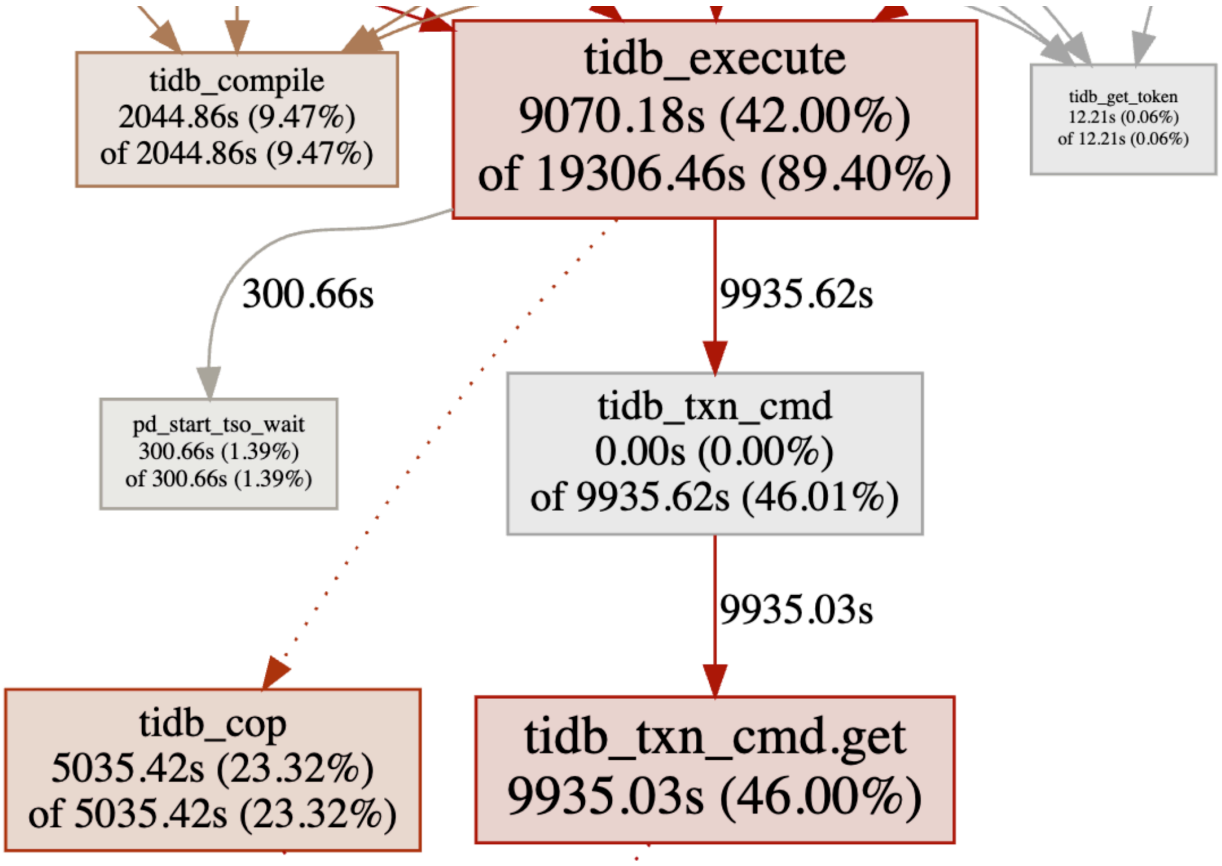


Figure 534: tidb_execute node relation example1

From the graph above, you can see the two child nodes of `tidb_execute`:

- `pd_start_tso_wait`: The total duration of waiting for the transaction's `start_tso`, which is 300.66 seconds.
- `tidb_txn_cmd`: The total duration of TiDB executing the relevant transaction commands, which is 9935.62 seconds.

In addition, `tidb_execute` also has a dotted arrow pointing to the `tidb_cop` box area, which indicates as follows:

`tidb_execute` includes the duration of the `tidb_cop` metric, but `cop` requests might be executed concurrently. For example, the `execute` duration of performing `join` queries on two tables is 60 seconds, during which table scan requests are concurrently executed on

the joined two tables. If the execution durations of `cop` requests are respectively 40 seconds and 30 seconds, the total duration of `cop` requests are 70 seconds. However, the `execute` duration is only 60 seconds. Therefore, if the duration of a parent node does not completely include the duration of a child node, the dotted arrow is used to point to the child node.

> **Note:**
>
> When a node have a dotted arrow pointing to its child node, the duration of this node itself is inaccurate. For example, in the `tidb_execute` node, the duration of the node itself is 9070.18 seconds (`9070.18 = 19306.46 - 300.66 - 9935.62`). In this equation, the duration of the `tidb_cop` child node is not calculated into the duration of `tidb_execute`'s child nodes. But in fact, this is not true. 9070.18 seconds, the duration of `tidb_execute` itself, includes a part of the `tidb_cop` duration, and the duration of this part cannot be determined.

`tidb_kv_request` and its parent nodes

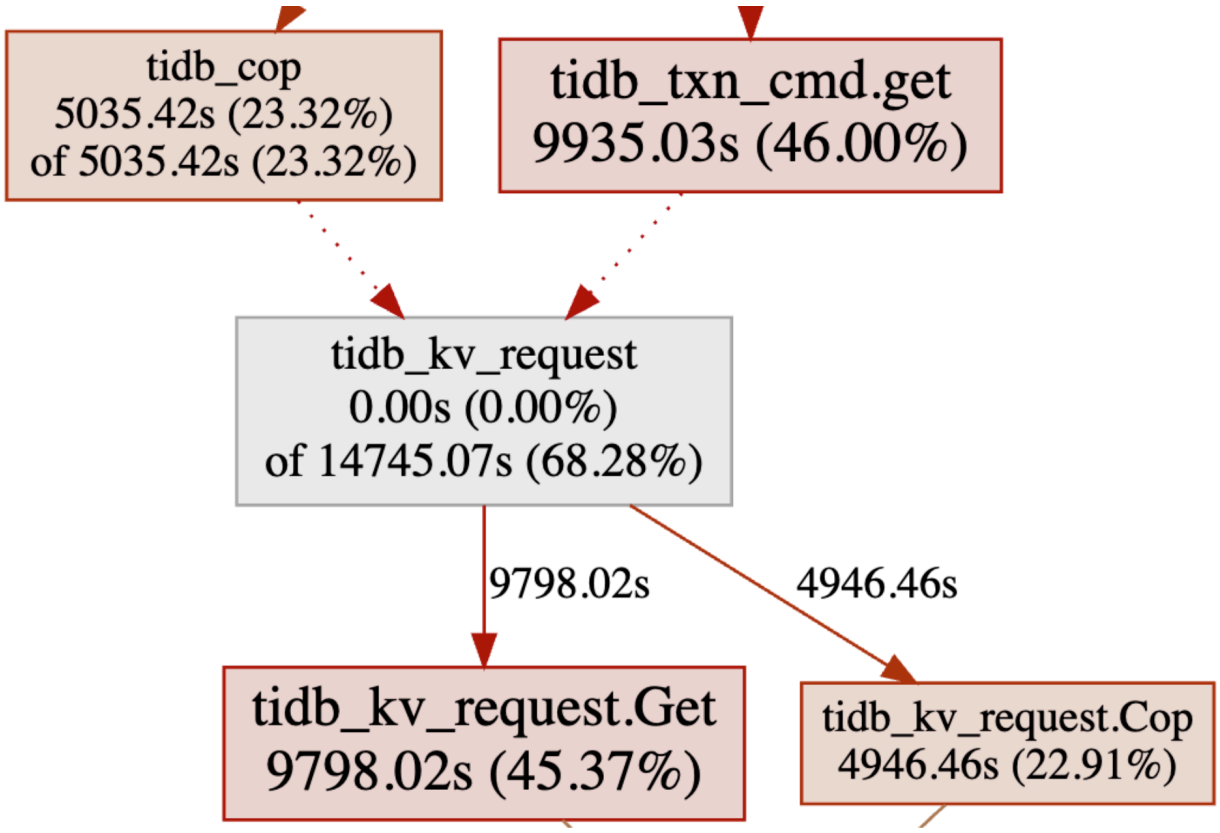


Figure 535: tidb_execute node relation example2

`tidb_cop` and `tidb_txn_cmd.get`, the parent nodes of `tidb_kv_request`, both have dotted arrows pointing to `tidb_kv_request`, which indicates as follows:

- The duration of `tidb_cop` includes a part of `tidb_kv_request`'s duration.
- The duration of `tidb_txn_cmd.get` also includes a part of `tidb_kv_request`'s duration.

However, it is hard to determine how much duration of `tidb_kv_request` is included in `tidb_cop`.

- `tidb_kv_request.Get`: The duration of TiDB sending the `Get` type key-value requests.
- `tidb_kv_request.Cop`: The duration of TiDB sending the `Cop` type key-value requests.

`tidb_kv_request` does not include `tidb_kv_request.Get` and `tidb_kv_request.Cop` nodes as its child nodes, but consists of the latter two nodes. The name prefix of the child node is the name of the parent node plus `.xxx`, which means that the child node is the sub-class of the parent node. You can understand this case in the following way:

The total duration of TiDB sending key-value requests is 14745.07 seconds, during which the key-value requests for the `Get` and `Cop` types respectively consume 9798.02 seconds and 4946.46 seconds.

14.11.1.9 SQL Statements Analysis

14.11.1.9.1 SQL Statements Page of TiDB Dashboard

The SQL statements page shows the execution status of all SQL statements in the cluster. This page is often used to analyze the SQL statement whose total or single execution time is long.

On this page, SQL queries with a consistent structure (even if the query parameters are inconsistent) are classified as the same SQL statement. For example, both `SELECT * FROM employee WHERE id IN (1, 2, 3)` and `select * from EMPLOYEE where ID in (4, 5)` are classified as the same `select * from employee where id in (...)` SQL statement.

Access the page

You can use one of the following two methods to access the SQL statement summary page:

- After logging into TiDB Dashboard, click **SQL Statements** on the left navigation menu:

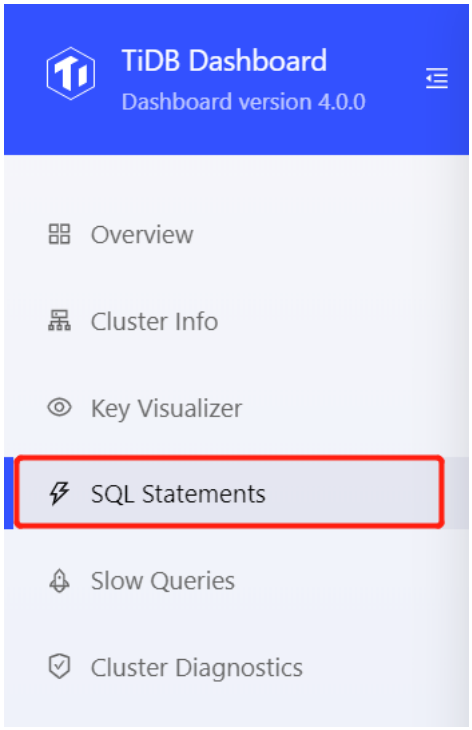


Figure 536: Access SQL statement summary page

- Visit http://127.0.0.1:2379/dashboard/#/statement in your browser. Replace `127.0.0.1:2379` with the actual PD instance address and port.

All the data shown on the SQL statement summary page are from the TiDB statement summary tables. For more details about the tables, see TiDB Statement Summary Tables.

> **Note:**
>
> In the **Mean Latency** column of the SQL statement summary page, the blue bar indicates the average execution time. If there is a yellow line on the blue bar for an SQL statement, the left and right sides of the yellow line respectively represent the minimum and maximum execution time of the SQL statement during the recent data collection cycle.

Change Filters

On the top of the SQL statement summary page, you can modify the time range of SQL executions to be displayed. You can also filter the list by database in which SQL statements are executed, or by SQL types. The following image shows all SQL executions over the recent data collection cycle (recent 30 minutes by default).

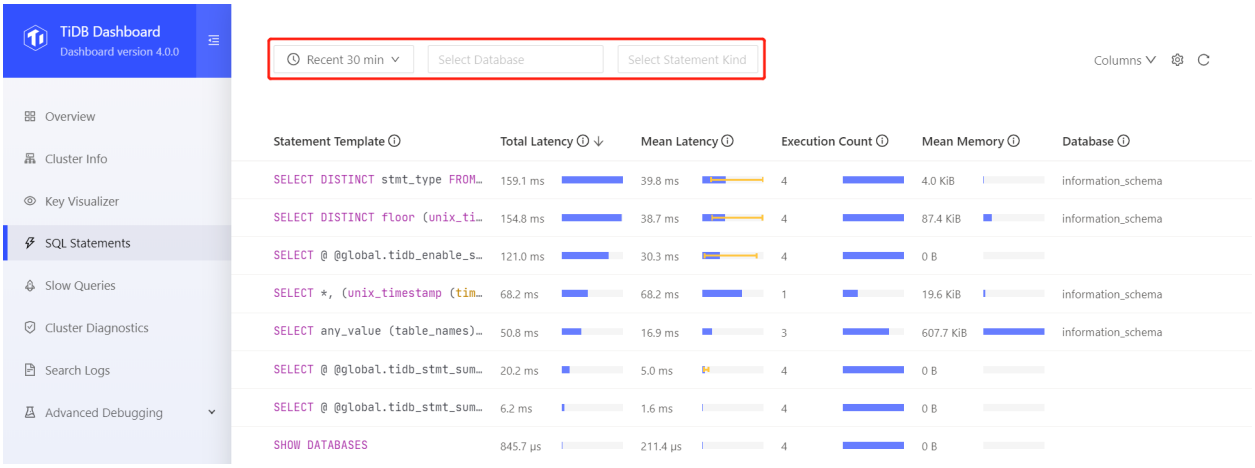


Figure 537: Modify filters

Display More Columns

Click **Columns** on the page and you can choose to see more columns. You can move your mouse to the **(i)** icon at the right side of a column name to view the description of this column:

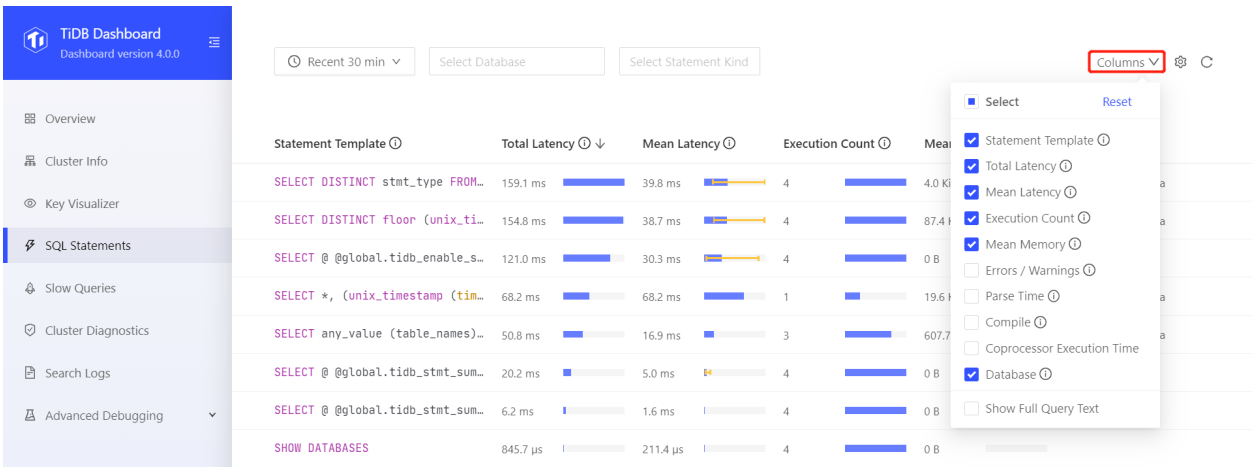


Figure 538: Choose columns

Sort by Column

By default, the list is sorted by **Total Latency** from high to low. Click on different column headings to modify the sorting basis or switch the sorting order:

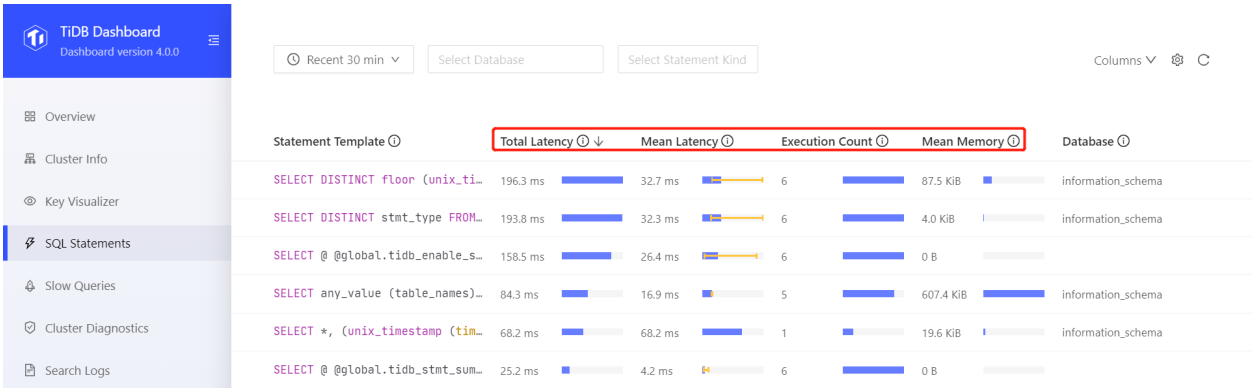


Figure 539: Modify list sorting

Change Settings

On the list page, click the **Settings** button on the top right to change the settings of the SQL statements feature:

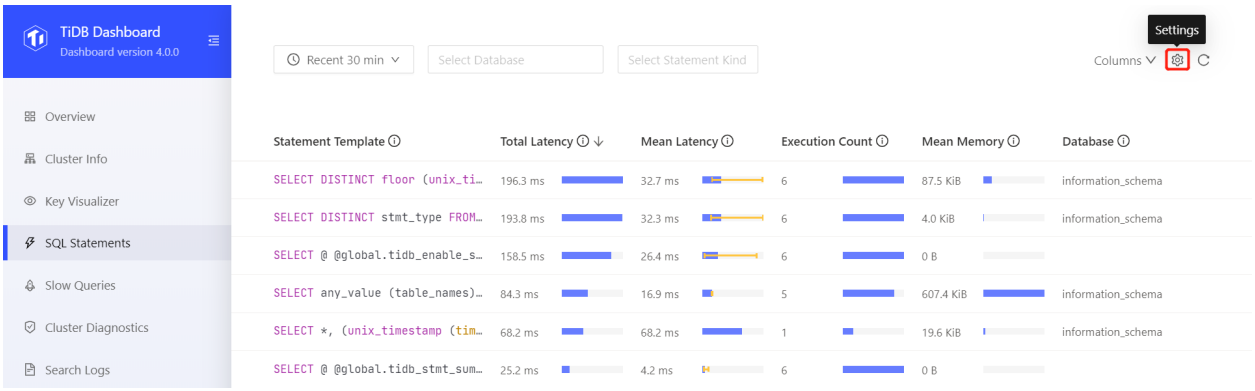


Figure 540: Settings entry

After clicking the **Settings** button, you can see the following setting dialog box:

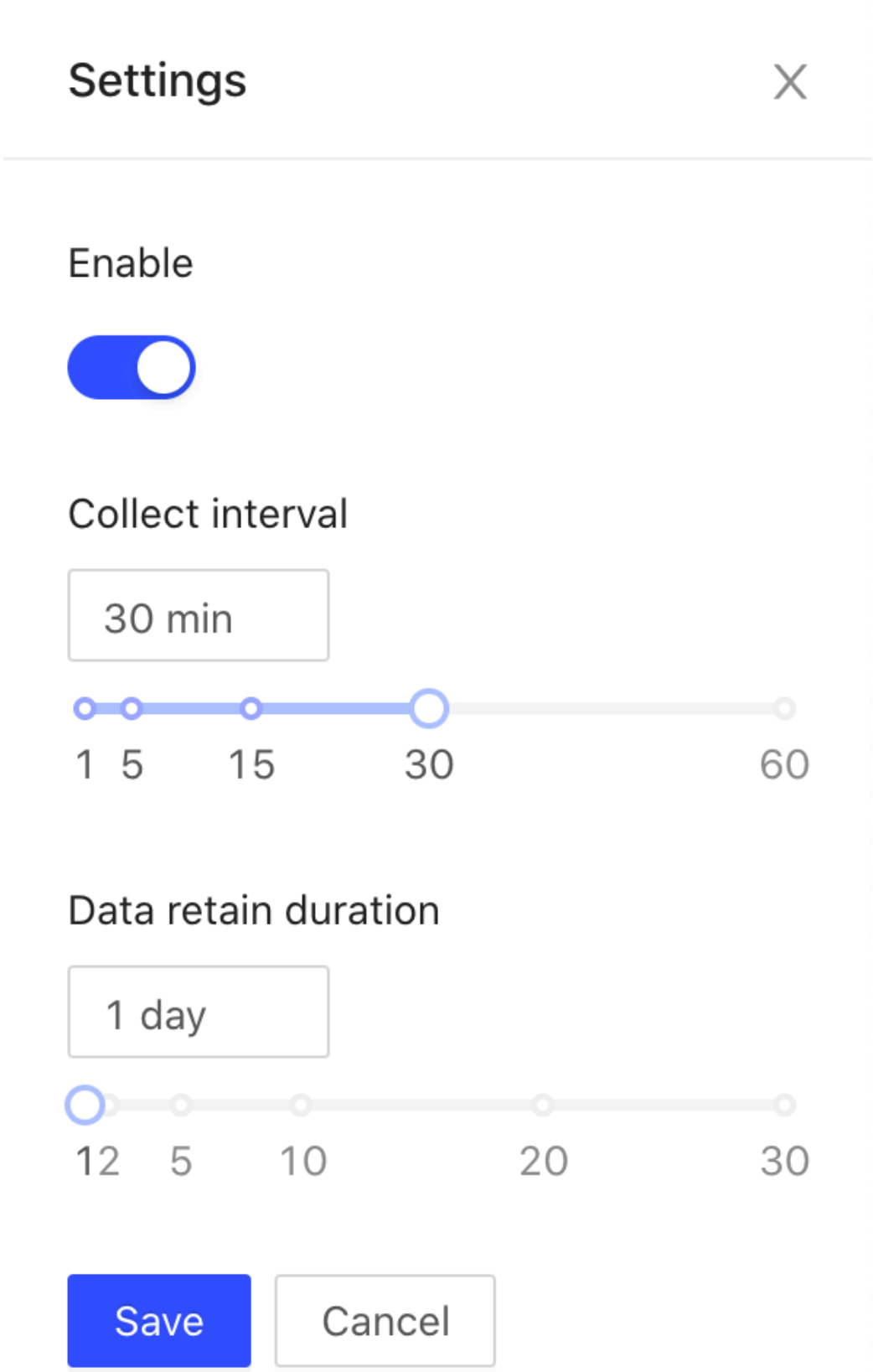


Figure 541: Settings

On the setting page, you can disable or enable the SQL statements feature. When the SQL statements feature is enabled, you can modify the following settings:

- Collect interval: The length of period for each SQL statement analysis, which is 30 minutes by default. The SQL statements feature summarizes and counts all SQL statements within a period of time. If the period is too long, the granularity of the summary is coarse, which is not good for locating problems; if the period is too short, the granularity of the statistics is fine, which is good for locating problems, but this will result in more records and more memory usage within the same data retention duration. Therefore, you need to adjust this value based on the actual situation, and properly lower this value when locating problems.
- Data retain duration: The retention duration of summary information, which is 1 day by default. Data retained longer than this duration will be deleted from system tables.

See Configurations of Statement Summary Tables for details.

Note:

- Because the statement system table is only stored in memory, after the SQL Statements feature is disabled, the data in the system table will be cleared.
- The values of `Collect interval` and `retain duration` affect the memory usage, so it is recommended to adjust these values according to the actual situation. The value of `retain duration` should not be set too large.

14.11.1.9.2 Statement Execution Details of TiDB Dashboard

Click any item in the list to enter the detail page of the SQL statement to view more detailed information. This information includes the following parts:

- The overview of SQL statements, which includes the SQL template, the SQL template ID, the current time range of displayed SQL executions, the number of execution plans and the database in which the SQL statement is executed (see area 1 in the image below).
- The execution plan list: If the SQL statement has multiple execution plans, this list is displayed. You can select different execution plans, and the details of the selected plans are displayed below the list. If there is only one execution plan, the list is not displayed (see area 2 below).
- Execution detail of plans, which displays the detailed information of the selected execution plans. See Execution plan in details (area 3 in the image below).

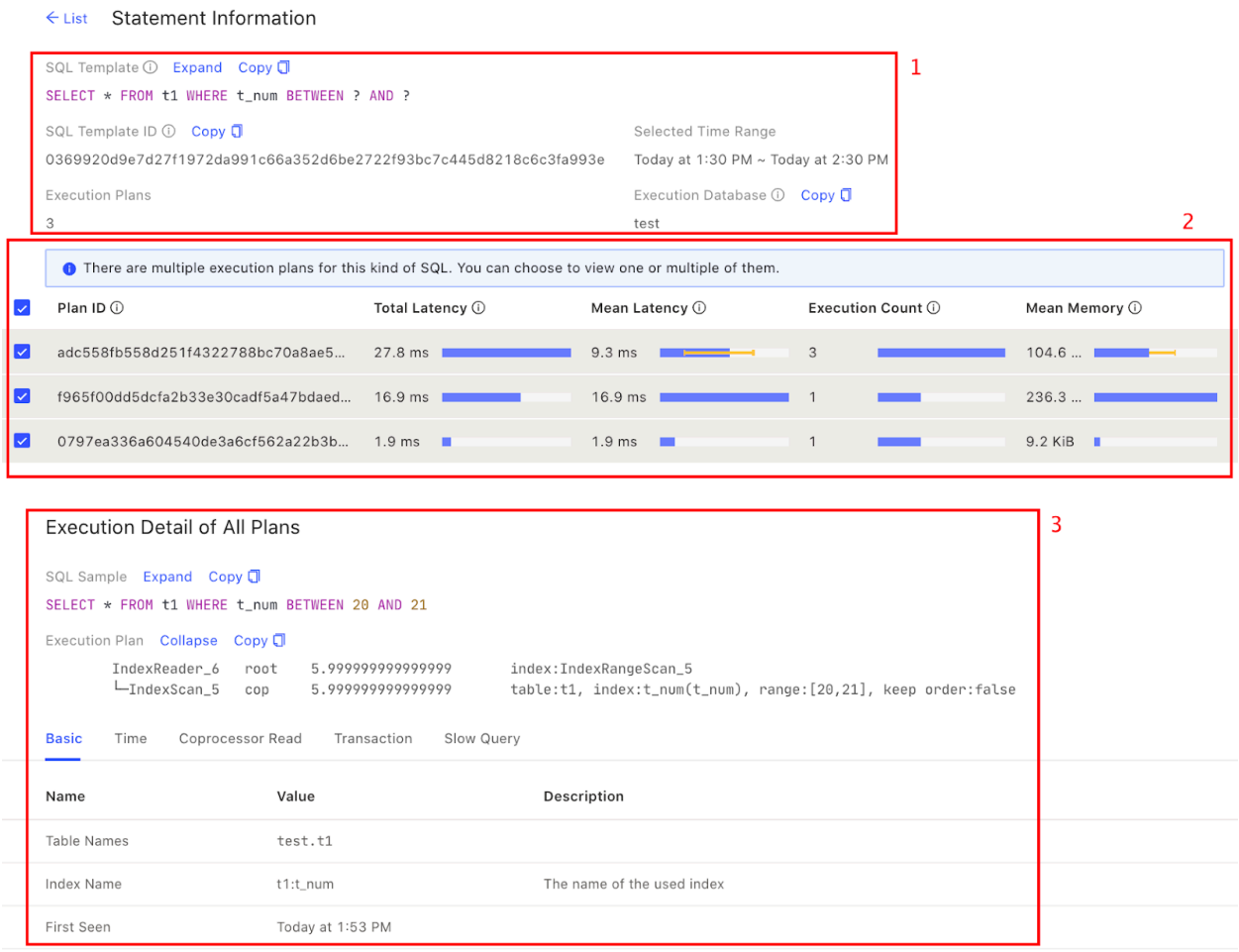


Figure 542: Details

Execution details of plans

The execution detail of plans includes the following information:

- SQL sample: The text of a certain SQL statement that is actually executed corresponding to the plan. Any SQL statement that has been executed within the time range might be used as a SQL sample.
- Execution plan: For details of the execution plan, see Understand the Query Execution Plan. If multiple execution plans are selected, only (any) one of them is displayed.
- For basic information, execution time, Coprocessor read, transaction, and slow query of the SQL statement, you can click the corresponding tab titles to switch among different information.

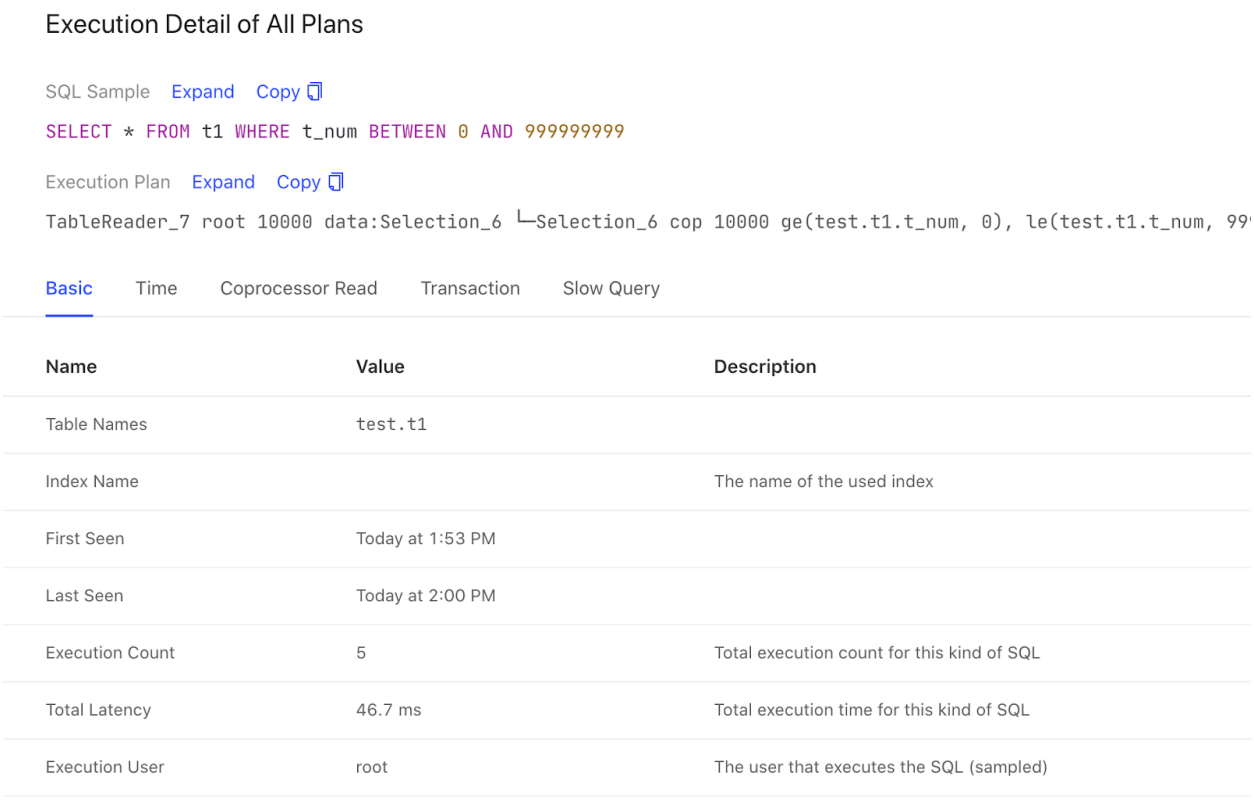


Figure 543: Execution details of plans

Basic Tab

The basic information of a SQL execution includes the table names, index name, execution count, and total latency. The **Description** column provides detailed description of each field.

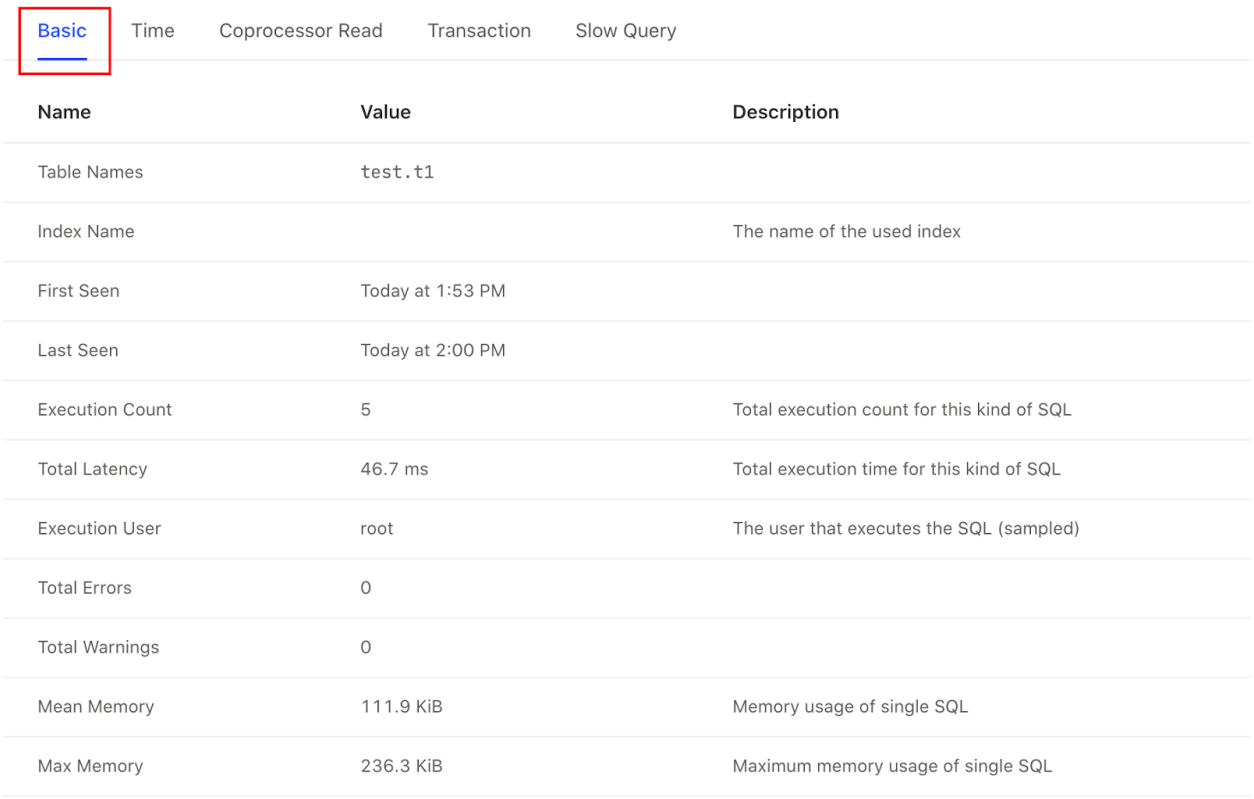


Figure 544: Basic information

Time Tab

Click the **Time** tab, and you can see how long each stage of the execution plan lasts.

> **Note:**
>
> Because some operations might be performed in parallel within a single SQL statement, the cumulative duration of each stage might exceed the actual execution time of the SQL statement.

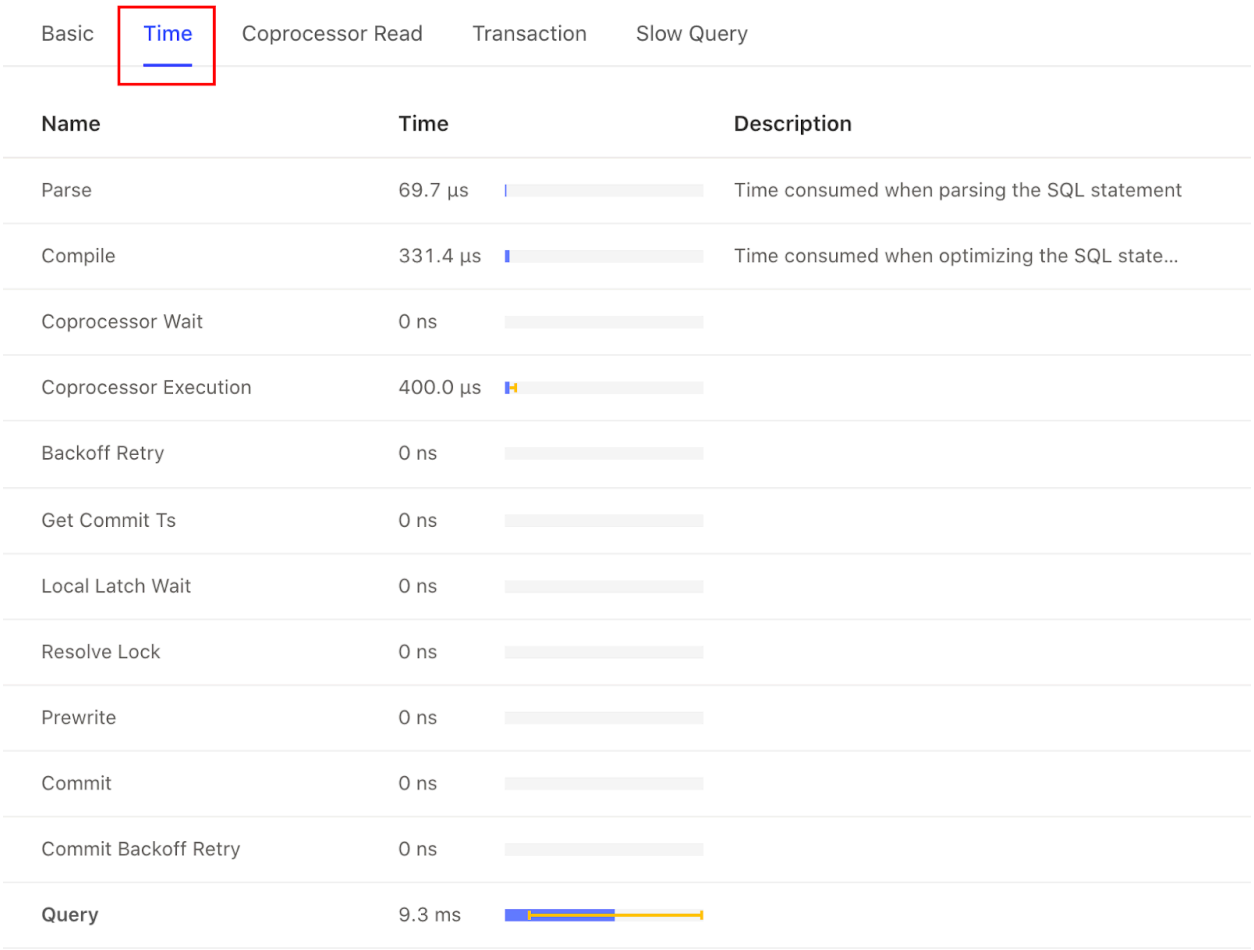


Figure 545: Execution time

Coprocessor Read Tab

Click the **Coprocessor Read** tab, and you can see information related to Coprocessor read.

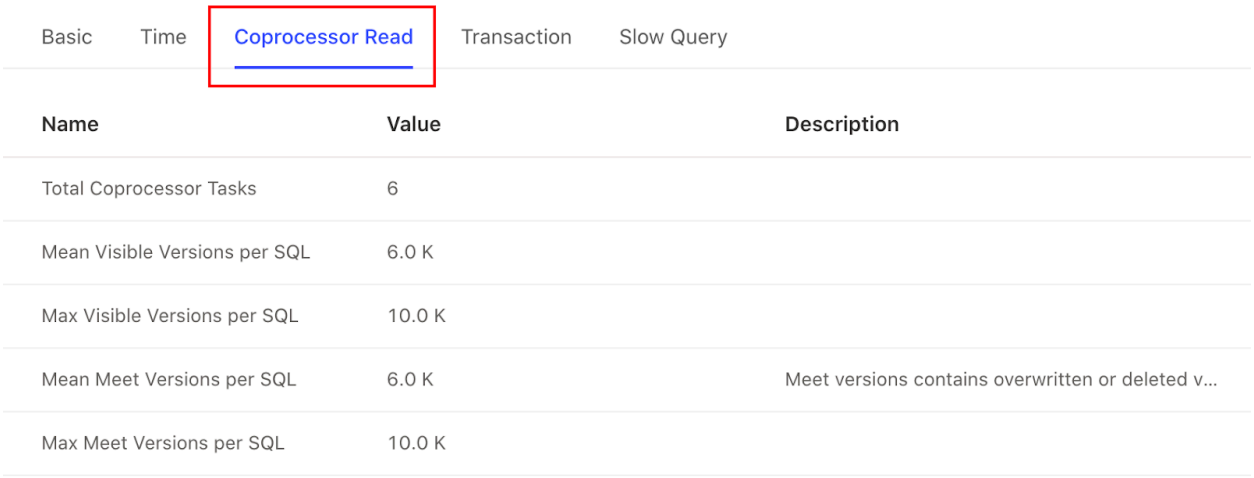


Figure 546: Coprocessor read

Transaction Tab

Click the **Transaction** tab, and you can see information related to execution plans and transactions, such as the average number of written keys or the maximum number of written keys.

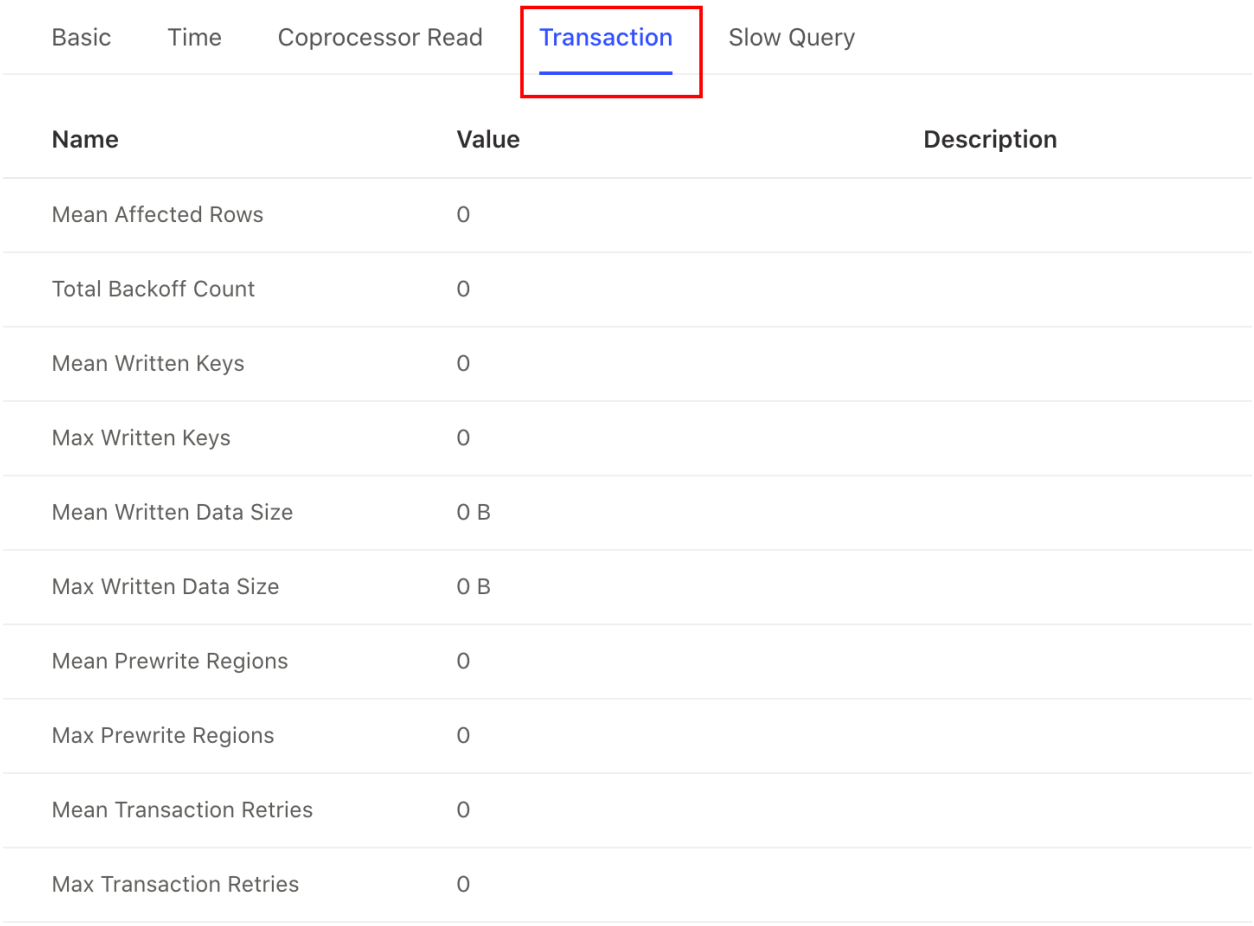


Figure 547: Transaction

Slow Query Tab

If an execution plan is executed too slowly, you can see its associated slow query records under the **Slow Query** tab.

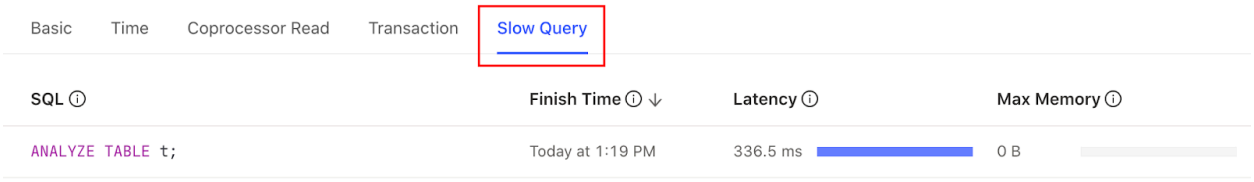


Figure 548: Slow Query

The information displayed in this area has the same structure with the slow query page. See TiDB Dashboard Slow Query Page for details.

14.11.1.10 Slow Queries Page of TiDB Dashboard

On the Slow Queries page of TiDB Dashboard, you can search and view all slow queries in the cluster.

By default, SQL queries with an execution time of more than 300 milliseconds are considered as slow queries. These queries are recorded in the slow query logs and can be searched via TiDB Dashboard. You can adjust the threshold of slow queries through the `tidb_slow_log_threshold` session variable or the `instance.tidb_slow_log_threshold` TiDB parameter.

> **Note:**
>
> If the slow query log is disabled, this feature will be unavailable. The slow query log is enabled by default, and you can enable or disable the slow query log through the `enable-slow-log` TiDB configuration item.

14.11.1.10.1 Access the page

You can use one of the following two methods to access the slow query page:

- After logging into TiDB Dashboard, click **Slow Queries** on the left navigation menu:

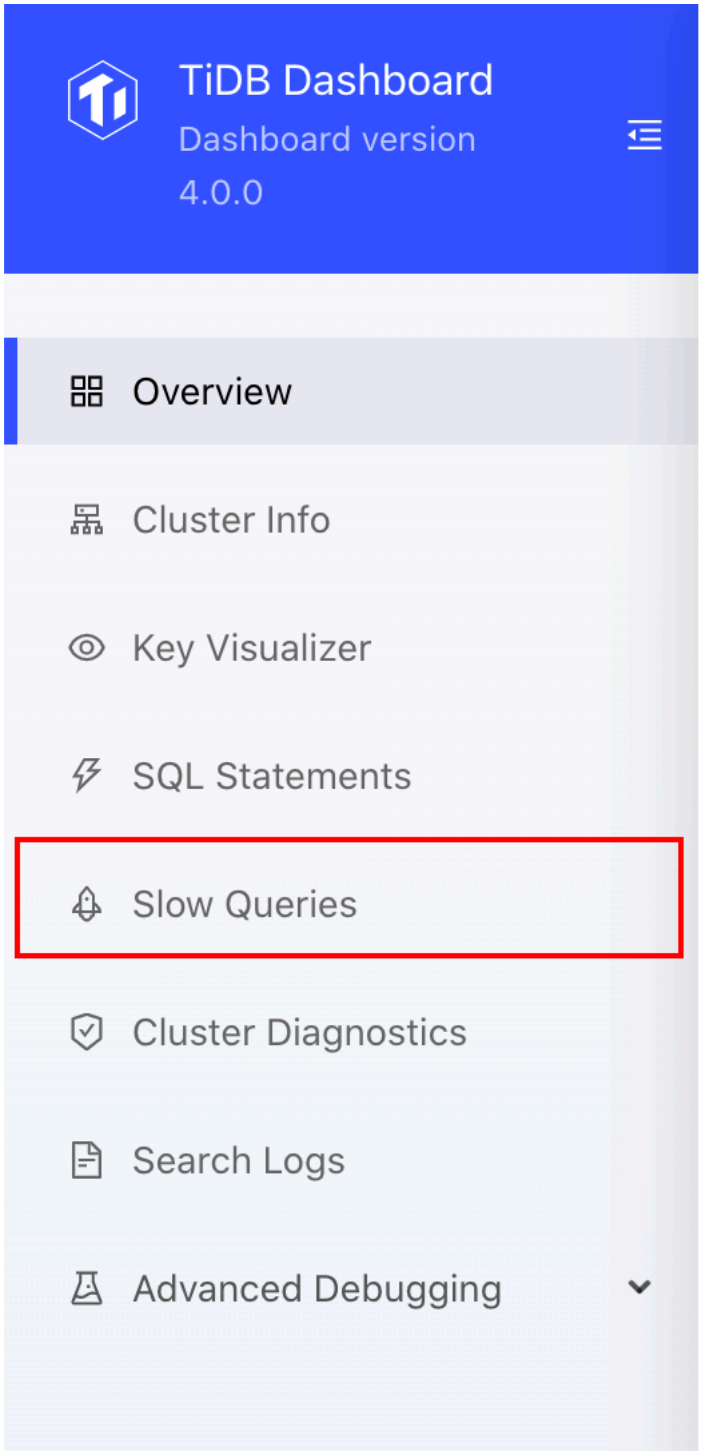


Figure 549: Access slow query page

- Visit http://127.0.0.1:2379/dashboard/#/slow_query in your browser. Replace `127.0.0.1:2379` with the actual PD address and port.

All data displayed on the slow query page comes from TiDB slow query system tables and slow query logs. See slow query logs for details.

Change Filters

You can filter slow queries based on the time range, the related database, SQL keywords, SQL types, the number of slow queries to be displayed. In the image below, 100 slow queries over the recent 30 minutes are displayed by default.

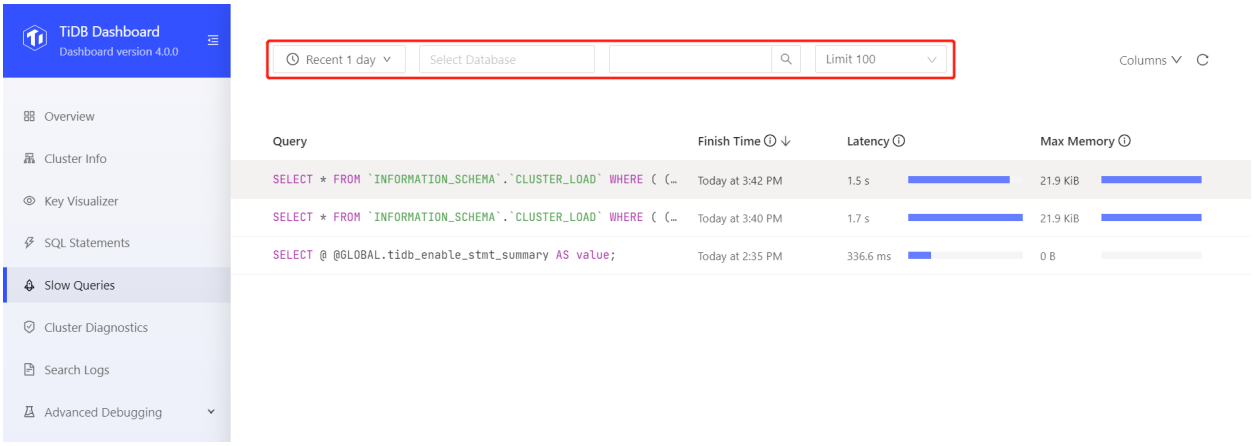


Figure 550: Modify list filters

Display More Columns

Click **Columns** on the page and you can choose to see more columns. You can move your mouse to the **(i)** icon at the right side of a column name to view the description of this column:

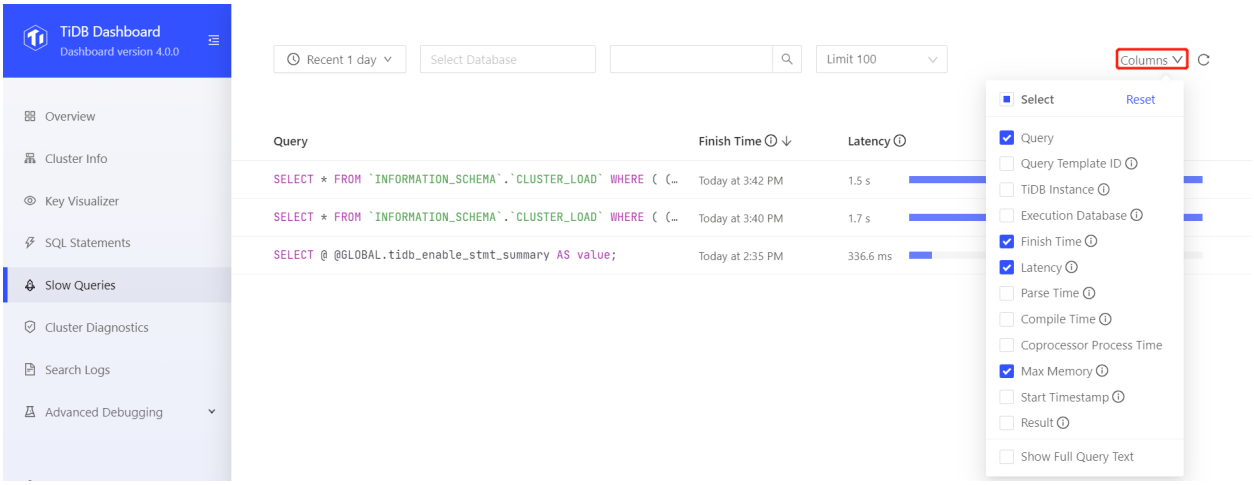


Figure 551: Show more columns

Sort by Column

By default, the list is sorted by **Finish Time** in the descending order. Click column headings to sort by the column or switch the sorting order:

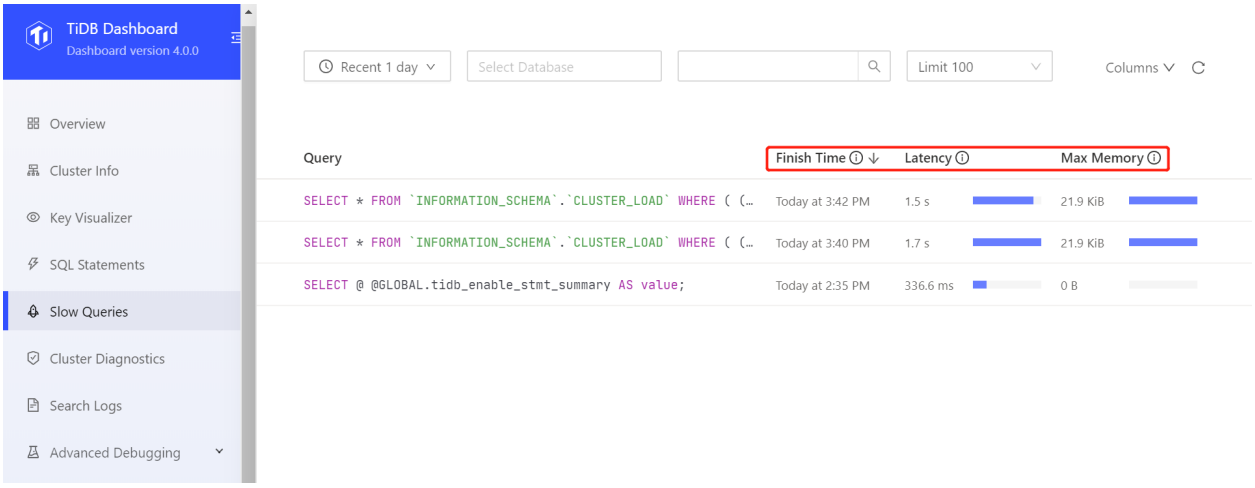


Figure 552: Modify sorting basis

14.11.1.10.2 View execution details

Click any item in the list to display detailed execution information of the slow query, including:

- Query: The text of the SQL statement (see area 1 in the image below);
- Plan: The execution plan of the slow query. See Understand the Query Execution Plan to learn how to read the execution plan (see area 2 in the image below);
- Other sorted SQL execution information (see area 3 in the image below).

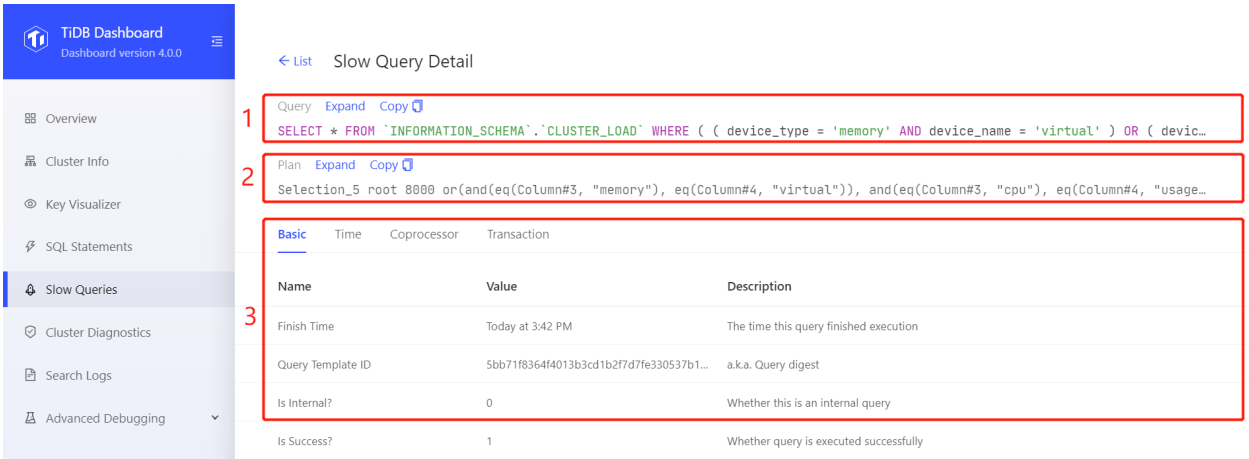


Figure 553: View execution details

Click the **Expand** link to show the detailed information of an item. Click the **Copy** link to copy the detailed information to the clipboard.

Click the corresponding tab titles to switch information of different sorted SQL executions.

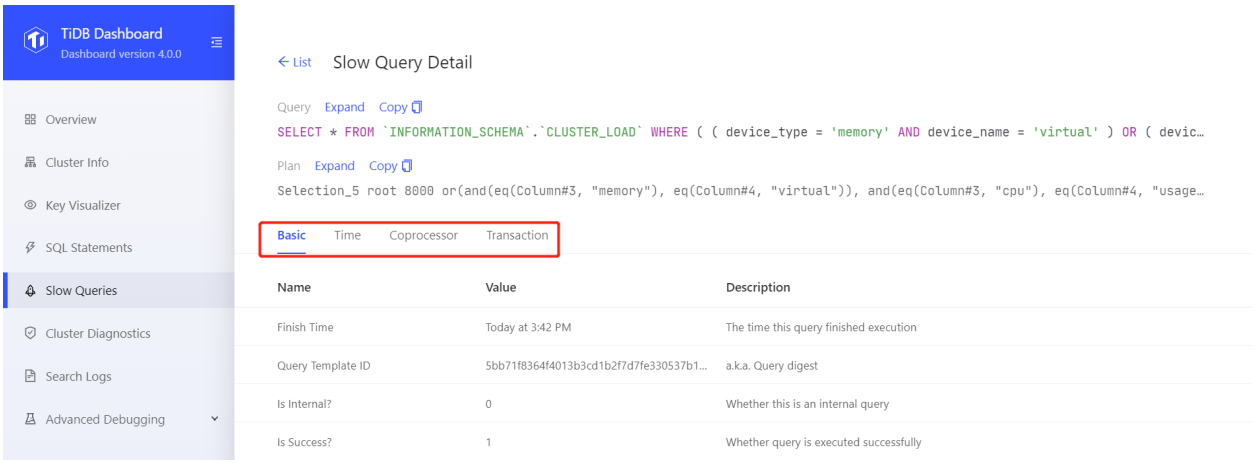


Figure 554: Show different sorted execution information

14.11.1.11 Cluster Diagnostics

14.11.1.11.1 TiDB Dashboard Cluster Diagnostics Page

> **Warning:**
>
> Diagnostics in TiDB Dashboard is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

The cluster diagnostics feature in TiDB Dashboard diagnoses the problems that might exist in a cluster within a specified time range, and summarizes the diagnostic results and the cluster-related load monitoring information into a diagnostic report. This diagnostic report is in the form of a web page. You can browse the page offline and circulate this page link after saving the page from a browser.

> **Note:**
>
> The cluster diagnostics feature depends on Prometheus deployed in the cluster. For details about how to deploy this monitoring component, see the

TiUP deployment document. If no monitoring component is deployed in the cluster, the generated diagnostic report will indicate a failure.

Access the page

You can use one of the following methods to access the cluster diagnostics page:

- After logging into TiDB Dashboard, click **Cluster Diagnostics** on the left navigation menu:

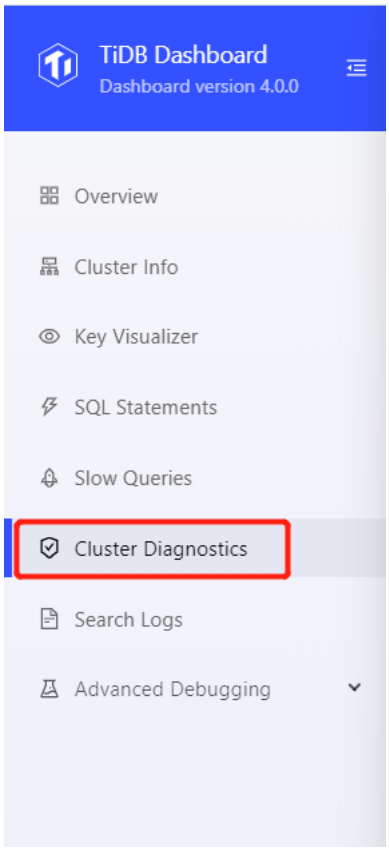


Figure 555: Access Cluster Diagnostics page

- Visit `http://127.0.0.1:2379/dashboard/#/diagnose` in your browser. Replace `127.0.0.1:2379` with the actual PD address and port number.

Generate diagnostic report

To diagnose a cluster within a specified time range and check the cluster load, you can take the following steps to generate a diagnostic report:

1. Set the **Range Start Time**, such as `2020-05-21 14:40:00`.
2. Set the **Range Duration**, such as `10 min`.
3. Click **Start**.

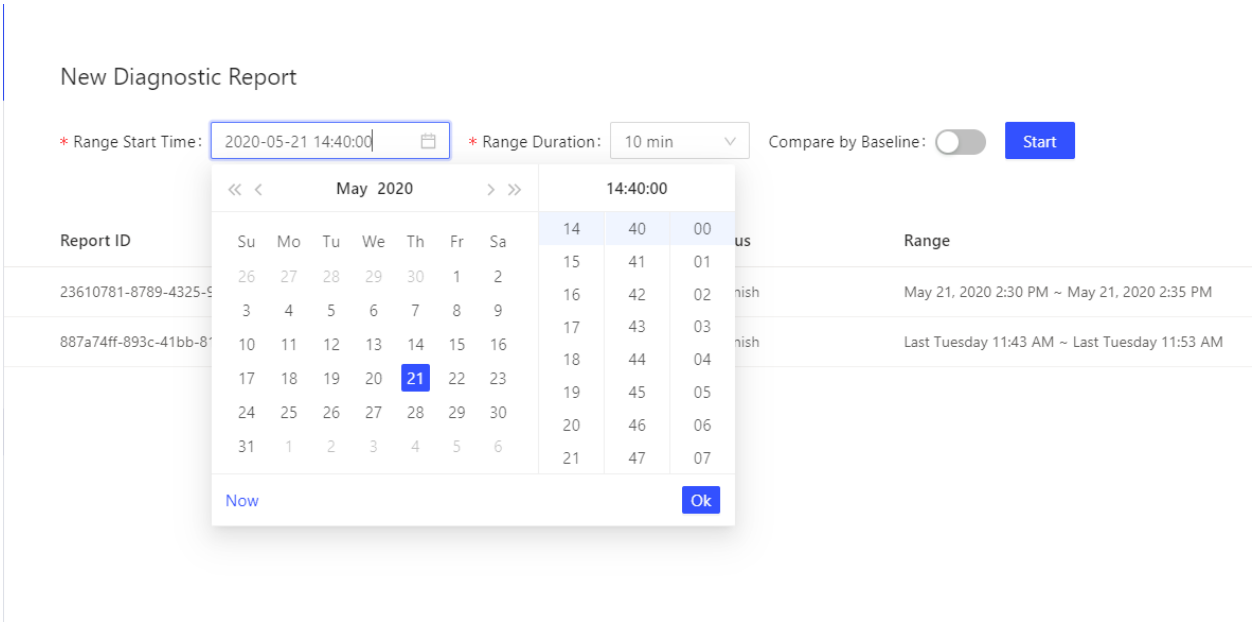


Figure 556: Generate diagnostic report

> **Note:**
>
> It is recommended that the **Range Duration** of the report is between 1 minute and 60 minutes. This **Range Duration** cannot exceed 60 minutes.

The steps above generate a diagnostic report for the time range from `2020-05-21 14:40:00` to `2020-05-21 14:50:00`. After clicking **Start**, you can see the interface below. **Progress** is the progress bar of the diagnostic report. After the report is generated, click **View Full Report**.

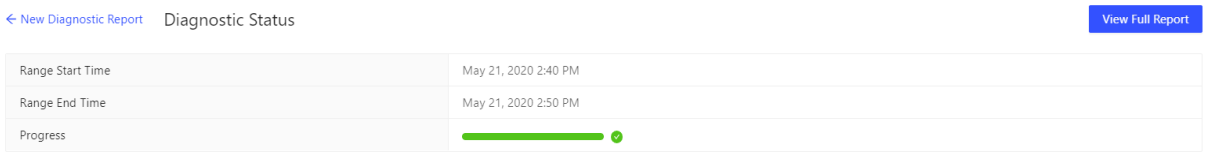


Figure 557: Report progress

Generate comparison report

If a system exception occurs at a certain point, for example, QPS jitter or higher latency, a diagnostic report can be generated. Particularly, this report compares the system in the abnormal time range with the system in the normal time range. For example:

- Abnormal time range: `2020-05-21 14:40:00-2020-05-21 14:45:00`. Within this time range, the system is abnormal.
- Normal time range: `2020-05-21 14:30:00 - 2020-05-21 14:35:00`. Within this time range, the system is normal.

You can take the following steps to generate a comparison report for the two time ranges above:

1. Set the **Range Start Time**, which is the start time of the range in which the system becomes abnormal, such as `2020-05-21 14:40:00`.
2. Set the **Range Duration**. Generally, this duration is the duration of system anomalies, such as 5 minutes.
3. Enable **Compare by baseline**.
4. Set the **Baseline Range Start Time**, which is the start time of the range (to be compared with) in which the system is normal, such as `2020-05-21 14:30:00`.
5. Click **Start**.

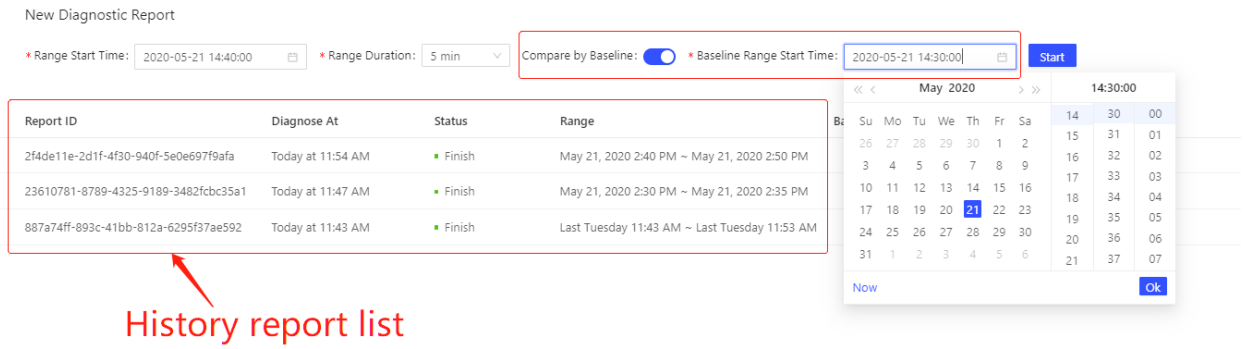


Figure 558: Generate comparison report

Then wait for the report to be generated and click **View Full Report**.

In addition, the historical diagnostic report is displayed in the list on the main page of the diagnostic report. You can click to view these historical reports directly.

14.11.1.11.2 TiDB Dashboard Diagnostic Report

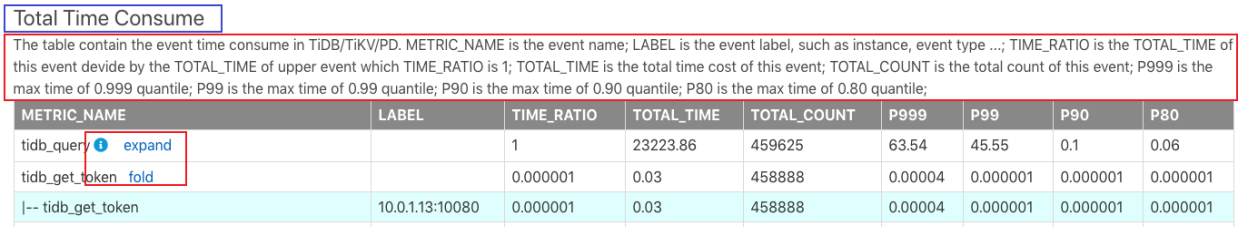
This document introduces the content of the diagnostic report and viewing tips. To access the cluster diagnostic page and generate reports, see TiDB Dashboard Cluster Diagnostics Page.

View report

The diagnostic report consists of the following parts:

- Basic information: Includes the time range of the diagnostic report, hardware information of the cluster, the version information of cluster topology.
- Diagnostic information: Shows the results of automatic diagnostics.
- Load information: Includes CPU, memory and other load information of the server, TiDB, PD, or TiKV.
- Overview information: Includes the consumed time and error information of each TiDB, PD, or TiKV module.
- TiDB/PD/TiKV monitoring information: Includes monitoring information of each component.
- Configuration information: Includes configuration information of each component.

An example of the diagnostic report is as follows:



Total Time Consume

The table contain the event time consume in TiDB/TiKV/PD. METRIC_NAME is the event name; LABEL is the event label, such as instance, event type ...; TIME_RATIO is the TOTAL_TIME of this event devide by the TOTAL_TIME of upper event which TIME_RATIO is 1; TOTAL_TIME is the total time cost of this event; TOTAL_COUNT is the total count of this event; P999 is the max time of 0.999 quantile; P99 is the max time of 0.99 quantile; P90 is the max time of 0.90 quantile; P80 is the max time of 0.80 quantile;

METRIC_NAME	LABEL	TIME_RATIO	TOTAL_TIME	TOTAL_COUNT	P999	P99	P90	P80
tidb_query expand		1	23223.86	459625	63.54	45.55	0.1	0.06
tidb_get_token fold		0.000001	0.03	458888	0.00004	0.000001	0.000001	0.000001
|-- tidb_get_token	10.0.1.13:10080	0.000001	0.03	458888	0.00004	0.000001	0.000001	0.000001

Figure 559: Sample report

In the image above, **Total Time Consume** in the top blue box is the report name. The information in the red box below explains the content of this report and the meaning of each field in the report.

In this report, some small buttons are described as follows:

- **i** icon: You can move your mouse to the **i** icon to see the explanatory note of the row.

- **expand**: Click **expand** to see details about this monitoring metric. For example, the detailed information of `tidb_get_token` in the image above includes the monitoring information of each TiDB instance's latency.
- **collapse**: Contrary to **expand**, the button is used to fold detailed monitoring information.

All monitoring metrics basically correspond to those on the TiDB Grafana monitoring dashboard. After a module is found to be abnormal, you can view more monitoring information on the TiDB Grafana.

In addition, the `TOTAL_TIME` and `TOTAL_COUNT` metrics in this report are monitoring data read from Prometheus, so calculation inaccuracy might exits in their statistics.

Each part of this report is introduced as follows.

Basic information

Diagnostics Time Range

The time range for generating the diagnostics report includes the start time and end time.

Report Time Range

START_TIME	END_TIME
2020-05-21 06:40:00	2020-05-21 06:50:00

Figure 560: Report time range

Cluster Hardware Info

Cluster Hardware Info includes information such as CPU, memory, and disk of each server in the cluster.

Cluster Hardware

HOST	INSTANCE	CPU_CORES	MEMORY (GB)	DISK (GB)	UPTIME (DAY)
10.0.1.14	tiflash*1	20/40	125.64	nvme01: 498.638	2.9132028513567314
10.0.1.11	pd*1	20/40	125.64	sda1: 965.834	2.921851028772416
10.0.1.12	tikv*1	20/40	125.64	nvme0n1: 122.78 sda1: 965.834	2.9197623066052247
10.0.1.13	tidb*1	20/40	125.64	sda1: 498.638	2.917983893669314

Figure 561: Cluster hardware report

The fields in the table above are described as follows:

- `HOST`: The IP address of the server.
- `INSTANCE`: The number of instances deployed on the server. For example, `pd * 1` means that this server has 1 PD instance deployed; `tidb * 2 pd * 1` means that this server has 2 TiDB instances and 1 PD instance deployed.

- `CPU_CORES`: Indicates the number of CPU cores (physical cores or logical cores) of the server.
- `MEMORY`: Indicates the memory size of the server. The unit is GB.
- `DISK`: Indicates the server disk size. The unit is GB.
- `UPTIME`: The uptime of the server. The unit is day.

Cluster Topology Info

The `Cluster Info` table shows the cluster topology information. The information in this table are from TiDB information_schema.cluster_info system table.

Cluster Info

TYPE	INSTANCE	STATUS_ADDRESS	VERSION	GIT_HASH	START_TIME	UPTIME
pd	10.0.1.11:2379	10.0.1.11:2379	4.1.0-alpha	a80b99ef0d70807d106bdc1b7c6e97a118679c7e	2020-05-18T13:47:01Z	65h40m54.520385474s
tidb	10.0.1.13:4000	10.0.1.13:10080	4.0.0-beta.2	ac30f5322e253125002663ad7c789807acfc9305	2020-05-18T13:47:10Z	65h40m45.520383242s
tiflash	10.0.1.14:3930	10.0.1.14:20292	v4.1.0-alpha-37-gacf3f673d	acf3f673dfbc3e6ffff0dda575760670993fefa6	2020-05-18T13:47:18Z	65h40m37.520390633s
tikv	10.0.1.12:20160	10.0.1.12:20180	4.1.0-alpha	6b09cadbf55311ac07fc51e1b43e3b57b54e71f2	2020-05-18T13:47:04Z	65h40m51.520389948s

Figure 562: Cluster info

The fields in the table above are described as follows:

- `TYPE`: The node type.
- `INSTANCE`: The instance address(es), which is a string in the `IP:PORT` format.
- `STATUS_ADDRESS`: The HTTP API service address.
- `VERSION`: The semantic version number of the corresponding node.
- `GIT_HASH`: Git Commit Hash when compiling the node version, which is used to identify whether the two nodes are absolutely the consistent version.
- `START_TIME`: The start time of the corresponding node.
- `UPTIME`: The uptime of the corresponding node.

Diagnostic information

TiDB has built-in automatic diagnostic results. For the description of each field, see information_schema.inspection-result system table.

Load Info

Node Load Info

The `Node Load Info` table shows the load information of the server node, including the average value (AVG), maximum value (MAX), minimum value (MIN) of the following metrics of the server within the time range:

- CPU usage (the maximum value is `100%`)

- Memory usage
- Disk I/O usage
- Disk write latency
- Disk read latency
- Disk read bytes per second
- Disk write bytes per second
- The number of bytes received by the node network per minute
- The number of bytes sent from the node network per minute
- The number of TCP connections in use by the node
- The number of all TCP connections of the node

Node Load Info

METRIC_NAME	instance	AVG	MAX	MIN
node_cpu_usage expand		40.42%	99.6%	6.98%
node_mem_usage expand		66.9%	96.28%	33.65%
node_disk_io_utilization expand		16%	97%	0%
Disk write latency expand		4000 us	50 ms	0 us
Disk read latency expand		2000 us	6000 us	0 us
tikv_disk_read_bytes expand		605.980 KB	15.842 MB	0.000 KB
tikv_disk_write_bytes expand		1.129 MB	12.235 MB	0.000 KB
node_network_in_traffic expand		225.706 KB	1.427 MB	0.000 KB
node_network_out_traffic expand		165.807 KB	1.659 MB	0.000 KB
node_tcp_in_use expand		19	47	5
node_tcp_connections expand		30	61	6

Figure 563: Server Load Info report

Instance CPU Usage

The `Instance CPU Usage` table shows the average value (AVG), maximum value (MAX), and minimum value (MIN) of the CPU usage of each TiDB/PD/TiKV process. The maximum CPU usage of the process is `100% * the number of CPU logical cores`.

INSTANCE	JOB	AVG	MAX	MIN
172.16.5.40:10089	tidb	1891%	1906%	1878%
172.16.5.40:22151	tikv	517%	571%	459%
172.16.5.40:20151	tikv	498%	562%	444%
172.16.5.40:23151	tikv	352%	399%	311%
172.16.5.40:24799	pd	39%	45%	36%

Figure 564: Instance CPU Usage report

Instance Memory Usage

The `Instance Memory Usage` table shows the average value (AVG), maximum value (MAX), and minimum value (MIN) of memory bytes occupied by each TiDB/PD/TiKV process.

INSTANCE	JOB	AVG	MAX	MIN
172.16.5.40:20151	tikv	9.562 GB	9.605 GB	9.523 GB
172.16.5.40:23151	tikv	9.528 GB	9.574 GB	9.496 GB
172.16.5.40:22151	tikv	9.433 GB	9.498 GB	9.345 GB
172.16.5.40:10089	tidb	904.500 MB	911.660 MB	894.336 MB
172.16.5.40:24799	pd	61.089 MB	61.270 MB	60.980 MB

Figure 565: Instance memory usage report

TiKV Thread CPU Usage

The `TiKV Thread CPU Usage` table shows the average value (AVG), maximum value (MAX) and minimum value (MIN) of CPU usage of each module thread in TiKV. The maximum CPU usage of the process is `100% * the thread count of the corresponding configuration`.

TiKV Thread CPU Usage

METRIC_NAME	INSTANCE	AVG	MAX	MIN	CONFIG_KEY	CURRENT_CONFIG_VALUE
grpc expand		14.83%	16%	12.96%		
schedule worker fold		8.76%	9.47%	7.56%		
|-- sched_worker	10.0.1.12:20180	8.76%	9.47%	7.56%	storage.scheduler-worker-pool-size	4
storage read pool expand		3.24%	3.64%	2.67%		
storage read pool normal expand		3.22%	3.64%	2.67%		
Async apply fold		3.14%	3.4%	2.71%		
|-- Async apply	10.0.1.12:20180	3.14%	3.4%	2.71%	raftstore.apply-pool-size	2
raftstore expand		2.7%	2.93%	2.38%		
rocksdb expand		0.54%	2.13%	0%		
unified read pool expand		0.45%	4.24%	0%		
split_check expand		0.26%	0.84%	0%		
gc expand		0.04%	0.4%	0%		
storage read pool high expand		0.02%	0.04%	0%		
snapshot expand		0.004%	0.02%	0%		
cop normal						
cop high						
cop low						
storage read pool low expand		0%	0%	0%		
cop						

Figure 566: TiKV Thread CPU Usage report

In the table above,

- `CONFIG_KEY`: The relevant thread configuration of the corresponding module.
- `CURRENT_CONFIG_VALUE`: The current value of the configuration when the report is generated.

Note:

`CURRENT_CONFIG_VALUE` is the value when the report is generated, not the value within the time range of this report. Currently, some configuration values of historical time cannot be obtained.

`TiDB/PD Goroutines Count`

The `TiDB/PD Goroutines Count` table shows the average value (AVG), maximum value (MAX), and minimum value (MIN) of the number of TiDB or PD goroutines. If the number of goroutines exceeds 2,000, the concurrency of the process is too high, which affects the overall request latency.

INSTANCE	JOB	AVG	MAX	MIN
172.16.5.40:10089	tidb	1434.33	1473	1394
172.16.5.40:24799	pd	157	157	157

Figure 567: TiDB/PD goroutines count report

Overview information

Time Consumed by Each Component

The `Time Consumed by Each Component` table shows the monitored consumed time and the time ratio of TiDB, PD, TiKV modules in the cluster. The default time unit is seconds. You can use this table to quickly locate which modules consume more time.

METRIC_NAME	LABEL	TIME_RATIO	TOTAL_TIME	TOTAL_COUNT	P999	P99	P90	P80
tidb_query expand		1	23223.86	459625	63.54	45.55	0.1	0.06
tidb_get_token expand		0.000001	0.03	458888	0.00004	0.000001	0.000001	0.000001
tidb_parse expand		0.00001	0.29	4603	0.0006	0.0004	0.0002	0.0001
tidb_compile expand		0.0008	17.82	463488	0.001	0.0009	0.0003	0.0002
tidb_execute expand		1	23237.39	463491	90.81	0.35	0.1	0.08
tidb_distsql_execution expand		0.0002	4.53	1064	0.13	0.1	0.02	0.01
tidb_cop expand		0.0005	12.39	1075	2.03	1.84	0.02	0.01
Transaction expand		1	23158.73	460023	8.16	7.91	5.38	0.06
tidb_transaction_local_latch_wait		0	0	0				
tidb_kv_backoff expand		0.00003	0.58	291	0.002	0.002	0.002	0.002
KV request expand		1.65	38325.58	2300949	2.03	1.84	0.04	0.03
Slow query expand		0.01	323.65	280	65.5	65.21	62.26	58.98
tidb_slow_query_cop_process		0	0	280	0.001	0.001	0.0009	0.0008
tidb_slow_query_cop_wait		0	0	280	0.001	0.001	0.0009	0.0008
DDL job		0	0	0				
tidb_ddl_worker		0	0	0				
tidb_ddl_update_self_version		0	0	0				
tidb_owner_handle_syncer		0	0	0				
Batch add index								
tidb_ddl_deploy_syncer		0	0	0				
Schema load expand		0.000008	0.19	27	0.06	0.06	0.06	0.05
TiDB meta operation expand		0.0001	3.21	1200	0.13	0.1	0.01	0.005
TiDB auto id request		0	0	0				
TiDB auto analyze expand		0.003	78.35	1	81.88	81.51	77.82	73.73
TiDB GC expand		0.000000001	0.00002	3	1	0.99	0.9	0.8
tidb_gc_push_task		0	0	0				
tidb_batch_client_unavailable		0	0	0				
tidb_batch_client_wait		0	0	0				
pd_tso_rpc expand		0.005	123.45	108540	0.02	0.008	0.003	0.002
pd_tso_wait expand		0.07	1662.29	920545	0.03	0.01	0.004	0.003
PD client cmd expand		0.15	3370.87	2761712	0.03	0.01	0.004	0.003
pd_client_request_rpc expand		0.005	123.45	108540	0.02	0.008	0.003	0.002
pd_grpc_completed_commands expand		0.00005	1.24	5255	0.01	0.01	0.006	0.005
pd_operator_finish								
pd_operator_step_finish								
Etcd transactions expand		0.000008	0.17	207	0.008	0.008	0.006	0.002
pd_region_heartbeat expand		0.00006	1.33	195	0	0	0	0
etcd_wal_fsync expand		0.00001	0.33	288	0.02	0.01	0.006	0.004
pd_peer_round_trip								

Figure 568: Time Consume report

The fields in columns of the table above are described as follows:

- `METRIC_NAME`: The name of the monitoring metric.
- `Label`: The label information for the monitoring metric. Click **expand** to view more detailed monitoring information of each label for a metric.
- `TIME_RATIO`: The ratio of the total time consumed by this monitoring metric to the total time of the monitoring row where `TIME_RATIO` is `1`. For example, the total

consumed time by `kv_request` is `1.65` (namely, `38325.58/23223.86`) times that of `tidb_query`. Because KV requests are executed concurrently, the total time of all KV requests might exceed the total query (`tidb_query`) execution time.

- `TOTAL_TIME`: The total time consumed by this monitoring metric.
- `TOTAL_COUNT`: The total number of times this monitoring metric is executed.
- `P999`: The maximum P999 time of this monitoring metric.
- `P99`: The maximum P99 time of this monitoring metric.
- `P90`: The maximum P90 time of this monitoring metric.
- `P80`: The maximum P80 time of this monitoring metric.

The following image shows the relationship of time consumption of the related modules in the monitoring metrics above.

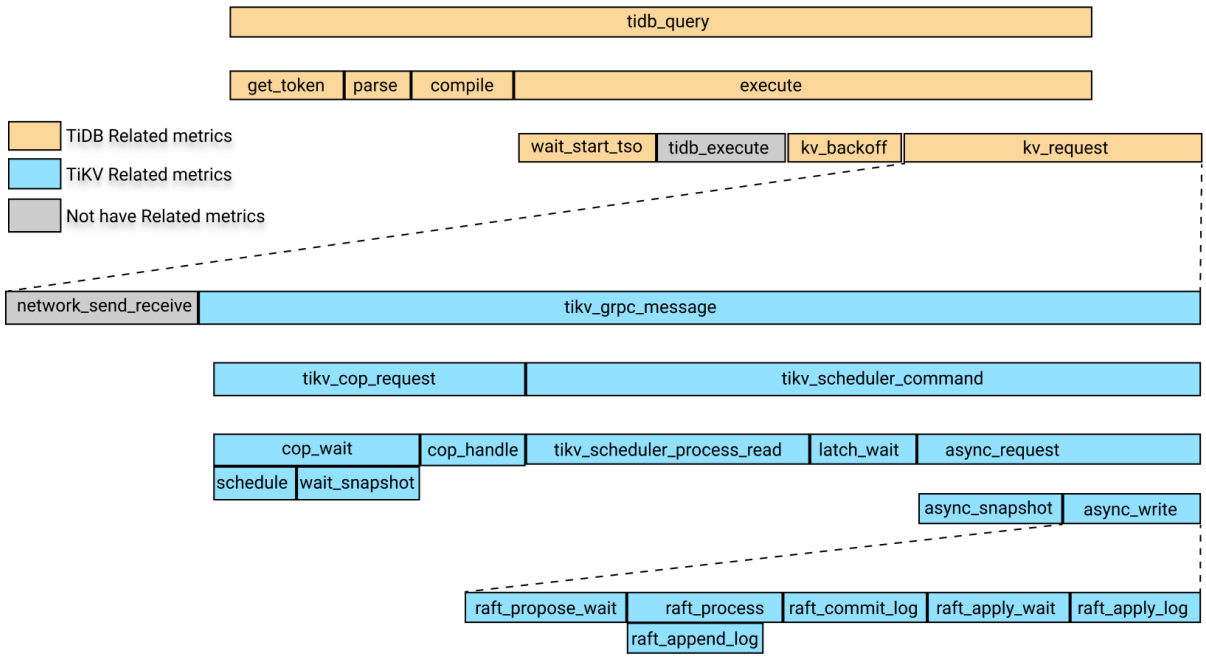


Figure 569: Time-consumption relationship of each module

In the image above, yellow boxes are the TiDB-related monitoring metrics.Blue boxes are TiKV-related monitoring metrics, and gray boxes temporarily do not correspond to specific monitoring metrics.

In the image above, the time consumption of `tidb_query` includes the following four parts:

- `get_token`
- `parse`
- `compile`

- `execute`

The `execute` time includes the following parts:

- `wait_start_tso`
- The execution time at the TiDB layer, which is currently not monitored
- KV request time
- `KV_backoff` time, which is the time for backoff after the KV request fails

Among the parts above, the KV request time includes the following parts:

- The time consumed by the network sending and receiving of requests. Currently, there is no monitoring metric for this item. You can subtract the time of `tikv_grpc_message` from the KV request time to roughly estimate this item.
- `tikv_grpc_message` time consumption.

Among the parts above, `tikv_grpc_message` time consumption includes the following parts:

- Coprocessor request time consumption, which refers to processing the COP type requests. This time consumption includes the following parts:
  - `tikv_cop_wait`: The time consumed by request queue.
  - `Coprocessor handle`: The time consumed to process Coprocessor requests.
- `tikv_scheduler_command` time consumption, which includes the following parts:
  - `tikv_scheduler_processing_read`: The time consumed to process read requests.
  - The time consumed to get snapshot in `tikv_storage_async_request` (snapshot is the label for this monitoring metric).
  - Time consumed to process write requests. This time consumption includes the following parts:
    - `tikv_scheduler_latch_wait`: The time consumed to wait for latch.
    - The time consumption of writes in `tikv_storage_async_request` (write is the label for this monitoring metric).

Among the above metrics, The time consumption of writes in `tikv_storage_async_request` refers to the time consumption of writing Raft KVs, including the following parts:

- `tikv_raft_propose_wait`
- `tikv_raft_process`, which mainly includes `tikv_raft_append_log`
- `tikv_raft_commit_log`
- `tikv_raft_apply_wait`

- `tikv_raft_apply_log`

You can use `TOTAL_TIME`, the P999 time, and the P99 time to determine which modules consume longer time according to the relationship between the time consumptions described above, and then look at the related monitoring metrics.

> **Note:**
>
> Because the Raft KVs writes might be processed in one batch, using `TOTAL_TIME` to measure the time consumed by each module is inapplicable to monitoring metrics related to Raft KV writes, specifically, `tikv_raft_process`, `tikv_raft_append_log`, `tikv_raft_commit_log`, `tikv_raft_apply_wait`, and `tikv_raft_apply_log`. In this situation, it is more reasonable to compare the time consumption of each module with the time of P999 and P99.
>
> The reason is that if there are 10 asynchronous write requests, Raft KVs internally pack 10 requests into a batch execution, and the execution time is 1 second. Therefore, the execution time of each request is 1 second, and the total time of 10 requests is 10 seconds, but the total time for Raft KV processing is 1 second. If you use `TOTAL_TIME` to measure the consumed time, you might not understand where the remaining 9 seconds are spent. You can also see the difference between the monitoring metric of Raft KV and other previous monitoring metrics from the total number of requests (`TOTAL_COUNT` ↪).

Errors Occurred in Each Component

The `Errors Occurred in Each Component` table shows the total number of errors in TiDB and TiKV, such as the failure to write binlog, `tikv server is busy`, `TiKV channel` ↪ `full`, `tikv write stall`. You can see the row comments for the specific meaning of each error.

METRIC_NAME	LABEL	TOTAL_COUNT
tidb_binlog_error_total_count		0
tidb_handshake_error_total_count		0
tidb_transaction_retry_error_total_count		
tidb_kv_region_error_total_count expand		903
tidb_schema_lease_error_total_count		0
tikv_grpc_error_total_count		0
tikv_critical_error_total_count		
tikv_scheduler_is_busy_total_count		
tikv_channel_full_total_count		
tikv_coprocessor_request_error_total_count expand		1
tikv_engine_write_stall		0
tikv_server_report_failures_total_count expand		1396
tikv_storage_async_request_error expand		1176
tikv_lock_manager_detect_error_total_count		
tikv_backup_errors_total_count		
node_network_in_errors_total_count		0
node_network_out_errors_total_count		0

Figure 570: Errors Occurred in Each Component report

Specific TiDB/PD/TiKV monitoring information

This part includes more specific monitoring information of TiDB, PD, or TiKV.

TiDB-related monitoring information

Time Consumed by TiDB Component

This table shows the time consumed by each TiDB module and the ratio of each time consumption, which is similar to the `time consume` table in the overview, but the label information of this table are more detailed.

TiDB Server Connections

This table shows the number of client connections for each TiDB instance.

TiDB Transaction

This table shows transaction-related monitoring metrics.

METRIC_NAME	LABEL	TOTAL_VALUE	TOTAL_COUNT	P999	P99	P90	P80
tidb_transaction_retry_num		0	0	0	0	0	0
tidb_transaction_statement_num expand		6854588	6852371	4	4	3	2
tidb_txn_region_num expand		1070770	706887	3	2	2	2
tidb_txn_kv_write_num expand		513386	181312	234	2	2	2
tidb_txn_kv_write_size expand		266.772 MB	181296	116.913 KB	1.996 KB	1.905 KB	1.805 KB
tidb_load_safepoint_total_num expand		16					
tidb_lock_resolver_total_num expand		420514					

Figure 571: Transaction report

- `TOTAL_VALUE`: The sum of all values (SUM) during the report time range.
- `TOTAL_COUNT`: The total number of occurrences of this monitoring metric.
- `P999`: The maximum P999 value of this monitoring metric.
- `P99`: The maximum P99 value of this monitoring metric.
- `P90`: The maximum P90 value of this monitoring metric.
- `P80`: The maximum P80 value of this monitoring metric.

Example:

In the table above, within the report time range, `tidb_txn_kv_write_size`: a total of about 181,296 transactions of KV writes, and the total KV write size is 266.772 MB, of which the maximum P999, P99, P90, P80 values for a single transaction of KV writes are 116.913 KB, 1.996 KB, 1.905 KB, and 1.805 KB.

DDL Owner

MIN_TIME	DDL OWNER
2020-05-21 14:40:00	10.0.1.13:10080

Figure 572: TiDB DDL Owner Report

The table above shows that from `2020-05-21 14:40:00`, the cluster's `DDL OWNER` is at the `10.0.1.13:10080` node. If the owner changes, multiple rows of data exist in the table above, where the `Min_Time` column indicates the minimum time of the corresponding known owner.

> **Note:**
>
> If the owner information is empty, it does not mean that no owner exists in this period of time. Because in this situation, the DDL owner is determined based on the monitoring information of `ddl_worker`, it might be that `ddl_worker` ↪ has not done any DDL job in this period of time, causing the owner information to be empty.

Other monitoring tables in TiDB are as follows:

- Statistics Info: Shows related monitoring metrics of TiDB statistical information.
- Top 10 Slow Query: Shows the Top 10 slow query information in the report time range.
- Top 10 Slow Query Group By Digest: Shows the Top 10 slow query information in the report time range, which is aggregated according to the SQL fingerprint.

- Slow Query With Diff Plan: The SQL statement whose execution plan changes within the report time range.

PD related monitoring information

The tables related to the monitoring information of PD modules are as follows:

- `Time Consumed by PD Component`: The time consumed by the monitoring metrics of related modules in PD.
- `Blance Leader/Region`: The monitoring information of `balance-region` and `balance leader` occurred in the cluster within the report time range, such as the number of leaders that are scheduled out from `tikv_note_1` or the number of leaders that are scheduled in.
- `Cluster Status`: The cluster status information, including total number of TiKV nodes, total cluster storage capacity, the number of Regions, and the number of offline TiKV nodes.
- `Store Status`: Record the status information of each TiKV node, including Region score, leader score, and the number of Regions/leaders.
- `Etcd Status`: etcd related information in PD.

TiKV related monitoring information

The tables related to the monitoring information of TiKV modules are as follows:

- `Time Consumed by TiKV Component`: The time consumed by related modules in TiKV.
- `Time Consumed by RocksDB`: The time consumed by RocksDB in TiKV.
- `TiKV Error`: The error information related to each module in TiKV.
- `TiKV Engine Size`: The size of stored data of column families on each node in TiKV.
- `Coprocessor Info`: Monitoring information related to the Coprocessor module in TiKV.
- `Raft Info`: Monitoring information of the Raft module in TiKV.
- `Snapshot Info`: Snapshot related monitoring information in TiKV.
- `GC Info`: Garbage Collection (GC) related monitoring information in TiKV.
- `Cache Hit`: The hit rate information of each cache of RocksDB in TiKV.

Configuration information

In the configuration information, the configuration values of some modules are shown within the report time range. But the historical values of some other configurations of these modules cannot be obtained, so the shown values of these configurations are the current (when the report is generated) values .

Within the report time range, the following tables include items whose values are configured at the start time of the report time range:

- `Scheduler Initial Config`: The initial value of PD scheduling-related configuration at the report's start time.
- `TiDB GC Initial Config`: The initial value of TiDB GC related-configuration at the report's start time
- `TiKV RocksDB Initial Config`: The initial value of TiKV RocksDB-related configuration at the report's start time
- `TiKV RaftStore Initial Config`: The initial value of TiKV RaftStore-related configuration at the report's start time

Within the report time range, if some configurations have been modified, the following tables include records of some configurations that have been modified:

- `Scheduler Config Change History`
- `TiDB GC Config Change History`
- `TiKV RocksDB Config Change History`
- `TiKV RaftStore Config Change History`

Example:

APPROXIMATE_CHANGE_TIME	CONFIG_ITEM	VALUE
2020-05-22T20:00:00+08:00	leader-schedule-limit	4
2020-05-22T20:07:00+08:00	leader-schedule-limit	8

Figure 573: Scheduler Config Change History report

The table above shows that the `leader-schedule-limit` configuration parameter has been modified within the report time range:

- `2020-05-22T20:00:00+08:00`: At the start time of the report, the configuration value of `leader-schedule-limit` is `4`, which does not mean that the configuration has been modified, but that at the start time in the report time range, its configuration value is `4`.
- `2020-05-22T20:07:00+08:00`: The `leader-schedule-limit` configuration value is `8` ↪ , which indicates that the value of this configuration has been modified around `2020-05-22T20:07:00+08:00`.

The following tables show the current configuration of TiDB, PD, and TiKV at the time when the report is generated:

- `TiDB's Current Config`
- `PD's Current Config`
- `TiKV's Current Config`

Comparison report

You can generate a comparison report for two time ranges. The report content is the same as the report for a single time range, except that a comparison column is added to show the difference between the two time ranges. The following sections introduce some unique tables in the comparison report and how to view the comparison report.

First, the `Compare Report Time Range` report in the basic information shows the two time ranges for comparison:

Compare Report Time Range

T1.START_TIME	T1.END_TIME	T2.START_TIME	T2.END_TIME
2020-05-21 06:30:00	2020-05-21 06:35:00	2020-05-21 06:40:00	2020-05-21 06:45:00

Figure 574: Compare Report Time Range report

In the table above, `t1` is the normal time range, or the reference time range. `t2` is the abnormal time range.

Tables related to slow queries are shown as follows:

- `Slow Queries In Time Range t2`: Shows slow queries that only appear in `t2` but not during `t1`.
- `Top 10 slow query in time range t1`: The Top 10 slow queries during `t1`.
- `Top 10 slow query in time range t2`: The Top 10 slow queries during `t2`.

DIFF_RATIO introduction

This section introduces `DIFF_RATIO` using the `Instance CPU Usage` table as an example.

Instance CPU Usage

INSTANCE	JOB	t1.AVG	t1.MAX	t1.MIN	t2.AVG	t2.MAX	t2.MIN	AVG_DIFF_RATIO	MAX_DIFF_RATIO	MIN_DIFF_RATIO
172.16.5.40:10089	tidb	410%	410%	410%	1240%	1240%	1240%	2.02	2.02	2.02
172.16.5.40:20151	tikv	221%	221%	221%	617%	617%	617%	1.79	1.79	1.79
172.16.5.40:20379	pd	82%	82%	82%	78%	78%	78%	-0.05	-0.05	-0.05

Figure 575: Compare Instance CPU Usage report

- `t1.AVG`, `t1.MAX`, `t1.Min` are the average value, maximum value, and minimum value of CPU usage in the `t1`.
- `t2.AVG`, `t2.MAX`, and `t2.Min` are the average value, maximum value, and minimum value of CPU usage during `t2`.
- `AVG_DIFF_RATIO` is `DIFF_RATIO` of the average values during `t1` and `t2`.
- `MAX_DIFF_RATIO` is `DIFF_RATIO` of the maximum values during `t1` and `t2`.
- `MIN_DIFF_RATIO` is `DIFF_RATIO` of the minimum values during `t1` and `t2`.

`DIFF_RATIO`: Indicates the difference value between the two time ranges. It has the following values:

- If the monitoring metric has a value only within `t2` and has no value within `t1`, the value of `DIFF_RATIO` is `1`.
- If the monitoring metric has a value only within `t1`, and has no value within `t2` time range, the value of `DIFF_RATIO` is `-1`.
- If the value of `t2` is greater than that of `t1`, then `DIFF_RATIO = (t2.value / t1.value)-1`
- If the value of `t2` is smaller than that of `t1`, then `DIFF_RATIO = 1-(t1.value / t2.value)`

For example, in the table above, the average CPU usage of the `tidb` node in `t2` is 2.02 times higher than that in `t1`, which is `2.02 = 1240/410-1`.

Maximum Different Item table

The `Maximum Different Item` table compares the monitoring metrics of two time ranges, and sorts them according to the difference of the monitoring metrics. Using this table, you can quickly find out which monitoring metric has the biggest difference in the two time ranges. See the following example:

Maximum Different Item

The maximum different metrics between two time ranges.

TABLE	METRIC_NAME	LABEL	MAX_DIFF	t1.VALUE	t2.VALUE	VALUE_TYPE
TiKV, coprocessor_info Expand	TiKV Coprocessor scan	172.16.5.40:20151,select,get,lock	28916.85	61.33	1773532	TOTAL_VALUE
TiDB, statistics_info	store_query_feedback_total_count	172.16.5.40:10089,ok	7219.00		7219	TOTAL_COUNT
overview, total_time_consume	tidb_parse	general	1201.00		1201	TOTAL_COUNT
TiDB, tidb_time_consume	tidb_slow_query_cop_wait	172.16.5.40:10089	800.00		800	TOTAL_COUNT
overview, total_time_consume	tidb_slow_query_cop_process	172.16.5.40:10089	800.00		800	TOTAL_COUNT
overview, total_time_consume	Slow query	172.16.5.40:10089	799.00	1	800	TOTAL_COUNT
TiKV, coprocessor_info	TiKV Coprocessor response	172.16.5.40:20151	534.43	15.822 MB	8.273 GB	TOTAL_VALUE
overview, total_time_consume	tidb_distsql_execution	dag	470.06	0.68	320.32	TOTAL_TIME
load, tikv_thread_cpu_usage	unified read pool	172.16.5.40:20151	412.29	0.07%	28.93%	MIN
TiKV, coprocessor_info Expand	TiKV Coprocessor scan keys	172.16.5.40:20151,select	198.39	1067055.38	212755381.33	TOTAL_VALUE
overview, total_time_consume Expand	tidb_ddl_worker	drop view	139.00		139	TOTAL_COUNT
overview, total_error Expand	tikv_storage_async_request_error	snapshot	89.00		89	TOTAL_COUNT
overview, total_time_consume Expand	Coprocessor handling request	index	78.99	0.67	53.59	TOTAL_TIME
overview, total_time_consume	tidb_cop	172.16.5.40:10089	75.11	5.93	451.31	TOTAL_TIME
overview, total_time_consume	KV request	Cop	75.10	5.93	451.26	TOTAL_TIME
overview, total_time_consume Expand	Coprocessor request	select	73.12	4.73	350.6	TOTAL_TIME
PD, pd_time_consume Expand	pd_region_heartbeat	172.16.5.40:20150,1	-73.00	73		TOTAL_COUNT
overview, total_time_consume Expand	tikv_grpc_messge	coprocessor	71.75	5.56	404.49	TOTAL_TIME
TiDB, tidb_time_consume	tidb_ddl_update_self_version	172.16.5.40:10089,ok	71.00		71	TOTAL_COUNT
overview, total_time_consume Expand	TiDB meta operation	update_ddl_job	71.00		71	TOTAL_COUNT
overview, total_time_consume Expand	tidb_owner_handle_syncer	update_global_version	71.00		71	TOTAL_COUNT
TiDB, tidb_time_consume Expand	tidb_query	172.16.5.40:10089,internal	-62.84	15.96	0.25	P999
TiKV, scheduler_info Expand	Scheduler stage	172.16.5.40:20151,scan_lock,snapshot_ok	55.45		55.45	TOTAL_VALUE
overview, total_time_consume Expand	tikv_grpc_messge	kv_scan_lock	55.00		55	TOTAL_COUNT
TiKV, tikv_time_consume Expand	tikv_scheduler_command	172.16.5.40:20151,scan_lock	55.00		55	TOTAL_COUNT
overview, total_time_consume Expand	PD client cmd	get_region	44.00		44	TOTAL_COUNT
overview, total_time_consume Expand	tikv_cop_wait	select	39.40	171	6908	TOTAL_COUNT
TiDB, tidb_time_consume	Schema load	172.16.5.40:10089	37.50	0.02	0.77	TOTAL_TIME
overview, total_time_consume	Schema load	172.16.5.40:10089	30.25	0.008	0.25	P999

Figure 576: Maximum Different Item table

- `Table`: Indicates this monitoring metric comes from which table in the comparison report. For example, `TiKV, coprocessor_info` indicates the `coprocessor_info` table in the TiKV component.
- `METRIC_NAME`: The monitoring metric name. Click `expand` to view the comparison of different labels of metrics.
- `LABEL`: The label corresponding to the monitoring metric. For example, the monitoring metric of `TiKV Coprocessor scan` has 2 labels, namely `instance`, `req`, `tag`, `sql_type` ↪ , which are the TiKV address, request type, operation type and operation column family.
- `MAX_DIFF`: Difference value, which is the `DIFF_RATIO` calculation of `t1.VALUE` and

`t2.VALUE`.

From the table above, you can see the `t2` time range has much more Coprocessor requests than the `t1` time range, and the SQL parsing time of TiDB in `t2` is much longer.

14.11.1.11.3 Locate Problems Using Diagnostic Report of TiDB Dashboard

This document introduces how to locate problems using diagnostic report of TiDB Dashboard.

Comparison diagnostics

This section demonstrates how to use the comparison diagnostic feature to diagnose QPS jitter or latency increase caused by large queries or writes.

Example 1

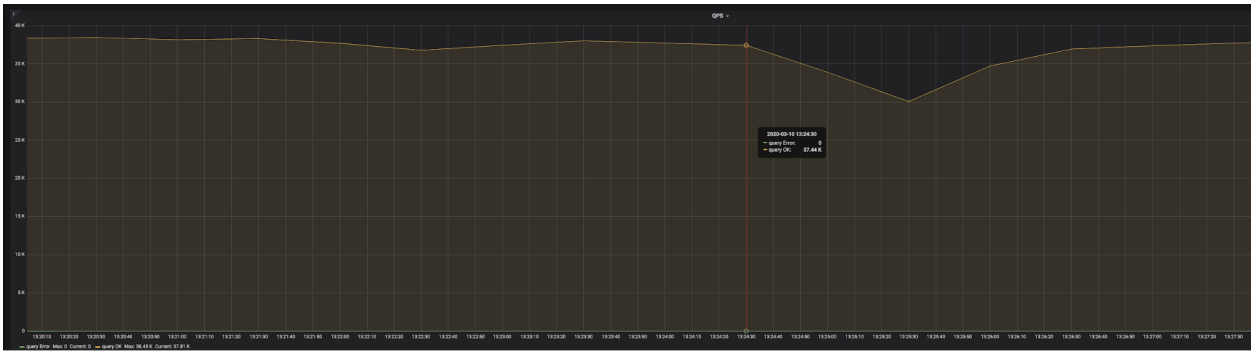


Figure 577: QPS example

The result of a `go-ycsb` pressure test is shown in the image above. You can see that at `2020-03-10 13:24:30`, QPS suddenly began to decrease. After 3 minutes, QPS began to return to normal. You can use diagnostic report of TiDB Dashboard to find out the cause.

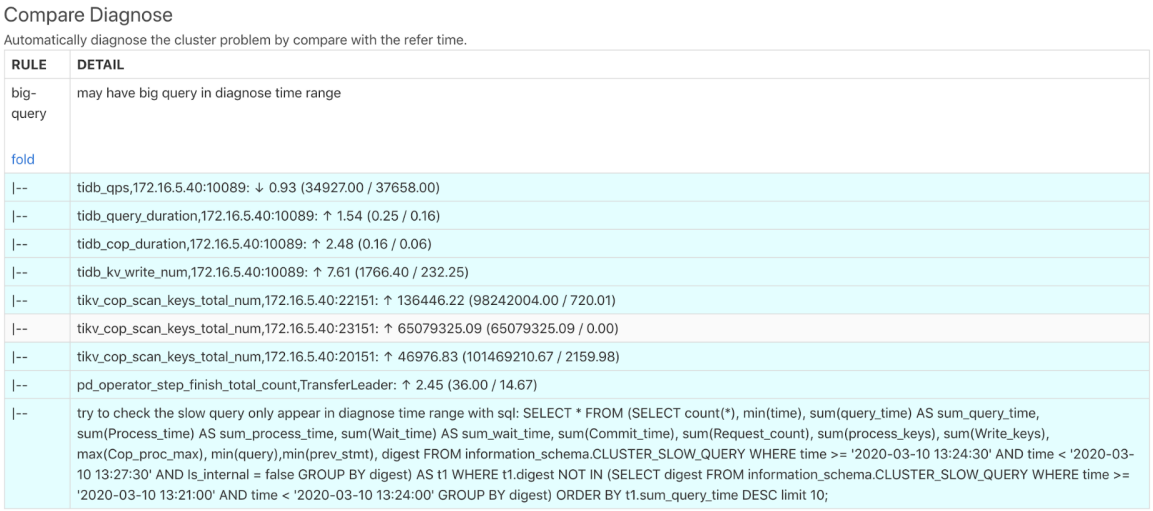
Generate a report that compares the system in the following two time ranges:

T1: `2020-03-10 13:21:00` to `2020-03-10 13:23:00`. In this range, the system is normal, which is called a reference range.

T2: `2020-03-10 13:24:30` to `2020-03-10 13:27:30`. In this range, QPS began to decrease.

Because the impact range of jitter is 3 minutes, the two time ranges above are both 3 minutes. Because some monitored average values are used for comparison during diagnostics, too long a range will cause the difference of average values to be insignificant, and then the problem cannot be accurately located.

After the report is generated, you can view this report on the **Compare Diagnose** page.



Compare Diagnose

Automatically diagnose the cluster problem by compare with the refer time.

RULE	DETAIL
big-query fold	may have big query in diagnose time range
|--	tidb_qps,172.16.5.40:10089: ↓ 0.93 (34927.00 / 37658.00)
|--	tidb_query_duration,172.16.5.40:10089: ↑ 1.54 (0.25 / 0.16)
|--	tidb_cop_duration,172.16.5.40:10089: ↑ 2.48 (0.16 / 0.06)
|--	tidb_kv_write_num,172.16.5.40:10089: ↑ 7.61 (1766.40 / 232.25)
|--	tikv_cop_scan_keys_total_num,172.16.5.40:22151: ↑ 136446.22 (98242004.00 / 720.01)
|--	tikv_cop_scan_keys_total_num,172.16.5.40:23151: ↑ 65079325.09 (65079325.09 / 0.00)
|--	tikv_cop_scan_keys_total_num,172.16.5.40:20151: ↑ 46976.83 (101469210.67 / 2159.98)
|--	pd_operator_step_finish_total_count,TransferLeader: ↑ 2.45 (36.00 / 14.67)
|--	try to check the slow query only appear in diagnose time range with sql: SELECT * FROM (SELECT count(*), min(time), sum(query_time) AS sum_query_time, sum(Process_time) AS sum_process_time, sum(Wait_time) AS sum_wait_time, sum(Commit_time), sum(Request_count), sum(process_keys), sum(Write_keys), max(Cop_proc_max), min(query),min(prev_stmt), digest FROM information_schema.CLUSTER_SLOW_QUERY WHERE time >= '2020-03-10 13:24:30' AND time < '2020-03-10 13:27:30' AND Is_internal = false GROUP BY digest) AS t1 WHERE t1.digest NOT IN (SELECT digest FROM information_schema.CLUSTER_SLOW_QUERY WHERE time >= '2020-03-10 13:21:00' AND time < '2020-03-10 13:24:00' GROUP BY digest) ORDER BY t1.sum_query_time DESC limit 10;

Figure 578: Comparison diagnostics

The diagnostic results above show that big queries might exist during the diagnostic time. Each **DETAIL** in the report above is described as follows:

- `tidb_qps`: QPS decreased by 0.93 times.
- `tidb_query_duration`: P999 query latency increased by 1.54 times.
- `tidb_cop_duration`: The processing latency of P999 coprocessor requests increased by 2.48 times.
- `tidb_kv_write_num`: The number written KVs in the P999 TiDB transaction increased by 7.61 times.
- `tikv_cop_scan_keys_total_nun`: The number of keys/values scanned by the TiKV Coprocessor has greatly improved on 3 TiKV instances.
- In `pd_operator_step_finish_total_count`, the number of transferred leaders increases by 2.45 times, which means that the scheduling in the abnormal time range is higher than that in the normal time range.
- The report indicates that there might be slow queries and indicates that you can use SQL statements to query slow queries. The execution result of the SQL statements are as follows:

```
SELECT * FROM (SELECT count(*), min(time), sum(query_time) AS
    ↪ sum_query_time, sum(Process_time) AS sum_process_time, sum(Wait_time)
    ↪  AS sum_wait_time, sum(Commit_time), sum(Request_count), sum(
    ↪ process_keys), sum(Write_keys), max(Cop_proc_max), min(query),min(
    ↪ prev_stmt), digest FROM information_schema.CLUSTER_SLOW_QUERY WHERE
    ↪ time >= '2020-03-10 13:24:30' AND time < '2020-03-10 13:27:30' AND
    ↪ Is_internal = false GROUP BY digest) AS t1 WHERE t1.digest NOT IN (
    ↪ SELECT digest FROM information_schema.CLUSTER_SLOW_QUERY WHERE time
```

```
    ↪ >= '2020-03-10 13:21:00' AND time < '2020-03-10 13:24:00' GROUP BY
    ↪ digest) ORDER BY t1.sum_query_time DESC limit 10\G
***************************[ 1. row ]***************************
count(*)          | 196
min(time)         | 2020-03-10 13:24:30.204326
sum_query_time    | 46.878509117
sum_process_time | 265.924
sum_wait_time     | 8.308
sum(Commit_time) | 0.926820886
sum(Request_count) | 6035
sum(process_keys) | 201453000
sum(Write_keys)  | 274500
max(Cop_proc_max) | 0.263
min(query)        | delete from test.tcs2 limit 5000;
min(prev_stmt)    |
digest            | 24
    ↪ bd6d8a9b238086c9b8c3d240ad4ef32f79ce94cf5a468c0b8fe1eb5f8d03df
```

From the result above, you can see that from `13:24:30`, there is a large write of batch deletion, which has been executed 196 times in total, each time deleting 5,000 rows of data, in a total duration of 46.8 seconds.

Example 2

If a large query has not been executed, the query is not recorded in the slow log. In this situation, this large query can still be diagnosed. See the following example:



Figure 579: QPS results

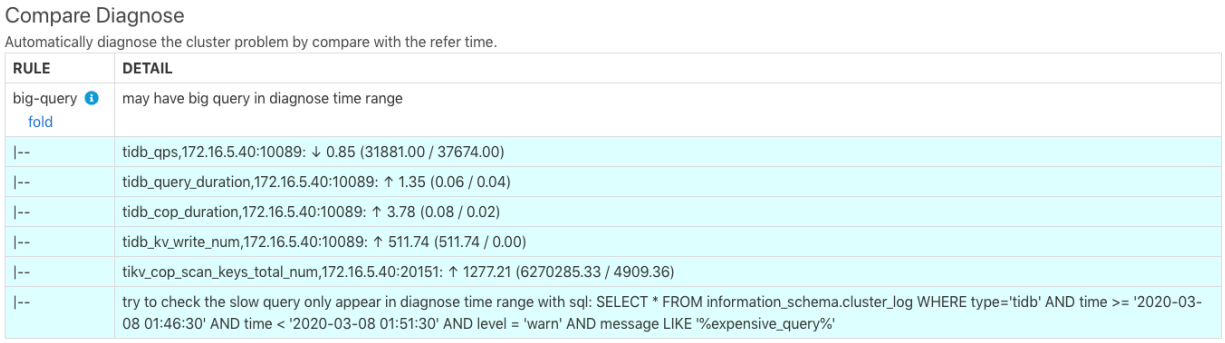
The result of another `go-ycsb` pressure test is shown in the image above. You can see that at `2020-03-08 01:46:30`, QPS suddenly began to drop and did not recover.

Generate a report that compares the system in the following two time ranges:

T1: `2020-03-08 01:36:00` to `2020-03-08 01:41:00`. In this range, the system is normal, which is called a reference range.

T2: `2020-03-08 01:46:30` to `2020-03-08 01:51:30`. In this range, QPS began to decrease.

After the report is generated, you can view this report on the **Compare Diagnose** page.



Compare Diagnose

Automatically diagnose the cluster problem by compare with the refer time.

RULE	DETAIL
big-query fold	may have big query in diagnose time range
|--	tidb_qps,172.16.5.40:10089: ↓ 0.85 (31881.00 / 37674.00)
|--	tidb_query_duration,172.16.5.40:10089: ↑ 1.35 (0.06 / 0.04)
|--	tidb_cop_duration,172.16.5.40:10089: ↑ 3.78 (0.08 / 0.02)
|--	tidb_kv_write_num,172.16.5.40:10089: ↑ 511.74 (511.74 / 0.00)
|--	tikv_cop_scan_keys_total_num,172.16.5.40:20151: ↑ 1277.21 (6270285.33 / 4909.36)
|--	try to check the slow query only appear in diagnose time range with sql: SELECT * FROM information_schema.cluster_log WHERE type='tidb' AND time >= '2020-03-08 01:46:30' AND time < '2020-03-08 01:51:30' AND level = 'warn' AND message LIKE '%expensive_query%'

Figure 580: Comparison diagnostics

The diagnostic result is similar to that of example 1. The last row of the image above indicates that there might be slow queries and indicate that you can use SQL statements to query the expensive queries in the TiDB log. The execution result of the SQL statements are as follows.

```
> SELECT * FROM information_schema.cluster_log WHERE type='tidb' AND time
    ↪ >= '2020-03-08 01:46:30' AND time < '2020-03-08 01:51:30' AND level =
    ↪  'warn' AND message LIKE '%expensive_query%'\G
TIME    | 2020/03/08 01:47:35.846
TYPE    | tidb
INSTANCE | 172.16.5.40:4009
LEVEL   | WARN
MESSAGE | [expensivequery.go:167] [expensive_query] [cost_time=60.085949605s
    ↪ ] [process_time=2.52s] [wait_time=2.52s] [request_count=9] [
    ↪ total_keys=996009] [process_keys=996000] [num_cop_tasks=9] [
    ↪ process_avg_time=0.28s] [process_p90_time=0.344s] [process_max_time
    ↪ =0.344s] [process_max_addr=172.16.5.40:20150] [wait_avg_time
    ↪ =0.000777777s] [wait_p90_time=0.003s] [wait_max_time=0.003s] [
    ↪ wait_max_addr=172.16.5.40:20150] [stats=t_wide:pseudo] [conn_id
    ↪ =19717] [user=root] [database=test] [table_ids="[80,80]"] [
    ↪ txn_start_ts=415132076148785201] [mem_max="23583169 Bytes
    ↪ (22.490662574768066 MB)"] [sql="select count(*) from t_wide as t1
    ↪ join t_wide as t2 where t1.c0>t2.c1 and t1.c2>0"]
```

The query result above shows that on this `172.16.5.40:4009` TiDB instance, at `2020/03/08 01:47:35.846` there is an expensive query that has been executed for 60

seconds, but the execution has not yet been finished. This query is a join of Cartesian products.

Locate problems using comparison diagnostic report

Because the diagnostics might be wrong, using a comparison report might help DBAs locate problems more quickly. See the following example.

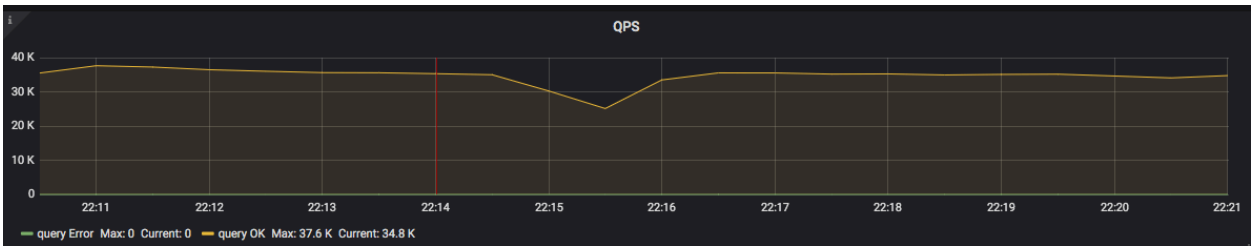


Figure 581: QPS results

The result of a `go-ycsb` pressure test is shown in the image above. You can see that at `2020-05-22 22:14:00`, QPS suddenly began to decrease. After 3 minutes, QPS began to return to normal. You can use the comparison diagnostic report of TiDB Dashboard to find out the cause.

Generate a report that compares the system in the following two time ranges:

T1: `2020-05-22 22:11:00` to `2020-05-22 22:14:00`. In this range, the system is normal, which is called a reference range.

T2: `2020-05-22 22:14:00 2020-05-22 22:17:00`. In this range, QPS began to decrease.

After generating the comparison report, check the **Max diff item** report. This report compares the monitoring items of the two time ranges above and sorts them according to the difference of the monitoring items. The result of this table is as follows:

Maximum Different Item

The maximum different metrics between two time ranges.

TABLE	METRIC_NAME	LABEL	MAX_DIFF	t1.VALUE	t2.VALUE	VALUE_TYPE
TiDB, transaction	Transaction KV write count	172.16.5.40:10089	-21616.00	43234	2	P999
TiKV, coprocessor_info Expand	TiKV Coprocessor scan	172.16.5.40:20151,select,get,lock	10499.24	64	672015.48	TOTAL_VALUE
TiKV, coprocessor_info Expand	TiKV Coprocessor scan keys	172.16.5.40:20151,select	4264.45	15431.85	65823838.67	TOTAL_VALUE
TiDB, transaction	Transaction KV write size	172.16.5.40:10089	-2702.69	5.278 MB	1.999 KB	P999
overview, total_time_consume	Coprocessor handling request	select	2132.50	0.06	128.01	TOTAL_TIME
TiKV, coprocessor_info	TiKV Coprocessor response	172.16.5.40:20151	1535.00	1.792 MB	2.688 GB	TOTAL_VALUE
TiDB, statistics_info	store_query_feedback_total_count	172.16.5.40:10089,ok	1447.00		1447	TOTAL_COUNT
TiKV, scheduler_info Expand	tikv_scheduler_keys_written	172.16.5.40:20151,commit	-860.00	861	1	P99
TiKV, tikv_time_consume	Coprocessor request	172.16.5.40:20151,select	853.87	0.15	128.23	TOTAL_TIME
load, tikv_thread_cpu_usage	unified read pool	172.16.5.40:20151	667.27	0.11%	73.51%	AVG
TiKV, tikv_time_consume	Coprocessor handling request	172.16.5.40:20151,index	444.00	0.002	0.89	P999
TiKV, tikv_time_consume Expand	tikv_grpc_messge	172.16.5.40:20151,coprocessor	419.97	0.33	138.92	TOTAL_TIME
load, node_load_info	tikv_disk_read_bytes	172.16.5.40:19110,sda	-273.41	0.267 KB	0.000 KB	MAX
overview, total_time_consume Expand	KV request	Cop	244.29	0.66	161.89	TOTAL_TIME

Figure 582: Comparison results

From the result above, you can see that the Coprocessor requests in T2 are much more than those in T1. It might be that some large queries appear in T2 that bring more load.

In fact, during the entire time range from T1 to T2, the `go-ycsb` pressure test is running. Then 20 `tpch` queries are running during T2. So it is the `tpch` queries that cause many Coprocessor requests.

If such a large query whose execution time exceeds the threshold of slow log is recorded in the slow log. You can check the `Slow Queries In Time Range t2` report to see whether there is any slow query. However, some queries in T1 might become slow queries in T2, because in T2, other loads cause their executions to slow down.

14.11.1.12 TiDB Dashboard Log Search Page

On the log search page of TiDB Dashboard, you can search logs of all nodes, preview the search result and download logs.

14.11.1.12.1 Access the page

After logging into TiDB Dashboard, you can click **Search Logs** to enter this log search homepage.

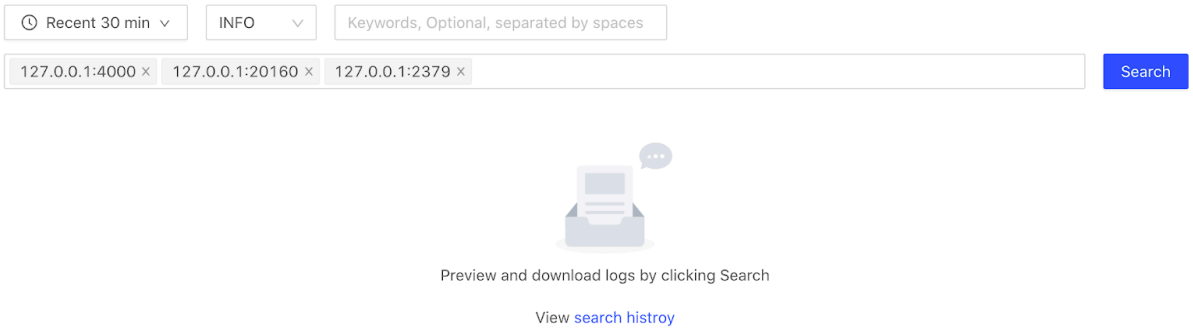


Figure 583: Log Search Page

This page provides the following search parameters:

- Time range: Specifies the time range of logs to search. The default value is the recent 30 minutes.
- Log level: Specifies the minimum log level. All logs above this log level are searched. The default value is the `INFO`.
- Keywords: The parameter is optional and its value can be any legal string. Multiple keywords are separated by a space. Regular expressions are supported (case-insensitive).
- Components: Selects the cluster components to search, which are multi-select and non-empty. By default, all components are selected.

After clicking the **Search** button, you enter the detail page of the search results.

14.11.1.12.2 Page of search result

The following image shows the page of the search results.



Figure 584: Search result

This page consists of the following three areas:

- Parameter options (area 1 in the image above): These options are the same as the parameter options on the search homepage. You can re-select the parameters in the boxes and start a new search.
- Progress (area 2 in the image above): The current search progress is shown on the right side of this page, including the log search status and statistics of each node.
- Search results (area 3 in the image above):
    - Time: The time at which the log is generated. The time zone is the same as the time zone of the front-end user.
    - Level: log level.
    - Component: Shows the component name and its address.
    - Log: The body part of each log record, excluding the log time and log level. Logs that are too long are automatically truncated. Click a row to expand the full content. The full log can show up to 512 characters.

Note:

At most 500 search results can be previewed on this page. You can get the complete search results by downloading them.

Search progress

In the search progress area, a search on a node is called a search task. A search task might have the following statuses:

- Running: After starting the search, all tasks enter the **Running** status.
- Success: After the task is completed, it automatically enters the **Success** status. At this time, the logs have been cached in the local disk where the Dashboard backend is located, and can be provided to the frontend to download.
- Failed: When you cancel the search task, or the task exits with an error, the task enters the **Failed** status. When the task fails, the local temporary files are automatically cleaned.

The search progress area has the following three control buttons:

- **Download Selected**: Click this button to download logs of the selected components (only the completed ones can be selected), and you will get a tar file. Unzip this tar file to get one or more zip files (each component corresponds to a zip file). Unzip the zip file(s) to get the log text file.
- **Cancel**: Click this button to cancel all running tasks. You can click this button only when there are running tasks.
- **Retry**: Click this button to retry all failed tasks. You can click this button only when there are failed tasks and no running tasks.

14.11.1.12.3 Search history list

Click the **View search history** link on the log search homepage to enter page of search history list:

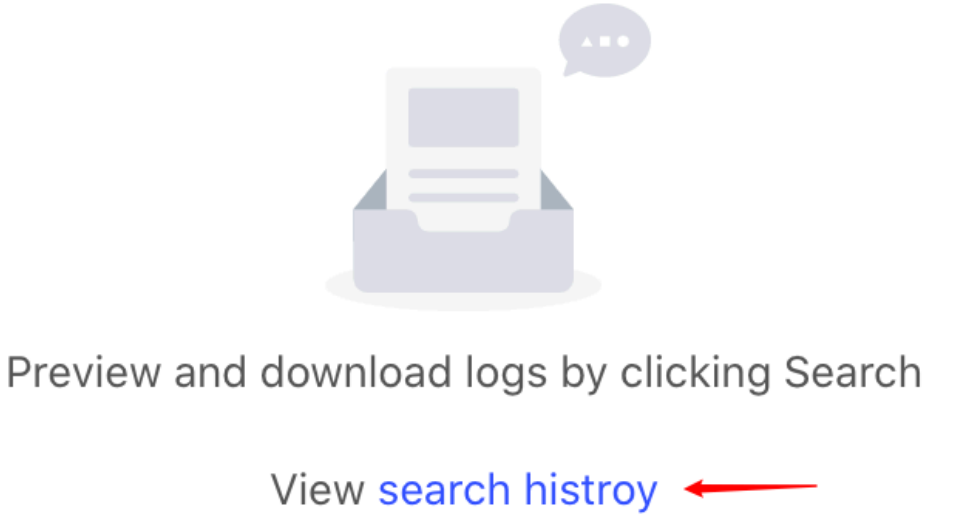


Figure 585: Search history entry

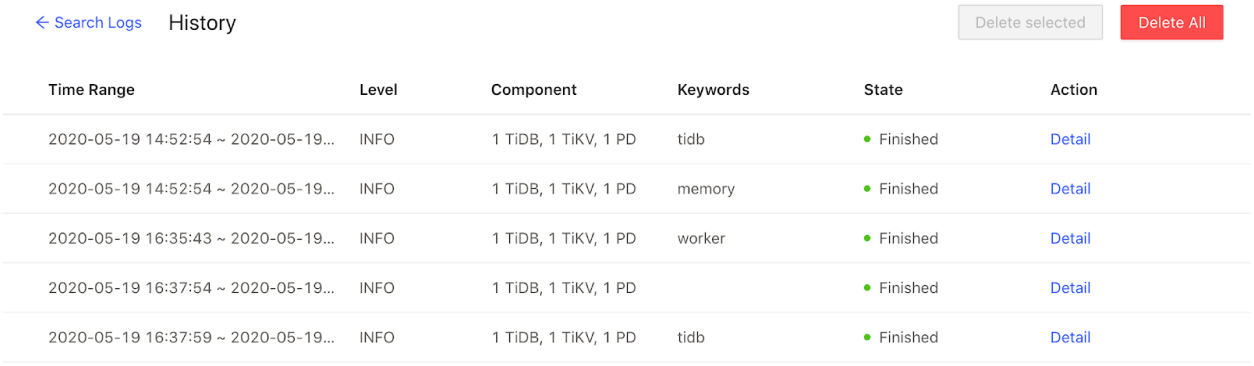


Figure 586: Search history list

The history list shows the time range, log level, components, keywords, and search status of each search log. Click the **Detail** link in the **Action** column to see the search result details:

You can delete the search history that you no longer need. Click **Delete All** in the upper right corner, or select the rows to be deleted and then click **Delete selected** to delete the history:

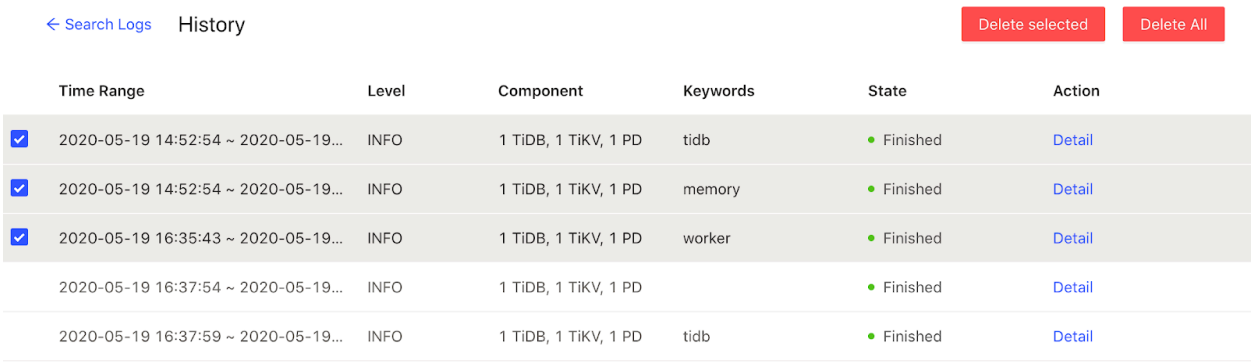


Figure 587: Delete search history

14.11.1.13 Instance Profiling

14.11.1.13.1 TiDB Dashboard Instance Profiling - Manual Profiling

> **Note:**
>
> This feature is designed for database experts. For non-expert users, it is recommended to use this feature under the guidance of PingCAP technical supports.

Manual Profiling allows users to collect current performance data **on demand** for each TiDB, TiKV, PD and TiFlash instances with a single click. The collected performance data can be visualized as FlameGraph or DAG.

With these performance data, experts can analyze current resource consumption details like instance's CPU and memory, to help pinpoint sophisticated ongoing performance problems, such as high CPU overhead, high memory usage, and process stalls.

After initiates the profiling, TiDB Dashboard collects current performance data for a period of time (30 seconds by default). Therefore this feature can only be used to analyze ongoing problems that the cluster is facing now and has no significant effect on historical problems. If you want to collect and analyze performance data **at any time**, see Continuous Profiling.

Supported performance data

The following performance data are currently supported:

- CPU: The CPU overhead of each internal function on TiDB, TiKV, PD and TiFlash instances

> The CPU overhead of TiKV and TiFlash instances is currently not supported in ARM architecture.

- Heap: The memory consumption of each internal function on TiDB and PD instances
- Mutex: The mutex contention states on TiDB and PD instances
- Goroutine: The running state and call stack of all goroutines on TiDB and PD instances

Access the page

You can access the instance profiling page using either of the following methods:

- After logging into TiDB Dashboard, click **Advanced Debugging** > **Profiling Instances** > **Manual Profiling** on the left navigation bar.

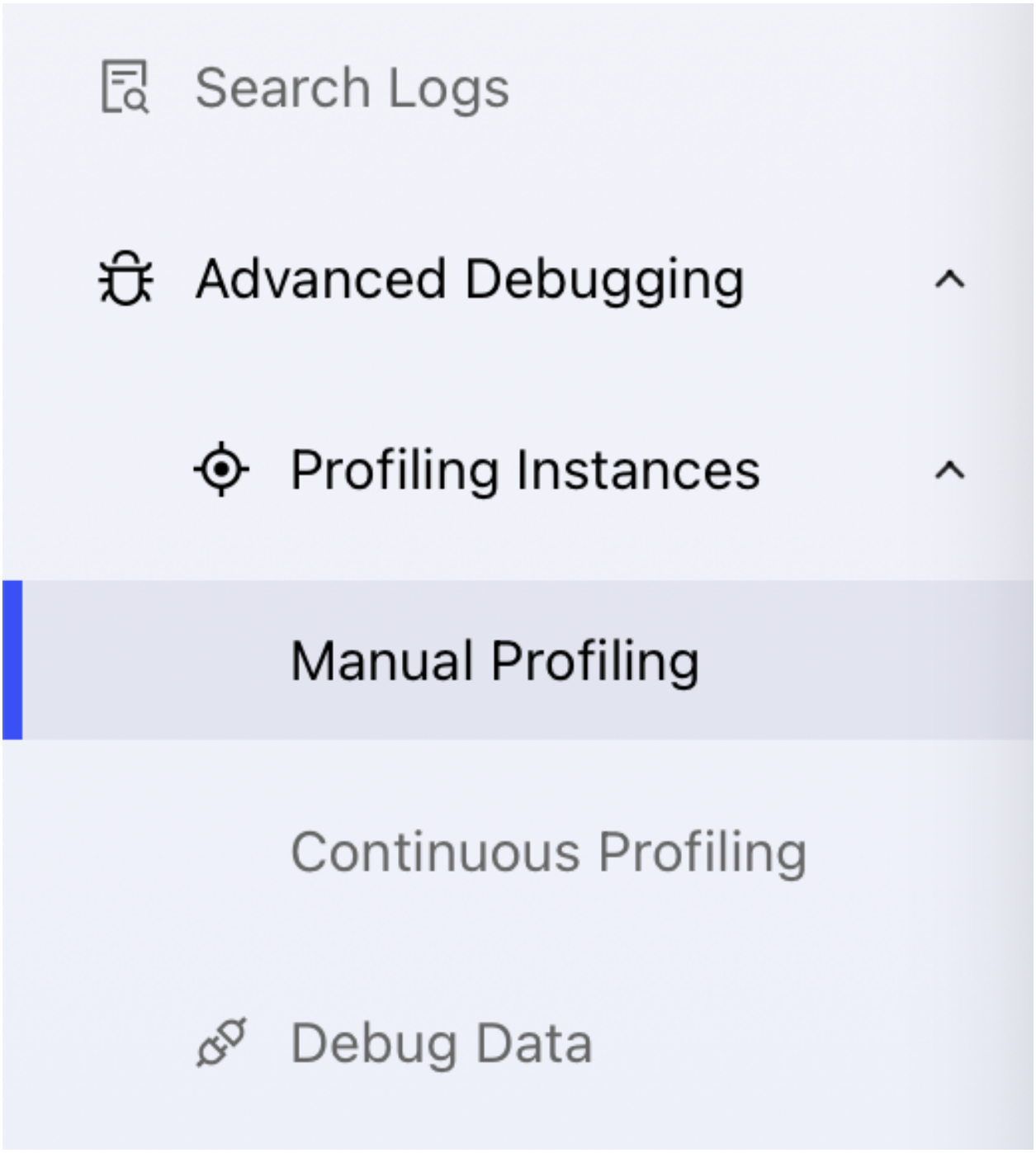


Figure 588: Access instance profiling page

- Visit http://127.0.0.1:2379/dashboard/#/instance_profiling in your browser. Replace `127.0.0.1:2379` with the actual PD instance address and port.

Start Profiling

In the instance profiling page, choose at least one target instance and click **Start Profiling** to start the instance profiling.

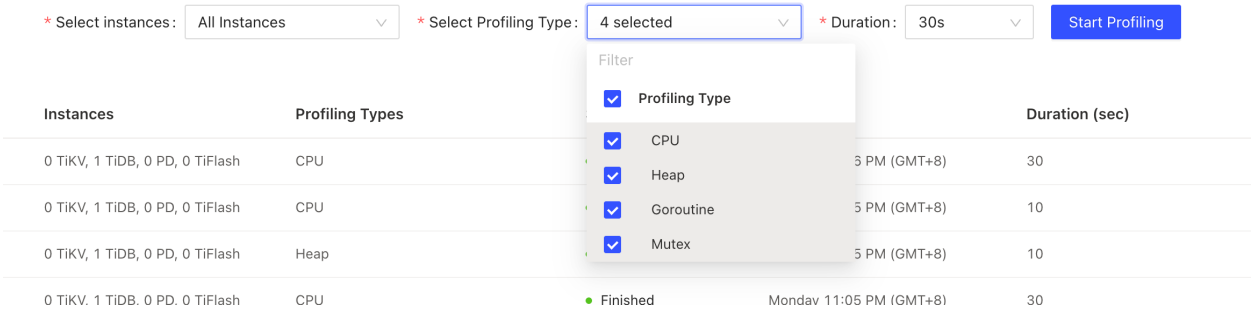


Figure 589: Start instance profiling

You can modify the profiling duration before starting the profiling. This duration is determined by the time needed for the profiling, which is 30 seconds by default. The 30-second duration takes 30 seconds to complete.

Manual Profiling cannot be initiated on clusters that have Continuous Profiling enabled. To view the performance data at the current moment, click on the most recent profiling result in the Continuous Profiling page.

View profiling status

After a profiling is started, you can view the profiling status and progress in real time.

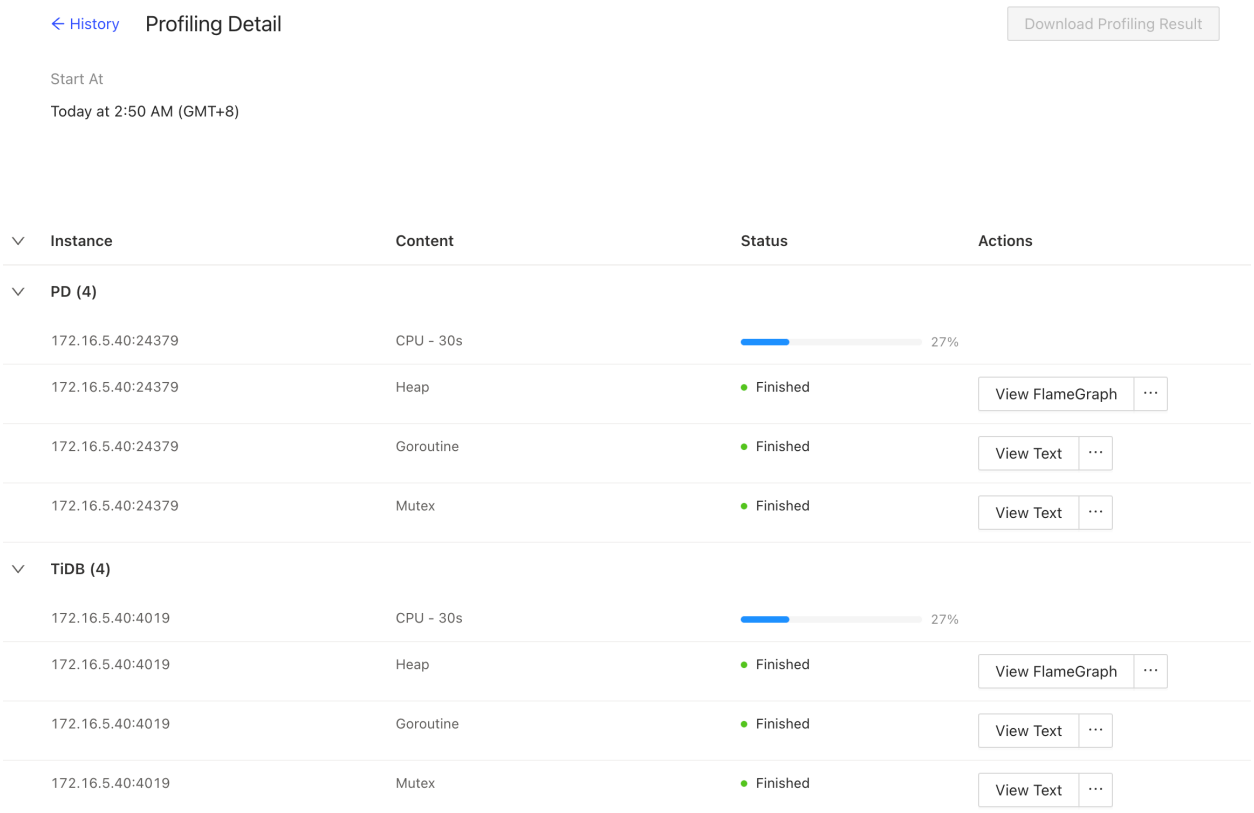


Figure 590: Profiling detail

The profiling runs in the background. Refreshing or exiting the current page does not stop the profiling task that is running.

Download performance data

After the profiling of all instances is completed, you can click **Download Profiling Result** in the upper right corner to download all performance data.

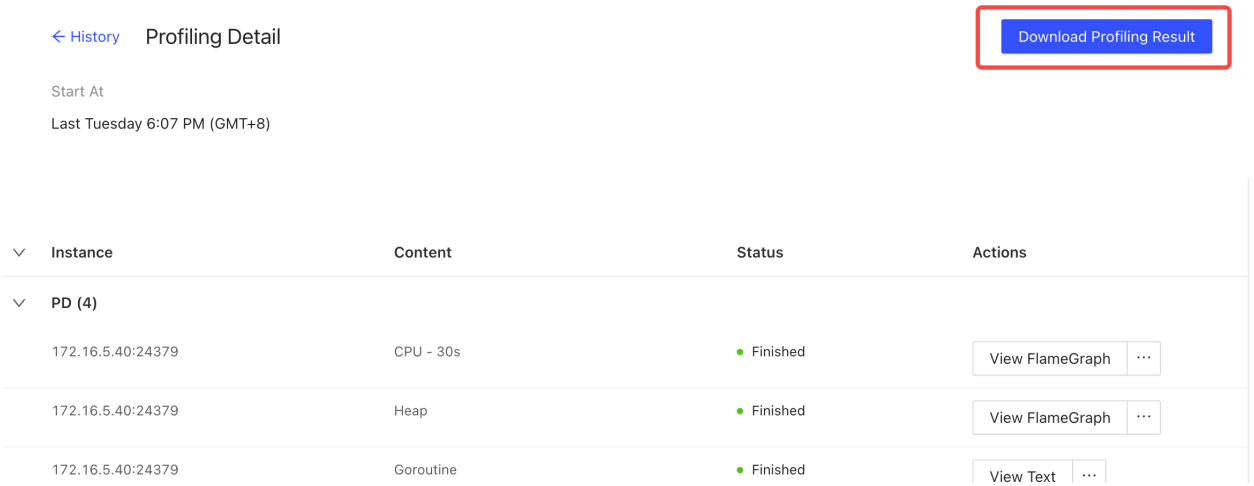


Figure 591: Download profiling result

You can also click an individual instance in the table to view its profiling result. Alternatively, you can hover on … to download raw data.

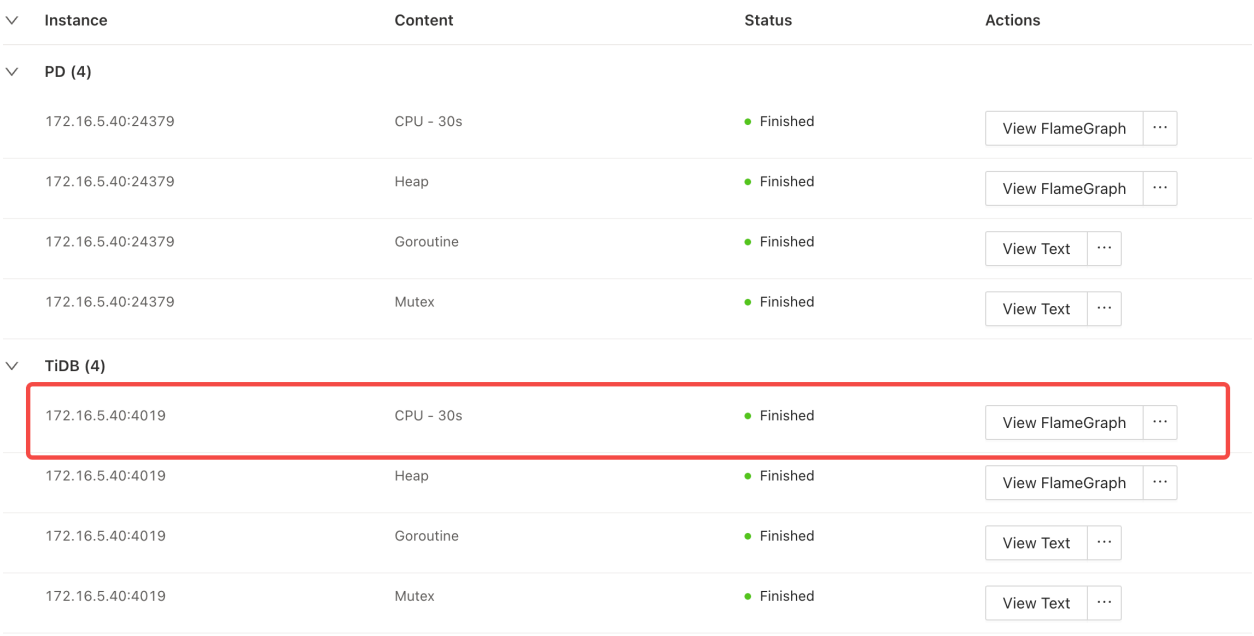
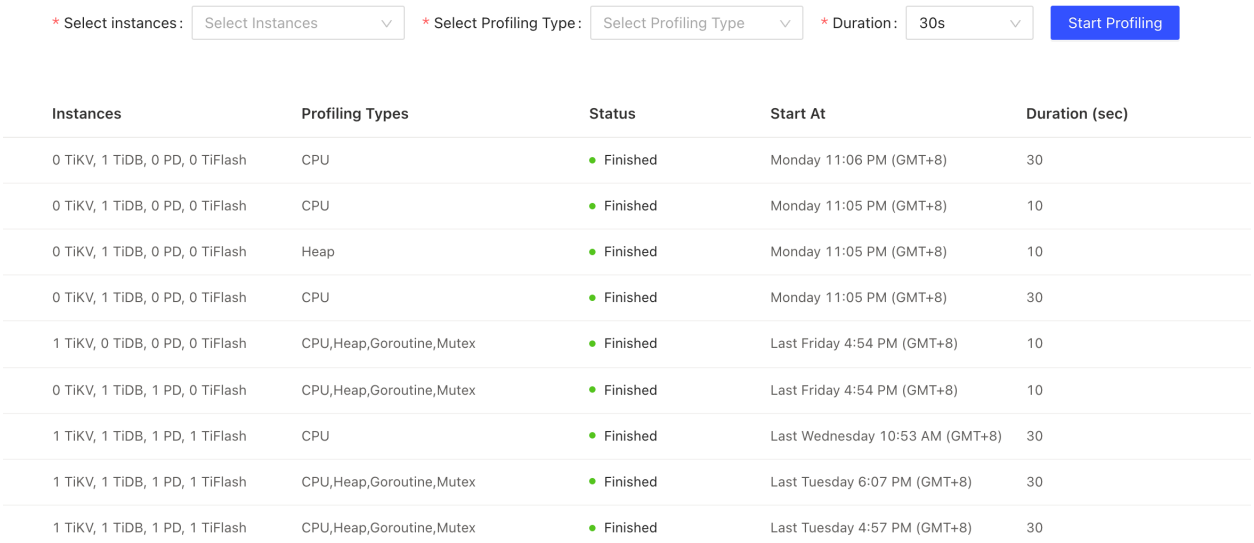


Figure 592: Single instance result

View profiling history

The on-demand profiling history is listed on the page. Click a row to view details.



Instances	Profiling Types	Status	Start At	Duration (sec)
0 TiKV, 1 TiDB, 0 PD, 0 TiFlash	CPU	Finished	Monday 11:06 PM (GMT+8)	30
0 TiKV, 1 TiDB, 0 PD, 0 TiFlash	CPU	Finished	Monday 11:05 PM (GMT+8)	10
0 TiKV, 1 TiDB, 0 PD, 0 TiFlash	Heap	Finished	Monday 11:05 PM (GMT+8)	10
0 TiKV, 1 TiDB, 0 PD, 0 TiFlash	CPU	Finished	Monday 11:05 PM (GMT+8)	30
1 TiKV, 0 TiDB, 0 PD, 0 TiFlash	CPU,Heap,Goroutine,Mutex	Finished	Last Friday 4:54 PM (GMT+8)	10
0 TiKV, 1 TiDB, 1 PD, 0 TiFlash	CPU,Heap,Goroutine,Mutex	Finished	Last Friday 4:54 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	CPU	Finished	Last Wednesday 10:53 AM (GMT+8)	30
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	CPU,Heap,Goroutine,Mutex	Finished	Last Tuesday 6:07 PM (GMT+8)	30
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	CPU,Heap,Goroutine,Mutex	Finished	Last Tuesday 4:57 PM (GMT+8)	30

Figure 593: View profiling history

For detailed operations on the profiling status page, see View Profiling Status.

14.11.1.13.2 TiDB Dashboard Instance Profiling - Continuous Profiling

> **Note:**
>
> This feature is designed for database experts. For non-expert users, it is recommended to use this feature under the guidance of PingCAP technical supports.

Continuous Profiling allows collecting performance data **continuously** from each TiDB, TiKV and PD instance. The collected performance data can be visualized as FlameGraph or DAG.

With these performance data, experts can analyze resource consumption details like instance's CPU and memory, to help pinpoint sophisticated performance problems at any time, such as high CPU overhead, high memory usage, process stalls, and so on. Even for problems cannot be reproduced, experts can dig deep into the problem by viewing the historical performance data collected at that moment. In this way, MTTR can be reduced effectively.

Compare with Manual Profiling

Continuous Profiling is an enhanced feature of Manual Profiling. They can be both used to collect and analyze different kinds of performance data for each instance. Differences between them are as follows:

- Manual Profiling only collects performance data for a short period of time (for example, 30 seconds) at the moment you initiate the profiling, while Continuous Profiling collects data continuously when it is enabled.
- Manual Profiling can only be used to analyze current occurring problems, while Continuous Profiling can be used to analyze both the current and historical problems.
- Manual Profiling allows to collect specific performance data for specific instances, while Continuous Profiling collects all performance data for all instances.
- Continuous Profiling stores more performance data, therefore it takes up more disk space.

Supported performance data

All performance data in Manual Profiling is collected.

- CPU: The CPU overhead of each internal function on TiDB, TiKV, TiFlash, and PD instances
- Heap: The memory consumption of each internal function on TiDB and PD instances
- Mutex: The mutex contention states on TiDB and PD instances
- Goroutine: The running state and call stack of all goroutines on TiDB and PD instances

Access the page

You can access the Continuous Profiling page using either of the following methods:

- After logging into TiDB Dashboard, click **Advanced Debugging** > **Profiling Instances** > **Continuous Profiling** on the left navigation bar.

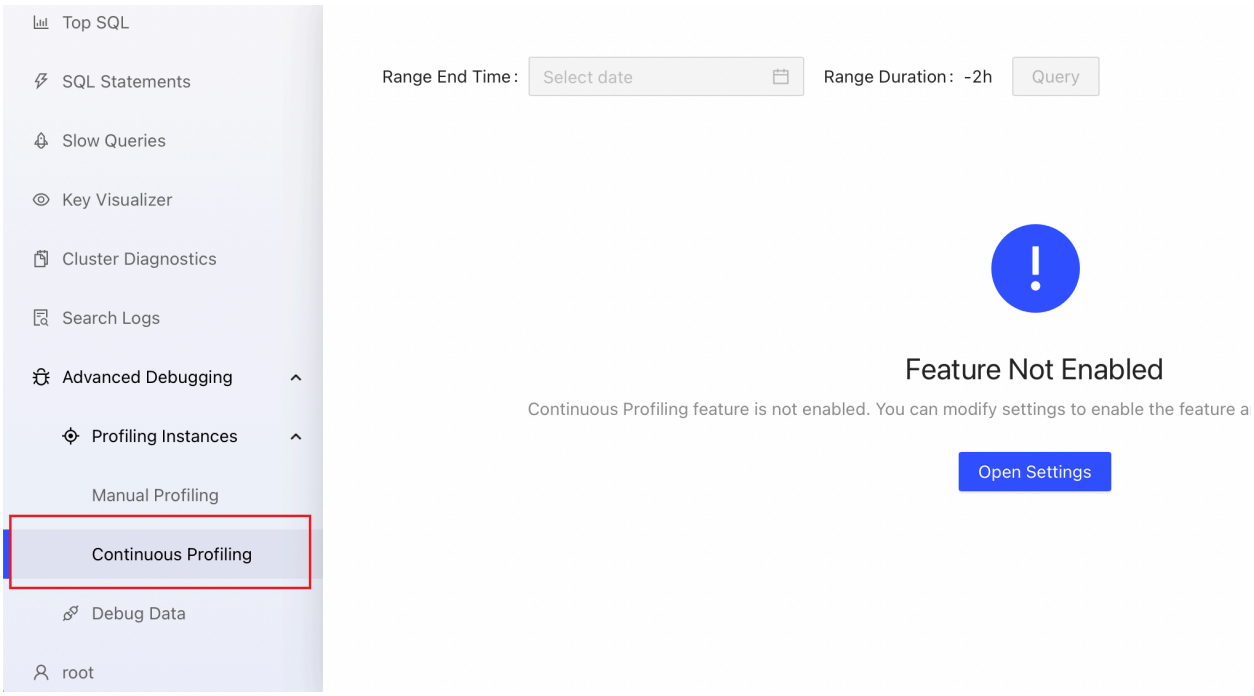


Figure 594: Access page

- Visit http://127.0.0.1:2379/dashboard/#/continuous_profiling in your browser. Replace `127.0.0.1:2379` with the actual PD instance address and port.

Enable Continuous Profiling

> **Note:**
>
> To use Continuous Profiling, your cluster should be deployed or upgraded with a recent version of TiUP (v1.9.0 or above) or TiDB Operator (v1.3.0 or above). If your cluster was upgraded using an earlier version of TiUP or TiDB Operator, see FAQ for instructions.

After enabling Continuous Profiling, you can have performance data continuously collected in the background without keeping the web pages always active. Data collected can be kept for a certain period of time and expired data is automatically cleared.

To enable this feature:

1. Visit the Continuous Profiling page.
2. Click **Open Settings**. In the **Settings** area on the right, switch **Enable Feature** on, and modify the default value of **Retention Duration** if necessary.

3. Click **Save**.

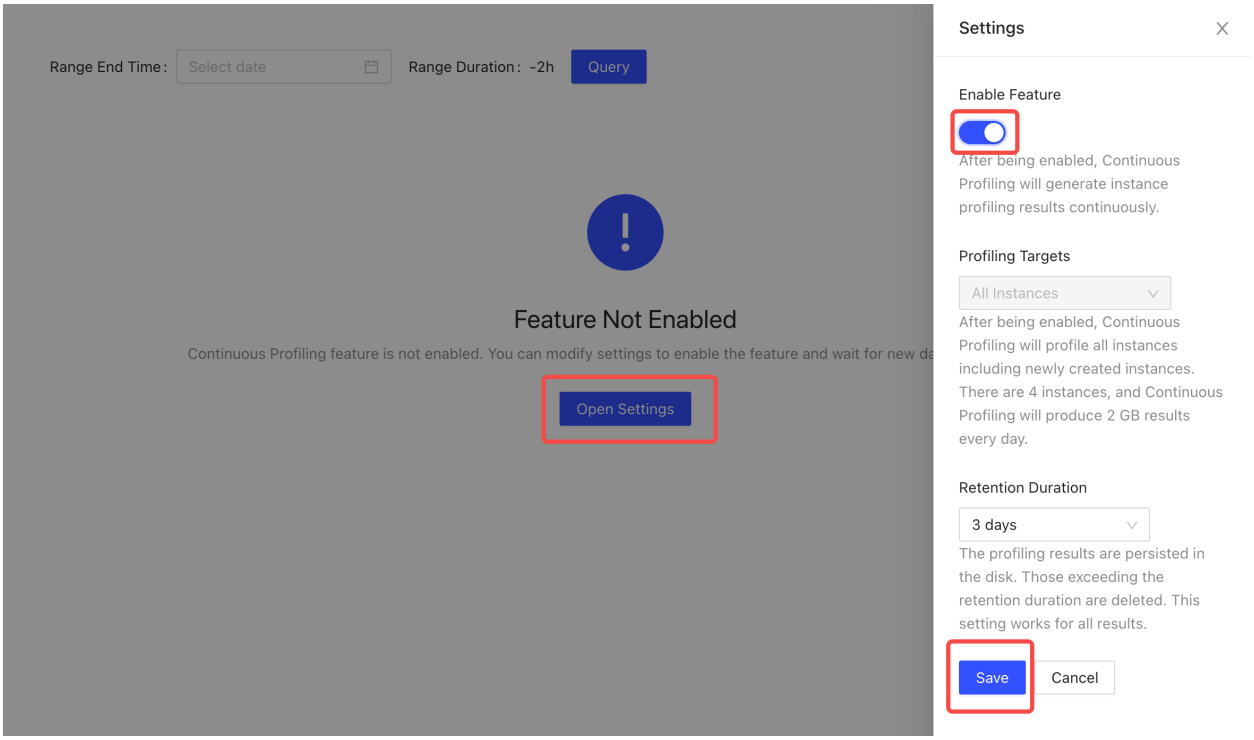


Figure 595: Enable feature

View current performance data

Manual Profiling cannot be initiated on clusters that have Continuous Profiling enabled. To view the performance data at the current moment, just click on the most recent profiling result.

View historical performance data

On the list page, you can see all performance data collected since the enabling of this feature.

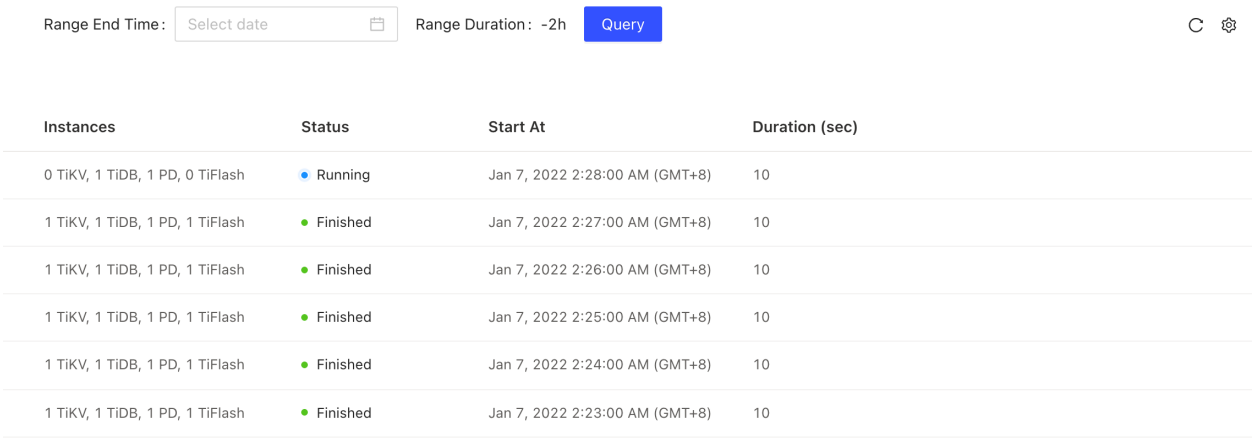


Figure 596: History results

Download performance data

On the profiling result page, you can click **Download Profiling Result** in the upper-right corner to download all profiling results.

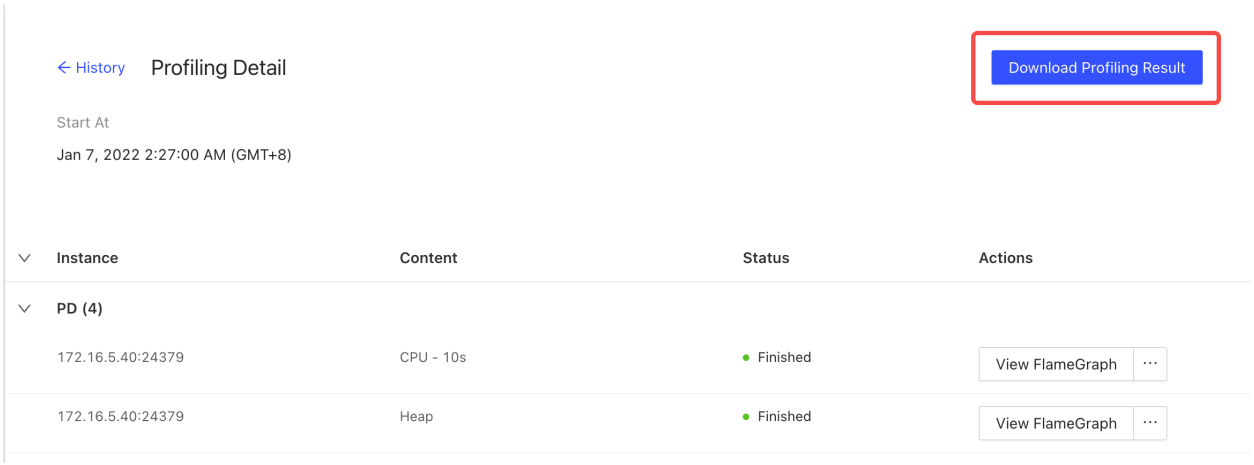


Figure 597: Download profiling result

You can also click an individual instance in the table to view its profiling result. Alternatively, you can hover on … to download raw data.

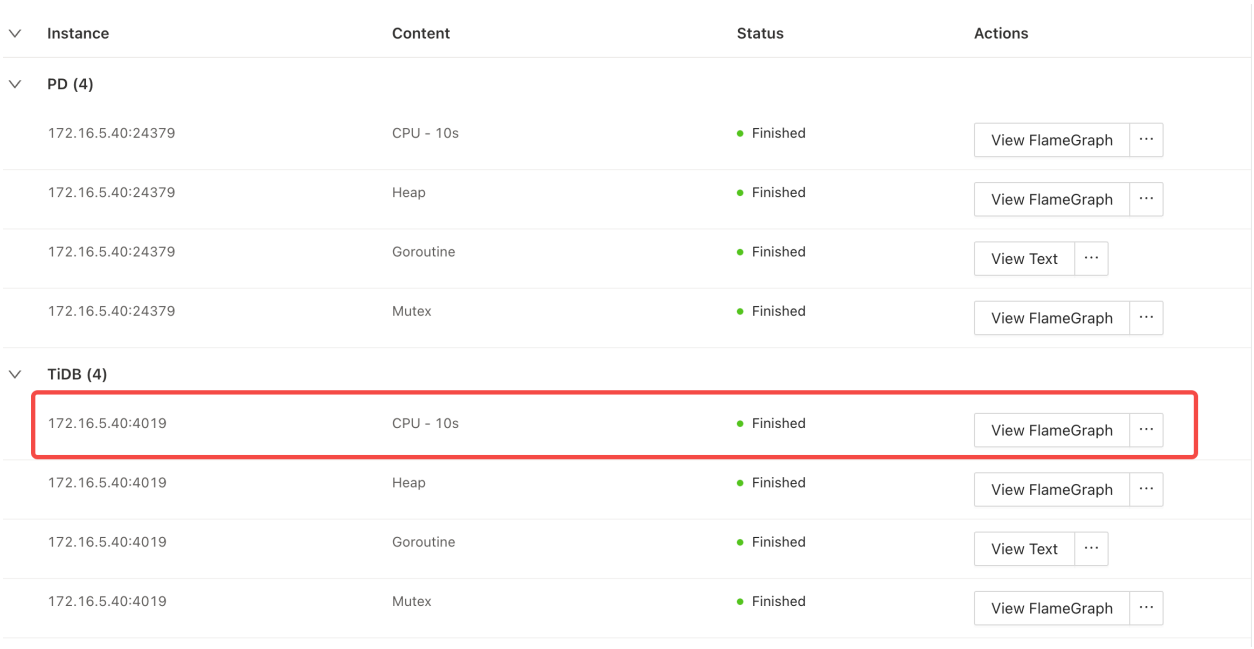


Figure 598: View profiling result

Disable Continuous Profiling

1. Visit the Continuous Profiling page.
2. Click the gear icon in the upper right corner to open the settings page. Switch **Enable Feature** off.
3. Click **Save**.
4. In the popped-up dialog box, click **Disable**.

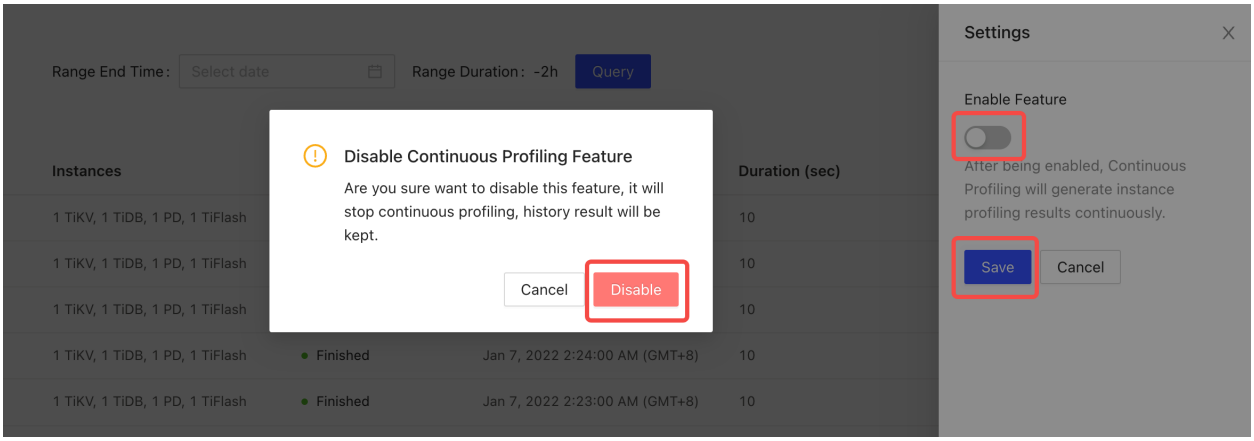


Figure 599: Disable feature

Frequently asked questions

1. Continuous Profiling cannot be enabled and the UI displays "required component NgMonitoring is not started".

See TiDB Dashboard FAQ.

2. Will performance be affected after enabling Continuous Profiling?

According to our benchmark, the average performance impact is less than 1% when the feature is enabled.

3. What is the status of this feature?

It is now a generally available (GA) feature and can be used in production environments.

14.11.1.14 Session Management and Configuration

14.11.1.14.1 Share TiDB Dashboard Sessions

You can share the current session of the TiDB Dashboard to other users so that they can access and operate the TiDB Dashboard without entering the user password.

Steps for the Inviter

1. Sign into TiDB Dashboard.
2. Click the username in the left sidebar to access the configuration page.
3. Click **Share Current Session**.

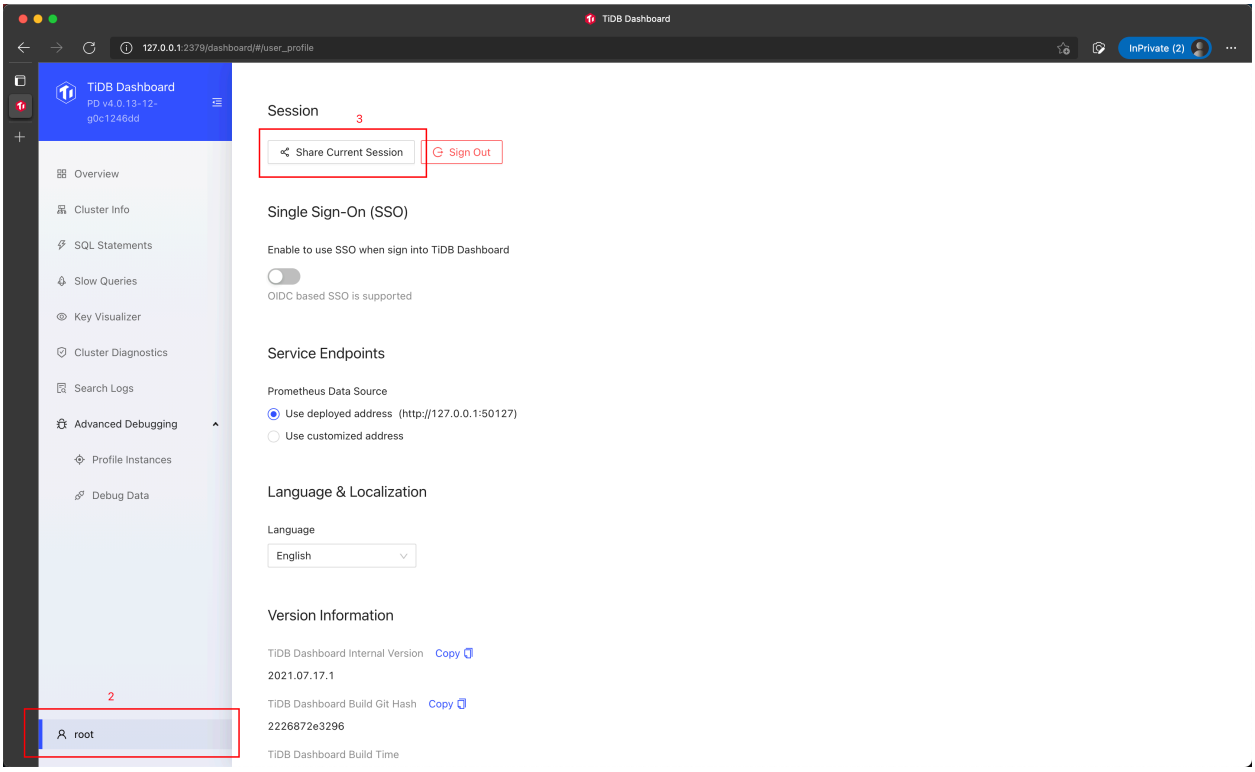


Figure 600: Sample Step

> **Note:**
>
> For security reasons, the shared session cannot be shared again.

4. Adjust sharing settings in the popup dialog:

- Expire in: How long the shared session will be effective. Signing out of the current session does not affect the effective time of the shared session.
- Share as read-only privilege: The shared session only permits read operations but not write operations (such as modifying configurations).

5. Click **Generate Authorization Code**.

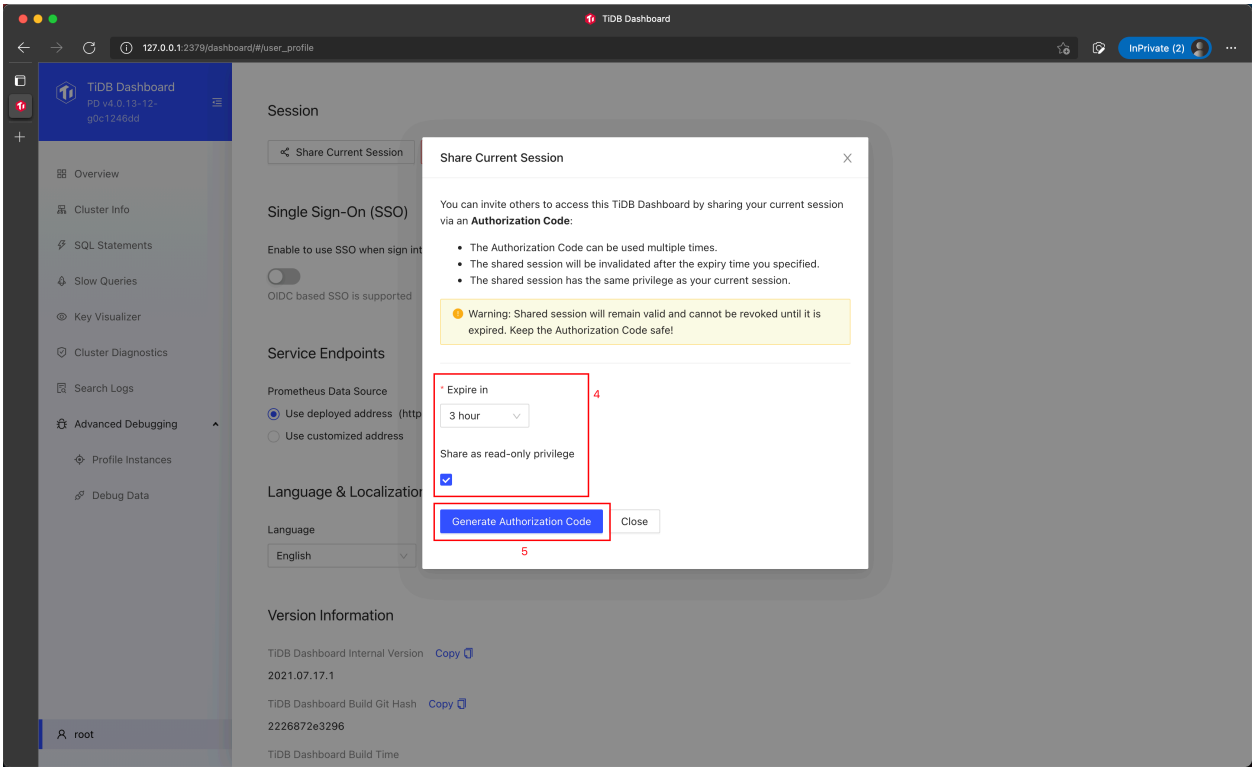


Figure 601: Sample Step

6. Provide the generated **Authorization Code** to the user to whom you want to share the session.

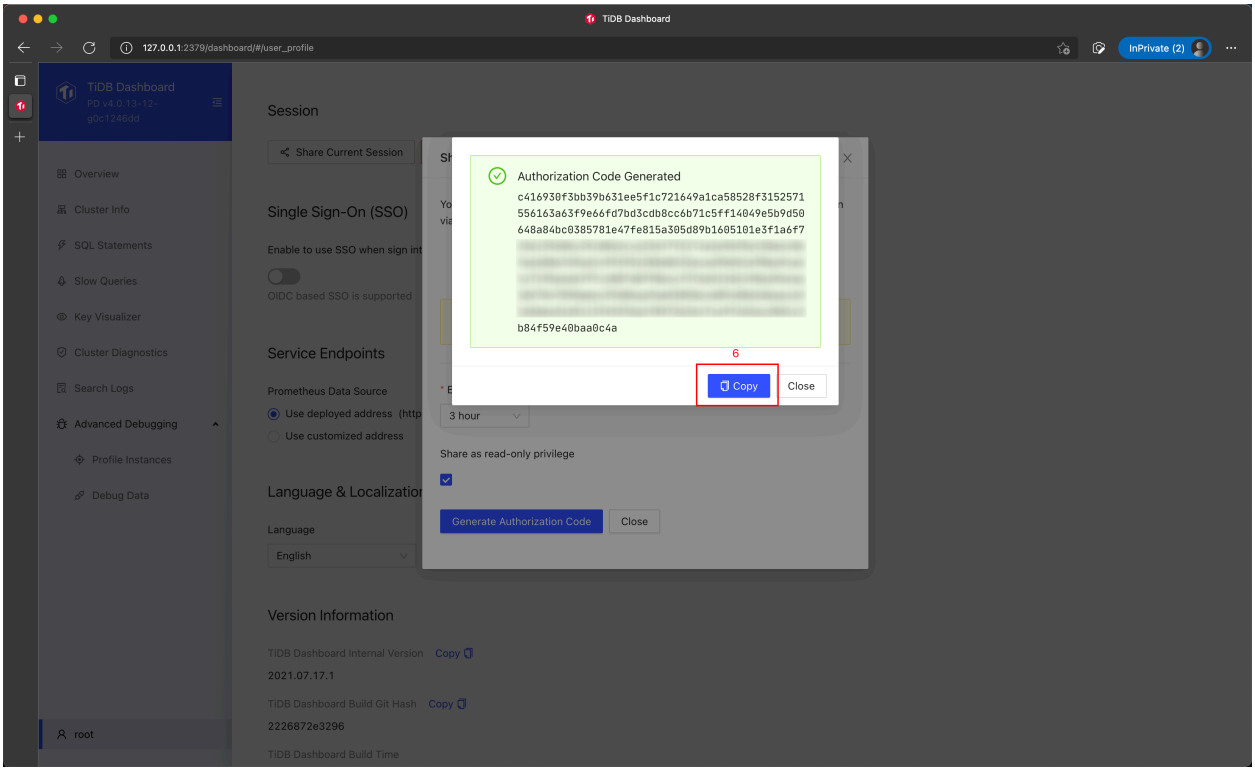


Figure 602: Sample Step

Warning:

Keep your authorization code secure and do not send it to anyone who is untrusted. Otherwise, they will be able to access and operate TiDB Dashboard without your authorization.

Steps for the Invitee

1. On the sign-in page of TiDB Dashboard, click **Use Alternative Authentication**.

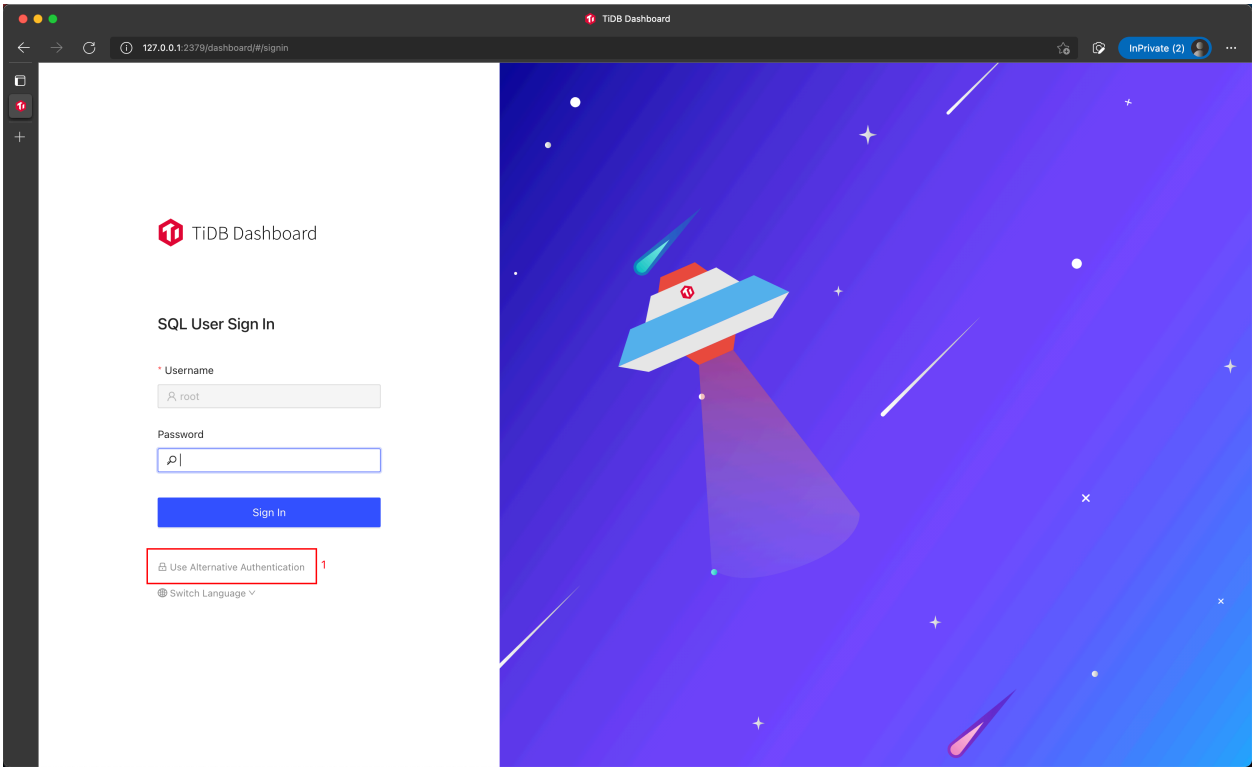


Figure 603: Sample Step

2. Click **Authorization Code** to use it to sign in.

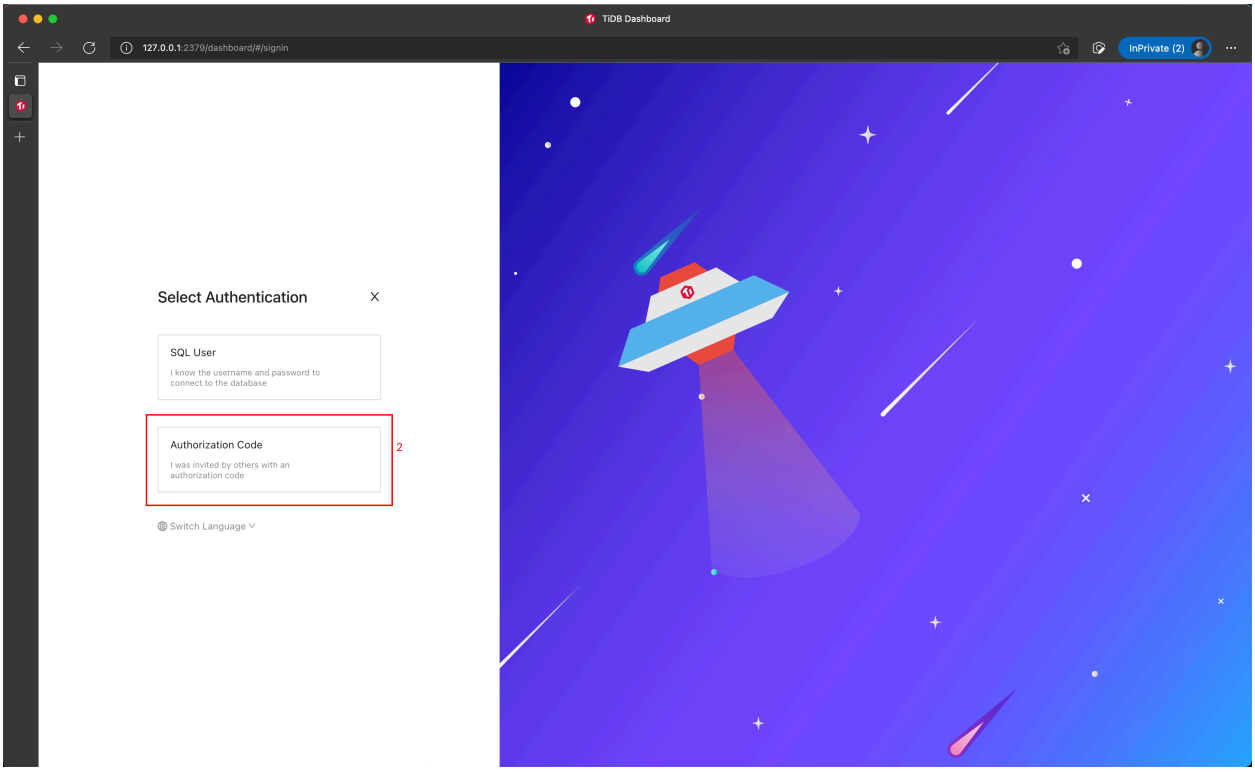


Figure 604: Sample Step

3. Enter the authorization code you have received from the inviter.
4. Click **Sign In**.

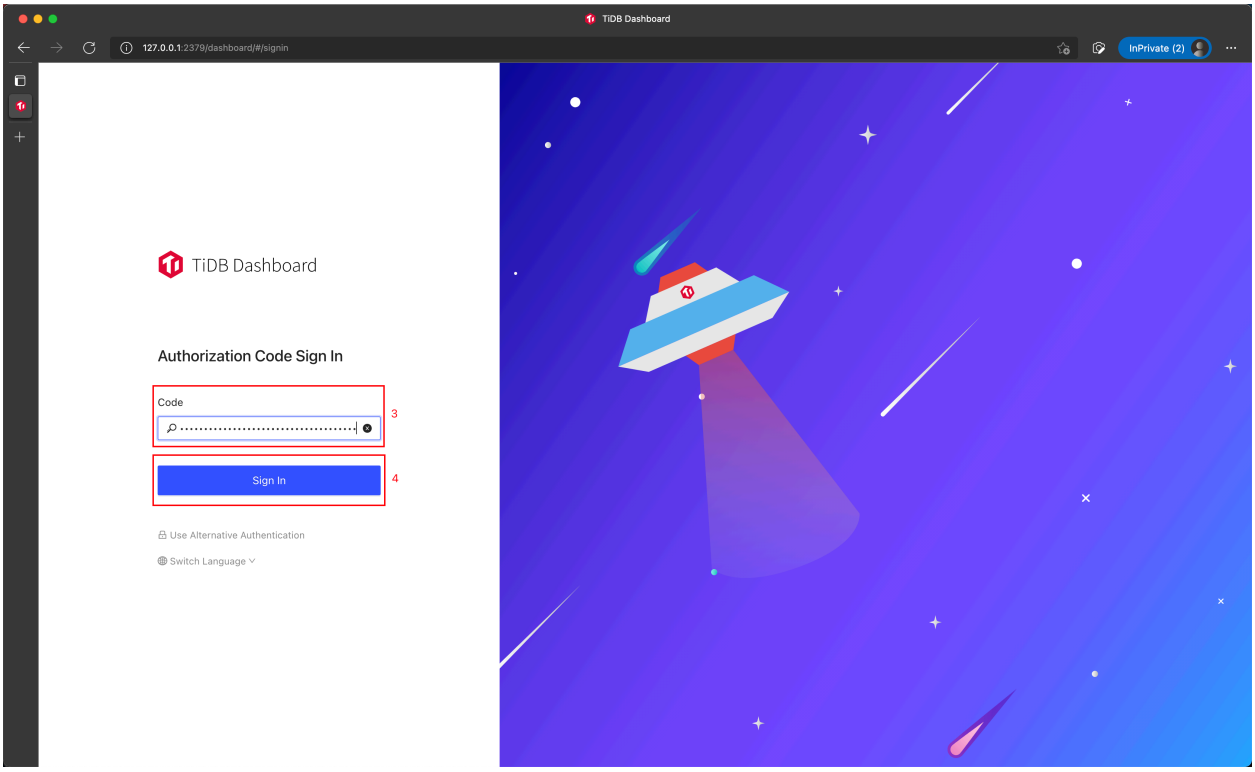


Figure 605: Sample Step

14.11.1.14.2 Configure SSO for TiDB Dashboard

TiDB Dashboard supports OIDC-based Single Sign-On (SSO). After enabling the SSO feature of TiDB Dashboard, the configured SSO service is used for your sign-in authentication and then you can access TiDB Dashboard without entering the SQL user password.

Configure OIDC SSO

Enable SSO

1. Sign into TiDB Dashboard.
2. Click the username in the left sidebar to access the configuration page.
3. In the **Single Sign-On** section, select **Enable to use SSO when sign into TiDB Dashboard**.
4. Fill the **OIDC Client ID** and the **OIDC Discovery URL** fields in the form.

Generally, you can obtain the two fields from the SSO service provider:

- OIDC Client ID is also called OIDC Token Issuer.
- OIDC Discovery URL is also called OIDC Token Audience.

5. Click **Authorize Impersonation** and input the SQL password.

TiDB Dashboard will store this SQL password and use it to impersonate a normal SQL sign-in after an SSO sign-in is finished.

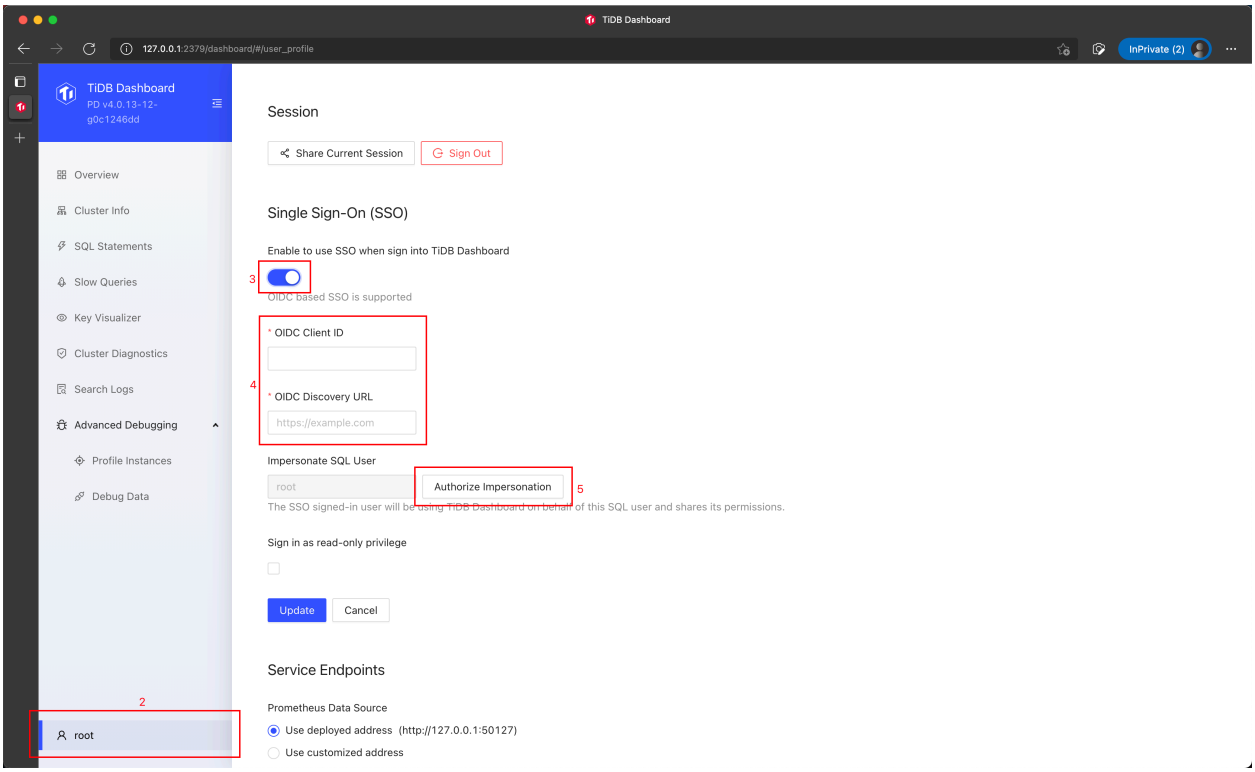


Figure 606: Sample Step

> **Note:**
>
> The password you have entered will be encrypted and stored. The SSO sign-in will fail after the password of the SQL user is changed. In this case, you can re-enter the password to bring SSO back.

6. Click **Authorize and Save**.

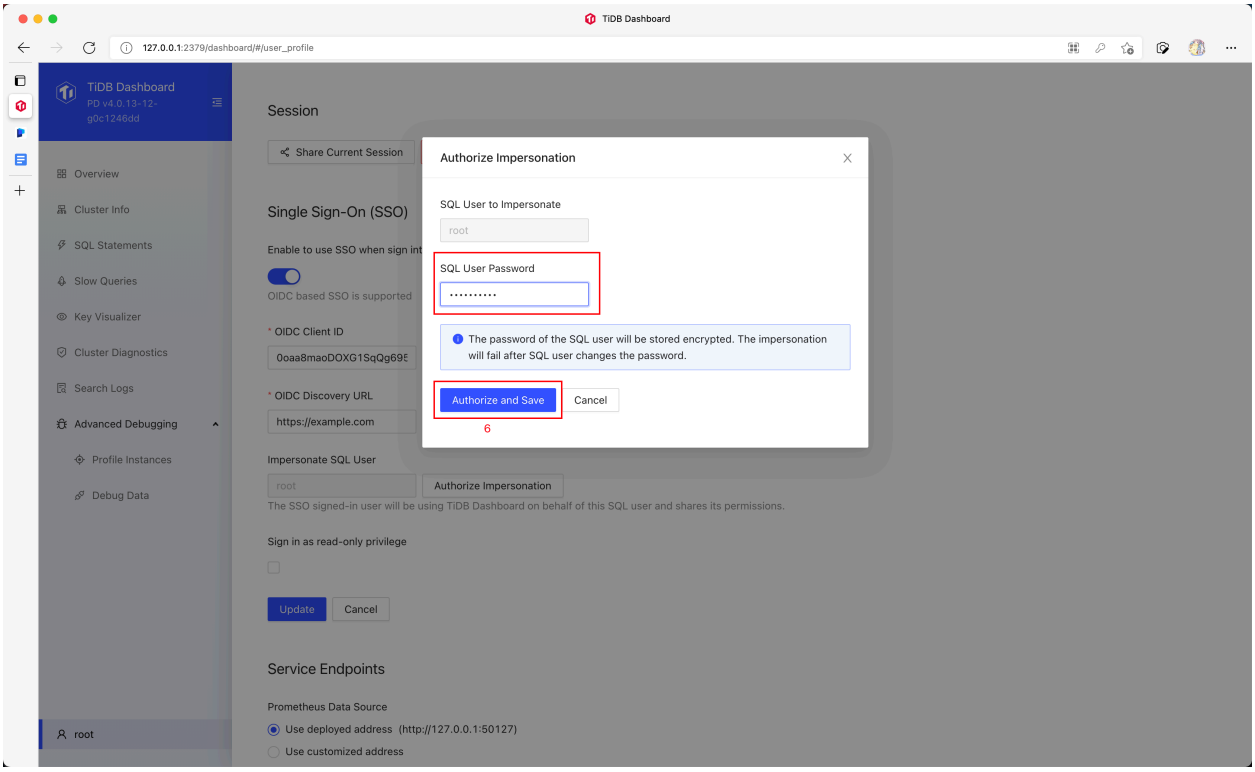


Figure 607: Sample Step

7. Click **Update** (Update) to save the configuration.

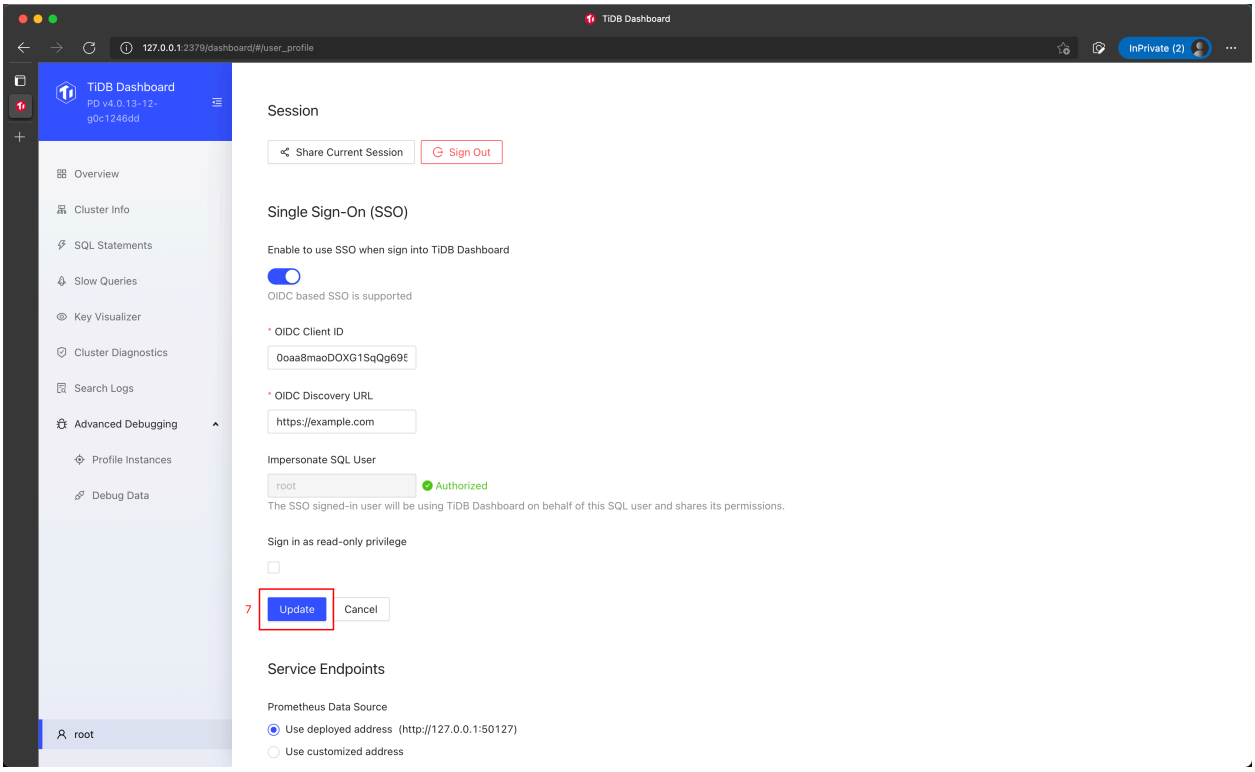


Figure 608: Sample Step

Now SSO sign-in has been enabled for TiDB Dashboard.

> **Note:**
>
> For security reasons, some SSO services require additional configuration for the SSO service, such as the trusted sign-in and sign-out URIs. Refer to the documentation of the SSO service for further information.

Disable SSO

You can disable the SSO, which will completely erase the stored SQL password:

1. Sign into TiDB Dashboard.
2. Click the username in the left sidebar to access the configuration page.
3. In the **Single Sign-On** section, deselect **Enable to use SSO when sign into TiDB Dashboard**.
4. Click **Update** (Update) to save the configuration.

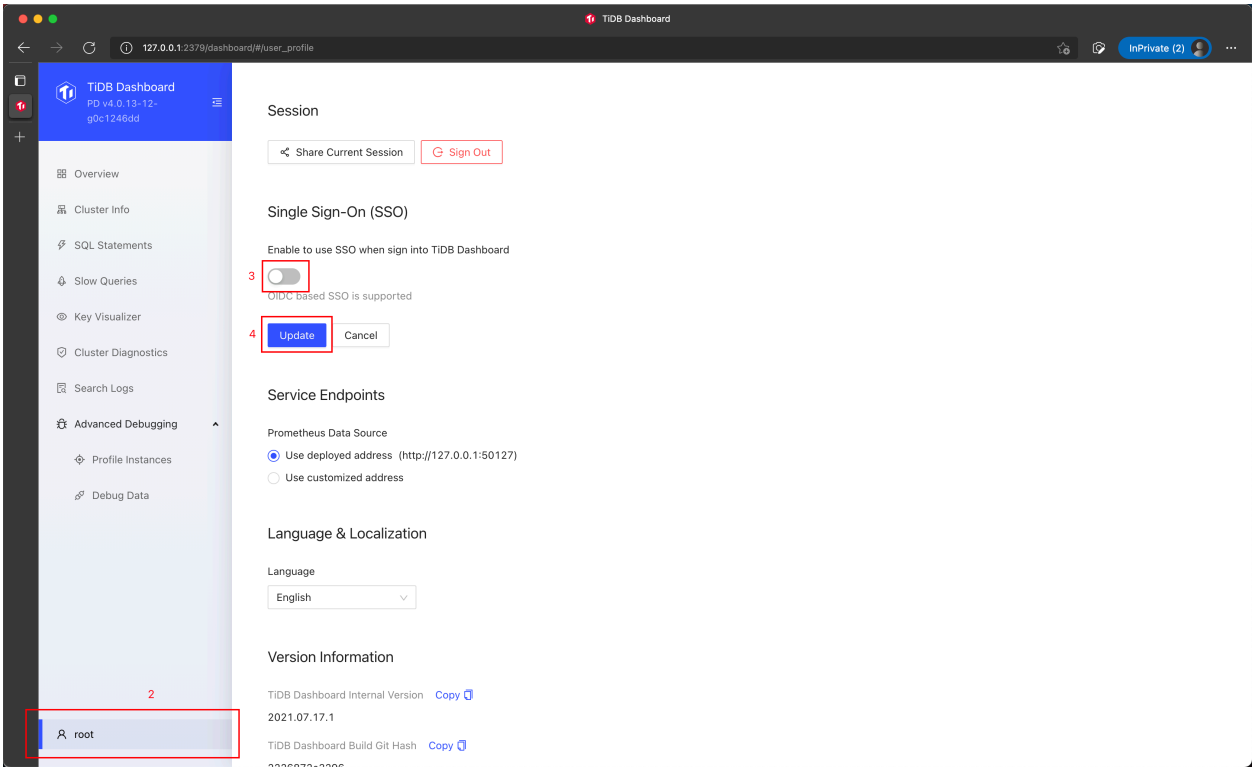


Figure 609: Sample Step

Re-enter the password after a password change

The SSO sign-in will fail once the password of the SQL user is changed. In this case, you can bring back the SSO sign-in by re-entering the SQL password:

1. Sign into TiDB Dashboard.
2. Click the username in the left sidebar to access the configuration page.
3. In the **Single Sign-On** section, Click **Authorize Impersonation** and input the updated SQL password.

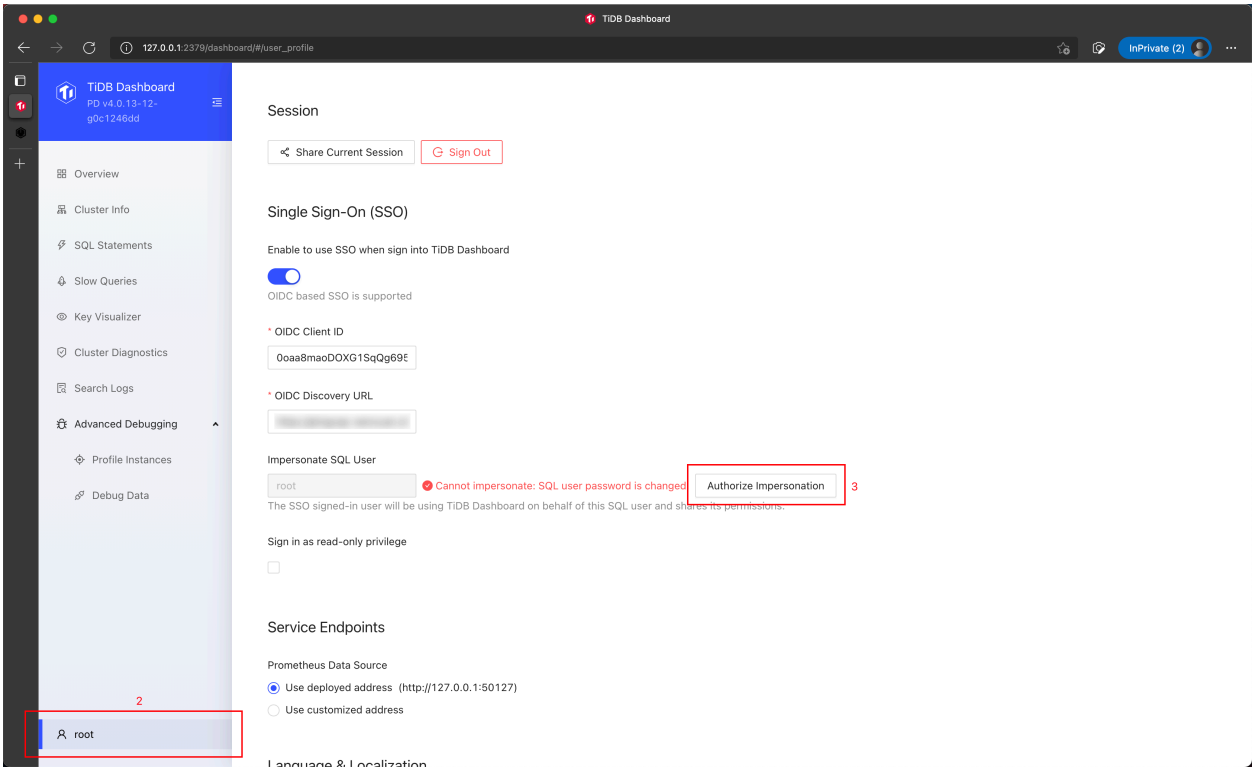


Figure 610: Sample Step

4. Click **Authorize and Save**.

Sign in via SSO

Once SSO is configured for TiDB Dashboard, you can sign in via SSO by taking following steps:

1. In the sign-in page of TiDB Dashboard, click **Sign in via Company Account**.

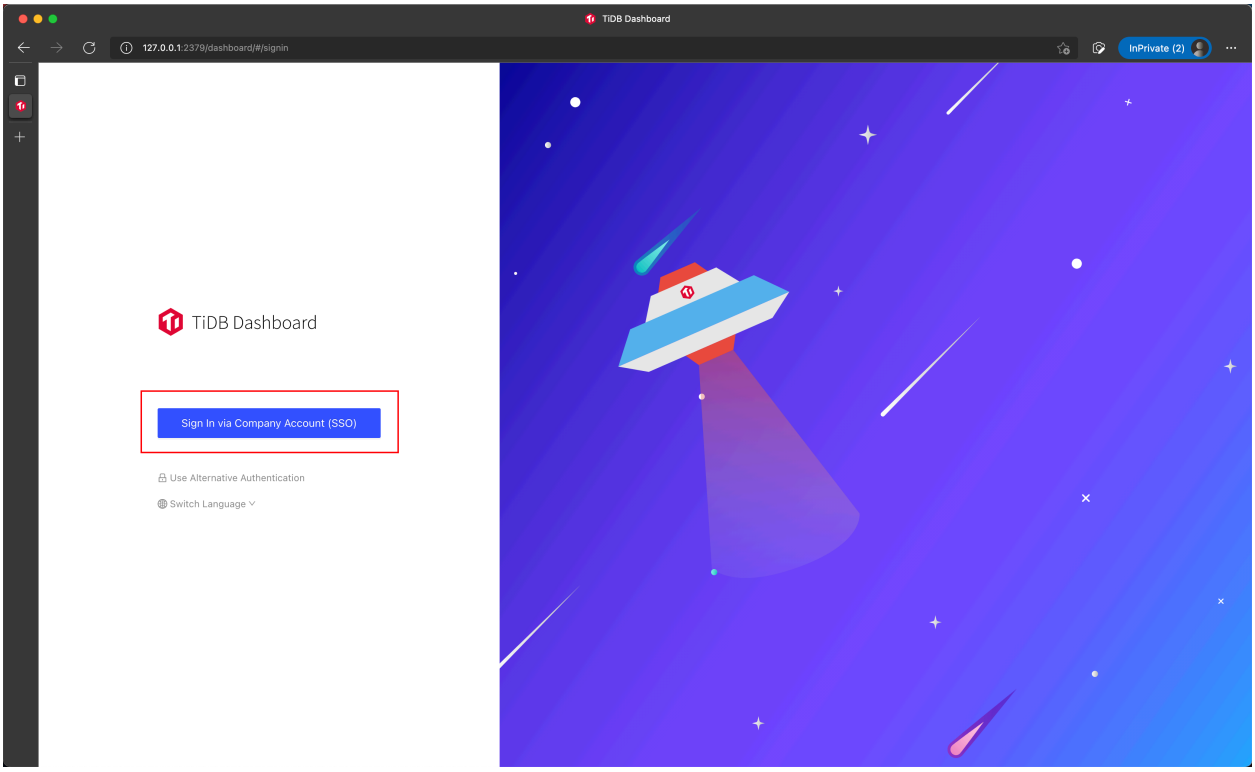


Figure 611: Sample Step

2. Sign into the system with SSO service configured.
3. You are redirected back to TiDB Dashboard to finish the sign-in.

Example 1: Use Okta for TiDB Dashboard SSO sign-in

Okta is an OIDC SSO identity service, which is compatible with the SSO feature of TiDB Dashboard. The steps below demonstrate how to configure Okta and TiDB Dashboard so that Okta can be used as the TiDB Dashboard SSO provider.

Step 1: Configure Okta

First, create an Okta Application Integration to integrate SSO.

1. Access the Okta administration site.
2. Navigate from the left sidebar **Applications** > **Applications**.
3. Click **Create App Integration**.

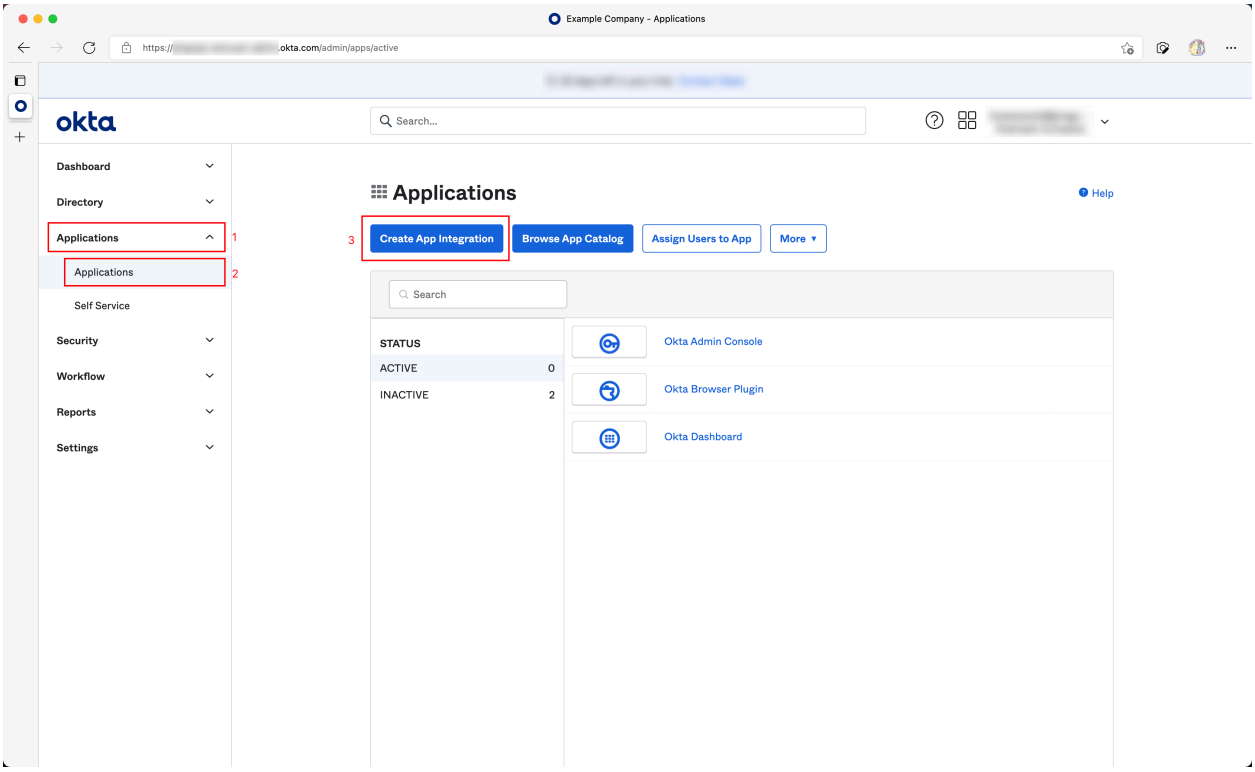


Figure 612: Sample Step

4. In the poped up dialog, choose **OIDC - OpenID Connect** in **Sign-in method**.
5. Choose **Single-Page Application** in **Application Type**.
6. Click the **Next** button.

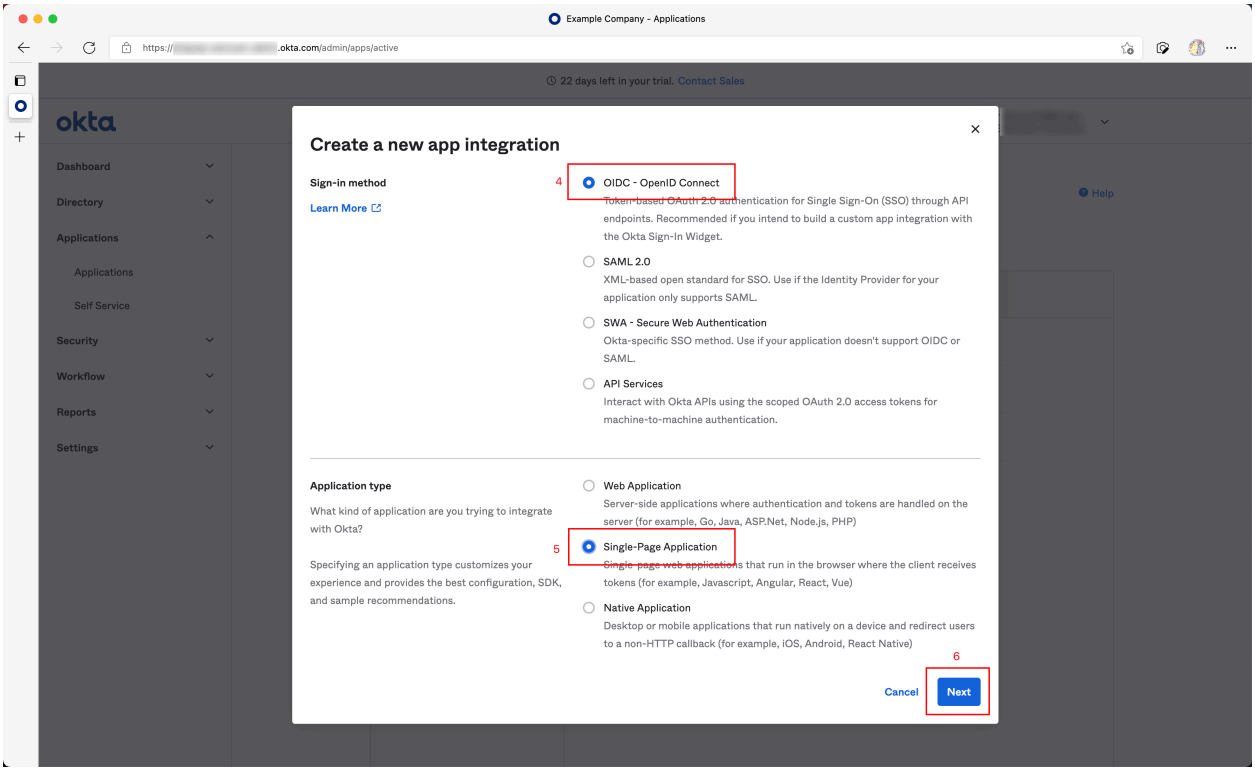


Figure 613: Sample Step

7. Fill **Sign-in redirect URIs** as follows:

```
http://DASHBOARD_IP:PORT/dashboard/?sso_callback=1
```

Substitute `DASHBOARD_IP:PORT` with the actual domain (or IP address) and port that you use to access the TiDB Dashboard in the browser.

8. Fill **Sign-out redirect URIs** as follows:

```
http://DASHBOARD_IP:PORT/dashboard/
```

Similarly, substitute `DASHBOARD_IP:PORT` with the actual domain (or IP address) and port.

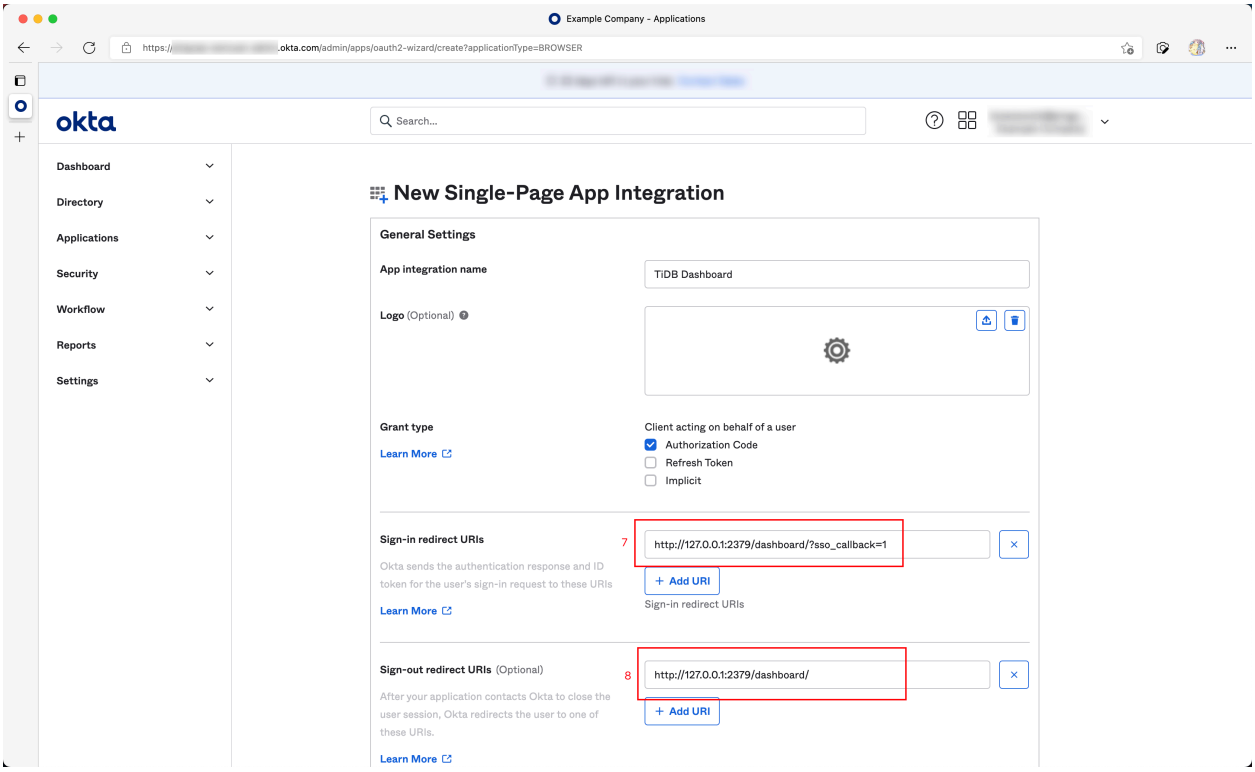


Figure 614: Sample Step

9. Configure what type of users in your organization is allowed for SSO sign-in in the **Assignments** field, and then click **Save** to save the configuration.

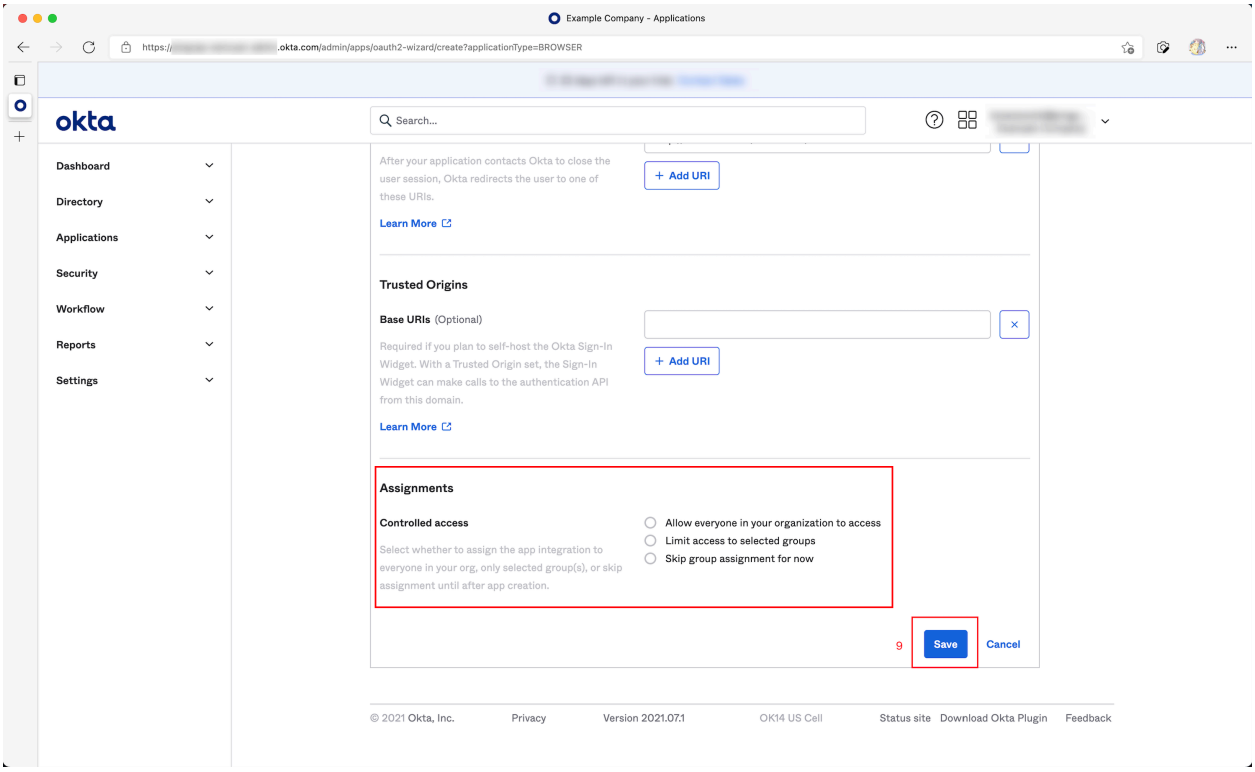


Figure 615: Sample Step

Step 2: Obtain OIDC information and fill in TiDB Dashboard

1. In the Application Integration just created in Okta, click **Sign On**.

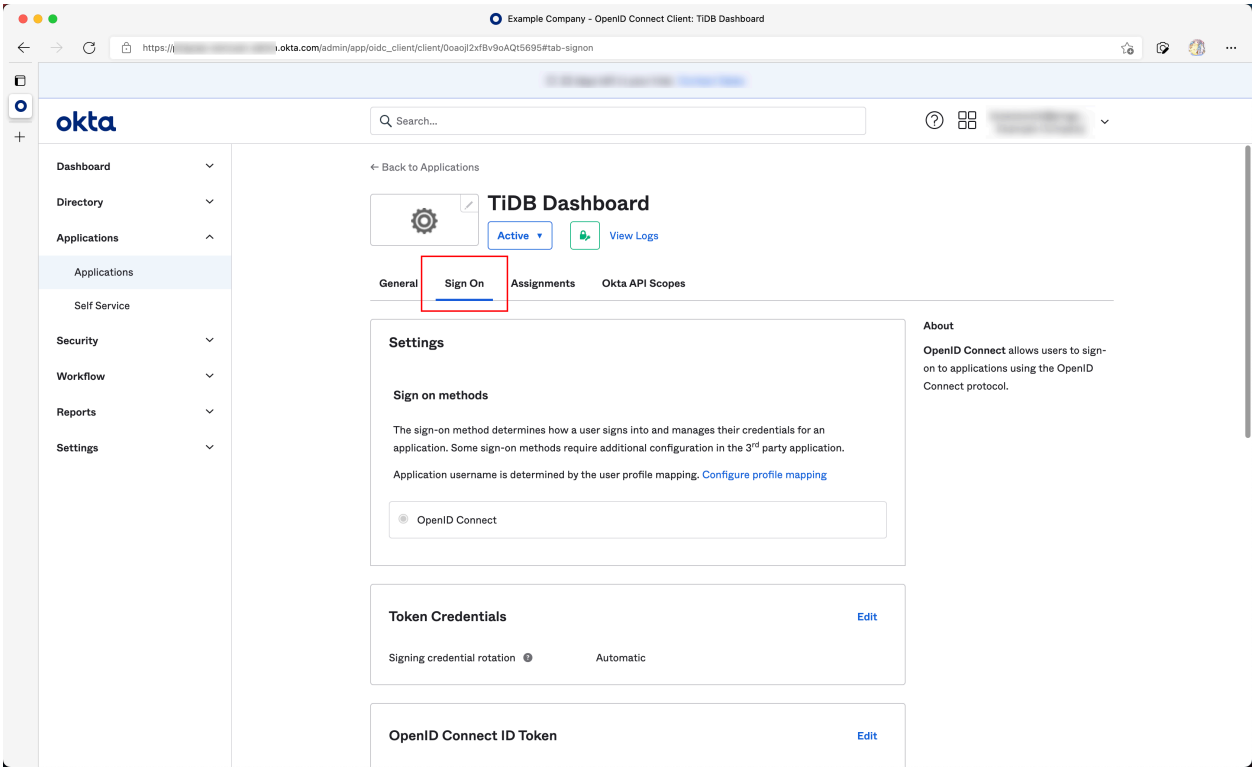


Figure 616: Sample Step 1

2. Copy values of the **Issuer** and **Audience** fields from the **OpenID Connect ID Token** section.

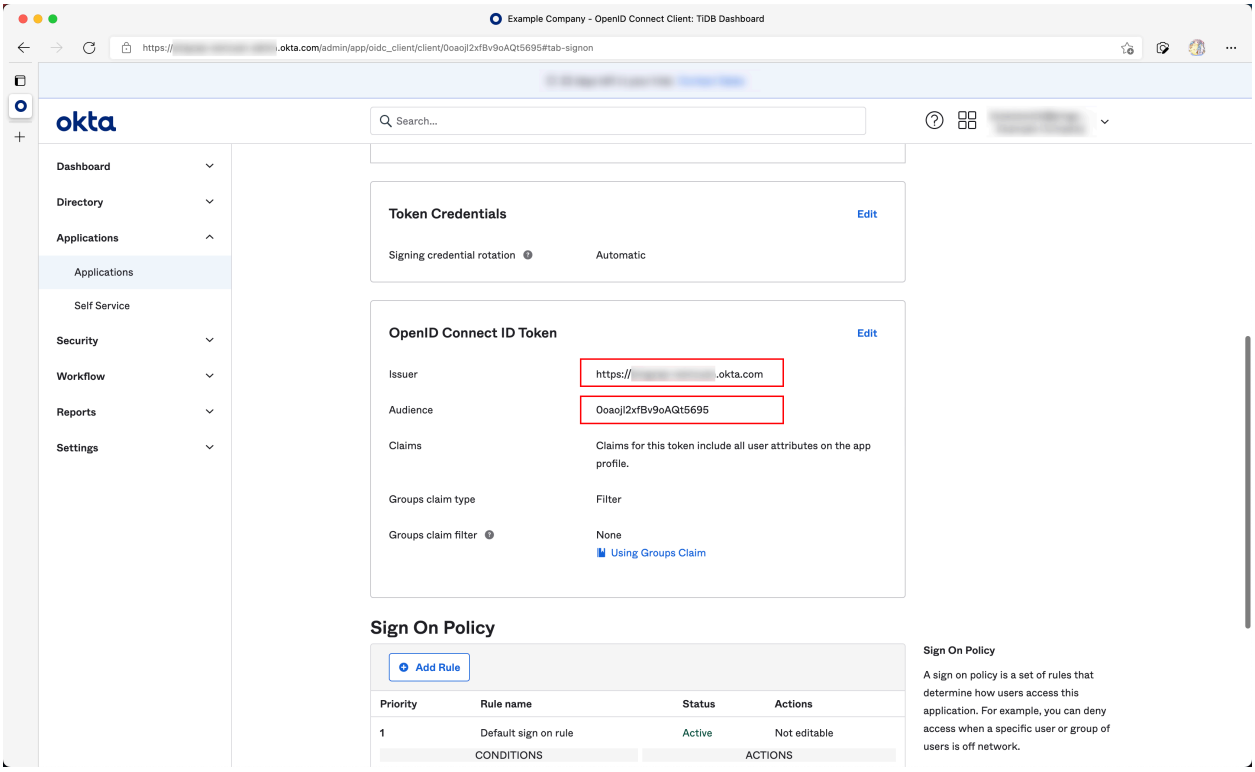


Figure 617: Sample Step 2

3. Open the TiDB Dashboard configuration page, fill **OIDC Client ID** with **Issuer** obtained from the last step and fill **OIDC Discovery URL** with **Audience**. Then finish the authorization and save the configuration. For example:

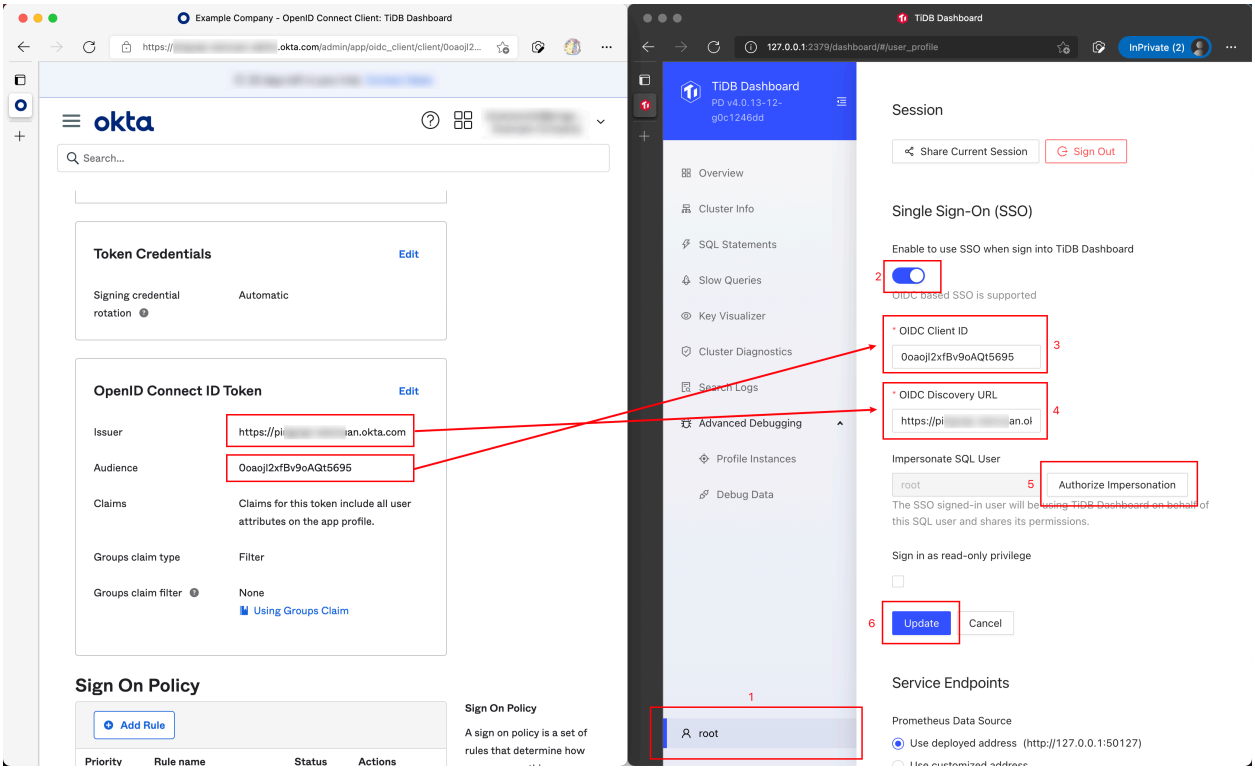


Figure 618: Sample Step 3

Now TiDB Dashboard has been configured to use Okta SSO for sign-in.

Example 2: Use Auth0 for TiDB Dashboard SSO sign-in

Similar to Okta, Auth0 also provides OIDC SSO identity service. The following steps describe how to configure Auth0 and TiDB Dashboard so that Auth0 can be used as the TiDB Dashboard SSO provider.

Step 1: Configure Auth0

1. Access the Auth0 administration site.
2. Navigate on the left sidebar **Applications** > **Applications**.
3. Click **Create App Integration**.

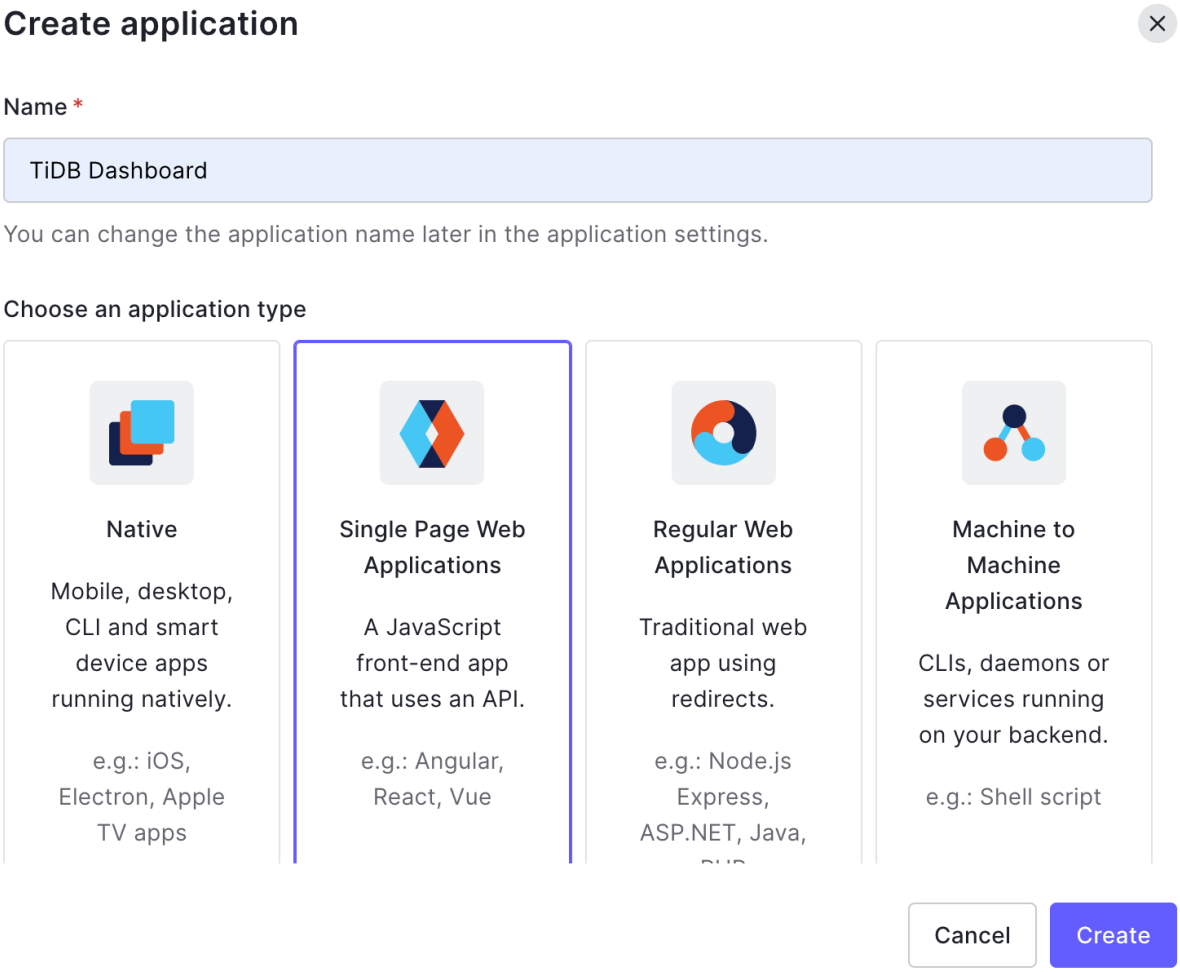


Figure 619: Create Application

```
In the popped-up dialog, fill **Name**, for example, "TiDB Dashboard".
   ↪ Choose **Single Page Web Applications** in **Choose an application
   ↪ type**. Click **Create**.
```

4. Click **Settings**.

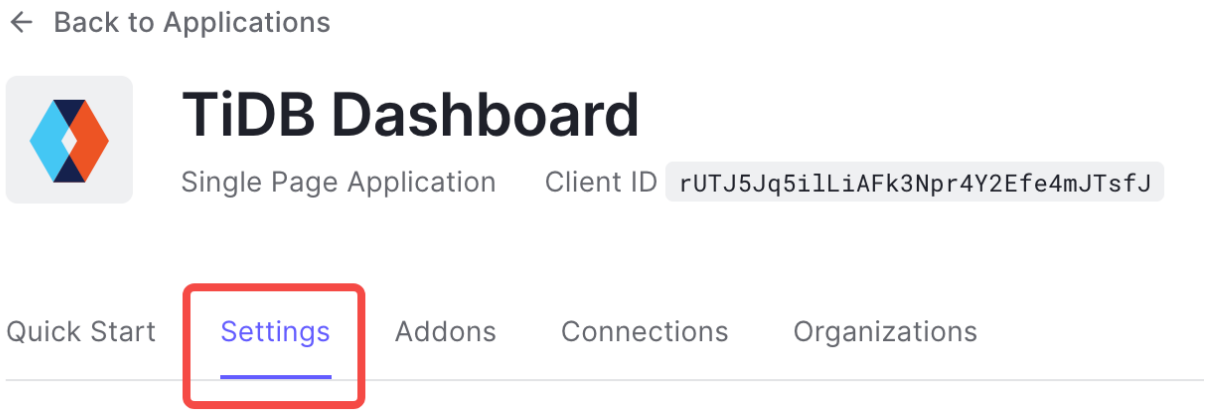


Figure 620: Settings

5. Fill **Allowed Callback URLs** as follows:

```
http://DASHBOARD_IP:PORT/dashboard/?sso_callback=1
```

Replace `DASHBOARD_IP:PORT` with the actual domain (or IP address) and port that you use to access the TiDB Dashboard in your browser.

6. Fill **Allowed Logout URLs** as follows:

```
http://DASHBOARD_IP:PORT/dashboard/
```

Similarly, replace `DASHBOARD_IP:PORT` with the actual domain (or IP address) and port.

Allowed Callback URLs

http://localhost:3001/dashboard/?sso_callback=1,
http://127.0.0.1:2379/dashboard/?sso_callback

After the user authenticates we will only call back to any of these URLs. You can specify multiple valid URLs by comma-separating them (typically to handle different environments like QA or testing). Make sure to specify the protocol (`https://`) otherwise the callback may fail in some cases. With the exception of custom URI schemes for native clients, all callbacks should use protocol `https://`. You can use Organization URL parameters in these URLs.

Allowed Logout URLs

http://localhost:3001/dashboard/,
http://127.0.0.1:2379/dashboard/

A set of URLs that are valid to redirect to after logout from Auth0. After a user logs out from Auth0 you can redirect them with the `returnTo` query parameter. The URL that you use in `returnTo` must be listed here. You can specify multiple valid URLs by comma-separating them. You can use the star symbol as a wildcard for subdomains (`*.google.com`). Query strings and hash information are not taken into account when validating these URLs. Read more about this at https://auth0.com/docs/login/logout

Figure 621: Settings

7. Keep the default values for other settings and click **Save Changes**.

Step 2: Obtain OIDC information and fill in TiDB Dashboard

1. Fill **OIDC Client ID** of TiDB Dashboard with **Client ID** in **Basic Information** under the **Settings** tab of Auth0.
2. Fill **OIDC Discovery URL** with the **Domain** field value prefixed with `https://` ↪ and suffixed with `/`, for example, `https://example.us.auth0.com/`. Complete authorization and save the configuration.

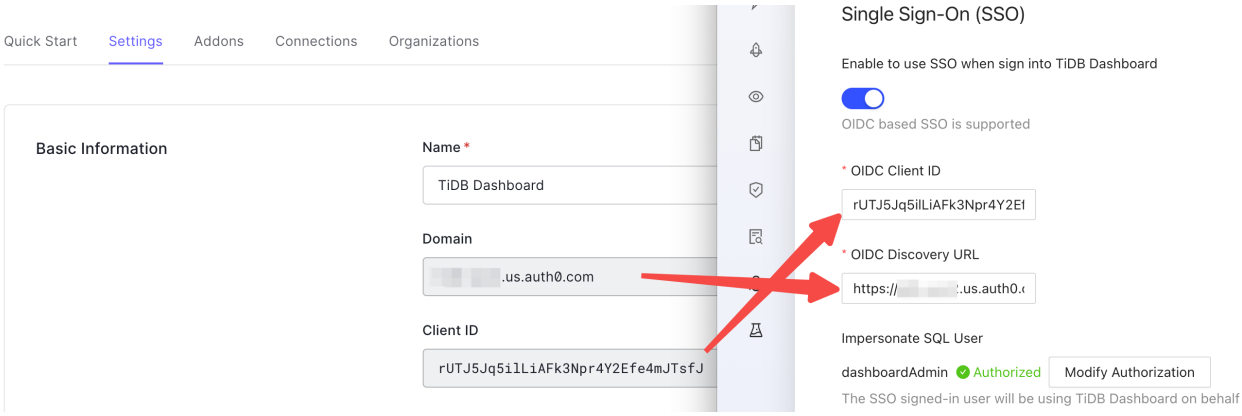


Figure 622: Settings

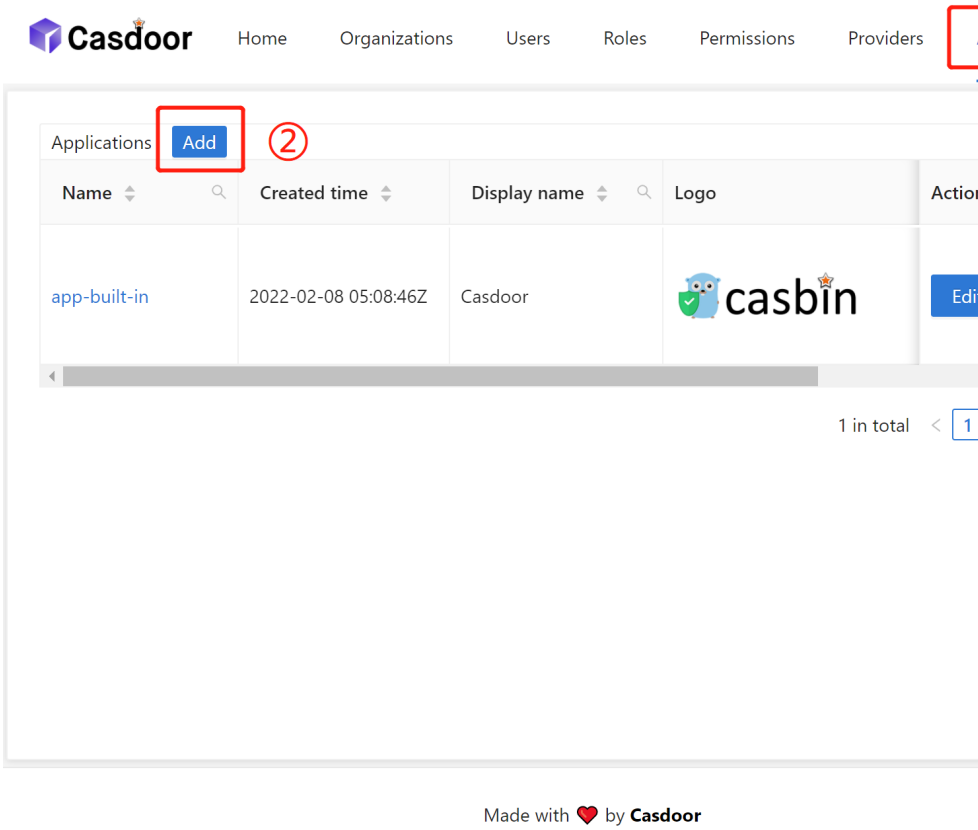
Now TiDB Dashboard has been configured to use Auth0 SSO for sign-in.

Example 3: Use Casdoor for TiDB Dashboard SSO sign-in

Casdoor is an open-source SSO platform that can be deployed in your own hosts. It is compatible with the SSO feature of TiDB Dashboard. The following steps describe how to configure Casdoor and TiDB Dashboard so that Casdoor can be used as the TiDB Dashboard SSO provider.

Step 1: Configure Casdoor

1. Deploy and access the Casdoor administration site.
2. Navigate from the top sidebar **Applications**.



3. Click **Applications - Add**.
4. Fill **Name** and **Display name**, for example, **TiDB Dashboard**.
5. Add **Redirect URLs** as follows:

```
http://DASHBOARD_IP:PORT/dashboard/?sso_callback=1
```

Replace `DASHBOARD_IP:PORT` with the actual domain (or IP address) and port that you use to access the TiDB Dashboard in your browser.

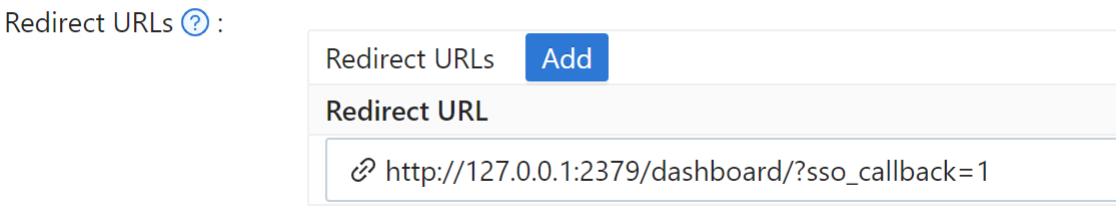


Figure 623: Settings

6. Keep the default values for other settings and click **Save & Exit**.
7. Save the **Client ID** seen on the page.

Step 2: Obtain OIDC information and fill in TiDB Dashboard

1. Fill **OIDC Client ID** of TiDB dashboard with **Client ID** saved in the previous step.
2. Fill **OIDC Discovery URL** with the **Domain** field value prefixed with `https://` and suffixed with `/`, for example, `https://casdoor.example.com/`. Complete authorization and save the configuration.

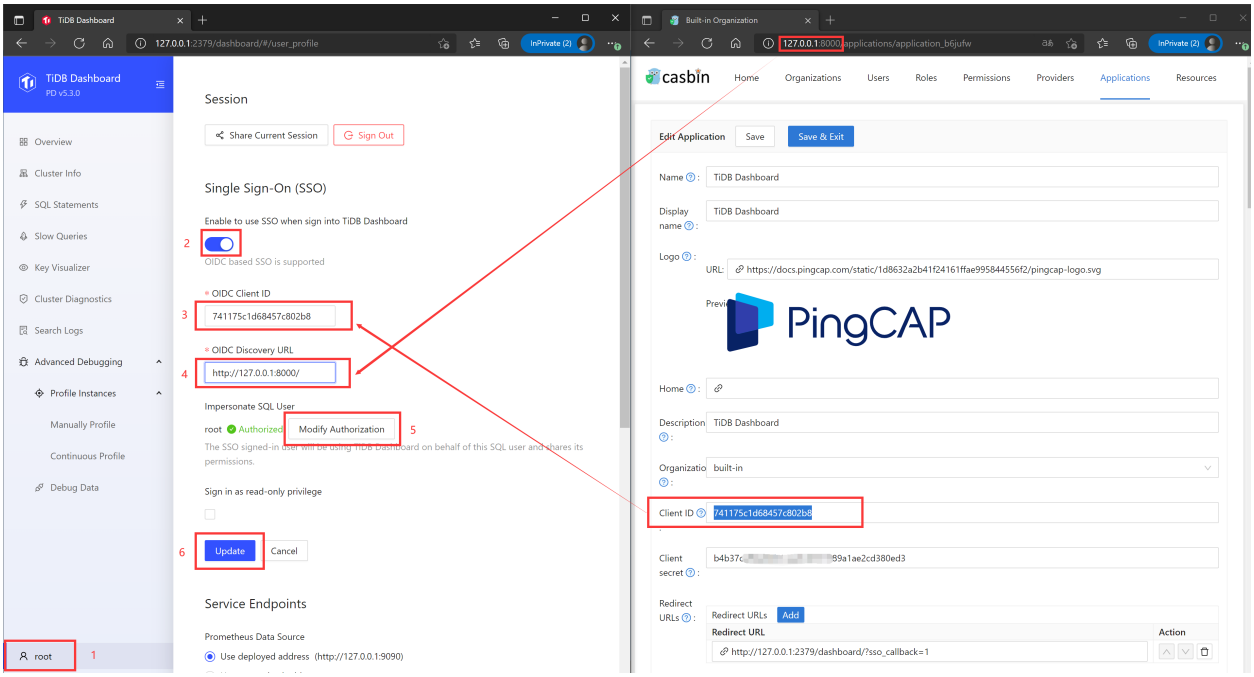


Figure 624: Settings

Now TiDB Dashboard has been configured to use Casdoor SSO for sign-in.

14.11.1.15 TiDB Dashboard FAQs

This document summarizes the frequently asked questions (FAQs) and answers about TiDB Dashboard. If a problem cannot be located or persists after you perform as instructed, get support from PingCAP or the community.

14.11.1.15.1 Access-related FAQ

When the firewall or reverse proxy is configured, I am redirected to an internal address other than TiDB Dashboard

When multiple Placement Driver (PD) instances are deployed in a cluster, only one of the PD instances actually runs the TiDB Dashboard service. If you access other PD instances instead of this one, your browser redirects you to another address. If the firewall

or reverse proxy is not properly configured for accessing TiDB Dashboard, when you visit the Dashboard, you might be redirected to an internal address that is protected by the firewall or reverse proxy.

- See TiDB Dashboard Multi-PD Instance Deployment to learn the working principle of TiDB Dashboard with multiple PD instances.
- See Use TiDB Dashboard through a Reverse Proxy to learn how to correctly configure a reverse proxy.
- See Secure TiDB Dashboard to learn how to correctly configure the firewall.

When TiDB Dashboard is deployed with dual network interface cards (NICs), TiDB Dashboard cannot be accessed using another NIC

For security reasons, TiDB Dashboard on PD only monitors the IP addresses specified during deployment (that is, it only listens on one NIC), not on `0.0.0.0`. Therefore, when multiple NICs are installed on the host, you cannot access TiDB Dashboard using another NIC.

If you have deployed TiDB using the `tiup cluster` or `tiup playground` command, currently this problem cannot be solved. It is recommended that you use a reverse proxy to safely expose TiDB Dashboard to another NIC. For details, see Use TiDB Dashboard behind a Reverse Proxy.

14.11.1.15.2 UI-related FAQ

A `prometheus_not_found` error is shown in **QPS** and **Latency** sections on the Overview page

The **QPS** and **Latency** sections on the **Overview** page require a cluster with Prometheus deployed. Otherwise, the error is shown. You can solve this problem by deploying a Prometheus instance in the cluster.

If you still encounter this problem when the Prometheus instance has been deployed, the possible reason is that your deployment tool is out of date (TiUP or TiDB Operator), and your tool does not automatically report metrics addresses, which makes TiDB Dashboard unable to query metrics. You can upgrade you deployment tool to the latest version and try again.

If your deployment tool is TiUP, take the following steps to solve this problem. For other deployment tools, refer to the corresponding documents of those tools.

1. Upgrade TiUP and TiUP Cluster:

```
tiup update --self
tiup update cluster --force
```

2. After the upgrade, when a new cluster is deployed with Prometheus instances, the metrics can be displayed normally.

3. After the upgrade, for an existing cluster, you can restart this cluster to report the metrics addresses. Replace `CLUSTER_NAME` with the actual cluster name:

```
tiup cluster start CLUSTER_NAME
```

Even if the cluster has been started, still execute this command. This command does not affect the normal application in the cluster, but refreshes and reports the metrics addresses, so that the monitoring metrics can be displayed normally in TiDB Dashboard.

An `invalid connection` error is shown on the **Slow Queries** page

The possible reason is that you have enabled the Prepared Plan Cache feature of TiDB. As an experimental feature, when enabled, Prepared Plan Cache might not function properly in specific TiDB versions, which could cause this problem in TiDB Dashboard (and other applications). You can disable Prepared Plan Cache by setting the system variable `tidb_enable_prepared_plan_cache = OFF`.

A `required component NgMonitoring is not started` error is shown

NgMonitoring is an advanced monitoring component built in TiDB clusters of v5.4.0 and later versions to support TiDB Dashboard features such as **Continuous Profiling** and **Top SQL**. NgMonitoring is automatically deployed when you deploy or upgrade a cluster with a newer version of TiUP. For clusters deployed using TiDB Operator, you can deploy NgMonitoring manually by referring to Enable Continuous Profiling.

If the web page shows `required component NgMonitoring is not started`, you can troubleshoot the deployment issue as follows:

Clusters Deployed using TiUP

Step 1. Check versions

1. Check the TiUP cluster version. NgMonitoring is deployed only when TiUP is v1.9.0 or later.

```
tiup cluster --version
```

The command output shows the TiUP version. For example:

```
tiup version 1.9.0 tiup
Go Version: go1.17.2
Git Ref: v1.9.0
```

2. If the TiUP cluster version is earlier than v1.9.0, upgrade TiUP and TiUP cluster to the latest version:

```
tiup update --all
```

Step 2. Add the ng_port configuration item on the control machine by using TiUP. Then reload Prometheus.

1. Open the cluster configuration file in editing mode:

```
tiup cluster edit-config ${cluster-name}
```

2. Under `monitoring_servers`, add the `ng_port:12020` parameter:

```
monitoring_servers:
- host: 172.16.6.6
  ng_port: 12020
```

3. Reload Prometheus:

```
tiup cluster reload ${cluster-name} --role prometheus
```

If the error message is still prompted after performing steps above, get support from PingCAP or the community.

Clusters Deployed using TiDB Operator

Deploy the NgMonitoring component by following instructions in the Enable Continuous Profiling section in TiDB Operator documentation.

Clusters Started using TiUP Playground

When starting the cluster, TiUP Playground (>= v1.8.0) automatically starts the NgMonitoring component. To update TiUP Playground to the latest version, run the following command:

```
tiup update --self
tiup update playground
```

An `unknown field` error is shown on the **Slow Queries** page

If the `unknown field` error appears on the **Slow Queries** page after the cluster upgrade, the error is related to a compatibility issue caused by the difference between TiDB Dashboard server fields (which might be updated) and user preferences fields (which are in the browser cache). This issue has been fixed. If your cluster is earlier than v5.0.3 or v4.0.14, perform the following steps to clear your browser cache:

1. Open TiDB Dashboard page.
2. Open Developer Tools. Different browsers have different ways of opening Developer Tools. After clicking the **Menu Bar**:
    - Firefox: **Menu** > **Web Developer** > **Toggle Tools**, or **Tools** > **Web Developer** > **Toggle Tools**.
    - Chrome: **More tools** > **Developer tools**.

- Safari: **Develop** > **Show Web Inspector**. If you can't see the **Develop** menu, go to **Safari** > **Preferences** > **Advanced**, and check the **Show Develop** menu in menu bar checkbox.

In the following example, Chrome is used.

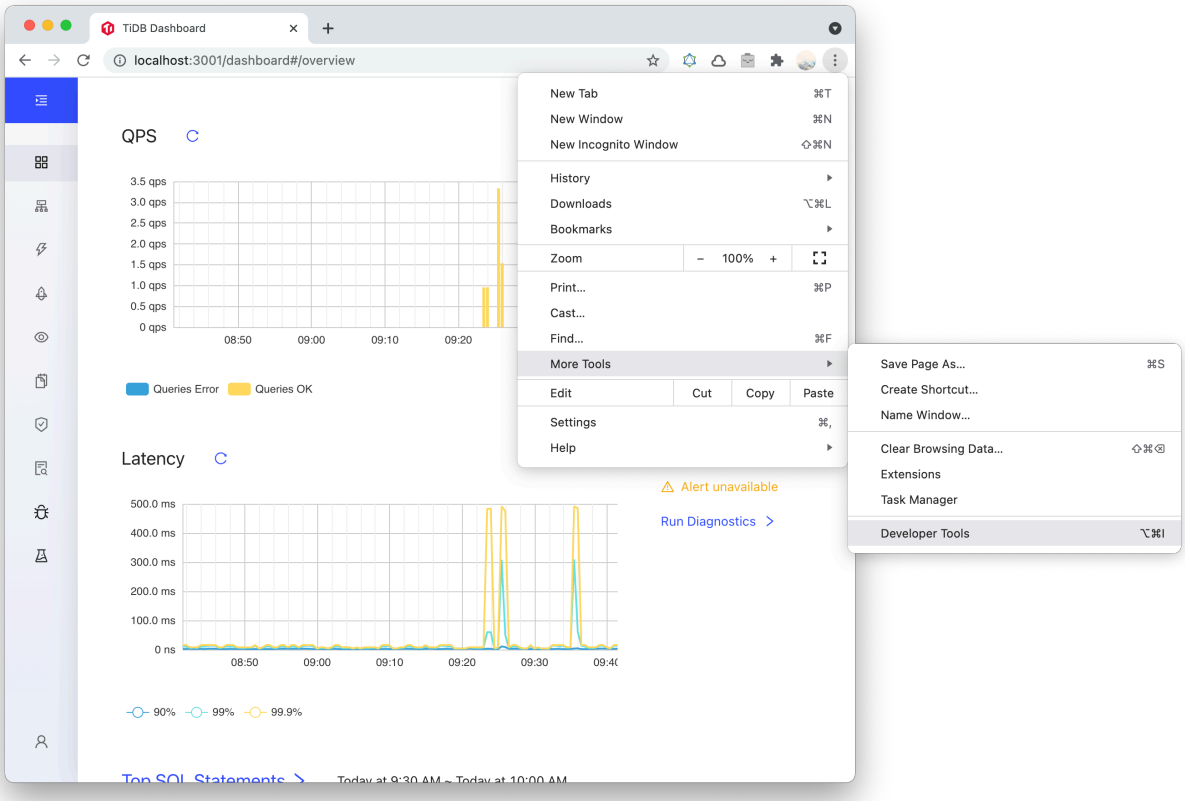


Figure 625: Opening DevTools from Chrome's main menu

3. Select the **Application** panel, expand the **Local Storage** menu and select the **TiDB Dashboard page domain**. Click the **Clear All** button.

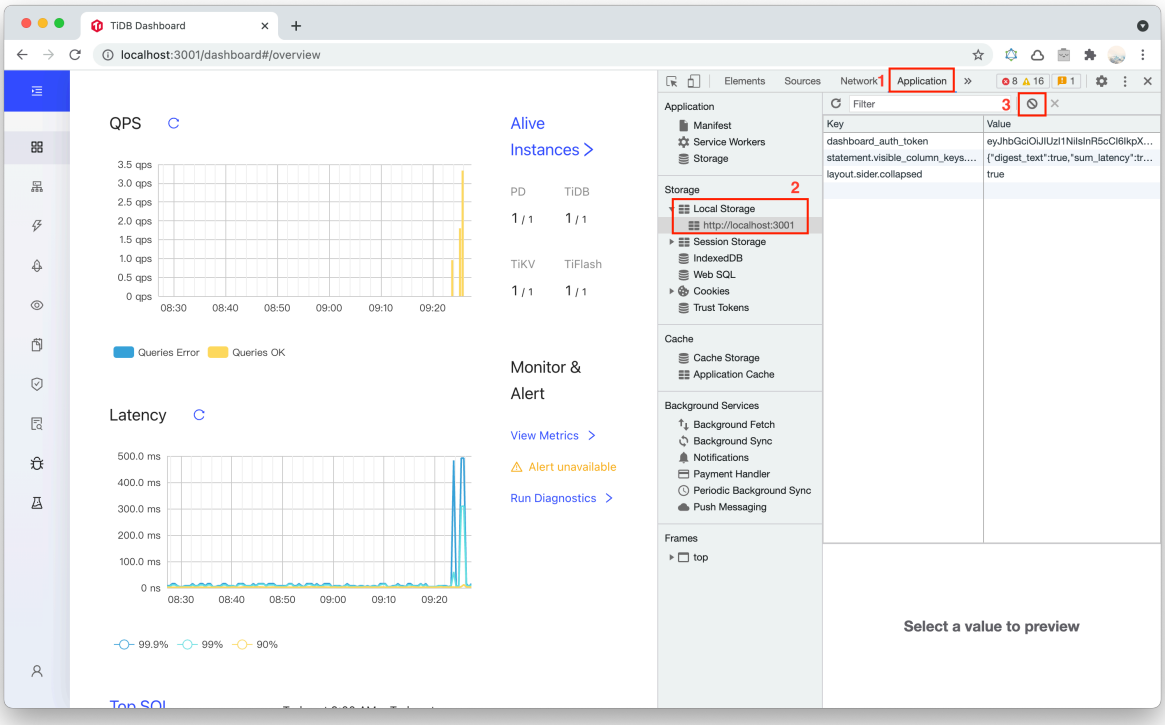


Figure 626: Clear the Local Storage

14.12 Telemetry

When the telemetry is enabled, TiDB, TiUP and TiDB Dashboard collect usage information and share the information with PingCAP to help understand how to improve the product. For example, this usage information helps prioritize new features.

Note:

- Starting from February 20, 2023, the telemetry feature is disabled by default in new versions of TiDB and TiDB Dashboard, and usage information is not collected and shared with PingCAP. Before upgrading to these versions, if the cluster uses the default telemetry configuration, the telemetry feature is disabled after the upgrade. See TiDB Release Timeline for a specific version.
- Starting from v1.11.3, the telemetry feature is disabled by default in newly deployed TiUP, and usage information is not collected. If you upgrade from a TiUP version earlier than v1.11.3 to v1.11.3 or a later

version, the telemetry feature keeps the same status as before the upgrade.

14.12.1 What is shared?

The following sections describe the shared usage information in detail for each component. The usage details that get shared might change over time. These changes (if any) will be announced in release notes.

Note:

In **ALL** cases, user data stored in the TiDB cluster will **NOT** be shared. You can also refer to PingCAP Privacy Policy.

14.12.1.1 TiDB

When the telemetry collection feature is enabled in TiDB, the TiDB cluster collects usage details on a 6-hour basis. These usage details include but are not limited to:

- A randomly generated telemetry ID.
- Deployment characteristics, such as the size of hardware (CPU, memory, disk), TiDB components versions, OS name.
- The status of query requests in the system, such as the number of query requests and the duration.
- Component usage, for example, whether the Async Commit feature is in use or not.
- Pseudonymized IP address of the TiDB telemetry data sender.

To view the full content of the usage information shared to PingCAP, execute the following SQL statement:

```
ADMIN SHOW TELEMETRY;
```

14.12.1.2 TiDB Dashboard

When the telemetry collection feature is enabled for TiDB Dashboard, usage details of the TiDB Dashboard web UI will be shared, including (but not limited to):

- A randomly generated telemetry ID.

- User operation information, such as the name of the TiDB Dashboard web page accessed by the user.
- Browser and OS information, such as browser name, OS name, and screen resolution.

To view the full content of the usage information shared to PingCAP, use the Network Activity Inspector of Chrome DevTools or the Network Monitor of Firefox Developer Tools.

14.12.1.3 TiUP

When the telemetry collection feature is enabled in TiUP, usage details of TiUP will be shared, including (but not limited to):

- A randomly generated telemetry ID.
- Execution status of TiUP commands, such as whether the execution is successful and the execution duration.
- Deployment characteristics, such as the size of hardware, TiDB components versions, and deployment configuration names that have been modified.

To view the full content of the usage information shared to PingCAP, set the `TIUP_CLUSTER_DEBUG=enable` environment variable when executing the TiUP command. For example:

```
TIUP_CLUSTER_DEBUG=enable tiup cluster list
```

14.12.1.4 TiSpark

> **Note:**
>
> Starting from v3.3, the telemetry collection is disabled by default in TiSpark, and usage information is not collected and shared with PingCAP.

When the telemetry collection feature is enabled for TiSpark, the Spark module will share the usage details of TiSpark, including (but not limited to):

- A randomly generated telemetry ID.
- Some configuration information of TiSpark, such as the read engine and whether streaming read is enabled.
- Cluster deployment information, such as the machine hardware information, OS information, and component version number of the node where TiSpark is located.

You can view TiSpark usage information that is collected in Spark logs. You can set the Spark log level to INFO or lower, for example:

```
cat {spark.log} | grep Telemetry report | tail -n 1
```

14.12.2 Disable telemetry

14.12.2.1 Disable TiDB telemetry at deployment

When the telemetry is enabled in existing TiDB clusters, you can configure `enable-telemetry = false` on each TiDB instance to disable the TiDB telemetry collection on that instance, which does not take effect until you restart the cluster.

Detailed steps to disable telemetry in different deployment tools are listed below.

Binary deployment

Create a configuration file `tidb_config.toml` with the following content:

```
enable-telemetry = false
```

Specify the `--config=tidb_config.toml` command-line parameter when starting TiDB for the configuration file above to take effect.

See TiDB Configuration Options and TiDB Configuration File for details.

Deployment using TiUP Playground

Create a configuration file `tidb_config.toml` with the following content:

```
enable-telemetry = false
```

When starting TiUP Playground, specify the `--db.config tidb_config.toml` command-line parameter for the configuration file above to take effect. For example:

```
tiup playground --db.config tidb_config.toml
```

See Quickly Deploy a Local TiDB Cluster for details.

Deployment using TiUP Cluster

Modify the deployment topology file `topology.yaml` to add the following content:

```
server_configs:
  tidb:
    enable-telemetry: false
```

Deployment on Kubernetes via TiDB Operator

Configure `spec.tidb.config.enable-telemetry: false` in `tidb-cluster.yaml` or TidbCluster Custom Resource.

See Deploy TiDB Operator on Kubernetes for details.

> **Note:**
>
> This configuration item requires TiDB Operator v1.1.3 or later to take effect.

14.12.2.2 Disable TiDB telemetry for deployed TiDB clusters

In existing TiDB clusters, you can also modify the system variable `tidb_enable_telemetry` ↪ to dynamically disable the TiDB telemetry collection:

```
SET GLOBAL tidb_enable_telemetry = 0;
```

> **Note:**
>
> When you disable telemetry, the configuration file has a higher priority over system variable. That is, after telemetry collection is disabled by the configuration file, the value of the system variable will be ignored.

14.12.2.3 Disable TiDB Dashboard telemetry

Configure `dashboard.enable-telemetry = false` to disable the TiDB Dashboard telemetry collection on all PD instances. You need to restart the running clusters for the configuration to take effect.

Detailed steps to disable telemetry for different deployment tools are listed below.

Binary deployment

Create a configuration file `pd_config.toml` with the following content:

```
[dashboard]
enable-telemetry = false
```

Specify the `--config=pd_config.toml` command-line parameter when starting PD to take effect.

See PD Configuration Flags and PD Configuration File for details.

Deployment using TiUP Playground

Create a configuration file `pd_config.toml` with the following content:

```
[dashboard]
enable-telemetry = false
```

When starting TiUP Playground, specify the `--pd.config pd_config.toml` command-line parameter to take effect, for example:

```
tiup playground --pd.config pd_config.toml
```

See Quickly Deploy a Local TiDB Cluster for details.

Deployment using TiUP Cluster

Modify the deployment topology file `topology.yaml` to add the following content:

```
server_configs:
  pd:
    dashboard.enable-telemetry: false
```

Deployment on Kubernetes via TiDB Operator

Configure `spec.pd.config.dashboard.enable-telemetry: false` in `tidb-cluster.yaml` or TidbCluster Custom Resource.

See Deploy TiDB Operator on Kubernetes for details.

> **Note:**
>
> This configuration item requires TiDB Operator v1.1.3 or later to take effect.

14.12.2.4 Disable TiUP telemetry

To disable the TiUP telemetry collection, execute the following command:

```
tiup telemetry disable
```

14.12.3 Check telemetry status

For TiDB telemetry, execute the following SQL statement to check the telemetry status:

```
ADMIN SHOW TELEMETRY;
```

If the `DATA_PREVIEW` column in the execution result is empty, TiDB telemetry is disabled. If not, TiDB telemetry is enabled. You can also check when the usage information was shared previously according to the `LAST_STATUS` column and whether the sharing was successful or not.

For TiUP telemetry, execute the following command to check the telemetry status:

```
tiup telemetry status
```

14.12.4 Compliance

To meet compliance requirements in different countries or regions, the usage information is sent to servers located in different countries according to the IP address of the sender machine:

- For IP addresses from the Chinese mainland, usage information is sent to and stored on cloud servers in the Chinese mainland.

- For IP addresses from outside of the Chinese mainland, usage information is sent to and stored on cloud servers in the US.

See PingCAP Privacy Policy for details.

14.13 Error Codes and Troubleshooting

This document describes the problems encountered during the use of TiDB and provides the solutions.

14.13.1 Error codes

TiDB is compatible with the error codes in MySQL, and in most cases returns the same error code as MySQL. For a list of error codes for MySQL, see MySQL 5.7 Error Message Reference. In addition, TiDB has the following unique error codes:

> **Note:**
>
> Some error codes stand for internal errors. Normally, TiDB handles the error rather than return it to the user, so some error codes are not listed here.
>
> If you encounter an error code that is not listed here, get support from PingCAP or the community.

- Error Number: 8001

  The memory used by the request exceeds the threshold limit for the TiDB memory usage.

  Increase the memory limit for a single SQL statement by configuring the system variable `tidb_mem_quota_query`.

- Error Number: 8002

  To guarantee consistency, a transaction with the `SELECT FOR UPDATE` statement cannot be retried when it encounters a commit conflict. TiDB rolls back the transaction and returns this error.

  The application can safely retry the whole transaction.

- Error Number: 8003

  If the data in a row is not consistent with the index when executing the `ADMIN CHECK TABLE` command, TiDB returns this error. This error is commonly seen when you check the data corruption in the table.

  You can get support from PingCAP or the community.

- Error Number: 8004

  A single transaction is too large.

  See the error message `transaction too large` for the cause and solution.

- Error Number: 8005

  The complete error message: `ERROR 8005 (HY000): Write Conflict, txnStartTS is stale`

  Transactions in TiDB encounter write conflicts. To handle this error, check whether `tidb_disable_txn_auto_retry` is set to `on`. If so, set it to `off`; if it is already `off`, increase the value of `tidb_retry_limit` until the error no longer occurs.

- Error Number: 8018

  When you reload a plugin, if the plugin has not been loaded before, this error is returned.

  You can execute an initial load of the plugin.

- Error Number: 8019

  The version of the plugin that is being reloaded is different from the previous version. Therefore, the plugin cannot be reloaded, and this error is returned.

  You can reload the plugin by ensuring that the plugin version is the same as the previous one.

- Error Number: 8020

  When the table is locked, if you perform a write operation on the table, this error is returned.

  Unlock the table and retry the write operation.

- Error Number: 8021

  When the key to be read from TiKV does not exist, this error is returned. This error is used internally, and the external result is an empty read.

- Error Number: 8022

  The transaction commit fails and has been rolled back.

  The application can safely retry the whole transaction.

- Error Number: 8023

  If you set an empty value when writing the transaction cache, this error is returned. This error is used and dealt with internally, and is not returned to the application.

- Error Number: 8024

  Invalid transactions. If TiDB finds that no transaction ID (Start Timestamp) is obtained for the transaction that is being executed, which means this transaction is invalid, this error is returned.

Usually this error does not occur. If you encounter this error, get support from PingCAP or the community.

- Error Number: 8025

  The single Key-Value pair being written is too large. The largest single Key-Value pair supported in TiDB is 6 MB by default.

  If a pair exceeds this limit, you need to properly adjust the `txn-entry-size-limit` configuration value to relax the limit.

- Error Number: 8026

  The interface function being used has not been implemented. This error is only used internally, and is not returned to the application.

- Error Number: 8027

  The table schema version is outdated. TiDB applies schema changes online. When the table schema version of the TiDB server is earlier than that of the entire system, this error is returned if you execute a SQL statement.

  When this error occurs, check the network between the TiDB server and the PD Leader.

- Error Number: 8028

  TiDB does not support table lock, which is called metadata lock in MySQL and might be called intention lock in other databases.

  When a transaction is executed, the transaction cannot recognize the table schema changes. Therefore, when committing a transaction, TiDB checks the table schema related the transaction. If the related table schema has changed during the execution, the transaction commit will fail and this error is returned.

  The application can safely retry the whole transaction.

- Error Number: 8029

  This error occurs when numeric conversion within the database encounters an error. This error is only used internally and is converted to a specific type of error for external applications.

- Error Number: 8030

  After an unsigned positive integer is converted to a signed integer, it exceeds the maximum value and displays as a negative integer. This error mostly occurs in the alert message.

- Error Number: 8031

  When being converted to an unsigned integer, a negative integer is converted to a positive integer. This error mostly occurs in the alert message.

- Error Number: 8032

  Invalid `year` format is used. `year` only accepts 1, 2 or 4 digits.

- Error Number: 8033

  Invalid `year` value is used. The valid range of `year` is (1901, 2155).

- Error Number: 8037

  Invalid `mode` format is used in the `week` function. `mode` must be 1 digit within [0, 7].

- Error Number: 8038

  The field fails to obtain the default value. This error is usually used internally, and is converted to a specific type of error for external applications.

- Error Number: 8040

  Unsupported operations are performed. For example, you perform a table locking operation on a view or a sequence.

- Error Number: 8047

  The value of the system variable is not supported. This error usually occurs in the alarm information when the user sets a variable value that is not supported in the database.

- Error Number: 8048

  An unsupported database isolation level is set.

  If you cannot modify the codes because you are using a third-party tool or framework, consider using `tidb_skip_isolation_level_check` to bypass this check.

```
set @@tidb_skip_isolation_level_check = 1;
```

- Error Number: 8050

  An unsupported privilege type is set.

  See Privileges required for TiDB operations for the solution.

- Error Number: 8051

  Unknown data type is encountered when TiDB parses the Exec argument list sent by the client.

  If you encounter this error, check the client. If the client is normal, get support from PingCAP or the community.

- Error Number: 8052

  The serial number of the data packet from the client is incorrect.

  If you encounter this error, check the client. If the client is normal, get support from PingCAP or the community.

- Error Number: 8055

  The current snapshot is too old. The data may have been garbage collected. You can increase the value of `tidb_gc_life_time` to avoid this problem. TiDB automatically reserves data for long-running transactions. Usually this error does not occur.

  See garbage collection overview and garbage collection configuration.

- Error Number: 8059

  The auto-random ID is exhausted and cannot be allocated. There is no way to recover from such errors currently. It is recommended to use bigint when using the auto random feature to obtain the maximum number of assignment. And try to avoid manually assigning values to the auto random column.

  See auto random for reference.

- Error Number: 8060

  Invalid auto-incrementing offset. Check the values of `auto_increment_increment` and `auto_increment_offset`.

- Error Number: 8061

  Unsupported SQL Hint.

  See Optimizer Hints to check and modify the SQL Hint.

- Error Number: 8062

  An invalid token is used in SQL Hint. It conflicts with reserved words in SQL Hint.

  See Optimizer Hints to check and modify the SQL Hint.

- Error Number: 8063

  The limited memory usage set in SQL Hint exceeds the upper limit of the system. The setting in SQL Hint is ignored.

  See Optimizer Hints to check and modify the SQL Hint.

- Error Number: 8064

  It fails to parse SQL Hint.

  See Optimizer Hints to check and modify the SQL Hint.

- Error Number: 8065

  An invalid integer is used in SQL Hint.

  See Optimizer Hints to check and modify the SQL Hint.

- Error Number: 8066

  The second parameter in the `JSON_OBJECTAGG` function is invalid.

- Error Number: 8101

  The format of plugin ID is incorrect.

  The correct format is `[name]-[version]`, and no `-` is allowed in `name` and `version`.

- Error Number: 8102

  Unable to read the plugin definition information.

  Check the configuration related to the plugin.

- Error Number: 8103

  The plugin name is incorrect.

  Check the configuration of the plugin.

- Error Number: 8104

  The plugin version does not match.

  Check the configuration of the plugin.

- Error Number: 8105

  The plugin is repeatedly loaded.

- Error Number: 8106

  The plugin defines a system variable whose name does not begin with the plugin name.

  Contact the developer of the plugin to modify, or get support from PingCAP or the community.

- Error Number: 8107

  The loaded plugin does not specify a version, or the specified version is too low.

  Check the configuration of the plugin.

- Error Number: 8108

  Unsupported execution plan type. This error is an internal error.

  If you encounter this error, get support from PingCAP or the community.

- Error Number: 8109

  The specified index cannot be found when the index is analyzed.

- Error Number: 8110

  The Cartesian product operation cannot be executed.

  Set `cross-join` in the configuration to `true`.

- Error Number: 8111

  When executing the `EXECUTE` statement, the corresponding `Prepare` statement cannot be found.

- Error Number: 8112

  The number of parameters in the `EXECUTE` statement is not consistent with the `Prepare` statement.

- Error Number: 8113

  The table schema related in the `EXECUTE` statement has changed after the `Prepare` statement is executed.

- Error Number: 8115

  It is not supported to prepare multiple lines of statements.

- Error Number: 8116

  It is not supported to prepare DDL statements.

- Error Number: 8120

  The `start tso` of transactions cannot be obtained.

  Check the state/monitor/log of the PD server and the network between the TiDB server and the PD server.

- Error Number: 8121

  Privilege check fails.

  Check the privilege configuration of the database.

- Error Number: 8122

  No corresponding table name is found, given the specified wild cards.

- Error Number: 8123

  An SQL query with aggregate functions returns non-aggregated columns, which violates the `only_full_group_by` mode.

  Modify the SQL statement or disable the `only_full_group_by` mode.

- Error Number: 8129

  TiDB does not yet support JSON objects with the key length >= 65536.

- Error Number: 8130

  The complete error message: `ERROR 8130 (HY000): client has multi-statement capability disabled`

  This error might occur after you upgrade from an earlier version of TiDB. To reduce the impact of SQL injection attacks, TiDB now prevents multiple queries from being executed in the same `COM_QUERY` call by default.

  The system variable `tidb_multi_statement_mode` can be used to control this behavior.

- Error Number: 8138

  The transaction attempts to write an incorrect row value. For more information, see Troubleshoot Inconsistency Between Data and Indexes.

- Error Number: 8139

  The transaction attempts to write a row whose handle is inconsistent with that in the index. For more information, see Troubleshoot Inconsistency Between Data and Indexes.

- Error Number: 8140

  The transaction attempts to write a row whose data is inconsistent with the index data. For more information, see Troubleshoot Inconsistency Between Data and Indexes.

- Error Number: 8141

  When a transaction is being committed, the existence assertion of a key fails. For more information,see Troubleshoot Inconsistency Between Data and Indexes.

- Error Number: 8143

  During the execution of a non-transactional DML statement, if a batch fails, the statement is stopped. For more information, see Non-transactional DML statements.

- Error Number: 8200

  The DDL syntax is not yet supported.

  See compatibility of MySQL DDL for reference.

- Error Number: 8214

  The DDL operation is terminated by the `admin cancel` operation.

- Error Number: 8215

  `ADMIN REPAIR TABLE` fails.

  If you encounter this error, get support from PingCAP or the community.

- Error Number: 8216

  The usage of automatic random columns is incorrect.

  See auto random to modify.

- Error Number: 8223

  This error occurs when detecting that the data is not consistent with the index.

  If you encounter this error, get support from PingCAP or the community.

- Error Number: 8224

  The DDL job cannot be found.

  Check whether the job id specified by the `restore` operation exists.

- Error Number: 8225

  The DDL operation is completed and cannot be canceled.

- Error Number: 8226

  The DDL operation is almost completed and cannot be canceled.

- Error Number: 8227

  Unsupported options are used when creating Sequence.

  See Sequence documentation to find the list of the supported options.

- Error Number: 8228

  Unsupported types are specified when using `setval` on Sequence.

  See Sequence documentation to find the example of the function.

- Error Number: 8229

  The transaction exceeds the survival time.

  Commit or roll back the current transaction, and start a new transaction.

- Error Number: 8230

  TiDB currently does not support using Sequence as the default value on newly added columns, and reports this error if you use it.

- Error Number: 9001

  The complete error message: `ERROR 9001 (HY000): PD Server Timeout`

  The PD request timed out.

  Check the status, monitoring data and log of the PD server, and the network between the TiDB server and the PD server.

- Error Number: 9002

  The complete error message: `ERROR 9002 (HY000): TiKV Server Timeout`

  The TiKV request timed out.

  Check the status, monitoring data and log of the TiKV server, and the network between the TiDB server and the TiKV server.

- Error Number: 9003

  The complete error message: `ERROR 9003 (HY000): TiKV Server is Busy`

  The TiKV server is busy and this usually occurs when the workload is too high.

  Check the status, monitoring data, and log of the TiKV server.

- Error Number: 9004

  The complete error message: `ERROR 9004 (HY000): Resolve Lock Timeout`

  A lock resolving timeout. This error occurs when a large number of transactional conflicts exist in the database.

  Check the application code to see whether lock contention exists in the database.

- Error Number: 9005

  The complete error message: `ERROR 9005 (HY000): Region is unavailable`

  The accessed Region or a certain Raft Group is not available, with possible reasons such as insufficient replicas. This error usually occurs when the TiKV server is busy or the TiKV node is down.

  Check the status, monitoring data and log of the TiKV server.

- Error Number: 9006

  The complete error message: `ERROR 9006 (HY000): GC life time is shorter than transaction duration`

  The interval of `GC Life Time` is too short. The data that should have been read by long transactions might be deleted. You can adjust `tidb_gc_life_time` using the following command:

```
SET GLOBAL tidb_gc_life_time = '30m';
```

> **Note:**
> "30m" means only cleaning up the data generated 30 minutes ago, which might consume some extra storage space.

- Error Number: 9500

  A single transaction is too large.

  See the error message `transaction too large` for the solution.

- Error Number: 9007

  The complete error message: `ERROR 9007 (HY000): Write Conflict`

  Transactions in TiKV encounter write conflicts.

  Check whether `tidb_disable_txn_auto_retry` is set to `on`. If so, set it to `off`; if it is already `off`, increase the value of `tidb_retry_limit` until the error no longer occurs.

- Error Number: 9008

  Too many requests are sent to TiKV at the same time. The number exceeds limit.

  Increase `tidb_store_limit` or set it to `0` to remove the limit on the traffic of requests.

- Error Number: 9010

  TiKV cannot process this raft log.

  Check the state/monitor/log of the TiKV server.

- Error Number: 9012

  The TiFlash request timed out.

  Check the state/monitor/log of the TiFlash server and the network between the TiDB server and TiFlash server.

- Error Number: 9013

  The TiFlash server is busy and this usually occurs when the workload is too high.

  Check the state/monitor/log of the TiFlash server.

14.13.1.1 MySQL native error messages

- Error Number: 2013 (HY000)

  The complete error message: `ERROR 2013 (HY000): Lost connection to MySQL server during query`

  You can handle this error as follows:

  - Check whether panic is in the log.
  - Check whether OOM exists in dmesg using `dmesg -T | grep -i oom`.
  - A long time of no access might also lead to this error. It is usually caused by TCP timeout. If TCP is not used for a long time, the operating system kills it.

- Error Number: 1105 (HY000)

  The complete error message: `ERROR 1105 (HY000): other error: unknown error Wire Error(InvalidEnumValue(4004))`

  This error usually occurs when the version of TiDB does not match with that of TiKV. To avoid version mismatch, upgrade all components when you upgrade the version.

- Error Number: 1148 (42000)

  The complete error message: `ERROR 1148 (42000): the used command is not allowed with this TiDB version`

  When you execute the `LOAD DATA LOCAL` statement but the MySQL client does not allow executing this statement (the value of the `local_infile` option is 0), this error occurs.

  The solution is to use the `--local-infile=1` option when you start the MySQL client. For example, run the command `mysql --local-infile=1 -u root -h 127.0.0.1 -P 4000`. The default value of `local-infile` varies in different versions of the MySQL client. Therefore, you need to configure it in specific MySQL clients.

- Error Number: 9001 (HY000)

  The complete error message: `ERROR 9001 (HY000): PD server timeout start timestamp may fall behind safe point`

  This error occurs when TiDB fails to access PD. A worker in the TiDB background continuously queries the safepoint from PD and reports this error if it fails to query within 100s. Generally, it is because the disk on PD is slow and busy or the network failed between TiDB and PD. For the details of common errors, see Error Number and Fault Diagnosis.

- TiDB log error message: EOF error

  When the client or proxy disconnects from TiDB, TiDB does not immediately notice the disconnection. Instead, TiDB notices the disconnection only when it begins to return data to the connection. At this time, the log prints an EOF error.

14.13.2 Troubleshooting

See the troubleshooting and FAQ documents.

14.14 Table Filter

The TiDB migration tools operate on all the databases by default, but oftentimes only a subset is needed. For example, you only want to work with the schemas in the form of `foo*` and `bar*` and nothing else.

Since TiDB 4.0, all TiDB migration tools share a common filter syntax to define subsets. This document describes how to use the table filter feature.

14.14.1 Usage

14.14.1.1 CLI

Table filters can be applied to the tools using multiple `-f` or `--filter` command line parameters. Each filter is in the form of `db.table`, where each part can be a wildcard (further explained in the next section). The following lists the example usage.

- BR:

```
./br backup full -f 'foo*.*' -f 'bar*.*' -s 'local:///tmp/backup'
```

```
./br restore full -f 'foo*.*' -f 'bar*.*' -s 'local:///tmp/backup'
```

- Dumpling:

```
./dumpling -f 'foo*.*' -f 'bar*.*' -P 3306 -o /tmp/data/
```

- TiDB Lightning:

```
./tidb-lightning -f 'foo*.*' -f 'bar*.*' -d /tmp/data/ --backend tidb
```

14.14.1.2 TOML configuration files

Table filters in TOML files are specified as array of strings. The following lists the example usage.

- TiDB Lightning:

```
[mydumper]
filter = ['foo*.*', 'bar*.*']
```

- TiCDC:

```
[filter]
rules = ['foo*.*', 'bar*.*']

[[sink.dispatchers]]
matcher = ['db1.*', 'db2.*', 'db3.*']
dispatcher = 'ts'
```

14.14.2 Syntax

14.14.2.1 Plain table names

Each table filter rule consists of a "schema pattern" and a "table pattern", separated by a dot (`.`). Tables whose fully-qualified name matches the rules are accepted.

```
db1.tbl1
db2.tbl2
db3.tbl3
```

A plain name must only consist of valid identifier characters, such as:

- digits (`0` to `9`)
- letters (`a` to `z`, `A` to `Z`)
- `$`
- `_`
- non ASCII characters (U+0080 to U+10FFFF)

All other ASCII characters are reserved. Some punctuations have special meanings, as described in the next section.

14.14.2.2 Wildcards

Each part of the name can be a wildcard symbol described in fnmatch(3):

- `*` — matches zero or more characters
- `?` — matches one character
- `[a-z]` — matches one character between "a" and "z" inclusively
- `[!a-z]` — matches one character except "a" to "z".

```
db[0-9].tbl[0-9a-f][0-9a-f]
data.*
*.backup_*
```

"Character" here means a Unicode code point, such as:

- U+00E9 (é) is 1 character.
- U+0065 U+0301 (é) are 2 characters.
- U+1F926 U+1F3FF U+200D U+2640 U+FE0F ( ) are 5 characters.

14.14.2.3 File import

To import a file as the filter rule, include an `@` at the beginning of the rule to specify the file name. The table filter parser treats each line of the imported file as additional filter rules.

For example, if a file `config/filter.txt` has the following content:

```
employees.*
*.WorkOrder
```

the following two invocations are equivalent:

```
./dumpling -f '@config/filter.txt'
./dumpling -f 'employees.*' -f '*.WorkOrder'
```

A filter file cannot further import another file.

14.14.2.4 Comments and blank lines

Inside a filter file, leading and trailing white-spaces of every line are trimmed. Furthermore, blank lines (empty strings) are ignored.

A leading `#` marks a comment and is ignored. `#` not at start of line is considered syntax error.

```
## this line is a comment
db.table  # but this part is not comment and may cause error
```

14.14.2.5 Exclusion

An `!` at the beginning of the rule means the pattern after it is used to exclude tables from being processed. This effectively turns the filter into a block list.

```
*.*
#^ note: must add the *.* to include all tables first
!*.Password
!employees.salaries
```

14.14.2.6 Escape character

To turn a special character into an identifier character, precede it with a backslash `\`.

```
db\.with\.dots.*
```

For simplicity and future compatibility, the following sequences are prohibited:

- `\` at the end of the line after trimming whitespaces (use `[ ]` to match a literal whitespace at the end).
- `\` followed by any ASCII alphanumeric character (`[0-9a-zA-Z]`). In particular, C-like escape sequences like `\0`, `\r`, `\n` and `\t` currently are meaningless.

14.14.2.7 Quoted identifier

Besides `\`, special characters can also be suppressed by quoting using `"` or `` ` ``.

```
"db.with.dots"."tbl\1"
`db.with.dots`.`tbl\2`
```

The quotation mark can be included within an identifier by doubling itself.

```
"foo""bar".`foo``bar`
## equivalent to:
foo\"bar.foo\`bar
```

Quoted identifiers cannot span multiple lines.

It is invalid to partially quote an identifier:

```
"this is "invalid*.*
```

14.14.2.8 Regular expression

In case very complex rules are needed, each pattern can be written as a regular expression delimited with `/`:

```
/^db\d{2,}$/./^tbl\d{2,}$/
```

These regular expressions use the Go dialect. The pattern is matched if the identifier contains a substring matching the regular expression. For instance, `/b/` matches `db01`.

> **Note:**
>
> Every `/` in the regular expression must be escaped as `\/`, including inside `[…]`. You cannot place an unescaped `/` between `\Q…\E`.

14.14.3 Multiple rules

When a table name matches none of the rules in the filter list, the default behavior is to ignore such unmatched tables.

To build a block list, an explicit `*.*` must be used as the first rule, otherwise all tables will be excluded.

```
## every table will be filtered out
./dumpling -f '!*.Password'

## only the "Password" table is filtered out, the rest are included.
./dumpling -f '*.*' -f '!*.Password'
```

In a filter list, if a table name matches multiple patterns, the last match decides the outcome. For instance:

```
## rule 1
employees.*
## rule 2
!*.dep*
## rule 3
*.departments
```

The filtered outcome is as follows:

Table name	Rule 1	Rule 2	Rule 3	Outcome
irrelevant.table				Default (reject)
employees.employees				Rule 1 (accept)
employees.dept_emp				Rule 2 (reject)
employees.departments				Rule 3 (accept)
else.departments				Rule 3 (accept)

Note:

In TiDB tools, the system schemas are always excluded in the default configuration. The system schemas are:

- `INFORMATION_SCHEMA`
- `PERFORMANCE_SCHEMA`
- `METRICS_SCHEMA`
- `INSPECTION_SCHEMA`
- `mysql`
- `sys`

14.15 Schedule Replicas by Topology Labels

Note:

TiDB v5.3.0 introduces Placement Rules in SQL. This offers a more convenient way to configure the placement of tables and partitions. Placement Rules in SQL might replace placement configuration with PD in future releases.

To improve the high availability and disaster recovery capability of TiDB clusters, it is recommended that TiKV nodes are physically scattered as much as possible. For example, TiKV nodes can be distributed on different racks or even in different data centers. According to the topology information of TiKV, the PD scheduler automatically performs scheduling at the background to isolate each replica of a Region as much as possible, which maximizes the capability of disaster recovery.

To make this mechanism effective, you need to properly configure TiKV and PD so that the topology information of the cluster, especially the TiKV location information, is reported to PD during deployment. Before you begin, see Deploy TiDB Using TiUP first.

14.15.1 Configure `labels` based on the cluster topology

14.15.1.1 Configure `labels` for TiKV

You can use the command-line flag or set the TiKV or TiFlash configuration file to bind some attributes in the form of key-value pairs. These attributes are called `labels`. After

TiKV and TiFlash are started, they report their `labels` to PD so users can identify the location of TiKV and TiFlash nodes.

Assume that the topology has four layers: zone > data center (dc) > rack > host, and you can use these labels (zone, dc, rack, host) to set location of the TiKV and TiFlash. To set labels for TiKV and TiFlash, you can use one of the following methods:

- Use the command-line flag to start a TiKV instance:

  ```
  shell tikv-server --labels zone=<zone>,dc=<dc>,rack=<rack>,host=<host>
  ```

- Configure in the TiKV configuration file:

  ```
  [server]
  [server.labels]
  zone = "<zone>"
  dc = "<dc>"
  rack = "<rack>"
  host = "<host>"
  ```

14.15.1.2 Configure `location-labels` for PD

According to the description above, the label can be any key-value pair used to describe TiKV attributes. But PD cannot identify the location-related labels and the layer relationship of these labels. Therefore, you need to make the following configuration for PD to understand the TiKV node topology.

Defined as an array of strings, `location-labels` is the configuration for PD. Each item of this configuration corresponds to the key of TiKV `labels`. Besides, the sequence of each key represents the layer relationship of different labels (the isolation levels decrease from left to right).

You can customize the value of `location-labels`, such as `zone`, `rack`, or `host`, because the configuration does not have default values. Also, this configuration has **no** restriction in the number of label levels (not mandatory for 3 levels) as long as they match with TiKV server labels.

Note:

- To make configurations take effect, you must configure `location-labels` for PD and `labels` for TiKV at the same time. Otherwise, PD does not perform scheduling according to the topology.
- If you use Placement Rules in SQL, you only need to configure `labels` for TiKV. Currently, Placement Rules in SQL is incompatible with the `location-labels` configuration of PD and ignores this configuration. It is not recommended to use `location-labels` and Placement Rules in SQL at the same time; otherwise, unexpected results might occur.

To configure `location-labels`, choose one of the following methods according to your cluster situation:

- If the PD cluster is not initialized, configure `location-labels` in the PD configuration file:

```
[replication]
location-labels = ["zone", "rack", "host"]
```

- If the PD cluster is already initialized, use the pd-ctl tool to make online changes:

```
pd-ctl config set location-labels zone,rack,host
```

14.15.1.3 Configure `isolation-level` for PD

If `location-labels` has been configured, you can further enhance the topological isolation requirements on TiKV clusters by configuring `isolation-level` in the PD configuration file.

Assume that you have made a three-layer cluster topology by configuring `location-labels` according to the instructions above: zone -> rack -> host, you can configure the `isolation-level` to `zone` as follows:

```
[replication]
isolation-level = "zone"
```

If the PD cluster is already initialized, you need to use the pd-ctl tool to make online changes:

```
pd-ctl config set isolation-level zone
```

The `location-level` configuration is an array of strings, which needs to correspond to a key of `location-labels`. This parameter limits the minimum and mandatory isolation level requirements on TiKV topology clusters.

> **Note:**
>
> `isolation-level` is empty by default, which means there is no mandatory restriction on the isolation level. To set it, you need to configure `location-labels` for PD and ensure that the value of `isolation-level` is one of `location-labels` names.

14.15.1.4 Configure a cluster using TiUP (recommended)

When using TiUP to deploy a cluster, you can configure the TiKV location in the initialization configuration file. TiUP will generate the corresponding configuration files for TiKV, PD, and TiFlash during deployment.

In the following example, a two-layer topology of `zone/host` is defined. The TiKV nodes of the cluster are distributed among three zones, z1, z2, and z3, with each zone having four hosts, h1, h2, h3, and h4. In z1, four TiKV instances are deployed on two hosts, `tikv-1` and `tikv-2` on h1, and `tikv-3` and `tikv-4` on h2. Two TiFlash instances are deployed on the other two hosts, `tiflash-1` on h3 and `tiflash-2` on h4. In z2 and z3, two TiKV instances are deployed on two hosts, and two TiFlash instances are deployed on the other two hosts. In the following example, `tikv-n` represents the IP address of the `n`th TiKV node, and `tiflash-n` represents the IP address of the `n`th TiFlash node.

```
server_configs:
  pd:
    replication.location-labels: ["zone", "host"]

tikv_servers:
## z1
  - host: tikv-1
    config:
      server.labels:
        zone: z1
        host: h1
   - host: tikv-2
    config:
      server.labels:
        zone: z1
        host: h1
  - host: tikv-3
    config:
      server.labels:
        zone: z1
        host: h2
  - host: tikv-4
    config:
      server.labels:
        zone: z1
        host: h2
## z2
  - host: tikv-5
    config:
      server.labels:
        zone: z2
        host: h1
```

```
   - host: tikv-6
    config:
      server.labels:
        zone: z2
        host: h2
## z3
  - host: tikv-7
    config:
      server.labels:
        zone: z3
        host: h1
  - host: tikv-8
    config:
      server.labels:
        zone: z3
        host: h2s
tiflash_servers:
## z1
  - host: tiflash-1
    learner_config:
      server.labels:
        zone: z1
        host: h3
   - host: tiflash-2
    learner_config:
      server.labels:
        zone: z1
        host: h4
## z2
  - host: tiflash-3
    learner_config:
      server.labels:
        zone: z2
        host: h3
   - host: tiflash-4
    learner_config:
      server.labels:
        zone: z2
        host: h4
## z3
  - host: tiflash-5
    learner_config:
      server.labels:
        zone: z3
        host: h3
```

```
- host: tiflash-6
  learner_config:
    server.labels:
      zone: z3
      host: h4
```

For details, see Geo-distributed Deployment topology.

> **Note:**
>
> If you have not configured `replication.location-labels` in the configuration file, when you deploy a cluster using this topology file, an error might occur. It is recommended that you confirm `replication.location-labels` is configured in the configuration file before deploying a cluster.

14.15.2 PD schedules based on topology label

PD schedules replicas according to the label layer to make sure that different replicas of the same data are scattered as much as possible.

Take the topology in the previous section as an example.

Assume that the number of cluster replicas is 3 (`max-replicas=3`). Because there are 3 zones in total, PD ensures that the 3 replicas of each Region are respectively placed in z1, z2, and z3. In this way, the TiDB cluster is still available when one data center fails.

Then, assume that the number of cluster replicas is 5 (`max-replicas=5`). Because there are only 3 zones in total, PD cannot guarantee the isolation of each replica at the zone level. In this situation, the PD scheduler will ensure replica isolation at the host level. In other words, multiple replicas of a Region might be distributed in the same zone but not on the same host.

In the case of the 5-replica configuration, if z3 fails or is isolated as a whole, and cannot be recovered after a period of time (controlled by `max-store-down-time`), PD will make up the 5 replicas through scheduling. At this time, only 4 hosts are available. This means that host-level isolation cannot be guaranteed and that multiple replicas might be scheduled to the same host. But if the `isolation-level` value is set to `zone` instead of being left empty, this specifies the minimum physical isolation requirements for Region replicas. That is to say, PD will ensure that replicas of the same Region are scattered among different zones. PD will not perform corresponding scheduling even if following this isolation restriction does not meet the requirement of `max-replicas` for multiple replicas.

For example, a TiKV cluster is distributed across three data zones z1, z2, and z3. Each Region has three replicas as required, and PD distributes the three replicas of the same Region to these three data zones respectively. If a power outage occurs in z1 and cannot

be recovered after a period of time (controlled by `max-store-down-time` and 30 minutes by default), PD determines that the Region replicas on z1 are no longer available. However, because `isolation-level` is set to `zone`, PD needs to strictly guarantee that different replicas of the same Region will not be scheduled on the same data zone. Because both z2 and z3 already have replicas, PD will not perform any scheduling under the minimum isolation level restriction of `isolation-level`, even if there are only two replicas at this moment.

Similarly, when `isolation-level` is set to `rack`, the minimum isolation level applies to different racks in the same data center. With this configuration, the isolation at the zone layer is guaranteed first if possible. When the isolation at the zone level cannot be guaranteed, PD tries to avoid scheduling different replicas to the same rack in the same zone. The scheduling works similarly when `isolation-level` is set to `host` where PD first guarantees the isolation level of rack, and then the level of host.

In summary, PD maximizes the disaster recovery of the cluster according to the current topology. Therefore, if you want to achieve a certain level of disaster recovery, deploy more machines on different sites according to the topology than the number of `max-replicas`. TiDB also provides mandatory configuration items such as `isolation-level` for you to more flexibly control the topological isolation level of data according to different scenarios.

15 FAQs

15.1 TiDB FAQ Summary

This document summarizes frequently asked questions (FAQs) about TiDB.

Category	Related documents
TiDB architecture and principles	TiDB Architecture FAQs

| Deployment |

Deployment FAQs

TiUP FAQs

TiDB on Kubernetes FAQs

| | Data migration |

Data Migration FAQs

Data import

TiDB Lightning FAQs

DM FAQs

Incremental data replication

TiCDC FAQs

TiDB Binlog FAQs

| | Data backup and restore | Backup & Restore FAQs | | SQL operations | SQL FAQs | | Cluster upgrade | TiDB Upgrade FAQs | | Cluster management | Cluster Management FAQs | | Monitor and alert |

Monitoring FAQs

TiDB Dashboard FAQs

TiDB Cluster Alert Rules

| | High availability and high reliability |

High Availability FAQs

High Reliability FAQs

| | Common error codes | Error Codes and Troubleshooting |

15.2 TiDB Architecture FAQs

This document lists the Most Frequently Asked Questions about TiDB.

15.2.1 TiDB introduction and architecture

15.2.1.1 What is TiDB?

TiDB is an open-source distributed SQL database that supports Hybrid Transactional and Analytical Processing (HTAP) workloads. It is MySQL compatible and features horizontal scalability, strong consistency, and high availability. The goal of TiDB is to provide users with a one-stop database solution that covers OLTP (Online Transactional Processing), OLAP (Online Analytical Processing), and HTAP services. TiDB is suitable for various use cases that require high availability and strong consistency with large-scale data.

15.2.1.2 What is TiDB's architecture?

The TiDB cluster has three components: the TiDB server, the PD (Placement Driver) server, and the TiKV server. For more details, see TiDB architecture, TiDB storage, TiDB computing, and TiDB scheduling.

15.2.1.3 Is TiDB based on MySQL?

No. TiDB supports MySQL syntax and protocol, but it is a new open source database that is developed and maintained by PingCAP, Inc.

15.2.1.4 What is the respective responsibility of TiDB, TiKV and PD (Placement Driver)?

- TiDB works as the SQL computing layer, mainly responsible for parsing SQL, specifying query plan, and generating executor.
- TiKV works as a distributed Key-Value storage engine, used to store the real data. In short, TiKV is the storage engine of TiDB.
- PD works as the cluster manager of TiDB, which manages TiKV metadata, allocates timestamps, and makes decisions for data placement and load balancing.

15.2.1.5 Is it easy to use TiDB?

Yes, it is. When all the required services are started, you can use TiDB as easily as a MySQL server. You can replace MySQL with TiDB to power your applications without changing a single line of code in most cases. You can also manage TiDB using the popular MySQL management tools.

15.2.1.6 How is TiDB compatible with MySQL?

Currently, TiDB supports the majority of MySQL 5.7 syntax, but does not support triggers, stored procedures, user-defined functions, and foreign keys. For more details, see Compatibility with MySQL.

15.2.1.7 Does TiDB support distributed transactions?

Yes. TiDB distributes transactions across your cluster, whether it is a few nodes in a single location or many nodes across multiple data centers.

Inspired by Google's Percolator, the transaction model in TiDB is mainly a two-phase commit protocol with some practical optimizations. This model relies on a timestamp allocator to assign the monotone increasing timestamp for each transaction, so conflicts can be detected. PD works as the timestamp allocator in a TiDB cluster.

15.2.1.8 What programming language can I use to work with TiDB?

Any language supported by MySQL client or driver.

15.2.1.9 Can I use other Key-Value storage engines with TiDB?

Yes. In addition to TiKV, TiDB supports standalone storage engines such as UniStore and MockTiKV. Note that in later TiDB releases, MockTiKV might NO LONGER be supported.

To check all TiDB-supported storage engines, use the following command:

```
./bin/tidb-server -h
```

The output is as follows:

```
Usage of ./bin/tidb-server:
  -L string
        log level: info, debug, warn, error, fatal (default "info")
  -P string
        tidb server port (default "4000")
  -V    print version information and exit (default false)
.........
  -store string
        registered store name, [tikv, mocktikv, unistore] (default "unistore
            ↪ ")
  ......
```

15.2.1.10 In addition to the TiDB documentation, are there any other ways to acquire TiDB knowledge?

- TiDB documentation: the most important and timely way to get TiDB related knowledge.
- TiDB blogs: learn technical articles, product insights, and case studies.
- PingCAP Education: take online courses and certification programs.

15.2.1.11 What is the length limit for the TiDB user name?

32 characters at most.

15.2.1.12 What are the limits on the number of columns and row size in TiDB?

- The maximum number of columns in TiDB defaults to 1017. You can adjust the number up to 4096.
- The maximum size of a single row defaults to 6 MB. You can increase the number up to 120 MB.

For more information, see TiDB Limitations.

15.2.1.13 Does TiDB support XA?

No. The JDBC driver of TiDB is MySQL Connector/J. When using Atomikos, set the data source to `type="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource"`. TiDB does not support the connection with MySQL JDBC XADataSource. MySQL JDBC XADataSource only works for MySQL (for example, using DML to modify the `redo` log).

After you configure the two data sources of Atomikos, set the JDBC drives to XA. When Atomikos operates TM and RM (DB), Atomikos sends the command including XA to the

JDBC layer. Taking MySQL for an example, when XA is enabled in the JDBC layer, JDBC will send a series of XA logic operations to InnoDB, including using DML to change the `redo` log. This is the operation of the two-phase commit. The current TiDB version does not support the upper application layer JTA/XA and does not parse XA operations sent by Atomikos.

As a standalone database, MySQL can only implement across-database transactions using XA; while TiDB supports distributed transactions using Google Percolator transaction model and its performance stability is higher than XA, so TiDB does not support JTA/XA and there is no need for TiDB to support XA.

15.2.1.14 How could TiDB support high concurrent `INSERT` or `UPDATE` operations to the columnar storage engine (TiFlash) without hurting performance?

- TiFlash introduces a special structure named DeltaTree to process the modification of the columnar engine.
- TiFlash acts as the learner role in a Raft group, so it does not vote for the log commit or writes. This means that DML operations do not have to wait for the acknowledgment of TiFlash, which is why TiFlash does not slow down the OLTP performance. In addition, TiFlash and TiKV work in separate instances, so they do not affect each other.

15.2.1.15 Is TiFlash eventually consistent?

Yes. TiFlash maintains strong data consistency by default.

15.2.2 TiDB techniques

15.2.2.1 TiKV for data storage

See TiDB Internal (I) - Data Storage.

15.2.2.2 TiDB for data computing

See TiDB Internal (II) - Computing.

15.2.2.3 PD for scheduling

See TiDB Internal (III) - Scheduling.

15.3 SQL FAQs

This document summarizes the FAQs related to SQL operations in TiDB.

15.3.1 Does TiDB support the secondary key?

Yes. You can have the `NOT NULL` constraint on a non-primary key column with a unique secondary index. In this case, the column works as a secondary key.

15.3.2 How does TiDB perform when executing DDL operations on a large table?

DDL operations of TiDB on large tables are usually not an issue. TiDB supports online DDL operations, and these DDL operations do not block DML operations.

For some DDL operations such as adding columns, deleting columns or dropping indexes, TiDB can perform these operations quickly.

For some heavy DDL operations such as adding indexes, TiDB needs to backfill data, which takes a longer time (depending on the size of the table) and consumes additional resources. The impact on online traffic is tunable. TiDB can do the backfill with multiple threads, and the resource consumed can be set by the following system variables:

- `tidb_ddl_reorg_worker_cnt`
- `tidb_ddl_reorg_priority`
- `tidb_ddl_error_count_limit`
- `tidb_ddl_reorg_batch_size`

15.3.3 How to choose the right query plan? Do I need to use hints? Or can I use hints?

TiDB includes a cost-based optimizer. In most cases, the optimizer chooses the optimal query plan for you. If the optimizer does not work well, you can still use optimizer hints to intervene with the optimizer.

In addition, you can also use the SQL binding to fix the query plan for a particular SQL statement.

15.3.4 How to prevent the execution of a particular SQL statement?

You can create SQL bindings with the `MAX_EXECUTION_TIME` hint to limit the execution time of a particular statement to a small value (for example, 1ms). In this way, the statement is terminated automatically by the threshold.

For example, to prevent the execution of `SELECT * FROM t1, t2 WHERE t1.id = t2.id`, you can use the following SQL binding to limit the execution time of the statement to 1ms:

```
CREATE GLOBAL BINDING for
    SELECT * FROM t1, t2 WHERE t1.id = t2.id
USING
    SELECT /*+ MAX_EXECUTION_TIME(1) */ * FROM t1, t2 WHERE t1.id = t2.id;
```

> **Note:**
>
> The precision of `MAX_EXECUTION_TIME` is roughly 100ms. Before TiDB terminates the SQL statement, the tasks in TiKV might be started. To reduce the TiKV resource consumption in such case, it is recommended to set `tidb_enable_paging` to `ON`.

Dropping this SQL binding will remove the limit.

```
DROP GLOBAL BINDING for
    SELECT * FROM t1, t2 WHERE t1.id = t2.id;
```

15.3.5 What are the MySQL variables that TiDB is compatible with?

See System Variables.

15.3.6 The order of results is different from MySQL when `ORDER BY` is omitted

It is not a bug. The default order of records depends on various situations without any guarantee of consistency.

The order of results in MySQL might appear stable because queries are executed in a single thread. However, it is common that query plans can change when upgrading to new versions. It is recommended to use `ORDER BY` whenever an order of results is desired.

The reference can be found in ISO/IEC 9075:1992, Database Language SQL- July 30, 1992, which states as follows:

> If an `<order by clause>` is not specified, then the table specified by the `<cursor specification>` is T and the ordering of rows in T is implementation-dependent.

In the following two queries, both results are considered legal:

```
> select * from t;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
```

```
|    2 |    2 |
+------+------+
2 rows in set (0.00 sec)
```

```
> select * from t; -- the order of results is not guaranteed
+------+------+
| a    | b    |
+------+------+
|    2 |    2 |
|    1 |    1 |
+------+------+
2 rows in set (0.00 sec)
```

If the list of columns used in the `ORDER BY` is non-unique, the statement is also considered non-deterministic. In the following example, the column `a` has duplicate values. Thus, only `ORDER BY a, b` is guaranteed deterministic:

```
> select * from t order by a;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    1 |
|    2 |    2 |
+------+------+
3 rows in set (0.00 sec)
```

In the following statement, the order of column `a` is guaranteed, but the order of `b` is not guaranteed.

```
> select * from t order by a;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    2 |    1 |
+------+------+
3 rows in set (0.00 sec)
```

In TiDB, you can also use the system variable `tidb_enable_ordered_result_mode` to sort the final output result automatically.

15.3.7 Does TiDB support `SELECT FOR UPDATE`?

Yes. When using pessimistic locking (the default since TiDB v3.0.8) the `SELECT FOR UPDATE` execution behaves similar to MySQL.

When using optimistic locking, `SELECT FOR UPDATE` does not lock data when the transaction is started, but checks conflicts when the transaction is committed. If the check reveals conflicts, the committing transaction rolls back.

For details, see description of the `SELECT` syntax elements.

15.3.8 Can the codec of TiDB guarantee that the UTF-8 string is memcomparable? Is there any coding suggestion if our key needs to support UTF-8?

TiDB uses the UTF-8 character set by default and currently only supports UTF-8. The string of TiDB uses the memcomparable format.

15.3.9 What is the maximum number of statements in a transaction?

The maximum number of statements in a transaction is 5000 by default.

In the optimistic transaction mode, When transaction retry is enabled, the default upper limit is 5000. You can adjust the limit by using the `stmt-count-limit` parameter.

15.3.10 Why does the auto-increment ID of the later inserted data is smaller than that of the earlier inserted data in TiDB?

The auto-increment ID feature in TiDB is only guaranteed to be automatically incremental and unique but is not guaranteed to be allocated sequentially. Currently, TiDB is allocating IDs in batches. If data is inserted into multiple TiDB servers simultaneously, the allocated IDs are not sequential. When multiple threads concurrently insert data to multiple `tidb-server` instances, the auto-increment ID of the later inserted data might be smaller. TiDB allows specifying `AUTO_INCREMENT` for the integer field, but allows only one `AUTO_INCREMENT` field in a single table. For details, see Auto-increment ID and the AUTO_INCREMENT attribute.

15.3.11 How do I modify the `sql_mode` in TiDB?

TiDB supports modifying the `sql_mode` system variables on a SESSION or GLOBAL basis.

- Changes to `GLOBAL` scoped variables propagate to the rest servers of the cluster and persist across restarts. This means that you do not need to change the `sql_mode` value on each TiDB server.
- Changes to `SESSION` scoped variables only affect the current client session. After restarting a server, the changes are lost.

15.3.12 Error: `java.sql.BatchUpdateExecption:statement count 5001 exceeds the transaction limitation` while using Sqoop to write data into TiDB in batches

In Sqoop, `--batch` means committing 100 statements in each batch, but by default each statement contains 100 SQL statements. So, 100 * 100 = 10000 SQL statements, which exceeds 5000, the maximum number of statements allowed in a single TiDB transaction.

Two solutions:

- Add the `-Dsqoop.export.records.per.statement=10` option as follows:

```
sqoop export \
    -Dsqoop.export.records.per.statement=10 \
    --connect jdbc:mysql://mysql.example.com/sqoop \
    --username sqoop ${user} \
    --password ${passwd} \
    --table ${tab_name} \
    --export-dir ${dir} \
    --batch
```

- You can also increase the limited number of statements in a single TiDB transaction, but this will consume more memory. For details, see Limitations on SQL statements.

15.3.13 Does TiDB have a function like the Flashback Query in Oracle? Does it support DDL?

Yes, it does. And it supports DDL as well. For details, see Read Historical Data Using the `AS OF TIMESTAMP` Clause.

15.3.14 Does TiDB release space immediately after deleting data?

None of the `DELETE`, `TRUNCATE` and `DROP` operations release data immediately. For the `TRUNCATE` and `DROP` operations, after the TiDB GC (Garbage Collection) time (10 minutes by default), the data is deleted and the space is released. For the `DELETE` operations, the data is deleted but the space is not immediately released until the compaction is performed.

15.3.15 Why does the query speed get slow after data is deleted?

Deleting a large amount of data leaves a lot of useless keys, affecting the query efficiency. To solve the problem, you can use the Region Merge feature. For details, see the deleting data section in TiDB Best Practices.

15.3.16 What should I do if it is slow to reclaim storage space after deleting data?

Because TiDB uses multi-version concurrency control (MVCC), when the old data is overwritten with new data, the old data is not replaced but retained along with the new data. Timestamps are used to identify the data version. Deleting data does not immediately reclaim space. Garbage collection is delayed so that concurrent transactions are able to see earlier versions of rows. This can be configured via the `tidb_gc_life_time` (default: `10m0s`) system variable.

15.3.17 Does `SHOW PROCESSLIST` display the system process ID?

The display content of TiDB `SHOW PROCESSLIST` is almost the same as that of MySQL `SHOW PROCESSLIST`. TiDB `SHOW PROCESSLIST` does not display the system process ID. The ID that it displays is the current session ID. The differences between TiDB `SHOW PROCESSLIST` and MySQL `SHOW PROCESSLIST` are as follows:

- As TiDB is a distributed database, the `tidb-server` instance is a stateless engine for parsing and executing the SQL statements (for details, see TiDB architecture). `SHOW PROCESSLIST` displays the session list executed in the `tidb-server` instance that the user logs in to from the MySQL client, not the list of all the sessions running in the cluster. But MySQL is a standalone database and its `SHOW PROCESSLIST` displays all the SQL statements executed in MySQL.
- The `State` column in TiDB is not continually updated during query execution. Because TiDB supports parallel query, each statement might be in multiple *states* at once, and so it is difficult to simplify to a single value.

15.3.18 How to control or change the execution priority of SQL commits?

TiDB supports changing the priority on a global or individual statement basis. Priority has the following meaning:

- `HIGH_PRIORITY`: this statement has a high priority, that is, TiDB gives priority to this statement and executes it first.
- `LOW_PRIORITY`: this statement has a low priority, that is, TiDB reduces the priority of this statement during the execution period.
- `DELAYED`: this statement has normal priority and is the same as the `NO_PRIORITY` setting for `tidb_force_priority`.

You can combine the above two parameters with the DML of TiDB to use them. For example:

1. Adjust the priority by writing SQL statements in the database:

```
SELECT HIGH_PRIORITY | LOW_PRIORITY | DELAYED COUNT(*) FROM table_name
    ↪ ;
INSERT HIGH_PRIORITY | LOW_PRIORITY | DELAYED INTO table_name
    ↪ insert_values;
DELETE HIGH_PRIORITY | LOW_PRIORITY | DELAYED FROM table_name;
UPDATE HIGH_PRIORITY | LOW_PRIORITY | DELAYED table_reference SET
    ↪ assignment_list WHERE where_condition;
REPLACE HIGH_PRIORITY | LOW_PRIORITY | DELAYED INTO table_name;
```

2. The full table scan statement automatically adjusts itself to a low priority. ANALYZE has a low priority by default.

15.3.19 What's the trigger strategy for `auto analyze` in TiDB?

When the number of rows in a table or a single partition of a partitioned table reaches 1000, and the ratio (the number of modified rows / the current total number of rows) of the table or partition is larger than tidb_auto_analyze_ratio, the ANALYZE statement is automatically triggered.

The default value of the `tidb_auto_analyze_ratio` system variable is `0.5`, indicating that this feature is enabled by default. It is not recommended to set the value of `tidb_auto_analyze_ratio` to be larger than or equal to pseudo-estimate-ratio (the default value is `0.8`), otherwise the optimizer might use pseudo statistics. TiDB v5.3.0 introduces the tidb_enable_pseudo_for_outdated_stats variable, and when you set it to `OFF`, pseudo statistics are not used even if the statistics are outdated.

To disable `auto analyze`, use the system variable tidb_enable_auto_analyze.

15.3.20 Can I use optimizer hints to override the optimizer behavior?

TiDB supports multiple ways to override the default query optimizer behavior, including hints and SQL Plan Management. The basic usage is similar to MySQL, with several TiDB specific extensions:

```
SELECT column_name FROM table_name USE INDEX ( index_name ) WHERE
    ↪ where_condition;
```

15.3.21 Why the `Information schema is changed` error is reported?

TiDB handles the SQL statement using the `schema` of the time and supports online asynchronous DDL change. A DML statement and a DDL statement might be executed at the same time and you must ensure that each statement is executed using the same `schema`. Therefore, when the DML operation meets the ongoing DDL operation, the `Information ↪ schema is changed` error might be reported. Some improvements have been made to prevent too many error reportings during the DML operation.

Now, there are still a few causes for this error reporting:

- Cause 1: Some tables involved in the DML operation are the same tables involved in the ongoing DDL operation. To check the ongoing DDL operations, use the `ADMIN SHOW DDL` statement.
- Cause 2: The DML operation goes on for a long time. During this period, many DDL statements have been executed, which causes more than 1024 `schema` version changes. You can modify this default value by modifying the `tidb_max_delta_schema_count` variable.
- Cause 3: The TiDB server that accepts the DML request is not able to load `schema information` for a long time (possibly caused by the connection failure between TiDB and PD or TiKV). During this period, many DDL statements have been executed, which causes more than 100 `schema` version changes.
- Cause 4: After TiDB restarts and before the first DDL operation is executed, the DML operation is executed and then encounters the first DDL operation (which means before the first DDL operation is executed, the transaction corresponding to the DML is started. And after the first `schema` version of the DDL is changed, the transaction corresponding to the DML is committed), this DML operation reports this error.

In the preceding causes, only Cause 1 is related to tables. Cause 1 and Cause 2 do not impact the application, as the related DML operations retry after failure. For cause 3, you need to check the network between TiDB and TiKV/PD.

Note:

- Currently, TiDB does not cache all the `schema` version changes.
- For each DDL operation, the number of `schema` version changes is the same with the number of corresponding `schema state` version changes.
- Different DDL operations cause different number of `schema` version changes. For example, the `CREATE TABLE` statement causes one `schema` version change while the `ADD COLUMN` statement causes four.

15.3.22 What are the causes of the "Information schema is out of date" error?

When executing a DML statement, if TiDB fails to load the latest schema within a DDL lease (45s by default), the `Information schema is out of date` error might occur. Possible causes are:

- The TiDB instance that executed this DML was killed, and the transaction execution corresponding to this DML statement took longer than a DDL lease. When the transaction was committed, the error occurred.

- TiDB failed to connect to PD or TiKV while executing this DML statement. As a result, TiDB failed to load schema within a DDL lease or disconnected from PD due to the keepalive setting.

15.3.23 Error is reported when executing DDL statements under high concurrency?

When you execute DDL statements (such as creating tables in batches) under high concurrency, a very few of these statements might fail because of key conflicts during the concurrent execution.

It is recommended to keep the number of concurrent DDL statements under 20. Otherwise, you need to retry the failed statements from the client.

15.3.24 SQL optimization

15.3.24.1 TiDB execution plan description

See Understand the Query Execution Plan.

15.3.24.2 Statistics collection

See Introduction to Statistics.

15.3.24.3 How to optimize `select count(1)`?

The `count(1)` statement counts the total number of rows in a table. Improving the degree of concurrency can significantly improve the speed. To modify the concurrency, refer to the `tidb_distsql_scan_concurrency` document. But it also depends on the CPU and I/O resources. TiDB accesses TiKV in every query. When the amount of data is small, all MySQL is in memory, and TiDB needs to conduct a network access.

Recommendations:

- Improve the hardware configuration. See Software and Hardware Requirements.
- Improve the concurrency. The default value is 10. You can improve it to 50 and have a try. But usually the improvement is 2-4 times of the default value.
- Test the `count` in the case of large amount of data.
- Optimize the TiKV configuration. See Tune TiKV Thread Performance and Tune TiKV Memory Performance.
- Enable the Coprocessor Cache.

15.3.24.4 How to view the progress of the current DDL job?

You can use `ADMIN SHOW DDL` to view the progress of the current DDL job. The operation is as follows:

```
ADMIN SHOW DDL;
```

```
*************************** 1. row ***************************
  SCHEMA_VER: 140
       OWNER: 1a1c4174-0fcd-4ba0-add9-12d08c4077dc
RUNNING_JOBS: ID:121, Type:add index, State:running, SchemaState:write
    ↪ reorganization, SchemaID:1, TableID:118, RowCount:77312, ArgLen:0,
    ↪ start time: 2018-12-05 16:26:10.652 +0800 CST, Err:<nil>, ErrCount:0,
    ↪  SnapshotVersion:404749908941733890
     SELF_ID: 1a1c4174-0fcd-4ba0-add9-12d08c4077dc
```

From the above results, you can get that the `ADD INDEX` operation is currently being processed. You can also get from the `RowCount` field of the `RUNNING_JOBS` column that now the `ADD INDEX` operation has added 77312 rows of indexes.

15.3.24.5 How to view the DDL job?

- `ADMIN SHOW DDL`: to view the running DDL job
- `ADMIN SHOW DDL JOBS`: to view all the results in the current DDL job queue (including tasks that are running and waiting to run) and the last ten results in the completed DDL job queue
- `ADMIN SHOW DDL JOBS QUERIES 'job_id' [, 'job_id'] ...`: to view the original SQL statement of the DDL task corresponding to the `job_id`; the `job_id` only searches the running DDL job and the last ten results in the DDL history job queue.

15.3.24.6 Does TiDB support CBO (Cost-Based Optimization)? If yes, to what extent?

Yes. TiDB uses the cost-based optimizer. The cost model and statistics are constantly optimized. TiDB also supports join algorithms like hash join and sort-merge join.

15.3.24.7 How to determine whether I need to execute `analyze` on a table?

View the `Healthy` field using `SHOW STATS_HEALTHY` and generally you need to execute `ANALYZE` on a table when the field value is smaller than 60.

15.3.24.8 What is the ID rule when a query plan is presented as a tree? What is the execution order for this tree?

No rule exists for these IDs but the IDs are unique. When IDs are generated, a counter works and adds one when one plan is generated. The execution order has nothing to do with the ID. The whole query plan is a tree and the execution process starts from the root node and the data is returned to the upper level continuously. For details about the query plan, see Understanding the TiDB Query Execution Plan.

15.3.24.9 In the TiDB query plan, `cop` tasks are in the same root. Are they executed concurrently?

Currently the computing tasks of TiDB belong to two different types of tasks: `cop task` and `root task`.

`cop task` is the computing task which is pushed down to the KV end for distributed execution; `root task` is the computing task for single point execution on the TiDB end.

Generally the input data of `root task` comes from `cop task`; when `root task` processes data, `cop task` of TiKV can processes data at the same time and waits for the pull of `root task` of TiDB. Therefore, `cop` tasks can be considered as executed concurrently with `root task`; but their data has an upstream and downstream relationship. During the execution process, they are executed concurrently during some time. For example, the first `cop task` is processing the data in [100, 200] and the second `cop task` is processing the data in [1, 100]. For details, see Understanding the TiDB Query Plan.

15.3.25 Database optimization

15.3.25.1 Edit TiDB options

See The TiDB Command Options.

15.3.25.2 How to avoid hotspot issues and achieve load balancing? Is hot partition or range an issue in TiDB?

To learn the scenarios that cause hotspots, refer to common hotpots. The following TiDB features are designed to help you solve hotspot issues:

- The `SHARD_ROW_ID_BITS` attribute. After setting this attribute, row IDs are scattered and written into multiple Regions, which can alleviate the write hotspot issue.
- The `AUTO_RANDOM` attribute, which helps resolve hotspots brought by auto-increment primary keys.
- Coprocessor Cache, for read hotspots on small tables.
- Load Base Split, for hotspots caused by unbalanced access between Regions, such as full table scans for small tables.
- Cached tables, for frequently accessed but rarely updated small hotspot tables.

If you have a performance issue caused by hotspot, refer to Troubleshoot Hotspot Issues to get it resolved.

15.3.25.3 Tune TiKV performance

See Tune TiKV Thread Performance and Tune TiKV Memory Performance.

15.4 TiDB Deployment FAQs

This document summarizes the FAQs related to TiDB deployment.

15.4.1 Software and hardware requirements

15.4.1.1 What operating systems does TiDB support?

For the TiDB-supported operating systems, see Software and Hardware Recommendations.

15.4.1.2 What is the recommended hardware configuration for a TiDB cluster in the development, test, or production environment?

You can deploy and run TiDB on the 64-bit generic hardware server platform in the Intel x86-64 architecture or on the hardware server platform in the ARM architecture. For the requirements and recommendations about server hardware configuration for development, test, and production environments, see Software and Hardware Recommendations - Server recommendations.

15.4.1.3 What's the purposes of 2 network cards of 10 gigabit?

As a distributed cluster, TiDB has a high demand on time, especially for PD, because PD needs to distribute unique timestamps. If the time in the PD servers is not consistent, it takes longer waiting time when switching the PD server. The bond of two network cards guarantees the stability of data transmission, and 10 gigabit guarantees the transmission speed. Gigabit network cards are prone to meet bottlenecks, therefore it is strongly recommended to use 10 gigabit network cards.

15.4.1.4 Is it feasible if we don't use RAID for SSD?

If the resources are adequate, it is recommended to use RAID 10 for SSD. If the resources are inadequate, it is acceptable not to use RAID for SSD.

15.4.1.5 What's the recommended configuration of TiDB components?

- TiDB has a high requirement on CPU and memory. If you need to enable TiDB Binlog, the local disk space should be increased based on the service volume estimation and the time requirement for the GC operation. But the SSD disk is not a must.
- PD stores the cluster metadata and has frequent Read and Write requests. It demands a high I/O disk. A disk of low performance will affect the performance of the whole cluster. It is recommended to use SSD disks. In addition, a larger number of Regions has a higher requirement on CPU and memory.
- TiKV has a high requirement on CPU, memory and disk. It is required to use SSD.

For details, see Software and Hardware Recommendations.

15.4.2 Installation and deployment

For the production environment, it is recommended to use TiUP to deploy your TiDB cluster. See Deploy a TiDB Cluster Using TiUP.

15.4.2.1 Why the modified `toml` configuration for TiKV/PD does not take effect?

You need to set the `--config` parameter in TiKV/PD to make the `toml` configuration effective. TiKV/PD does not read the configuration by default. Currently, this issue only occurs when deploying using Binary. For TiKV, edit the configuration and restart the service. For PD, the configuration file is only read when PD is started for the first time, after which you can modify the configuration using pd-ctl. For details, see PD Control User Guide.

15.4.2.2 Should I deploy the TiDB monitoring framework (Prometheus + Grafana) on a standalone machine or on multiple machines? What is the recommended CPU and memory?

The monitoring machine is recommended to use standalone deployment. It is recommended to use an 8 core CPU with 16 GB+ memory and a 500 GB+ hard disk.

15.4.2.3 Why the monitor cannot display all metrics?

Check the time difference between the machine time of the monitor and the time within the cluster. If it is large, you can correct the time and the monitor will display all the metrics.

15.4.2.4 What is the function of supervise/svc/svstat service?

- supervise: the daemon process, to manage the processes
- svc: to start and stop the service
- svstat: to check the process status

15.4.2.5 Description of inventory.ini variables

Variable	Description
`cluster_name`	the name of a cluster, adjustable
`tidb_version`	the version of TiDB

Variable	Description
`deployment_method`	the method of deployment, binary by default, Docker optional
`process_supervision`	the supervision way of processes, systemd by default, supervise optional
`timezone`	the timezone of the managed node, adjustable, `Asia/Shanghai` by default, used with the `set_timezone` variable

Variable	Description
`set_timezone`	to edit the timezone of the managed node, True by default; False means closing
`enable_chrony`	currently not supported
`enable_firewalld`	to enable the firewall, closed by default
`enable_ntpd`	to monitor the NTP service of the managed node, True by default; do not close it

Variable	Description
`machine_to_benchmark`	to monitor the disk IOPS of the managed node, True by default; do not close it unless
`set_hostname`	True to set the hostname of the managed node based on the IP, False by default

Variable	Description
`enable_binlog`	whether to deploy Pump and enable the binlog, False by default, dependent on the Kafka cluster; see the `zookeeper_addrs` variable
`zookeeper_addrs`	the ZooKeeper address of the binlog Kafka cluster

Variable	Description
`enable_slow_query_log` ↪	record the slow query log of TiDB into a single file: ({{ deploy_dir }}/log/tidb_slow_query.log). False by default, to record it into the TiDB log

Variable	Description
`deploy_without_tidb` ↪	Key-Value mode, deploy only PD, TiKV and the monitoring service, not TiDB; set the IP of the tidb_servers host group to null in the `inventory` ↪ . ↪ `ini` file

15.4.2.6 How to separately record the slow query log in TiDB? How to locate the slow query SQL statement?

1. The slow query definition for TiDB is in the TiDB configuration file. The `tidb_slow_log_threshold: 300` parameter is used to configure the threshold value of the slow query (unit: millisecond).
2. If a slow query occurs, you can locate the `tidb-server` instance where the slow query is and the slow query time point using Grafana and find the SQL statement information recorded in the log on the corresponding node.
3. In addition to the log, you can also view the slow query using the `ADMIN SHOW SLOW` command. For details, see `ADMIN SHOW SLOW` command.

15.4.2.7 How to add the `label` configuration if `label` of TiKV was not configured when I deployed the TiDB cluster for the first time?

The configuration of TiDB `label` is related to the cluster deployment architecture. It is important and is the basis for PD to execute global management and scheduling. If you did not configure `label` when deploying the cluster previously, you should adjust the deployment structure by manually adding the `location-labels` information using the PD management tool `pd-ctl`, for example, `config set location-labels "zone,rack,host"` (you should configure it based on the practical `label` level name).

For the usage of `pd-ctl`, see PD Control User Guide.

15.4.2.8 Why does the `dd` command for the disk test use the `oflag=direct` option?

The Direct mode wraps the Write request into the I/O command and sends this command to the disk to bypass the file system cache and directly test the real I/O Read/Write performance of the disk.

15.4.2.9 How to use the `fio` command to test the disk performance of the TiKV instance?

- Random Read test:

```
./fio -ioengine=psync -bs=32k -fdatasync=1 -thread -rw=randread -size
    ↪ =10G -filename=fio_randread_test.txt -name='fio randread test' -
    ↪ iodepth=4 -runtime=60 -numjobs=4 -group_reporting --output-format
    ↪ =json --output=fio_randread_result.json
```

- The mix test of sequential Write and random Read:

```
./fio -ioengine=psync -bs=32k -fdatasync=1 -thread -rw=randrw -
    ↪ percentage_random=100,0 -size=10G -filename=
    ↪ fio_randread_write_test.txt -name='fio mixed randread and
    ↪ sequential write test' -iodepth=4 -runtime=60 -numjobs=4 -
    ↪ group_reporting --output-format=json --output=
    ↪ fio_randread_write_test.json
```

15.4.3 What public cloud vendors are currently supported by TiDB?

TiDB supports deployment on Google GKE, AWS EKS, and Alibaba Cloud ACK.

In addition, TiDB is currently available on JD Cloud and UCloud.

15.5 Migration FAQs

This document summarizes the frequently asked questions (FAQs) related to TiDB data migration.

For the frequently asked questions about migration-related tools, click the corresponding links in the list below:

- Backup & Restore FAQs
- TiDB Binlog FAQ
- TiDB Lightning FAQs
- TiDB Data Migration (DM) FAQs
- TiCDC FAQs

15.5.1 Full data export and import

15.5.1.1 How to migrate an application running on MySQL to TiDB?

Because TiDB supports most MySQL syntax, generally you can migrate your applications to TiDB without changing a single line of code in most cases.

15.5.1.2 Data import and export is slow, and many retries and EOF errors appear in the log of each component without other errors

If no other logical errors occur, retries and EOF errors might be caused by network issues. It is recommended to first use tools to check the network connectivity. In the following example, iperf is used for troubleshooting:

- Execute the following command on the server-side node where the retries and EOF errors occur:

```
iperf3 -s
```

- Execute the following command on the client-side node where the retries and EOF errors occur:

```
iperf3 -c <server-IP>
```

The following example is the output of a client node with a good network connection:

```
$ iperf3 -c 192.168.196.58
Connecting to host 192.168.196.58, port 5201
[  5] local 192.168.196.150 port 55397 connected to 192.168.196.58 port 5201
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00  sec  18.0 MBytes  150 Mbits/sec
[  5]   1.00-2.00  sec  20.8 MBytes  175 Mbits/sec
[  5]   2.00-3.00  sec  18.2 MBytes  153 Mbits/sec
[  5]   3.00-4.00  sec  22.5 MBytes  188 Mbits/sec
[  5]   4.00-5.00  sec  22.4 MBytes  188 Mbits/sec
[  5]   5.00-6.00  sec  22.8 MBytes  191 Mbits/sec
```

```
[  5]   6.00-7.00  sec  20.8 MBytes  174 Mbits/sec
[  5]   7.00-8.00  sec  20.1 MBytes  168 Mbits/sec
[  5]   8.00-9.00  sec  20.8 MBytes  175 Mbits/sec
[  5]   9.00-10.00 sec  21.8 MBytes  183 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval        Transfer    Bitrate
[  5]   0.00-10.00 sec  208 MBytes  175 Mbits/sec                  sender
[  5]   0.00-10.00 sec  208 MBytes  174 Mbits/sec                  receiver

iperf Done.
```

If the output shows low network bandwidth and high bandwidth fluctuations, a large number of retries and EOF errors might appear in each component log. In this case, you need to consult your network service provider to improve the network quality.

If the output of each metric looks good, try to update each component. If the problem persists after the updating, get support from PingCAP or the community.

15.5.1.3 If I accidentally import the MySQL user table into TiDB, or forget the password and cannot log in, how to deal with it?

Restart the TiDB service, add the `-skip-grant-table=true` parameter in the configuration file. Log into the cluster without password and recreate the user, or recreate the `mysql.user` table. For the specific table schema, search the official documentation.

15.5.1.4 How to export the data in TiDB?

You can use the following methods to export the data in TiDB:

- Export data using mysqldump and the `WHERE` clause.
- Use the MySQL client to export the results of `select` to a file.

15.5.1.5 How to migrate from DB2 or Oracle to TiDB?

To migrate all the data or migrate incrementally from DB2 or Oracle to TiDB, see the following solution:

- Use the official migration tool of Oracle, such as OGG, Gateway, CDC (Change Data Capture).
- Develop a program for importing and exporting data.
- Export Spool as text file, and import data using Load infile.
- Use a third-party data migration tool.

Currently, it is recommended to use OGG.

15.5.1.6 Error: `java.sql.BatchUpdateExecption:statement count 5001 exceeds the transaction limitation` while using Sqoop to write data into TiDB in `batches`

In Sqoop, `--batch` means committing 100 `statements` in each batch, but by default each `statement` contains 100 SQL statements. So, 100 * 100 = 10000 SQL statements, which exceeds 5000, the maximum number of statements allowed in a single TiDB transaction.

Two solutions:

- Add the `-Dsqoop.export.records.per.statement=10` option as follows:

```
sqoop export \
    -Dsqoop.export.records.per.statement=10 \
    --connect jdbc:mysql://mysql.example.com/sqoop \
    --username sqoop ${user} \
    --password ${passwd} \
    --table ${tab_name} \
    --export-dir ${dir} \
    --batch
```

- You can also increase the limited number of statements in a single TiDB transaction, but this will consume more memory.

15.5.1.7 Why does Dumpling return `The local disk space is insufficient` error or cause the upstream database to run out of memory when exporting a table?

This issue might have the following causes:

- The database's primary keys are not evenly distributed (for example, when you enable `SHARD_ROW_ID_BITS`).
- The upstream database is TiDB and the exported table is a partitioned table.

For the above cases, Dumpling splits excessively large data chunk for the export and sends queries with excessively large results. To address the issue, you can get the latest version of Dumpling.

15.5.1.8 Does TiDB have a function like the Flashback Query in Oracle? Does it support DDL?

Yes, it does. And it supports DDL as well. For details, see how TiDB reads data from history versions.

15.5.2 Migrate the data online

15.5.2.1 Is there a current solution to replicating data from TiDB to other databases like HBase and Elasticsearch?

No. Currently, the data replication depends on the application itself.

15.5.3 Migrate the traffic

15.5.3.1 How to migrate the traffic quickly?

It is recommended to migrate application data from MySQL to TiDB using TiDB Data Migration tool. You can migrate the read and write traffic in batches by editing the network configuration as needed. Deploy a stable network LB (such as HAproxy, LVS, F5, and DNS) on the upper layer, in order to implement seamless migration by directly editing the network configuration.

15.5.3.2 Is there a limit for the total write and read capacity in TiDB?

The total read capacity has no limit. You can increase the read capacity by adding more TiDB servers. Generally the write capacity has no limit as well. You can increase the write capacity by adding more TiKV nodes.

15.5.3.3 The error message `transaction too large` is displayed

Due to the limitation of the underlying storage engine, each key-value entry (one row) in TiDB should be no more than 6MB. You can adjust the `txn-entry-size-limit` configuration value up to 120MB.

Distributed transactions need two-phase commit and the bottom layer performs the Raft replication. If a transaction is very large, the commit process would be quite slow and the write conflict is more likely to occur. Moreover, the rollback of a failed transaction leads to an unnecessary performance penalty. To avoid these problems, we limit the total size of key-value entries to no more than 100MB in a transaction by default. If you need larger transactions, modify the value of `txn-total-size-limit` in the TiDB configuration file. The maximum value of this configuration item is up to 10G. The actual limitation is also affected by the physical memory of the machine.

There are similar limits on Google Cloud Spanner.

15.5.3.4 How to import data in batches?

When you import data, insert in batches and keep the number of rows within 10,000 for each batch.

15.5.3.5 Does TiDB release space immediately after deleting data?

None of the `DELETE`, `TRUNCATE` and `DROP` operations release data immediately. For the `TRUNCATE` and `DROP` operations, after the TiDB GC (Garbage Collection) time (10 minutes by default), the data is deleted and the space is released. For the `DELETE` operation, the data is deleted but the space is not released according to TiDB GC. When subsequent data is written into RocksDB and executes `COMPACT`, the space is reused.

15.5.3.6 Can I execute DDL operations on the target table when loading data?

No. None of the DDL operations can be executed on the target table when you load data, otherwise the data fails to be loaded.

15.5.3.7 Does TiDB support the `replace into` syntax?

Yes.

15.5.3.8 Why does the query speed getting slow after deleting data?

Deleting a large amount of data leaves a lot of useless keys, affecting the query efficiency. Currently the Region Merge feature is in development, which is expected to solve this problem. For details, see the deleting data section in TiDB Best Practices.

15.5.3.9 What is the most efficient way of deleting data?

When deleting a large amount of data, it is recommended to use `Delete from t where xx limit 5000;`. It deletes through the loop and uses `Affected Rows == 0` as a condition to end the loop, so as not to exceed the limit of transaction size. With the prerequisite of meeting business filtering logic, it is recommended to add a strong filter index column or directly use the primary key to select the range, such as `id >= 5000*n+m and id < 5000*(n+1)+m`.

If the amount of data that needs to be deleted at a time is very large, this loop method will get slower and slower because each deletion traverses backward. After deleting the previous data, lots of deleted flags remain for a short period (then all will be processed by Garbage Collection) and influence the following Delete statement. If possible, it is recommended to refine the Where condition. See details in TiDB Best Practices.

15.5.3.10 How to improve the data loading speed in TiDB?

- The TiDB Lightning tool is developed for distributed data import. It should be noted that the data import process does not perform a complete transaction process for performance reasons. Therefore, the ACID constraint of the data being imported during the import process cannot be guaranteed. The ACID constraint of the imported data can only be guaranteed after the entire import process ends. Therefore, the

applicable scenarios mainly include importing new data (such as a new table or a new index) or the full backup and restoring (truncate the original table and then import data).
- Data loading in TiDB is related to the status of disks and the whole cluster. When loading data, pay attention to metrics like the disk usage rate of the host, TiClient Error, Backoff, Thread CPU and so on. You can analyze the bottlenecks using these metrics.

15.6 Upgrade and After Upgrade FAQs

This document introduces some FAQs and their solutions when or after you upgrade TiDB.

15.6.1 Upgrade FAQs

This section lists some FAQs and their solutions when you upgrade TiDB.

15.6.1.1 What are the effects of rolling updates?

When you apply rolling updates to the TiDB services, the running application is affected to varying degrees. Therefore, it is not recommended that you perform a rolling update during business peak hours. You need to configure the minimum cluster topology (TiDB * 2, PD * 3, TiKV * 3). If the Pump or Drainer service is involved in the cluster, it is recommended to stop Drainer before rolling updates. When you upgrade TiDB, Pump is also upgraded.

15.6.1.2 Can I upgrade the TiDB cluster during the DDL execution?

DO NOT upgrade a TiDB cluster when a DDL statement is being executed in the cluster (usually for the time-consuming DDL statements such as `ADD INDEX` and the column type changes).

Before the upgrade, it is recommended to use the `ADMIN SHOW DDL` command to check whether the TiDB cluster has an ongoing DDL job. If the cluster has a DDL job, to upgrade the cluster, wait until the DDL execution is finished or use the `ADMIN CANCEL DDL` command to cancel the DDL job before you upgrade the cluster.

In addition, during the cluster upgrade, **DO NOT** execute any DDL statement. Otherwise, the issue of undefined behavior might occur.

15.6.1.3 How to upgrade TiDB using the binary?

It is not recommended to upgrade TiDB using the binary. Instead, it is recommended to upgrade TiDB using TiUP or upgrade a TiDB cluster on Kubernetes, which ensures both version consistency and compatibility.

15.6.2 After upgrade FAQs

This section lists some FAQs and their solutions after you upgrade TiDB.

15.6.2.1 The character set (charset) errors when executing DDL operations

In v2.1.0 and earlier versions (including all versions of v2.0), the character set of TiDB is UTF-8 by default. But starting from v2.1.1, the default character set has been changed into UTF8MB4.

If you explicitly specify the charset of a newly created table as UTF-8 in v2.1.0 or earlier versions, then you might fail to execute DDL operations after upgrading TiDB to v2.1.1.

To avoid this issue, you need to pay attention to:

- Before v2.1.3, TiDB does not support modifying the charset of the column. Therefore, when you execute DDL operations, you need to make sure that the charset of the new column is consistent with that of the original column.
- Before v2.1.3, even if the charset of the column is different from that of the table, `show create table` does not show the charset of the column. But as shown in the following example, you can view it by obtaining the metadata of the table through the HTTP API.

15.6.2.1.1 `unsupported modify column charset utf8mb4 not match origin utf8`

- Before upgrading, the following operations are executed in v2.1.0 and earlier versions.

```
create table t(a varchar(10)) charset=utf8;
```

```
Query OK, 0 rows affected
Time: 0.106s
```

```
show create table t;
```

```
+-------+-------------------------------------------------------+
| Table | Create Table                                          |
+-------+-------------------------------------------------------+
| t     | CREATE TABLE `t` (                                    |
|       |   `a` varchar(10) DEFAULT NULL                        |
|       | ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin |
+-------+-------------------------------------------------------+
1 row in set
Time: 0.006s
```

- After upgrading, the following error is reported in v2.1.1 and v2.1.2 but there is no such error in v2.1.3 and the later versions.

```
alter table t change column a a varchar(20);
```

```
ERROR 1105 (HY000): unsupported modify column charset utf8mb4 not match
    ↪  origin utf8
```

Solution:

You can explicitly specify the column charset as the same with the original charset.

```
alter table t change column a a varchar(22) character set utf8;
```

- According to Point #1, if you do not specify the column charset, UTF8MB4 is used by default, so you need to specify the column charset to make it consistent with the original one.
- According to Point #2, you can obtain the metadata of the table through the HTTP API, and find the column charset by searching the column name and the keyword "Charset".

```
curl "http://$IP:10080/schema/test/t" | python -m json.tool
```

A python tool is used here to format JSON, which is not required and only for the convenience to add the comments.

```
{
    "ShardRowIDBits": 0,
    "auto_inc_id": 0,
    "charset": "utf8", # The charset of the table.
    "collate": "",
    "cols": [          # The relevant information about the columns.
        {
            ...
            "id": 1,
            "name": {
                "L": "a",
                "O": "a"  # The column name.
            },
            "offset": 0,
            "origin_default": null,
            "state": 5,
            "type": {
                "Charset": "utf8", # The charset of column a.
                "Collate": "utf8_bin",
```

```
                "Decimal": 0,
                "Elems": null,
                "Flag": 0,
                "Flen": 10,
                "Tp": 15
            }
        }
    ],
    ...
}
```

15.6.2.1.2 unsupported modify charset from utf8mb4 to utf8

- Before upgrading, the following operations are executed in v2.1.1 and v2.1.2.

```
create table t(a varchar(10)) charset=utf8;
```

```
Query OK, 0 rows affected
Time: 0.109s
```

```
show create table t;
```

```
+-------+-------------------------------------------------------+
| Table | Create Table                                          |
+-------+-------------------------------------------------------+
| t     | CREATE TABLE `t` (                                    |
|       |   `a` varchar(10) DEFAULT NULL                        |
|       | ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin |
+-------+-------------------------------------------------------+
```

In the above example, `show create table` only shows the charset of the table, but the charset of the column is actually UTF8MB4, which can be confirmed by obtaining the schema through the HTTP API. However, when a new table is created, the charset of the column should stay consistent with that of the table. This bug has been fixed in v2.1.3.

- After upgrading, the following operations are executed in v2.1.3 and the later versions.

```
show create table t;
```

```
+-------+-------------------------------------------------------------------+
    ↪
| Table | Create Table                                                      |
+-------+-------------------------------------------------------------------+
    ↪
```

```
| t     | CREATE TABLE `t` (                                                  |
|       |   `a` varchar(10) CHARSET utf8mb4 COLLATE utf8mb4_bin DEFAULT
    ↪ NULL |
|       | ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin      |
+-------+-------------------------------------------------------------------+
    ↪
1 row in set
Time: 0.007s
```

```
alter table t change column a a varchar(20);
```

```
ERROR 1105 (HY000): unsupported modify charset from utf8mb4 to utf8
```

Solution:

- Starting from v2.1.3, TiDB supports modifying the charsets of the column and the table, so it is recommended to modify the table charset into UTF8MB4.

```
alter table t convert to character set utf8mb4;
```

- You can also specify the column charset as done in Issue #1, making it stay consistent with the original column charset (UTF8MB4).

```
alter table t change column a a varchar(20) character set utf8mb4;
```

15.6.2.1.3 ERROR 1366 (HY000): incorrect utf8 value f09f8c80( ) for column a

In TiDB v2.1.1 and earlier versions, if the charset is UTF-8, there is no UTF-8 Unicode encoding check on the inserted 4-byte data. But in v2.1.2 and the later versions, this check is added.

- Before upgrading, the following operations are executed in v2.1.1 and earlier versions.

```
create table t(a varchar(100) charset utf8);
```

```
Query OK, 0 rows affected
```

```
insert t values (unhex('f09f8c80'));
```

```
Query OK, 1 row affected
```

- After upgrading, the following error is reported in v2.1.2 and the later versions.

```
insert t values (unhex('f09f8c80'));
```

```
ERROR 1366 (HY000): incorrect utf8 value f09f8c80( ) for column a
```

Solution:

- In v2.1.2: this version does not support modifying the column charset, so you have to skip the UTF-8 check.

```
set @@session.tidb_skip_utf8_check=1;
```

```
Query OK, 0 rows affected
```

```
insert t values (unhex('f09f8c80'));
```

```
Query OK, 1 row affected
```

- In v2.1.3 and the later versions: it is recommended to modify the column charset into UTF8MB4. Or you can set `tidb_skip_utf8_check` to skip the UTF-8 check. But if you skip the check, you might fail to replicate data from TiDB to MySQL because MySQL executes the check.

```
alter table t change column a a varchar(100) character set utf8mb4;
```

```
Query OK, 0 rows affected
```

```
insert t values (unhex('f09f8c80'));
```

```
Query OK, 1 row affected
```

Specifically, you can use the variable `tidb_skip_utf8_check` to skip the legal UTF-8 and UTF8MB4 check on the data. But if you skip the check, you might fail to replicate the data from TiDB to MySQL because MySQL executes the check.

If you only want to skip the UTF-8 check, you can set `tidb_check_mb4_value_in_utf8` ↪ . This variable is added to the `config.toml` file in v2.1.3, and you can modify `check-mb4-value-in-utf8` in the configuration file and then restart the cluster to enable it.

Starting from v2.1.5, you can set `tidb_check_mb4_value_in_utf8` through the HTTP API and the session variable:

- HTTP API (the HTTP API can be enabled only on a single server)
  - To enable HTTP API:

```
curl -X POST -d "check_mb4_value_in_utf8=1" http://{TiDBIP
    ↪ }:10080/settings
```

- To disable HTTP API:

```
curl -X POST -d "check_mb4_value_in_utf8=0" http://{TiDBIP
    ↪ }:10080/settings
```

- Session variable
  - To enable session variable:

```
set @@session.tidb_check_mb4_value_in_utf8 = 1;
```

  - To disable session variable:

```
set @@session.tidb_check_mb4_value_in_utf8 = 0;
```

15.7 TiDB Monitoring FAQs

This document summarizes the FAQs related to TiDB monitoring.

- For details of Prometheus monitoring framework, see Overview of the Monitoring Framework.
- For details of key metrics of monitoring, see Key Metrics.

15.7.1 Is there a better way of monitoring the key metrics?

The monitoring system of TiDB consists of Prometheus and Grafana. From the dashboard in Grafana, you can monitor various running metrics of TiDB which include the monitoring metrics of system resources, of client connection and SQL operation, of internal communication and Region scheduling. With these metrics, the database administrator can better understand the system running status, running bottlenecks and so on. In the practice of monitoring these metrics, we list the key metrics of each TiDB component. Generally you only need to pay attention to these common metrics. For details, see official documentation.

15.7.2 The Prometheus monitoring data is deleted every 15 days by default. Could I set it to two months or delete the monitoring data manually?

Yes. Find the startup script on the machine where Prometheus is started, edit the startup parameter and restart Prometheus.

```
--storage.tsdb.retention="60d"
```

15.7.3 Region Health monitor

In TiDB 2.0, Region health is monitored in the PD metric monitoring page, in which the `Region Health` monitoring item shows the statistics of all the Region replica status. `miss` means shortage of replicas and `extra` means the extra replica exists. In addition, `Region Health` also shows the isolation level by `label`. `level-1` means the Region replicas are isolated physically in the first `label` level. All the Regions are in `level-0` when `location label` is not configured.

15.7.4 What is the meaning of `selectsimplefull` in Statement Count monitor?

It means full table scan but the table might be a small system table.

15.7.5 What is the difference between `QPS` and `Statement OPS` in the monitor?

The `QPS` statistics is about all the SQL statements, including `use database`, `load data`, `begin`, `commit`, `set`, `show`, `insert` and `select`.

The `Statement OPS` statistics is only about applications related SQL statements, including `select`, `update` and `insert`, therefore the `Statement OPS` statistics matches the applications better.

15.8 TiDB Cluster Management FAQs

This document summarizes the FAQs related to TiDB cluster management.

15.8.1 Daily management

This section describes common problems you might encounter during daily cluster management, their causes, and solutions.

15.8.1.1 How to log into TiDB?

You can log into TiDB like logging into MySQL. For example:

```
mysql -h 127.0.0.1 -uroot -P4000
```

15.8.1.2 How to modify the system variables in TiDB?

Similar to MySQL, TiDB includes static and solid parameters. You can directly modify static parameters using `SET GLOBAL xxx = n`, but the new value of a parameter is only effective within the life cycle in this instance.

15.8.1.3 Where and what are the data directories in TiDB (TiKV)?

TiKV data is located in the `--data-dir`, which include four directories of backup, db, raft, and snap, used to store backup, data, Raft data, and mirror data respectively.

15.8.1.4 What are the system tables in TiDB?

Similar to MySQL, TiDB includes system tables as well, used to store the information required by the server when it runs. See TiDB system table.

15.8.1.5 Where are the TiDB/PD/TiKV logs?

By default, TiDB/PD/TiKV outputs standard error in the logs. If a log file is specified by `--log-file` during the startup, the log is output to the specified file and executes rotation daily.

15.8.1.6 How to safely stop TiDB?

- If a load balancer is running (recommended): Stop the load balancer and execute the SQL statement `SHUTDOWN`. Then TiDB waits for a period as specified by `graceful-wait-before-shutdown` until all sessions are terminated. Then TiDB stops running.
- If no load balancer is running: Execute the `SHUTDOWN` statement. Then TiDB components are gracefully stopped.

15.8.1.7 Can `kill` be executed in TiDB?

- Kill DML statements:

  First use `information_schema.cluster_processlist` to find TiDB instance address and session ID, and then run the kill command.

  TiDB v6.1.0 introduces the Global Kill feature (controlled by the `enable-global-kill` configuration, which is enabled by default). If Global Kill is enabled, just execute `kill session_id`.

  If the TiDB version is earlier than v6.1.0, or the Global Kill feature is not enabled, `kill session_id` does not take effect by default. To terminate a DML statement, you should connect the client directly to the TiDB instance that is executing the DML statement and then execute the `kill tidb session_id` statement. If the client connects to another TiDB instance or there is a proxy between the client and the TiDB cluster, the `kill tidb session_id` statement might be routed to another TiDB instance, which might incorrectly terminate another session. For details, see `KILL`.

- Kill DDL statements: First use `admin show ddl jobs` to find the ID of the DDL job you need to terminate, and then run `admin cancel ddl jobs 'job_id' [, 'job_id'] ...`. For more details, see the `ADMIN` statement.

15.8.1.8 Does TiDB support session timeout?

TiDB currently supports two timeouts, `wait_timeout` and `interactive_timeout`.

15.8.1.9 What is the TiDB version management strategy?

For details about TiDB version management, see TiDB versioning.

15.8.1.10 How about the operating cost of deploying and maintaining a TiDB cluster?

TiDB provides a few features and tools, with which you can manage the clusters easily at a low cost:

- For maintenance operations, TiUP works as the package manager, which simplifies the deployment, scaling, upgrade, and other maintenance tasks.
- For monitoring, the TiDB monitoring framework uses Prometheus to store the monitoring and performance metrics, and uses Grafana to visualize these metrics. Dozens of built-in panels are available with hundreds of metrics.
- For troubleshooting, the TiDB Troubleshooting Map summarizes common issues of the TiDB server and other components. You can use this map to diagnose and resolve issues when you encounter related problems.

15.8.1.11 What's the difference between various TiDB master versions?

The TiDB community is highly active. The engineers have been keeping optimizing features and fixing bugs. Therefore, the TiDB version is updated quite fast. If you want to keep informed of the latest version, see TiDB Release Timeline.

It is recommeneded to deploy TiDB using TiUP or using TiDB Operator. TiDB has a unified management of the version number. You can view the version number using one of the following methods:

- `select tidb_version()`
- `tidb-server -V`

15.8.1.12 Is there a graphical deployment tool for TiDB?

Currently no.

15.8.1.13 How to scale out a TiDB cluster?

You can scale out your TiDB cluster without interrupting the online services.

- If your cluster is deployed using TiUP, refer to Scale a TiDB Cluster Using TiUP.
- If your cluster is deployed using TiDB Operator on Kubernetes, refer to Manually Scale TiDB on Kubernetes.

15.8.1.14 How to scale TiDB horizontally?

As your business grows, your database might face the following three bottlenecks:

- Lack of storage resources which means that the disk space is not enough.
- Lack of computing resources such as high CPU occupancy.
- Not enough write and read capacity.

You can scale TiDB as your business grows.

- If the disk space is not enough, you can increase the capacity simply by adding more TiKV nodes. When the new node is started, PD will migrate the data from other nodes to the new node automatically.
- If the computing resources are not enough, check the CPU consumption situation first before adding more TiDB nodes or TiKV nodes. When a TiDB node is added, you can configure it in the Load Balancer.
- If the capacity is not enough, you can add both TiDB nodes and TiKV nodes.

15.8.1.15 If Percolator uses distributed locks and the crash client keeps the lock, will the lock not be released?

For more details, see Percolator and TiDB Transaction Algorithm in Chinese.

15.8.1.16 Why does TiDB use gRPC instead of Thrift? Is it because Google uses it?

Not really. We need some good features of gRPC, such as flow control, encryption and streaming.

15.8.1.17 What does the 92 indicate in `like(bindo.customers.name, jason%, 92)`?

The 92 indicates the escape character, which is ASCII 92 by default.

15.8.1.18 Why does the data length shown by `information_schema.tables.data_length` differ from the store size on the TiKV monitoring panel?

Two reasons:

- The two results are calculated in different ways. `information_schema.tables.` ↪ `data_length` is an estimated value by calculating the averaged length of each row, while the store size on the TiKV monitoring panel sums up the length of the data files (the SST files of RocksDB) in a single TiKV instance.

- `information_schema.tables.data_length` is a logical value, while the store size is a physical value. The redundant data generated by multiple versions of the transaction is not included in the logical value, while the redundant data is compressed by TiKV in the physical value.

15.8.1.19 Why does the transaction not use the Async Commit or the one-phase commit feature?

In the following situations, even you have enabled the Async Commit feature and the one-phase commit feature using the system variables, TiDB will not use these features:

- If you have enabled TiDB Binlog, restricted by the implementation of TiDB Binlog, TiDB does not use the Async Commit or one-phase commit feature.
- TiDB uses the Async Commit or one-phase commit features only when no more than 256 key-value pairs are written in the transaction and the total size of keys is no more than 4 KB. This is because, for transactions with a large amount of data to write, using Async Commit cannot greatly improve the performance.

15.8.2 PD management

This section describes common problems you may encounter during PD management, their causes, and solutions.

15.8.2.1 The `TiKV cluster is not bootstrapped` message is displayed when I access PD

Most of the APIs of PD are available only when the TiKV cluster is initialized. This message is displayed if PD is accessed when PD is started while TiKV is not started when a new cluster is deployed. If this message is displayed, start the TiKV cluster. When TiKV is initialized, PD is accessible.

15.8.2.2 The `etcd cluster ID mismatch` message is displayed when starting PD

This is because the `--initial-cluster` in the PD startup parameter contains a member that doesn't belong to this cluster. To solve this problem, check the corresponding cluster of each member, remove the wrong member, and then restart PD.

15.8.2.3 The `[PD:encryption:ErrEncryptionNewMasterKey]fail to get encryption key from file /root/path/file%!(EXTRA string=open /root/path/file: permission denied)` message is displayed when enabling encryption at rest for PD

Encryption at rest does not support storing the key file in the `root` directory or its subdirectories. Even if you grant read permissions, the same error occurs. To resolve this issue, store the key file in a location outside the `root` directory.

15.8.2.4 What's the maximum tolerance for time synchronization error of PD?

PD can tolerate any synchronization error, but a larger error value means a larger gap between the timestamp allocated by the PD and the physical time, which will affect functions such as read of historical versions.

15.8.2.5 How does the client connection find PD?

The client connection can only access the cluster through TiDB. TiDB connects PD and TiKV. PD and TiKV are transparent to the client. When TiDB connects to any PD, the PD tells TiDB who is the current leader. If this PD is not the leader, TiDB reconnects to the leader PD.

15.8.2.6 What is the relationship between each status (Up, Disconnect, Offline, Down, Tombstone) of a TiKV store?

For the relationship between each status, refer to Relationship between each status of a TiKV store.

You can use PD Control to check the status information of a TiKV store.

15.8.2.7 What is the difference between the `leader-schedule-limit` and `region-schedule-limit` scheduling parameters in PD?

- The `leader-schedule-limit` scheduling parameter is used to balance the Leader number of different TiKV servers, affecting the load of query processing.
- The `region-schedule-limit` scheduling parameter is used to balance the replica number of different TiKV servers, affecting the data amount of different nodes.

15.8.2.8 Is the number of replicas in each region configurable? If yes, how to configure it?

Yes. Currently, you can only update the global number of replicas. When started for the first time, PD reads the configuration file (conf/pd.yml) and uses the max-replicas configuration in it. If you want to update the number later, use the pd-ctl configuration command `config set max-replicas $num` and view the enabled configuration using `config show all`. The updating does not affect the applications and is configured in the background.

Make sure that the total number of TiKV instances is always greater than or equal to the number of replicas you set. For example, 3 replicas need 3 TiKV instances at least. Additional storage requirements need to be estimated before increasing the number of replicas. For more information about pd-ctl, see PD Control User Guide.

15.8.2.9 How to check the health status of the whole cluster when lacking command line cluster management tools?

You can determine the general status of the cluster using the pd-ctl tool. For detailed cluster status, you need to use the monitor to determine.

15.8.2.10 How to delete the monitoring data of a cluster node that is offline?

The offline node usually indicates the TiKV node. You can determine whether the offline process is finished by the pd-ctl or the monitor. After the node is offline, perform the following steps:

1. Manually stop the relevant services on the offline node.
2. Delete the `node_exporter` data of the corresponding node from the Prometheus configuration file.

15.8.3 TiDB server management

This section describes common problems you may encounter during TiDB server management, their causes, and solutions.

15.8.3.1 How to set the `lease` parameter in TiDB?

The lease parameter (`--lease=60`) is set from the command line when starting a TiDB server. The value of the lease parameter impacts the Database Schema Changes (DDL) speed of the current session. In the testing environments, you can set the value to 1s for to speed up the testing cycle. But in the production environments, it is recommended to set the value to minutes (for example, 60) to ensure the DDL safety.

15.8.3.2 What is the processing time of a DDL operation?

The processing time is different for different scenarios. Generally, you can consider the following three scenarios:

1. The `Add Index` operation with a relatively small number of rows in the corresponding data table: about 3s
2. The `Add Index` operation with a relatively large number of rows in the corresponding data table: the processing time depends on the specific number of rows and the QPS at that time (the `Add Index` operation has a lower priority than ordinary SQL operations)
3. Other DDL operations: about 1s

If the TiDB server instance that receives the DDL request is the same TiDB server instance that the DDL owner is in, the first and third scenarios above may cost only dozens to hundreds of milliseconds.

15.8.3.3 Why it is very slow to run DDL statements sometimes?

Possible reasons:

- If you run multiple DDL statements together, the last few DDL statements might run slowly. This is because the DDL statements are executed serially in the TiDB cluster.
- After you start the cluster successfully, the first DDL operation may take a longer time to run, usually around 30s. This is because the TiDB cluster is electing the leader that processes DDL statements.
- The processing time of DDL statements in the first ten minutes after starting TiDB would be much longer than the normal case if you meet the following conditions: 1) TiDB cannot communicate with PD as usual when you are stopping TiDB (including the case of power failure); 2) TiDB fails to clean up the registration data from PD in time because TiDB is stopped by the `kill -9` command. If you run DDL statements during this period, for the state change of each DDL, you need to wait for 2 * lease (lease = 45s).
- If a communication issue occurs between a TiDB server and a PD server in the cluster, the TiDB server cannot get or update the version information from the PD server in time. In this case, you need to wait for 2 * lease for the state processing of each DDL.

15.8.3.4 Can I use S3 as the backend storage engine in TiDB?

No. Currently, TiDB only supports the distributed storage engine and the Goleveldb/RocksDB/BoltDB engine.

15.8.3.5 Can the `Information_schema` support more real information?

As part of MySQL compatibility, TiDB supports a number of `INFORMATION_SCHEMA` ↪ tables. Many of these tables also have a corresponding SHOW command. For more information, see Information Schema.

15.8.3.6 What's the explanation of the TiDB Backoff type scenario?

In the communication process between the TiDB server and the TiKV server, the `Server` ↪ `is busy` or `backoff.maxsleep 20000ms` log message is displayed when processing a large volume of data. This is because the system is busy while the TiKV server processes data. At this time, usually you can view that the TiKV host resources usage rate is high. If this occurs, you can increase the server capacity according to the resources usage.

15.8.3.7 What is the main reason of TiDB TiClient type?

The TiClient Region Error indicator describes the error types and metrics that appear when the TiDB server as a client accesses the TiKV server through the KV interface to perform data operations. The error types include `not_leader` and `stale_epoch`. These errors occur when the TiDB server manipulates the Region leader data according to its own cache information, the Region leader has migrated, or the current TiKV Region information

and the routing information of the TiDB cache are inconsistent. Generally, in this case, the TiDB server will automatically retrieve the latest routing data from PD and redo the previous operation.

15.8.3.8 What's the maximum number of concurrent connections that TiDB supports?

By default, there is no limit on the maximum number of connections per TiDB server. If too large concurrency leads to an increase of response time, it is recommended to increase the capacity by adding TiDB nodes.

15.8.3.9 How to view the creation time of a table?

The `create_time` of tables in the `information_schema` is the creation time.

15.8.3.10 What is the meaning of `EXPENSIVE_QUERY` in the TiDB log?

When TiDB is executing a SQL statement, the query will be `EXPENSIVE_QUERY` if each operator is estimated to process over 10,000 rows. You can modify the `tidb-server` configuration parameter to adjust the threshold and then restart the `tidb-server`.

15.8.3.11 How do I estimate the size of a table in TiDB?

To estimate the size of a table in TiDB, you can use the following query statement.

```
SELECT
  db_name,
  table_name,
  ROUND(SUM(total_size / cnt), 2) Approximate_Size,
  ROUND(
    SUM(
      total_size / cnt / (
        SELECT
          ROUND(AVG(value), 2)
        FROM
          METRICS_SCHEMA.store_size_amplification
        WHERE
          value > 0
      )
    ),
    2
  ) Disk_Size
FROM
  (
    SELECT
      db_name,
```

```
    table_name,
    region_id,
    SUM(Approximate_Size) total_size,
    COUNT(*) cnt
  FROM
    information_schema.TIKV_REGION_STATUS
  WHERE
    db_name = @dbname
    AND table_name IN (@table_name)
  GROUP BY
    db_name,
    table_name,
    region_id
) tabinfo
GROUP BY
db_name,
table_name;
```

When using the above statement, you need to fill in and replace the following fields in the statement as appropriate.

- `@dbname`: the name of the database.
- `@table_name`: the name of the target table.

In addition, in the above statement:

- `store_size_amplification` indicates the average of the cluster compression ratio. In addition to using `SELECT * FROM METRICS_SCHEMA.store_size_amplification;` to query this information, you can also check the **Size amplification** metric for each node on the **Grafana Monitoring PD - statistics balance** panel. The average of the cluster compression ratio is the average of the Size amplification for all nodes.
- `Approximate_Size` indicates the size of the table in a replica before compression. Note that this is an approximate value, not an accurate one.
- `Disk_Size` indicates the size of the table after compression. This is an approximate value and can be calculated according to `Approximate_Size` and `store_size_amplification`.

15.8.4 TiKV server management

This section describes common problems you might encounter during TiKV server management, their causes, and solutions.

15.8.4.1 How to specify the location of data for compliance or multi-tenant applications?

You can use Placement Rules to specify the location of data for compliance or multi-tenant applications.

Placement Rules in SQL is designed to control the attributes of any continuous data range, such as the number of replicas, the Raft role, the placement location, and the key ranges in which the rules take effect.

15.8.4.2 What is the recommended number of replicas in the TiKV cluster? Is it better to keep the minimum number for high availability?

3 replicas for each Region is sufficient for a testing environment. However, you should never operate a TiKV cluster with under 3 nodes in a production scenario. Depending on infrastructure, workload, and resiliency needs, you may wish to increase this number. It is worth noting that the higher the copy, the lower the performance, but the higher the security.

15.8.4.3 The `cluster ID mismatch` message is displayed when starting TiKV

This is because the cluster ID stored in local TiKV is different from the cluster ID specified by PD. When a new PD cluster is deployed, PD generates random cluster IDs. TiKV gets the cluster ID from PD and stores the cluster ID locally when it is initialized. The next time when TiKV is started, it checks the local cluster ID with the cluster ID in PD. If the cluster IDs don't match, the `cluster ID mismatch` message is displayed and TiKV exits.

If you previously deploy a PD cluster, but then you remove the PD data and deploy a new PD cluster, this error occurs because TiKV uses the old data to connect to the new PD cluster.

15.8.4.4 The `duplicated store address` message is displayed when starting TiKV

This is because the address in the startup parameter has been registered in the PD cluster by other TiKVs. Common conditions that cause this error: There is no data folder in the path specified by TiKV `--data-dir` (no update –data-dir after deleting or moving), restart the TiKV with the previous parameters.Please try store delete function of pd-ctl, delete the previous store, and then restart TiKV.

15.8.4.5 TiKV primary node and secondary node use the same compression algorithm, why the results are different?

Currently, some files of TiKV primary node have a higher compression rate, which depends on the underlying data distribution and RocksDB implementation. It is normal that the data size fluctuates occasionally. The underlying storage engine adjusts data as needed.

15.8.4.6 What are the features of TiKV block cache?

TiKV implements the Column Family (CF) feature of RocksDB. By default, the KV data is eventually stored in the 3 CFs (default, write and lock) within RocksDB.

- The default CF stores real data and the corresponding parameter is in `[rocksdb.defaultcf]`.
- The write CF stores the data version information (MVCC) and index-related data, and the corresponding parameter is in `[rocksdb.writecf]`.
- The lock CF stores the lock information and the system uses the default parameter.
- The Raft RocksDB instance stores Raft logs. The default CF mainly stores Raft logs and the corresponding parameter is in `[raftdb.defaultcf]`.
- All CFs have a shared block-cache to cache data blocks and improve RocksDB read speed. The size of block-cache is controlled by the `block-cache-size` parameter. A larger value of the parameter means more hot data can be cached and is more favorable to read operation. At the same time, it consumes more system memory.
- Each CF has an individual write-buffer and the size is controlled by the `write-buffer-size` parameter.

15.8.4.7 Why is the TiKV channel full?

- The Raftstore thread is too slow or blocked by I/O. You can view the CPU usage status of Raftstore.
- TiKV is too busy (such as CPU and disk I/O) and cannot manage to handle it.

15.8.4.8 Why does TiKV frequently switch Region leader?

- Network problem results in the communication stuck among nodes. You can check Report failures monitoring.
- The node of the original main Leader is stuck, resulting in failure to reach out to the Follower in time.
- Raftstore thread stuck.

15.8.4.9 If a node is down, will the service be affected? If yes, how long?

TiKV uses Raft to replicate data among multiple replicas (by default 3 replicas for each Region). If one replica goes wrong, the other replicas can guarantee data safety. Based on the Raft protocol, if a single leader fails as the node goes down, a follower in another node is soon elected as the Region leader after a maximum of 2 * lease time (lease time is 10 seconds).

15.8.4.10 What are the TiKV scenarios that take up high I/O, memory, CPU, and exceed the parameter configuration?

Writing or reading a large volume of data in TiKV takes up high I/O, memory and CPU. Executing very complex queries costs a lot of memory and CPU resources, such as the scenario that generates large intermediate result sets.

15.8.4.11 Does TiKV support SAS/SATA disks or mixed deployment of SSD/SAS disks?

No. For OLTP scenarios, TiDB requires high I/O disks for data access and operation. As a distributed database with strong consistency, TiDB has some write amplification such as replica replication and bottom layer storage compaction. Therefore, it is recommended to use NVMe SSD as the storage disks in TiDB best practices. Mixed deployment of TiKV and PD is not supported.

15.8.4.12 Is the Range of the Key data table divided before data access?

No. It differs from the table splitting rules of MySQL. In TiKV, the table Range is dynamically split based on the size of Region.

15.8.4.13 How does Region split?

Region is not divided in advance, but it follows a Region split mechanism. When the Region size exceeds the value of the `region-max-size` or `region-max-keys` parameters, split is triggered. After the split, the information is reported to PD.

15.8.4.14 Does TiKV have the `innodb_flush_log_trx_commit` parameter like MySQL, to guarantee the security of data?

Yes. Currently, the standalone storage engine uses two RocksDB instances. One instance is used to store the raft-log. When the `sync-log` parameter in TiKV is set to true, each commit is mandatorily flushed to the raft-log. If a crash occurs, you can restore the KV data using the raft-log.

15.8.4.15 What is the recommended server configuration for WAL storage, such as SSD, RAID level, cache strategy of RAID card, NUMA configuration, file system, I/O scheduling strategy of the operating system?

WAL belongs to ordered writing, and currently, we do not apply a unique configuration to it. Recommended configuration is as follows:

- SSD
- RAID 10 preferred
- Cache strategy of RAID card and I/O scheduling strategy of the operating system: currently no specific best practices; you can use the default configuration in Linux 7 or later

- NUMA: no specific suggestion; for memory allocation strategy, you can use `interleave = all`
- File system: ext4

15.8.4.16 How is the write performance in the most strict data available mode (`sync-log = true`)?

Generally, enabling `sync-log` reduces about 30% of the performance. For write performance when `sync-log` is set to `false`, see Performance test result for TiDB using Sysbench.

15.8.4.17 Can Raft + multiple replicas in the TiKV architecture achieve absolute data safety? Is it necessary to apply the most strict mode (`sync-log = true`) to a standalone storage?

Data is redundantly replicated between TiKV nodes using the Raft Consensus Algorithm to ensure recoverability should a node failure occur. Only when the data has been written into more than 50% of the replicas will the application return ACK (two out of three nodes). However, theoretically, two nodes might crash. Therefore, except for scenarios with less strict requirement on data safety but extreme requirement on performance, it is strongly recommended that you enable the `sync-log` mode.

As an alternative to using `sync-log`, you may also consider having five replicas instead of three in your Raft group. This would allow for the failure of two replicas, while still providing data safety.

For a standalone TiKV node, it is still recommended to enable the `sync-log` mode. Otherwise, the last write might be lost in case of a node failure.

15.8.4.18 Since TiKV uses the Raft protocol, multiple network roundtrips occur during data writing. What is the actual write delay?

Theoretically, TiDB has a write delay of 4 more network roundtrips than standalone databases.

15.8.4.19 Does TiDB have an InnoDB memcached plugin like MySQL which can directly use the KV interface and does not need the independent cache?

TiKV supports calling the interface separately. Theoretically, you can take an instance as the cache. Because TiDB is a distributed relational database, we do not support TiKV separately.

15.8.4.20 What is the Coprocessor component used for?

- Reduce the data transmission between TiDB and TiKV
- Make full use of the distributed computing resources of TiKV to execute computing pushdown.

15.8.4.21 The error message `IO error: No space left on device While appending to file` is displayed

This is because the disk space is not enough. You need to add nodes or enlarge the disk space.

15.8.4.22 Why does the OOM (Out of Memory) error occur frequently in TiKV?

The memory usage of TiKV mainly comes from the block-cache of RocksDB, which is 40% of the system memory size by default. When the OOM error occurs frequently in TiKV, you should check whether the value of `block-cache-size` is set too high. In addition, when multiple TiKV instances are deployed on a single machine, you need to explicitly configure the parameter to prevent multiple instances from using too much system memory that results in the OOM error.

15.8.4.23 Can both TiDB data and RawKV data be stored in the same TiKV cluster?

It depends on your TiDB version and whether TiKV API V2 is enabled (`storage.api-version = 2`).

- If your TiDB version is v6.1.0 or later and TiKV API V2 is enabled, TiDB data and RawKV data can be stored in the same TiKV cluster.
- Otherwise, the answer is no because the key format of TiDB data (or data created using the transactional API) is incompatible with data created using the RawKV API (or data from other RawKV-based services).

15.8.5 TiDB testing

This section describes common problems you might encounter during TiDB testing, their causes, and solutions.

15.8.5.1 What is the performance test result for TiDB using Sysbench?

At the beginning, many users tend to do a benchmark test or a comparison test between TiDB and MySQL. We have also done a similar official test and find the test result is consistent at large, although the test data has some bias. Because the architecture of TiDB differs greatly from MySQL, it is hard to find a benchmark point. The suggestions are as follows:

- Do not spend too much time on the benchmark test. Pay more attention to the difference of scenarios using TiDB.
- See Performance test result for TiDB using Sysbench.

15.8.5.2 What's the relationship between the TiDB cluster capacity (QPS) and the number of nodes? How does TiDB compare to MySQL?

- Within 10 nodes, the relationship between TiDB write capacity (Insert TPS) and the number of nodes is roughly 40% linear increase. Because MySQL uses single-node write, its write capacity cannot be scaled.
- In MySQL, the read capacity can be increased by adding secondary database, but the write capacity cannot be increased except using sharding, which has many problems.
- In TiDB, both the read and write capacity can be easily increased by adding more nodes.

15.8.5.3 The performance test of MySQL and TiDB by our DBA shows that the performance of a standalone TiDB is not as good as MySQL

TiDB is designed for scenarios where sharding is used because the capacity of a MySQL standalone is limited, and where strong consistency and complete distributed transactions are required. One of the advantages of TiDB is pushing down computing to the storage nodes to execute concurrent computing.

TiDB is not suitable for tables of small size (such as below ten million level), because its strength in concurrency cannot be shown with a small size of data and limited Regions. A typical example is the counter table, in which records of a few lines are updated high frequently. In TiDB, these lines become several Key-Value pairs in the storage engine, and then settle into a Region located on a single node. The overhead of background replication to guarantee strong consistency and operations from TiDB to TiKV leads to a poorer performance than a MySQL standalone.

15.8.6 Backup and restoration

This section describes common problems you may encounter during backup and restoration, their causes, and solutions.

15.8.6.1 How to back up data in TiDB?

Currently, for the backup of a large volume of data (more than 1 TB), the preferred method is using Backup & Restore (BR). Otherwise, the recommended tool is Dumpling. Although the official MySQL tool `mysqldump` is also supported in TiDB to back up and restore data, its performance is no better than BR and it needs much more time to back up and restore large volumes of data.

For more FAQs about BR, see BR FAQs.

15.8.6.2 How is the speed of backup and restore?

When BR is used to perform backup and restore tasks, the backup is processed at about 40 MB/s per TiKV instance, and restore is processed at about 100 MB/s per TiKV instance.

15.9 High Availability FAQs

This document summarizes the FAQs related to high availability of TiDB.

15.9.1 How is TiDB strongly consistent?

Data is redundantly replicated between TiKV nodes using the Raft consensus algorithm to ensure recoverability when a node failure occurs.

At the bottom layer, TiKV uses a model of replication log + State Machine to replicate data. For the write requests, the data is written to a Leader and the Leader then replicates the command to its Followers in the form of log. When the majority of nodes in the cluster receive this log, this log is committed and can be applied into the State Machine.

15.9.2 What's the recommended solution for the deployment of three geo-distributed data centers?

The architecture of TiDB guarantees that it fully supports geo-distribution and multi-activeness. Your data and applications are always-on. All the outages are transparent to your applications and your data can recover automatically. The operation depends on the network latency and stability. It is recommended to keep the latency within 5ms. Currently, TiDB already has similar use cases. For details, see Three Data Centers in Two Cities Deployment.

15.10 High Reliability FAQs

This document summarizes the FAQs related to high reliability of TiDB.

15.10.1 Does TiDB support data encryption?

Yes. To encrypt data in the network traffic, you can enable TLS between TiDB clients and servers. To encrypt data in the storage engine, you can enable transparent data encryption (TDE).

15.10.2 Does TiDB support modifying the MySQL version string of the server to a specific one that is required by the security vulnerability scanning tool?

- Since v3.0.8, TiDB supports modifying the version string of the server by modifying `server-version` in the configuration file.
- Since v4.0, if you deploy TiDB using TiUP, you can also specify the proper version string by executing `tiup cluster edit-config <cluster-name>` to edit the following section:

```
server_configs:
  tidb:
    server-version: 'YOUR_VERSION_STRING'
```

Then, use the `tiup cluster reload <cluster-name> -R tidb` command to make the preceding modification effective to avoid the failure of security vulnerability scan.

15.10.3 What authentication protocols does TiDB support? What's the process?

Like MySQL, TiDB supports the SASL protocol for user login authentication and password processing.

When the client connects to TiDB, the challenge-response authentication mode starts. The process is as follows:

1. The client connects to the server.
2. The server sends a random string challenge to the client.
3. The client sends the username and response to the server.
4. The server verifies the response.

15.10.4 How to modify the user password and privilege?

To modify the user password in TiDB, it is recommended to use `ALTER USER` (for example, `ALTER USER 'test'@'localhost' IDENTIFIED BY 'mypass';`), not `UPDATE mysql.user` which might lead to the condition that the password in other nodes is not refreshed timely.

It is recommended to use the official standard statements when modifying the user password and privilege. For details, see TiDB user account management.

16 Release Notes

16.1 TiDB Release Notes

16.1.1 6.1

- 6.1.7: 2023-07-12
- 6.1.6: 2023-04-12
- 6.1.5: 2023-02-28
- 6.1.4: 2023-02-08
- 6.1.3: 2022-12-05
- 6.1.2: 2022-10-24
- 6.1.1: 2022-09-01
- 6.1.0: 2022-06-13

16.1.2 6.0

- 6.0.0-DMR: 2022-04-07

16.1.3 5.4

- 5.4.3: 2022-10-13
- 5.4.2: 2022-07-08
- 5.4.1: 2022-05-13
- 5.4.0: 2022-02-15

16.1.4 5.3

- 5.3.4: 2022-11-24
- 5.3.3: 2022-09-14
- 5.3.2: 2022-06-29
- 5.3.1: 2022-03-03
- 5.3.0: 2021-11-30

16.1.5 5.2

- 5.2.4: 2022-04-26
- 5.2.3: 2021-12-03
- 5.2.2: 2021-10-29
- 5.2.1: 2021-09-09
- 5.2.0: 2021-08-27

16.1.6 5.1

- 5.1.5: 2022-12-28
- 5.1.4: 2022-02-22
- 5.1.3: 2021-12-03
- 5.1.2: 2021-09-27
- 5.1.1: 2021-07-30
- 5.1.0: 2021-06-24

16.1.7 5.0

- 5.0.6: 2021-12-31
- 5.0.5: 2021-12-03
- 5.0.4: 2021-09-27
- 5.0.3: 2021-07-02
- 5.0.2: 2021-06-10
- 5.0.1: 2021-04-24

- 5.0.0: 2021-04-07
- 5.0.0-rc: 2021-01-12

16.1.8 4.0

- 4.0.16: 2021-12-17
- 4.0.15: 2021-09-27
- 4.0.14: 2021-07-27
- 4.0.13: 2021-05-28
- 4.0.12: 2021-04-02
- 4.0.11: 2021-02-26
- 4.0.10: 2021-01-15
- 4.0.9: 2020-12-21
- 4.0.8: 2020-10-30
- 4.0.7: 2020-09-29
- 4.0.6: 2020-09-15
- 4.0.5: 2020-08-31
- 4.0.4: 2020-07-31
- 4.0.3: 2020-07-24
- 4.0.2: 2020-07-01
- 4.0.1: 2020-06-12
- 4.0.0: 2020-05-28
- 4.0.0-rc.2: 2020-05-15
- 4.0.0-rc.1: 2020-04-28
- 4.0.0-rc: 2020-04-08
- 4.0.0-beta.2: 2020-03-18
- 4.0.0-beta.1: 2020-02-28
- 4.0.0-beta: 2020-01-17

16.1.9 3.1

- 3.1.2: 2020-06-04
- 3.1.1: 2020-04-30
- 3.1.0: 2020-04-16
- 3.1.0-rc: 2020-04-02
- 3.1.0-beta.2: 2020-03-09
- 3.1.0-beta.1: 2020-01-10
- 3.1.0-beta: 2019-12-20

16.1.10 3.0

- 3.0.20: 2020-12-25
- 3.0.19: 2020-09-25
- 3.0.18: 2020-08-21

- 3.0.17: 2020-08-03
- 3.0.16: 2020-07-03
- 3.0.15: 2020-06-05
- 3.0.14: 2020-05-09
- 3.0.13: 2020-04-22
- 3.0.12: 2020-03-16
- 3.0.11: 2020-03-04
- 3.0.10: 2020-02-20
- 3.0.9: 2020-01-14
- 3.0.8: 2019-12-31
- 3.0.7: 2019-12-04
- 3.0.6: 2019-11-28
- 3.0.5: 2019-10-25
- 3.0.4: 2019-10-08
- 3.0.3: 2019-08-29
- 3.0.2: 2019-08-07
- 3.0.1: 2019-07-16
- 3.0.0: 2019-06-28
- 3.0.0-rc.3: 2019-06-21
- 3.0.0-rc.2: 2019-05-28
- 3.0.0-rc.1: 2019-05-10
- 3.0.0-beta.1: 2019-03-26
- 3.0.0-beta: 2019-01-19

16.1.11 2.1

- 2.1.19: 2019-12-27
- 2.1.18: 2019-11-04
- 2.1.17: 2019-09-11
- 2.1.16: 2019-08-15
- 2.1.15: 2019-07-18
- 2.1.14: 2019-07-04
- 2.1.13: 2019-06-21
- 2.1.12: 2019-06-13
- 2.1.11: 2019-06-03
- 2.1.10: 2019-05-22
- 2.1.9: 2019-05-06
- 2.1.8: 2019-04-12
- 2.1.7: 2019-03-28
- 2.1.6: 2019-03-15
- 2.1.5: 2019-02-28
- 2.1.4: 2019-02-15
- 2.1.3: 2019-01-28
- 2.1.2: 2018-12-22

- 2.1.1: 2018-12-12
- 2.1.0: 2018-11-30
- 2.1.0-rc.5: 2018-11-12
- 2.1.0-rc.4: 2018-10-23
- 2.1.0-rc.3: 2018-09-29
- 2.1.0-rc.2: 2018-09-14
- 2.1.0-rc.1: 2018-08-24
- 2.1.0-beta: 2018-06-29

16.1.12 2.0

- 2.0.11: 2019-01-03
- 2.0.10: 2018-12-18
- 2.0.9: 2018-11-19
- 2.0.8: 2018-10-16
- 2.0.7: 2018-09-07
- 2.0.6: 2018-08-06
- 2.0.5: 2018-07-06
- 2.0.4: 2018-06-15
- 2.0.3: 2018-06-01
- 2.0.2: 2018-05-21
- 2.0.1: 2018-05-16
- 2.0.0: 2018-04-27
- 2.0.0-rc.5: 2018-04-17
- 2.0.0-rc.4: 2018-03-30
- 2.0.0-rc.3: 2018-03-23
- 2.0.0-rc.1: 2018-03-09
- 1.1.0-beta: 2018-02-24
- 1.1.0-alpha: 2018-01-19

16.1.13 1.0

- 1.0.8: 2018-02-11
- 1.0.7: 2018-01-22
- 1.0.6: 2018-01-08
- 1.0.5: 2017-12-26
- 1.0.4: 2017-12-11
- 1.0.3: 2017-11-28
- 1.0.2: 2017-11-13
- 1.0.1: 2017-11-01
- 1.0.0: 2017-10-16
- Pre-GA: 2017-08-30
- rc4: 2017-08-04
- rc3: 2017-06-16

- rc2: 2017-03-01
- rc1: 2016-12-23

16.2 TiDB Release Timeline

This document shows all the released TiDB versions in reverse chronological order.

Version	Release Date
6.1.7	2023-07-12
6.1.6	2023-04-12
6.1.5	2023-02-28
6.1.4	2023-02-08
5.1.5	2022-12-28
6.1.3	2022-12-05
5.3.4	2022-11-24
6.1.2	2022-10-24
5.4.3	2022-10-13
5.3.3	2022-09-14
6.1.1	2022-09-01
6.2.0-DMR	2022-08-23
5.4.2	2022-07-08
5.3.2	2022-06-29
6.1.0	2022-06-13
5.4.1	2022-05-13
5.2.4	2022-04-26
6.0.0-DMR	2022-04-07
5.3.1	2022-03-03
5.1.4	2022-02-22
5.4.0	2022-02-15
5.0.6	2021-12-31
4.0.16	2021-12-17
5.1.3	2021-12-03
5.0.5	2021-12-03
5.2.3	2021-12-03
5.3.0	2021-11-30
5.2.2	2021-10-29
5.1.2	2021-09-27
5.0.4	2021-09-27
4.0.15	2021-09-27
5.2.1	2021-09-09
5.2.0	2021-08-27
5.1.1	2021-07-30
4.0.14	2021-07-27
5.0.3	2021-07-02

Version	Release Date
5.1.0	2021-06-24
5.0.2	2021-06-10
4.0.13	2021-05-28
5.0.1	2021-04-24
5.0.0	2021-04-07
4.0.12	2021-04-02
4.0.11	2021-02-26
4.0.10	2021-01-15
5.0.0-rc	2021-01-12
3.0.20	2020-12-25
4.0.9	2020-12-21
4.0.8	2020-10-30
4.0.7	2020-09-29
3.0.19	2020-09-25
4.0.6	2020-09-15
4.0.5	2020-08-31
3.0.18	2020-08-21
3.0.17	2020-08-03
4.0.4	2020-07-31
4.0.3	2020-07-24
3.0.16	2020-07-03
4.0.2	2020-07-01
4.0.1	2020-06-12
3.0.15	2020-06-05
3.1.2	2020-06-04
4.0.0	2020-05-28
4.0.0-rc.2	2020-05-15
3.0.14	2020-05-09
3.1.1	2020-04-30
4.0.0-rc.1	2020-04-28
3.0.13	2020-04-22
3.1.0	2020-04-16
4.0.0-rc	2020-04-08
3.1.0-rc	2020-04-02
4.0.0-beta.2	2020-03-18
3.0.12	2020-03-16
3.1.0-beta.2	2020-03-09
3.0.11	2020-03-04
4.0.0-beta.1	2020-02-28
3.0.10	2020-02-20
4.0.0-beta	2020-01-17
3.0.9	2020-01-14
3.1.0-beta.1	2020-01-10

Version	Release Date
3.0.8	2019-12-31
2.1.19	2019-12-27
3.1.0-beta	2019-12-20
3.0.7	2019-12-04
3.0.6	2019-11-28
2.1.18	2019-11-04
3.0.5	2019-10-25
3.0.4	2019-10-08
2.1.17	2019-09-11
3.0.3	2019-08-29
2.1.16	2019-08-15
3.0.2	2019-08-07
2.1.15	2019-07-18
3.0.1	2019-07-16
2.1.14	2019-07-04
3.0.0	2019-06-28
3.0.0-rc.3	2019-06-21
2.1.13	2019-06-21
2.1.12	2019-06-13
2.1.11	2019-06-03
3.0.0-rc.2	2019-05-28
2.1.10	2019-05-22
3.0.0-rc.1	2019-05-10
2.1.9	2019-05-06
2.1.8	2019-04-12
2.1.7	2019-03-28
3.0.0-beta.1	2019-03-26
2.1.6	2019-03-15
2.1.5	2019-02-28
2.1.4	2019-02-15
2.1.3	2019-01-28
3.0.0-beta	2019-01-19
2.0.11	2019-01-03
2.1.2	2018-12-22
2.0.10	2018-12-18
2.1.1	2018-12-12
2.1.0	2018-11-30
2.0.9	2018-11-19
2.1.0-rc.5	2018-11-12
2.1.0-rc.4	2018-10-23
2.0.8	2018-10-16
2.1.0-rc.3	2018-09-29
2.1.0-rc.2	2018-09-14

Version	Release Date
2.0.7	2018-09-07
2.1.0-rc.1	2018-08-24
2.0.6	2018-08-06
2.0.5	2018-07-06
2.1.0-beta	2018-06-29
2.0.4	2018-06-15
2.0.3	2018-06-01
2.0.2	2018-05-21
2.0.1	2018-05-16
2.0.0	2018-04-27
2.0.0-rc.5	2018-04-17
2.0.0-rc.4	2018-03-30
2.0.0-rc.3	2018-03-23
2.0.0-rc.1	2018-03-09
1.1.0-beta	2018-02-24
1.0.8	2018-02-11
1.0.7	2018-01-22
1.1.0-alpha	2018-01-19
1.0.6	2018-01-08
1.0.5	2017-12-26
1.0.4	2017-12-11
1.0.3	2017-11-28
1.0.2	2017-11-13
1.0.1	2017-11-01
1.0.0	2017-10-16
Pre-GA	2017-08-30
rc4	2017-08-04
rc3	2017-06-16
rc2	2017-03-01
rc1	2016-12-23

16.3 TiDB Versioning

It is recommended to always upgrade to the latest patch release of your release series.

TiDB offers two release series:

- Long-Term Support Releases
- Development Milestone Releases (introduced in TiDB v6.0.0)

To learn about the support policy for major releases of TiDB, see TiDB Release Support Policy.

16.3.1 Release versioning

TiDB versioning has the form of `X.Y.Z`. `X.Y` represents a release series.

- Since TiDB 1.0, `X` increments every year. Each `X` release introduces new features and improvements.
- `Y` increments from 0. Each `Y` release introduces new features and improvements.
- In the first release of a release series, `Z` is set to 0 by default. For patch releases, `Z` increments from 1.

For the versioning system of TiDB v5.0.0 and earlier versions, refer to Historical versioning.

16.3.2 Long-Term Support releases

Long-Term Support (LTS) versions are released approximately every six months and introduce new features, improvements, bug fixes and security vulnerability fixes.

LTS releases are versioned as `X.Y.Z`. `Z` defaults to 0.

Example versions:

- 6.1.0
- 5.4.0

During the lifecycle of LTS, patch releases are made available on demand. Patch releases contain bug fixes and security vulnerability fixes, and do not introduce new features.

Patch releases are versioned as `X.Y.Z`. `X.Y` is consistent with the corresponding LTS versioning. The patch number `Z` increments from 1.

Example version:

- 6.1.1

v5.1.0, v5.2.0, v5.3.0, v5.4.0 were released only two months after their preceding releases, but all four releases are LTS and provide patch releases.

16.3.3 Development Milestone Releases

Development Milestone Releases (DMR) are released approximately every two months that do not contain LTS. DMR versions introduce new features, improvements and bug fixes. TiDB does not provide patch releases based on DMR, and any related bugs are fixed in the subsequent release series.

DMRs are versioned as `X.Y.Z`. `Z` defaults to 0. A `-DMR` suffix is appended to the version number.

Example version:

- 6.0.0-DMR

16.3.4 Versioning of TiDB ecosystem tools

Some TiDB tools are released together with the TiDB server and use the same version numbering system, such as TiDB Lightning. Some TiDB tools are released separately from the TiDB server and use their own version numbering system, such as TiUP and TiDB Operator.

16.3.5 Historical versioning (deprecated)

16.3.5.1 General Availability releases

General Availability (GA) releases are stable versions of the current release series of TiDB. GA versions are released after Release Candidate (RC) versions. GA can be used in production environments.

Example versions:

- 1.0
- 2.1 GA
- 5.0 GA

16.3.5.2 Release Candidate releases

Release Candidate (RC) releases introduce new features and improvements. RC versions are significantly more stable than Beta versions. RC can be used for early testing, but are not suitable for production.

Example versions:

- RC1
- 2.0-RC1
- 3.0.0-rc.1

16.3.5.3 Beta releases

Beta releases introduces new features and improvements. Beta versions are greatly improved over Alpha versions and have eliminated critical bugs, but still contain some bugs. Beta releases are available for users to test the latest features.

Example versions:

- 1.1 Beta
- 2.1 Beta
- 4.0.0-beta.1

16.3.5.4 Alpha releases

Alpha releases are internal releases for testing and introduce new features and improvements. Alpha releases are the initial versions of the current release series. Alpha releases might have some bugs and are available for users to test the latest features.

Example version:

- 1.1 Alpha

16.4 TiDB Installation Packages

Before deploying TiUP offline, you need to download the binary packages of TiDB as described in Prepare the TiUP offline component package.

TiDB provides two binary packages: `TiDB-community-server` and `TiDB-community-toolkit`

The `TiDB-community-server` package contains the following contents.

Content	Change history
tidb-{version}-linux-amd64.tar.gz	
tikv-{version}-linux-amd64.tar.gz	
tiflash-{version}-linux-amd64.tar.gz	
pd-{version}-linux-amd64.tar.gz	
ctl-{version}-linux-amd64.tar.gz	
grafana-{version}-linux-amd64.tar.gz	
alertmanager-{version}-linux-amd64.tar.gz	
blackbox_exporter-{version}-linux-amd64.tar.gz	
prometheus-{version}-linux-amd64.tar.gz	
node_exporter-{version}-linux-amd64.tar.gz	
tiup-linux-amd64.tar.gz	
tiup-{version}-linux-amd64.tar.gz	
local_install.sh	
cluster-{version}-linux-amd64.tar.gz	
insight-{version}-linux-amd64.tar.gz	
diag-{version}-linux-amd64.tar.gz	New in v6.0.0
influxdb-{version}-linux-amd64.tar.gz	
playground-{version}-linux-amd64.tar.gz	

The `TiDB-community-toolkit` package contains the following contents.

Content	Change history
tikv-importer-{version}-linux-amd64.tar.gz	
pd-recover-{version}-linux-amd64.tar.gz	

Content	Change history
etcdctl	New in v6.0.0
tiup-linux-amd64.tar.gz	
tiup-{version}-linux-amd64.tar.gz	
tidb-lightning-{version}-linux-amd64.tar.gz	
tidb-lightning-ctl	
dumpling-{version}-linux-amd64.tar.gz	
cdc-{version}-linux-amd64.tar.gz	
dm-{version}-linux-amd64.tar.gz	
dm-worker-{version}-linux-amd64.tar.gz	
dm-master-{version}-linux-amd64.tar.gz	
dmctl-{version}-linux-amd64.tar.gz	
br-{version}-linux-amd64.tar.gz	
spark-{version}-any-any.tar.gz	
tispark-{version}-any-any.tar.gz	
package-{version}-linux-amd64.tar.gz	
bench-{version}-linux-amd64.tar.gz	
errdoc-{version}-linux-amd64.tar.gz	
dba-{version}-linux-amd64.tar.gz	
PCC-{version}-linux-amd64.tar.gz	
pump-{version}-linux-amd64.tar.gz	
drainer-{version}-linux-amd64.tar.gz	
binlogctl	New in v6.0.0
sync_diff_inspector	
reparo	
arbiter	
mydumper	New in v6.0.0
server-{version}-linux-amd64.tar.gz	New in v6.1.1
grafana-{version}-linux-amd64.tar.gz	New in v6.1.1
alertmanager-{version}-linux-amd64.tar.gz	New in v6.1.1
prometheus-{version}-linux-amd64.tar.gz	New in v6.1.1
blackbox_exporter-{version}-linux-amd64.tar.gz	New in v6.1.1
node_exporter-{version}-linux-amd64.tar.gz	New in v6.1.1

16.4.1 See also

Deploy TiUP offline

16.5 v6.1

16.5.1 TiDB 6.1.7 Release Notes

Release date: July 12, 2023

TiDB version: 6.1.7

Quick access: Quick start | Production deployment

16.5.1.1 Improvements

- TiDB
  - Use pessimistic transactions in internal transaction retry to avoid retry failure and reduce time consumption #38136 @jackysp
- Tools
  - TiCDC
    - Support batch `UPDATE` DML statements to improve TiCDC replication performance #8084 @amyangfei
  - TiDB Lightning
    - Verify checksum through SQL after the import to improve stability of verification #41941 @GMHDBJD

16.5.1.2 Bug fixes

- TiDB
  - Fix the panic issue caused by empty `processInfo` #43829 @zimulala
  - Fix the issue that `resolve lock` might hang when there is a sudden change in PD time #44822 @zyguan
  - Fix the issue that queries containing Common Table Expressions (CTEs) might cause insufficient disk space #44477 @guo-shaoge
  - Fix the issue that using CTEs and correlated subqueries simultaneously might result in incorrect query results or panic #44649 #38170 #44774 @winoros @guo-shaoge
  - Fix the issue that the query result of the `SELECT CAST(n AS CHAR)` statement is incorrect when `n` in the statement is a negative number #44786 @xhebox
  - Fix the query panic issue of TiDB in certain cases #40857 @Dousir9
  - Fix the issue that SQL compile error logs are not redacted #41831 @lance6716
  - Fix the issue that the `SELECT` statement returns an error for a partitioned table if the table partition definition uses the `FLOOR()` function to round a partitioned column #42323 @jiyfhust
  - Fix the issue that querying partitioned tables might cause errors during Region split #43144 @lcwangchao
  - Fix the issue of unnecessary memory usage during reading statistical information #42052 @xuyifangreeneyes

- Fix the issue of excessive memory usage after creating a large number of empty partitioned tables #44308 @hawkingrei
- Fix the issue that queries might return incorrect results when `tidb_opt_agg_push_down` is enabled #44795 @AilinKid
- Fix the issue that the join result of common table expressions might be wrong #38170 @wjhuang2016
- Fix the issue that in some rare cases, residual pessimistic locks of pessimistic transactions might affect data correctness when GC resolves locks #43243 @MyonKeminta
- Fix the issue that after a new column is added in the cache table, the value is `NULL` instead of the default value of the column #42928 @lqs
- Fix the issue that TiDB returns an error when the corresponding rows in partitioned tables cannot be found in the probe phase of index join #43686 @AilinKid @mjonss
- Fix the issue that dropping a database causes slow GC progress #33069 @tiancaiamao
- Fix the issue that data and indexes are inconsistent when the `ON UPDATE` statement does not correctly update the primary key #44565 @zyguan
- Fix the issue that TiCDC might lose some row changes during table renaming #43338 @tangenta
- Fix the behavior issue of Placement Rules in partitioned tables, so that the Placement Rules in deleted partitions can be correctly set and recycled #44116 @lcwangchao
- Fix the issue that when `tidb_scatter_region` is enabled, Region does not automatically split after a partition is truncated #43174 #43028
- Fix the issue of DDL retry caused by write conflict when executing `TRUNCATE TABLE` for partitioned tables with many partitions and TiFlash replicas #42940 @mjonss
- Fix the issue of incorrect execution plans when pushing down window functions to TiFlash #43922 @gengliqi
- Fix the issue that incorrect results might be returned when using a common table expression (CTE) in statements with non-correlated subqueries #44051 @winoros
- Fix the issue that using `memTracker` with cursor fetch causes memory leaks #44254 @YangKeao
- Fix the issue that the data length in the `QUERY` column of the `INFORMATION_SCHEMA.DDL_JOBS` table might exceed the column definition #42440 @tiancaiamao
- Fix the issue that the `min, max` query result is incorrect #43805 @wshwsh12
- Fix the issue that TiDB reports syntax errors when analyzing tables #43392 @guo-shaoge
- Fix the issue that the `SHOW PROCESSLIST` statement cannot display the TxnStart of the transaction of the statement with a long subquery time #40851 @crazycs520
- Fix the issue of missing table names in the `ADMIN SHOW DDL JOBS` result when a `DROP TABLE` operation is being executed #42268 @tiancaiamao
- Fix the issue of displaying the incorrect TiDB address in IPv6 environment #43260 @nexustar

- Fix the issue that the SQL statement reports the `runtime error: index out of range` error when using the `AES_DECRYPT` expression #43063 @lcwangchao
- Fix the issue that there is no warning when using `SUBPARTITION` to create partitioned tables #41198 #41200 @mjonss
- Fix the issue that the query with CTE causes TiDB to hang #43749 #36896 @guo-shaoge
- Fix the issue that truncating a partition of a partitioned table might cause the Placement Rule of the partition to become invalid #44031 @lcwangchao
- Fix the issue that CTE results are incorrect when pushing down predicates #43645 @winoros
- Fix the issue that `auto-commit` change affects transaction commit behaviours #36581 @cfzjywxk

- TiKV
  - Fix the issue that TiDB Lightning might cause SST file leakage #14745 @YuJuncen
  - Fix the issue that encryption key ID conflict might cause the deletion of the old keys #14585 @tabokie
  - Fix the issue of file handle leakage in Continuous Profiling #14224 @tabokie

- PD
  - Fix the issue that gRPC returns errors with unexpected formats #5161 @HuSharp

- Tools
  - Backup & Restore (BR)
    - Fix the issue that `resolved lock timeout` is falsely reported in some cases #43236 @YuJuncen
    - Fix the issue of backup slowdown when a TiKV node crashes in a cluster #42973 @YuJuncen
  - TiCDC
    - Fix the issue that TiCDC cannot create a changefeed with a downstream Kafka-on-Pulsar #8892 @hi-rustin
    - Fix the issue that TiCDC cannot automatically recover when PD address or leader fails #8812 #8877 @asddongmen
    - Fix the issue that when the downstream is Kafka, TiCDC queries the downstream metadata too frequently and causes excessive workload in the downstream #8957 #8959 @hi-rustin
    - Fix the issue that TiCDC gets stuck when PD fails such as network isolation or PD Owner node reboot #8808 #8812 #8877 @asddongmen
  - TiDB Lightning

- Fix the issue that in Logical Import Mode, deleting tables downstream during import might cause TiDB Lightning metadata not to be updated in time #44614 @dsdashun
- Fix the issue that disk quota might be inaccurate due to competing conditions #44867 @D3Hunter
- Fix the issue of `write to tikv with no leader returned` when importing a large amount of data #43055 @lance6716
- Fix a possible OOM problem when there is an unclosed delimiter in the data file #40400 @buchuitoudegou
- Fix the issue that OOM might occur when importing a wide table #43728 @D3Hunter

- TiDB Binlog
  - Fix the issue that the etcd client does not automatically synchronize the latest node information during initialization #1236 @lichunzhu
  - Fix the panic issue of Drainer due to an old TiKV client version by upgrading the TiKV client #1170 @lichunzhu
  - Fix the issue that unfiltered failed DDL statements cause task errors #1228 @lichunzhu

16.5.2 TiDB 6.1.6 Release Notes

Release date: April 12, 2023

TiDB version: 6.1.6

Quick access: Quick start | Production deployment

16.5.2.1 Compatibility changes

- TiCDC fixes the issue of incorrect encoding of FLOAT data in Avro #8490 @3AceShowHand

  When upgrading the TiCDC cluster to v6.1.6 or a later v6.1.x version, if a table replicated using Avro contains the `FLOAT` data type, you need to manually adjust the compatibility policy of Confluent Schema Registry to `None` before upgrading so that the changefeed can successfully update the schema. Otherwise, after upgrading, the changefeed will be unable to update the schema and enter an error state.

16.5.2.2 Improvements

- TiDB
  - Supports caching the execution plan for `BatchPointGet` in Prepared Plan Cache #42125 @qw4990

- Support more SQL formats for Index Join #40505 @Yisaer

- TiKV

  - Support starting TiKV on a CPU with less than 1 core #13586 #13752 #14017 @andreid-db @andreid-db

16.5.2.3 Bug fixes

- TiDB

  - Fix the issue that the `ignore_plan_cache` hint might not work for `INSERT` statements #40079 #39717 @qw4990
  - Fix the issue that TiDB might panic after `indexMerge` encounters an error #41047 #40877 @guo-shaoge @windtalker
  - Fix the issue that incorrect results might be returned when TopN operators with virtual columns are mistakenly pushing down to TiKV or TiFlash #41355 @Dousir9
  - Fix the PD OOM issue when there is a large number of Regions but the table ID cannot be pushed down when querying some virtual tables using `Prepare` or `Execute` #39605 @djshow832
  - Fix the issue that Plan Cache might cache FullScan plans when processing `int_col in (decimal...)` conditions #40224 @qw4990
  - Fix the issue that IndexMerge plans might generate incorrect ranges on the SET type columns #41273 #41293 @time-and-fate
  - Fix the issue of potential wrong results when comparing unsigned `TINYINT` ↪ `/SMALLINT/INT` values with `DECIMAL/FLOAT/DOUBLE` values smaller than `0` #41736 @LittleFall
  - Fix the issue that the TiDB server might run out of memory when the `INFORMATION_SCHEMA.CLUSTER_SLOW_QUERY` table is queried. This issue can be triggered when you check slow queries on the Grafana dashboard #33893 @crazycs520
  - Fix the issue that range partitions allow multiple `MAXVALUE` partitions #36329 @u5surf
  - Fix the issue that Plan Cache might cache Shuffle operators and return incorrect results #38335 @qw4990
  - Fix the issue that data race in time zone might cause data-index inconsistency #40710 @wjhuang2016
  - Fix the issue that goroutine leak might occur in `indexMerge` #41545 #41605 @guo-shaoge @guo-shaoge
  - Fix the issue that, when using Cursor Fetch and running other statements among Execute, Fetch, and Close, the Fetch and Close commands might return incorrect results or cause TiDB to panic #40094 [@YangKeao](https://github.com/YangKeao)

- Fix the issue that when modifying the floating-point type using DDL to keep the length unchanged and reduce the decimal places, the old data still remains the same #41281 [@zimulala](https://github.com/zimulala)
- Fix the issue that joining the `information_schema.columns` table causes TiDB to panic #32459 [@tangenta](https://github.com/tangenta)
- Fix the issue that TiDB panic occurs due to inconsistent InfoSchema being obtained when generating the execution plan #41622 [@tiancaiamao](https://github.com/tiancaiamao)
- Fix the issue that TiFlash reports an error for generated columns during execution #40663 @guo-shaoge
- Fix the issue that TiDB might produce incorrect results when different partitioned tables appear in a single SQL statement #42135 @mjonss
- Fix the issue that Plan Cache might cache Shuffle operators and return incorrect results #38335 @qw4990 @fzzf678
- Fix the issue that using Index Merge to read a table containing the `SET` type column might lead to incorrect results #41293 @time-and-fate
- Fix the issue that full index scans might cause errors when prepared plan cache is enabled #42150 @fzzf678
- Fix the issue that SQL statements using `PointGet` to read a table during the execution of a DDL statement might throw a panic #41622 @tiancaiamao
- Fix the issue that after executing `PointUpdate` within a transaction, TiDB returns incorrect results for the `SELECT` statement #28011 @zyguan
- Clear expired region cache regularly to avoid memory leak and performance degradation #40461 @sticnarf @zyguan
- Fix the issue that `INSERT IGNORE` and `REPLACE` statements do not lock keys that do not modify values #42121 @zyguan

- TiKV

    - Fix an error that occurs when casting the `const Enum` type to other types #14156 @wshwsh12
    - Fix the issue of CPU quota limitation 13084 @BornChanger
    - Fix the issue of incorrect snapshot last index 12618 @LintianShi

- PD

    - Fix the issue that the Region Scatter might cause uneven distribution of leader #6017 @HunDunDM
    - Fix the issue that the timeout mechanism of Online Unsafe Recovery does not work #6107 @v01dstar

- TiFlash

    - Fix the issue that semi-joins use excessive memory when calculating Cartesian products #6730 @gengliqi
    - Fix the issue that TiFlash log search is too slow #6829 @hehechen

- Fix the issue that TopN/Sort operators produce incorrect results after enabling the new collation #6807 @xzhangxian1008
- Fix the issue that Decimal cast rounds up incorrectly in certain cases #6994 @windtalker
- Fix the issue that TiFlash cannot recognize generated columns #6801 @guo-shaoge
- Fix the issue that Decimal division does not round up the last digit in certain cases #7022 @LittleFall

- Tools
  - TiCDC
    - Fix the issue that the disorder of `UPDATE` and `INSERT` statements during data replication might cause the `Duplicate entry` error #8597 @sdojjy
    - Fix the abnormal exit issue of the TiCDC service caused by network isolation between PD and TiCDC #8562 @overvenus
    - Fix the data inconsistency that occurs when replicating data to a TiDB or MySQL sink and when `CHARACTER SET` is specified on the column that has the non-null unique index without a primary key #8420 @zhaoxinyu
    - Fix the issue that the memory usage of `db sorter` is not controlled by `cgroup memory limit` #8588 @amyangfei
    - Optimize the error message of `cdc cli` for invalid input #7903 @CharlesCheung96
    - Fix the issue of insufficient duration that redo log can tolerate for S3 storage failure #8089 @CharlesCheung96
    - Fix the issue that pausing a changefeed when PD is abnormal results in incorrect status #8330 @sdojjy
  - TiDB Lightning
    - Fix the issue that the conflict resolution logic (`duplicate-resolution`) might lead to inconsistent checksums #40657 @sleepymole
    - Fix the issue that TiDB Lightning panics in the split-region phase #40934 @lance6716
    - Fix the issue that when importing data in Local Backend mode, the target columns do not automatically generate data if the compound primary key of the imported target table has an `auto_random` column and no value for the column is specified in the source data #41454 @D3Hunter
    - Fix the issue that TiDB Lightning might incorrectly skip conflict resolution when all but the last TiDB Lightning instance encounters a local duplicate record during a parallel import #40923 @lichunzhu

16.5.3 TiDB 6.1.5 Release Notes

Release date: February 28, 2023

TiDB version: 6.1.5

Quick access: Quick start | Production deployment

16.5.3.1 Compatibility changes

- Starting from February 20, 2023, the telemetry feature is disabled by default in new versions of TiDB and TiDB Dashboard, including v6.1.5, and usage information is not collected and shared with PingCAP. Before upgrading to these versions, if the cluster uses the default telemetry configuration, the telemetry feature is disabled after the upgrade. See TiDB Release Timeline for a specific version.
    - The default value of the `tidb_enable_telemetry` system variable is changed from `ON` to `OFF`.
    - The default value of the TiDB `enable-telemetry` configuration item is changed from `true` to `false`.
    - The default value of the PD `enable-telemetry` configuration item is changed from `true` to `false`.
- Starting from v1.11.3, the telemetry feature is disabled by default in newly deployed TiUP, and usage information is not collected. If you upgrade from a TiUP version earlier than v1.11.3 to v1.11.3 or a later version, the telemetry feature keeps the same status as before the upgrade.

16.5.3.2 Improvements

- TiDB
    - Support the `AUTO_RANDOM` column as the first column of the clustered composite index #38572 @tangenta

16.5.3.3 Bug fixes

- TiDB
    - Fix the issue that data race might cause TiDB to restart #27725 @XuHuaiyu
    - Fix the issue that the `UPDATE` statement might not read the latest data when the Read Committed isolation level is used #41581 @cfzjywxk
- PD
    - Fix the PD OOM issue that occurs when the calls of `ReportMinResolvedTS` are too frequent #5965 @HundunDM
- Tools
    - TiCDC

- Fix the issue that applying redo log might cause OOM when the replication lag is excessively high #8085 @CharlesCheung96
- Fix the issue that the performance degrades when redo log is enabled to write meta #8074 @CharlesCheung96

- TiDB Data Migration (DM)
  - Fix the issue that the `binlog-schema delete` command fails to execute #7373 @liumengya94
  - Fix the issue that the checkpoint does not advance when the last binlog is a skipped DDL #8175 @D3Hunter

16.5.4 TiDB 6.1.4 Release Notes

Release date: February 8, 2023

TiDB version: 6.1.4

Quick access: Quick start | Production deployment

16.5.4.1 Compatibility changes

- TiDB
  - No longer support modifying column types on partitioned tables because of potential correctness issues #40620 @mjonss

16.5.4.2 Improvements

- TiFlash
  - Reduce the IOPS by up to 95% and the write amplification by up to 65% for TiFlash instances under high update throughput workloads #6460 @flowbehappy

- Tools
  - TiCDC
    - Add the DML batch operation mode to improve the throughput when SQL statements are generated in batches #7653 @asddongmen
    - Support storing redo logs to GCS- or Azure-compatible object storage #7987 @CharlesCheung96
  - TiDB Lightning
    - Change the severity of the precheck items `clusterResourceCheckItem` and `emptyRegionCheckItem` from `Critical` to `Warning` #37654 @niubell

16.5.4.3 Bug fixes

- TiDB
    - Fix the issue that when you create a table, the default value and the type of a column are not consistent and are not automatically corrected #34881 @Lloyd-Pottiger @mjonss
    - Fix the data race issue in the `LazyTxn.LockKeys` function #40355 @HuSharp
    - Fix the issue that the `INSERT` or `REPLACE` statements might panic in long session connections #40351 @fanrenhoo
    - Fix the issue that reading data using the "cursor read" method might return an error because of GC #39447 @zyguan
    - Fix the issue that the `pessimistic-auto-commit` configuration item does not take effect for point-get queries #39928 @zyguan
    - Fix the issue that querying the `INFORMATION_SCHEMA.TIKV_REGION_STATUS` table returns an incorrect result #37436 @zimulala
    - Fix the issue that the `IN` and `NOT IN` subqueries in some patterns report the `Can't find column` error #37032 @AilinKid @lance6716
- PD
    - Fix the issue that PD might unexpectedly add multiple Learners to a Region #5786 @HunDunDM
- TiKV
    - Fix the issue that TiDB fails to start on Gitpod when there are multiple `cgroup` and `mountinfo` records #13660 @tabokie
    - Fix the issue that tikv-ctl is terminated unexpectedly when executing the `reset -to-version` command #13829 @tabokie
    - Fix the issue that TiKV mistakenly reports a `PessimisticLockNotFound` error #13425 @sticnarf
    - Fix the issue that TiKV might panic when the size of one single write exceeds 2 GiB #13848 @YuJuncen
    - Fix the data inconsistency issue caused by network failure between TiDB and TiKV during the execution of a DML after a failed pessimistic DML #14038 @MyonKeminta
    - Fix the issue that `_` in the `LIKE` operator cannot match non-ASCII characters when new collation is not enabled #13769 @YangKeao @tonyxuqqi
- TiFlash
    - Fix the issue that TiFlash global locks are blocked for a long time occasionally #6418 @SeaRise
    - Fix the issue that high throughput writes cause OOM #6407 @JaySon-Huang

- Tools
    - Backup & Restore (BR)
        - Fix the issue that restore is interrupted due to failure in getting the Region size #36053 @YuJuncen
        - Fix the issue that causes panic when BR debugs the `backupmeta` file #40878 @MoCuishle28
    - TiCDC
        - Fix the issue that the checkpoint cannot advance when TiCDC replicates an excessively large number of tables #8004 @asddongmen
        - Fix the issue that `transaction-atomicity` and `protocol` cannot be updated via the configuration file #7935 @CharlesCheung96
        - Fix the issue that TiCDC mistakenly reports an error when the version of TiFlash is later than that of TiCDC #7744 @overvenus
        - Fix the issue that OOM occurs when TiCDC replicates large transactions #7913 @overvenus
        - Fix a bug that the context deadline is exceeded when TiCDC replicates data without splitting large transactions #7982 @hi-rustin
        - Fix the issue that `sasl-password` in the `changefeed query` result is not masked #7182 @dveeden
        - Fix the issue that data is lost when a user quickly deletes a replication task and then creates another one with the same task name #7657 @overvenus
    - TiDB Data Migration (DM)
        - Fix a bug that DM might raise an error during precheck when the downstream database name in `SHOW GRANTS` contains a wildcard ("*") #7645 @lance6716
        - Fix the issue that DM prints too many logs caused by "COMMIT" in binlog query events #7525 @liumengya94
        - Fix the issue that the DM task fails to start when only `ssl-ca` is configured for SSL #7941 @liumengya94
        - Fix a bug that when the expression filters of both "update" and "non-update" types are specified in one table, all `UPDATE` statements are skipped #7831 @lance6716
        - Fix a bug that when only one of `update-old-value-expr` or `update-new-value-expr` is set for a table, the filter rule does not take effect or DM panics #7774 @lance6716
    - TiDB Lightning
        - Fix the memory leakage issue when TiDB Lightning imports a huge source data file #39331 @dsdashun
        - Fix the issue that TiDB Lightning prechecks cannot find dirty data left by previously failed imports #39477 @dsdashun

16.5.5 TiDB 6.1.3 Release Notes

Release date: December 5, 2022

TiDB version: 6.1.3

Quick access: Quick start | Production deployment

16.5.5.1 Compatibility changes

- Tools
  - TiCDC
    - Change the default value of `transaction-atomicity` from `table` to `none`, which helps reduce replication latency and reduce OOM risks, and ensures that only a few transactions (the size of a single transaction exceeds 1024 rows) are split, instead of all transactions #7505 #5231 @asddongmen

16.5.5.2 Improvements

- PD
  - Optimize the granularity of locks to reduce lock contention and improve the capability of processing heartbeat in high concurrency #5586 @rleungx
- Tools
  - TiCDC
    - Enable transaction split and disable the safe mode of a changefeed in TiCDC by default to improve performance #7505 @asddongmen
    - Improve the performance of Kafka protocol encoder #7540, #7532, #7543 @sdojjy @3AceShowHand
- Others
  - Upgrade the Go compiler version of TiDB from go1.18 to go1.19, which improves the TiDB stability. Specifically, a Go environment variable `GOMEMLIMIT` is introduced to keep the memory usage of TiDB below a certain threshold. This helps mitigate most OOM issues. For more information, see Mitigate OOM issues by configuring the `GOMEMLIMIT`.

16.5.5.3 Bug fixes

- TiDB
    - Fix the issue that the `grantor` field is missing in the `mysql.tables_priv` table #38293 @CbcWestwolf
    - Fix the issue of the wrong query result that occurs when the mistakenly pushed-down conditions are discarded by Join Reorder #38736 @winoros
    - Fix the issue that the lock acquired by `get_lock()` cannot hold for more than 10 minutes #38706 @tangenta
    - Fix the issue that the auto-increment column cannot be used with check constraint #38894 @YangKeao
    - Fix the issue that the gPRC log is output to a wrong file #38941 @xhebox
    - Fix the issue that the TiFlash sync status of a table is not deleted from etcd when the table is truncated or dropped #37168 @CalvinNeo
    - Fix the issue that data files can be accessed unrestrainedly via data source name injection (CVE-2022-3023) #38541 @lance6716
    - Fix the issue that the function `str_to_date` returns wrong result in the `NO_ZERO_DATE` SQL mode #39146 @mengxin9014
    - Fix the issue that statistics collection tasks in the background might panic #35421 @lilinghai
    - Fix the issue that in some scenarios the pessimistic lock is incorrectly added to the non-unique secondary index #36235 @ekexium
- PD
    - Fix inaccurate Stream timeout and accelerate leader switchover #5207 @CabinfeverB
- TiKV
    - Fix abnormal Region competition caused by expired lease during snapshot acquisition #13553 @SpadeA-Tang
- TiFlash
    - Fix the issue that logical operators return wrong results when the argument type is `UInt8` #6127 @xzhangxian1008
    - Fix the issue that wrong data input for `CAST(value AS DATETIME)` causing high TiFlash sys CPU #5097 @xzhangxian1008
    - Fix the issue that heavy write pressure might generate too many column files in the delta layer #6361 @lidezhu
    - Fix the issue that column files in the delta layer cannot be compacted after restarting TiFlash #6159 @lidezhu
- Tools

- Backup & Restore (BR)
    - Fix the issue that restore tasks fail when using old framework for collations in databases or tables #39150 @MoCuishle28
- TiCDC
    - Fix data loss occurred in the scenario of executing DDL statements first and then pausing and resuming the changefeed #7682 @asddongmen
    - Fix the issue that the sink component gets stuck if the downstream network is unavailable #7706 @hicqu
- TiDB Data Migration (DM)
    - Fix the issue that when `collation_compatible` is set to `"strict"`, DM might generate SQL with duplicated collations #6832 @lance6716
    - Fix the issue that DM tasks might stop with an `Unknown placement policy` error #7493 @lance6716
    - Fix the issue that relay logs might be pulled from upstream again in some cases #7525 @liumengya94
    - Fix the issue that data is replicated for multiple times when a new DM worker is scheduled before the existing worker exits #7658 @GMHDBJD

16.5.6 TiDB 6.1.2 Release Notes

Release date: October 24, 2022

TiDB version: 6.1.2

Quick access: Quick start | Production deployment

16.5.6.1 Improvements

- TiDB
    - Allow setting placement rules and TiFlash replicas at the same time in one table #37171 @lcwangchao
- TiKV
    - Support configuring the `unreachable_backoff` item to avoid Raftstore broadcasting too many messages after one peer becomes unreachable #13054 @5kbpers
    - Support configuring the RocksDB write stall settings to a value smaller than the flow control threshold #13467 @tabokie
- Tools
    - TiDB Lightning

- Add retryable errors during checksum to improve robustness #37690 @D3Hunter

- TiCDC
  - Enhance the performance of the region worker by handling resolved TS in a batch #7078 @sdojjy

16.5.6.2 Bug fixes

- TiDB
  - Fix the issue that database-level privileges are incorrectly cleaned up #38363 @dveeden
  - Fix the incorrect output of `SHOW CREATE PLACEMENT POLICY` #37526 @xhebox
  - Fix the issue that when one PD node goes down, the query of `information_schema.TIKV_REGION_STATUS` fails due to not retrying other PD nodes #35708 @tangenta
  - Fix the issue that the `UNION` operator might return unexpected empty result #36903 @tiancaiamao
  - Fix the wrong result that occurs when enabling dynamic mode in partitioned tables for TiFlash #37254 @wshwsh12
  - Fix the issue that the Region cache is not cleaned up in time when the Region is merged #37141 @sticnarf
  - Fix the issue that the KV client sends unnecessary ping messages #36861 @jackysp
  - Fix the issue that the `EXPLAIN ANALYZE` statement with DML executors might return result before the transaction commit finishes #37373 @cfzjywxk
  - Fix the issue that `GROUP CONCAT` with `ORDER BY` might fail when the `ORDER BY` clause contains a correlated subquery #18216 @winoros
  - Fix the issue that `Can't find column` is reported if an `UPDATE` statement contains common table expressions (CTE) #35758 @AilinKid
  - Fix the issue that the `EXECUTE` might throw an unexpected error in specific scenarios #37187 @Reminiscent

- TiKV
  - Fix the issue that the snapshot data might be incomplete caused by batch snapshot across Regions #13553 @SpadeA-Tang
  - Fix the issue of QPS drop when flow control is enabled and `level0_slowdown_trigger` is set explicitly #11424 @Connor1996
  - Fix the issue that causes permission denied error when TiKV gets an error from the web identity provider and fails back to the default provider #13122 @3pointer
  - Fix the issue that the TiKV service is unavailable for several minutes when a TiKV instance is in an isolated network environment #12966 @cosven

- PD
    - Fix the issue that the statistics of the Region tree might be inaccurate #5318 @rleungx
    - Fix the issue that the TiFlash learner replica might not be created #5401 @HunDunDM
    - Fix the issue that PD cannot correctly handle dashboard proxy requests #5321 @HunDunDM
    - Fix the issue that unhealthy Region might cause PD panic #5491 @nolouch
- TiFlash
    - Fix the issue that I/O Limiter might incorrectly throttle the I/O throughput of query requests after bulk writes, which reduces the query performance #5801 @JinheLin
    - Fix the issue that a window function might cause TiFlash to crash when the query is canceled #5814 @SeaRise
    - Fix the panic that occurs after creating the primary index with a column containing the `NULL` value #5859 @JaySon-Huang
- Tools
    - TiDB Lightning
        - Fix panic of TiDB Lightning caused by invalid metric counters #37338 @D3Hunter
    - TiDB Data Migration (DM)
        - Fix the issue that upstream table structure information is lost when DM tasks enter the sync unit and are interrupted #7159 @lance6716
        - Fix large transaction errors by splitting SQL statements when saving checkpoints #5010 @lance6716
        - Fix the issue that DM precheck requires the `SELECT` privilege on `INFORMATION_SCHEMA` #7317 @lance6716
        - Fix the issue that DM-worker triggers a deadlock error after running DM tasks with fast/full validators #7241 @buchuitoudegou
        - Fix the issue that DM reports the `Specified key was too long` error #5315 @lance6716
        - Fix the issue that latin1 data might be corrupted during replication #7028 @lance6716
    - TiCDC
        - Fix the issue that the cdc server might panic if it receives an HTTP request before the cdc server fully starts #6838 @asddongmen
        - Fix the log flooding issue during upgrade #7235 @hi-rustin
        - Fix the issue that changefeed's redo log files might be deleted by mistake #6413 @hi-rustin

- Fix the issue that TiCDC might become unavailable when too many operations in an etcd transaction are committed #7131 @hi-rustin
- Fix the issue that data inconsistency might occur when non-reentrant DDL statements in redo logs are executed twice #6927 @hicqu

- Backup & Restore (BR)
  - Fix the issue that the regions are not balanced because the concurrency is set too large during the restoration #37549 @3pointer
  - Fix the issue that might lead to backup and restoration failure if special characters exist in the authorization key of external storage #37469 @MoCuishle28

16.5.7 TiDB 6.1.1 Release Notes

Release date: September 1, 2022

TiDB version: 6.1.1

Quick access: Quick start | Production deployment

16.5.7.1 Compatibility changes

- TiDB
  - Make the `SHOW DATABASES LIKE ···` statement case-insensitive #34766 @e1ijah1
  - Change the default value of `tidb_enable_outer_join_reorder` from `1` to `0`, which disables Join Reorder's support for Outer Join by default.

- Diagnosis
  - Disable the Continuous Profiling feature by default, which avoids the possible TiFlash crash issue that occurs when this feature is enabled. For details, see #5687 @mornyx

16.5.7.2 Other changes

- Add the following contents in the `TiDB-community-toolkit` binary package. For details, see TiDB Installation Packages.
  - `server-{version}-linux-amd64.tar.gz`
  - `grafana-{version}-linux-amd64.tar.gz`
  - `alertmanager-{version}-linux-amd64.tar.gz`
  - `prometheus-{version}-linux-amd64.tar.gz`
  - `blackbox_exporter-{version}-linux-amd64.tar.gz`
  - `node_exporter-{version}-linux-amd64.tar.gz`

- Introduce multi-level support for different quality standards on the combination of operating systems and CPU architectures. See OS and platform requirements.

16.5.7.3 Improvements

- TiDB
  - Add a new optimizer `SEMI_JOIN_REWRITE` to improve the performance of `EXISTS` queries #35323 @winoros
- TiKV
  - Support compressing the metrics response using gzip to reduce the HTTP body size #12355 @winoros
  - Support reducing the amount of data returned for each request by filtering out some metrics using the `server.simplify-metrics` configuration item #12355 @glorv
  - Support dynamically modifying the number of sub-compaction operations performed concurrently in RocksDB (`rocksdb.max-sub-compactions`) #13145 @ethercflow
- PD
  - Improve the scheduling speed of Balance Region in specific stages #4990 @bufferflies
- Tools
  - TiDB Lightning
    - Add a retry mechanism on errors such as `stale command` to improve import success rate #36877 @D3Hunter
  - TiDB Data Migration (DM)
    - Users can manually set the amount of concurrency for lightning loader #5505 @buchuitoudegou
  - TiCDC
    - Add a sink uri parameter `transaction-atomicity` to support splitting the large transaction in a changefeed. This can greatly reduce the latency and memory consumption of large transactions #5231 @CharlesCheung96
    - Reduce performance overhead caused by runtime context switching in multi-Region scenarios #5610 @hicqu
    - Enhance the MySQL sink to turn off the safe mode automatically #5611 @overvenus

16.5.7.4 Bug fixes

- TiDB
    - Fix the issue that `INL_HASH_JOIN` might hang when used with `LIMIT` #35638 @guo-shaoge
    - Fix the issue that TiDB might panic when executing the `UPDATE` statement #32311 @Yisaer
    - Fix a bug that TiDB might send coprocessor requests when executing the `SHOW COLUMNS` statement #36496 @tangenta
    - Fix a bug that TiDB might return the `invalid memory address or nil pointer dereference` error when executing the `SHOW WARNINGS` statement #31569 @zyguan
    - Fix a bug that in the static partition prune mode, SQL statements with an aggregate condition might return wrong result when the table is empty #35295 @tiancaiamao
    - Fix the issue that the Join Reorder operation will mistakenly push down its Outer Join condition #37238 @winoros
    - Fix the issue that CTE-schema hash code is cloned mistakenly, which causes the `Can't find column ... in schema ...` error when CTE is referenced more than once #35404 @AilinKid
    - Fix the issue that the wrong join reorder in some right outer join scenarios causes wrong query result #36912 @winoros
    - Fix the issue of incorrectly inferred null flag of the TiFlash `firstrow` aggregate function in the EqualAll case #34584 @fixdb
    - Fix the issue that Plan Cache does not work when a binding is created with the `IGNORE_PLAN_CACHE` hint #34596 @fzzf678
    - Fix the issu that an `EXCHANGE` operator is missing between the hash-partition window and the single-partition window #35990 @LittleFall
    - Fix the issue that partitioned tables cannot fully use indexes to scan data in some cases #33966 @mjonss
    - Fix the issue of wrong query result when a wrong default value is set for partial aggregation after the aggregation is pushed down #35295 @tiancaiamao
    - Fix the issue that querying partitioned tables might get the `index-out-of-range` error in some cases #35181 @mjonss
    - Fix the issue that a partition is incorrectly pruned if a partition key is used in the query condition and the collate is different from the one in the query partition table #32749 @mjonss
    - Fix the issue that when TiDB Binlog is enabled, executing the `ALTER SEQUENCE` statement might cause a wrong metadata version and cause Drainer to exit #36276 @AilinKid
    - Fix the issue of incorrect TiDB status that might appear on startup in some extreme cases #36791 @xhebox
    - Fix the potential `UnknownPlanID` issue that occurs when querying the execution plans for partitioned tables in TiDB Dashboard #35153 @time-and-fate

- Fix the issue that the column list does not work in the LOAD DATA statement #35198 @SpadeA-Tang
- Fix the issue of the `data and columnID count not match` error that occurs when inserting duplicated values with TiDB Binlog enabled #33608 @zyguan
- Remove the limitation of `tidb_gc_life_time` #35392 @TonsnakeLin
- Fix the `LOAD DATA` statement dead loop when an empty filed terminator is used #33298 @zyguan
- Avoid sending requests to unhealthy TiKV nodes to improve availability #34906 @sticnarf

- TiKV

  - Fix a bug that Regions might be overlapped if Raftstore is busy #13160 @5kbpers
  - Fix the issue that PD does not reconnect to TiKV after the Region heartbeat is interrupted #12934 @bufferflies
  - Fix the issue that TiKV panics when performing type conversion for an empty string #12673 @wshwsh12
  - Fix the issue of inconsistent Region size configuration between TiKV and PD #12518 @5kbpers
  - Fix the issue that encryption keys are not cleaned up when Raft Engine is enabled #12890 @tabokie
  - Fix the panic issue that might occur when a peer is being split and destroyed at the same time #12825 @BusyJay
  - Fix the panic issue that might occur when the source peer catches up logs by snapshot in the Region merge process #12663 @BusyJay
  - Fix the issue of frequent PD client reconnection that occurs when the PD client meets an error #12345 @Connor1996
  - Fix potential panic when parallel recovery is enabled for Raft Engine #13123 @tabokie
  - Fix the issue that the Commit Log Duration of a new Region is too high, which causes QPS to drop #13077 @Connor1996
  - Fix rare panics when Raft Engine is enabled #12698 @tabokie
  - Avoid redundant log warnings when proc filesystem (procfs) cannot be found #13116 @tabokie
  - Fix the wrong expression of `Unified Read Pool CPU` in dashboard #13086 @glorv
  - Fix the issue that when a Region is large, the default `region-split-check-diff` might be larger than the bucket size #12598 @tonyxuqqi
  - Fix the issue that TiKV might panic when Apply Snapshot is aborted and Raft Engine is enabled #12470 @tabokie
  - Fix the issue that the PD client might cause deadlocks #13191 @bufferflies #12933 @BurtonQin

- PD

- Fix the issue that the online progress is inaccurate when label configurations of cluster nodes are invalid #5234 @rleungx
- Fix PD panics caused by the issue that gRPC handles errors inappropriately when `enable-forwarding` is enabled #5373 @bufferflies
- Fix the issue that `/regions/replicated` might return a wrong status #5095 @rleungx

- TiFlash

  - Fix the issue that TiFlash crashes after dropping a column of a table with clustered indexes in some situations #5154 @hongyunyan
  - Fix the issue that the `format` function might return a `Data truncated` error #4891 @xzhangxian1008
  - Fix the issue that some obsolete data might persist in storage and cannot be deleted #5659 @lidezhu
  - Fix unnecessary CPU usage in some edge cases #5409 @breezewish
  - Fix a bug that TiFlash cannot work in a cluster using IPv6 #5247 @solotzg
  - Fix a bug that TiFlash might crash due to an error in parallel aggregation #5356 @gengliqi
  - Fix a bug that thread resources might leak in case of `MinTSOScheduler` query errors #5556 @windtalker

- Tools

  - TiDB Lightning

    - Fix the issue that TiDB Lightning fails to connect to TiDB when TiDB uses an IPv6 host #35880 @D3Hunter
    - Fix the `read index not ready` error by adding a retry mechanism #36566 @D3Hunter
    - Fix the issue that sensitive information in logs is printed in server mode #36374 @lichunzhu
    - Fix the issue that TiDB Lightning does not support columns starting with slash, number, or non-ascii characters in Parquet files #36980 @D3Hunter
    - Fix the issue that de-duplication might cause TiDB Lightning to panic in extreme cases #34163 @ForwardStar

  - TiDB Data Migration (DM)

    - Fix the issue that the `txn-entry-size-limit` configuration item does not take effect in DM #6161 @ForwardStar
    - Fix the issue that the `check-task` command cannot handle special characters #5895 @Ehco1996
    - Fix the issue of possible data race in `query-status` #4811 @lyzx2001
    - Fix the different output format for the `operate-schema` command #5688 @ForwardStar
    - Fix goroutine leak when relay meets an error #6193 @lance6716

- Fix the issue that DM Worker might get stuck when getting DB Conn #3733 @lance6716
- Fix the issue that DM fails to start when TiDB uses an IPv6 host #6249 @D3Hunter

- TiCDC
  - Fix the wrong maximum compatible version number #6039 @hi-rustin
  - Fix a bug that may cause the cdc server to panic when it receives an HTTP request before it fully starts #5639 @asddongmen
  - Fix the ddl sink panic issue when the changefeed sync-point is enabled #4934 @asddongmen
  - Fix the issue that a changefeed is stuck in some scenarios when sync-point is enabled #6827 @hicqu
  - Fix a bug that changefeed API does not work properly after the cdc server restarts #5837 @asddongmen
  - Fix the data race issue in the black hole sink #6206 @asddongmen
  - Fix the TiCDC panic issue when you set `enable-old-value = false` #6198 @hi-rustin
  - Fix the data consistency issue when the redo log feature is enabled #6189 #6368 #6277 #6456 #6695 #6764 #6859 @asddongmen
  - Fix poor redo log performance by writing redo events asynchronously #6011 @CharlesCheung96
  - Fix the issue that the MySQL sink cannot connect to IPv6 addresses #6135 @hi-rustin

- Backup & Restore (BR)
  - Fix a bug that BR reports `ErrRestoreTableIDMismatch` in RawKV mode #35279 @3pointer
  - Adjust the backup data directory structure to fix backup failure caused by S3 rate limiting in large cluster backup #30087 @MoCuishle28
  - Fix incorrect backup time in the summary log #35553 @ixuh12

- Dumpling
  - Fix the issue that GetDSN does not support IPv6 #36112 @D3Hunter

- TiDB Binlog
  - Fix a bug that Drainer cannot send requests correctly to Pump when `compressor` is set to `gzip` #1152 @lichunzhu

16.5.8 TiDB 6.1.0 Release Notes

Release date: June 13, 2022

TiDB version: 6.1.0

Quick access: Quick start | Production deployment

In 6.1.0, the key new features or improvements are as follows:

- List partitioning and list COLUMNS partitioning become GA, compatible with MySQL 5.7
- TiFlash partitioned table (dynamic pruning) becomes GA
- Support user-level lock management, compatible with MySQL
- Support non-transactional DML statements (only support `DELETE`)
- TiFlash supports on-demand data compaction
- MPP introduces the window function framework
- TiCDC supports replicating changelogs to Kafka via Avro
- TiCDC supports splitting large transactions during replication, which significantly reduces replication latency caused by large transactions
- The optimistic mode for merging and migrating sharded tables becomes GA

16.5.8.1 New Features

16.5.8.1.1 SQL

- List partitioning and list COLUMNS partitioning become GA. Both are compatible with MySQL 5.7.

  User documents: List partitioning, List COLUMNS partitioning

- TiFlash supports initiating a compact command. (experimental)

  TiFlash v6.1.0 introduces the `ALTER TABLE ... COMPACT` statement, which provides a manual way to compact physical data based on the existing background compaction mechanism. With this statement, you can update data in earlier formats and improve read/write performance any time as appropriate. It is recommended that you execute this statement to compact data after upgrading your cluster to v6.1.0. This statement is an extension of the standard SQL syntax and therefore is compatible with MySQL clients. For scenarios other than TiFlash upgrade, usually there is no need to use this statement.

  User document, #4145

- TiFlash implements the window function framework and supports the following window functions:

  - `RANK()`
  - `DENSE_RANK()`
  - `ROW_NUMBER()`

  User document, #33072

16.5.8.1.2 Observability

- Continuous Profiling supports the ARM architecture and TiFlash.

  User document

- Grafana adds a Performance Overview dashboard to provide a system-level entry for overall performance diagnosis.

  As a new dashboard in the TiDB visualized monitoring component Grafana, Performance Overview provides a system-level entry for overall performance diagnosis. According to the top-down performance analysis methodology, the Performance Overview dashboard reorganizes TiDB performance metrics based on database time breakdown and displays these metrics in different colors. By checking these colors, you can identify performance bottlenecks of the entire system at the first glance, which significantly reduces performance diagnosis time and simplifies performance analysis and diagnosis.

  User document

16.5.8.1.3 Performance

- Support customized Region size

  Starting from v6.1.0, you can configure `coprocessor.region-split-size` to set Regions to a larger size. This can effectively reduce the number of Regions, make Regions easier to manage, and improve the cluster performance and stability.

  User document, #11515

- Support using buckets to increase concurrency (experimental)

  To help you further improve the query concurrency after setting Regions to a larger size, TiDB introduces the concept of bucket, which is a smaller range within a Region. Using buckets as the query unit can optimize concurrent query performance when Regions are set to a larger size. Using buckets as the query unit can also dynamically adjust the sizes of hotspot Regions to ensure the scheduling efficiency and load balance. This feature is currently experimental. It is not recommended to use it in production environments.

  User document, #11515

- Use Raft Engine as the default log storage engine

  Since v6.1.0, TiDB uses Raft Engine as the default storage engine for logs. Compared with RocksDB, Raft Engine can reduce TiKV I/O write traffic by up to 40% and CPU usage by 10%, while improving foreground throughput by about 5% and reducing tail latency by 20% under certain loads.

  User document, #95

- Support the join order hint syntax

- The `LEADING` hint reminds the optimizer to use the specified order as the prefix of join operations. A good prefix of join can quickly reduce the amount of data at the early phase of join and improve the query performance.
- The `STRAIGHT_JOIN` hint reminds the optimizer to join tables in an order that is consistent with the order of tables in the `FROM` clause.

This provides a method for you to fix the order of table joins. A proper use of the hints can effectively enhance the SQL performance and cluster stability.

User document: `LEADING`, `STRAIGHT_JOIN`, #29932

- TiFlash supports four more functions:

  - `FROM_DAYS`
  - `TO_DAYS`
  - `TO_SECONDS`
  - `WEEKOFYEAR`

  User document, #4679, #4678, #4677

- TiFlash supports partitioned tables in dynamic pruning mode.

  To enhance performance in OLAP scenarios, dynamic pruning mode is supported for partitioned tables. If your TiDB is upgraded from versions earlier than v6.0.0, it is recommended that you manually update statistics of existing partitioned tables, so as to maximize the performance (not required for new installations or new partitions created after upgrade to v6.1.0).

  User documents: Access partitioned tables in the MPP mode, Dynamic pruning mode, #3873

16.5.8.1.4 Stability

- Automatic recovery from SST corruption

  When RocksDB detects a damaged SST file in the background, TiKV will try to schedule the affected Peer and recover its data using other replicas. You can set the maximum allowable time for the recovery using the `background-error-recovery-window` parameter. If the recovery operation is not completed within the time window, TiKV will panic. This feature automatically detects and recovers recoverable damaged storage, thus improving the cluster stability.

  User document, #10578

- Support non-transactional DML statement

  In the scenarios of large data processing, a single SQL statement with a large transaction might have a negative impact on the cluster stability and performance. Since v6.1.0, TiDB supports providing a syntax in which a `DELETE` statement is split into

multiple statements for batch processing. The split statements compromise transactional atomicity and isolation but greatly improve the cluster stability. For detailed syntax, see BATCH.

User document

- TiDB supports configuring the maximum GC wait time

  The transaction of TiDB adopts the Multi-Version Concurrency Control (MVCC) mechanism. When the newly written data overwrites the old data, the old data is not replaced, and both versions of data are stored. The old data is cleaned up by the Garbage Collection (GC) task periodically, which helps reclaim storage space to improve the performance and stability of the cluster. GC is triggered every 10 minutes by default. To ensure that long-running transactions can access the corresponding historical data, when there are transactions in execution, the GC task is delayed. To ensure that the GC task is not delayed indefinitely, TiDB introduces the system variable tidb_gc_max_wait_time to control the maximum delay time of the GC task. If the maximum delay time is exceeded, the GC will be forcibly executed. The default value of the variable is 24 hours. This feature enables you to control the relationship between the GC waiting time and the long-running transaction, which improves the stability of the cluster.

  User document

- TiDB supports configuring the maximum execution time for automatic statistics collection tasks

  Databases can effectively understand the distribution of data by collecting statistics, which helps generate reasonable execution plans and improve the efficiency of SQL execution. TiDB regularly collects statistics on frequently changed data objects in the background. However, collecting statistics takes up cluster resources and might affect the stable operation of the business during business peaks.

  Starting from v6.1.0, TiDB introduces tidb_max_auto_analyze_time to control the maximum execution time for background statistics collection, which is 12 hours by default. When the application does not encounter a resource bottleneck, it is recommended not to modify this variable so that TiDB can timely collect statistics.

  User document

16.5.8.1.5 Ease of use

- Support a one-stop online data recovery when multiple replicas are lost

  Before TiDB v6.1.0, when multiple Region replicas are lost because of machine failure, users have to stop all TiKV servers and use TiKV Control to recover TiKV one by one. Since TiDB v6.1.0, the recovery process is fully automated, does not require to stop TiKV, and does not affect other applications online. The recovery process can be triggered using PD Control and provides a more user-friendly summary information.

  User document, #10483

- Support viewing history statistics collection tasks

  You can use the `SHOW ANALYZE STATUS` statement to show cluster-level statistics collection tasks. Before TiDB v6.1.0, the `SHOW ANALYZE STATUS` statement shows instance-level tasks only, and history task records are cleared after a TiDB restart. Therefore, you cannot view history statistics collection time and details. Starting from TiDB v6.1.0, history records of statistics collection tasks are persisted and can be queried after a cluster restart, which provides a reference for troubleshooting query performance issues caused by statistics anomalies.

  User document

- Support modifying TiDB, TiKV, and TiFlash configurations dynamically

In earlier TiDB versions, after modifying a configuration item, you must restart the cluster to make the modification effective. This might interrupt online services. To address this issue, TiDB v6.1.0 introduces the dynamic configuration feature, which allows you to validate a parameter change without restarting the cluster. The specific optimizations are as follows:

```
* Transform some TiDB configuration items to system variables, so that they
    ↪ can be modified dynamically and persisted. Note that the original
    ↪ configuration items are deprecated after transformation. For a
    ↪ detailed list of the transformed configuration items, see [
    ↪ Configuration file parameters](#configuration-file-parameters).
* Support configuring some TiKV parameters online. For a detailed list of
    ↪ the parameters, see [Others](#others).
* Transform the TiFlash configuration item `max_threads` to a system
    ↪ variable `tidb_max_tiflash_threads`, so that the configuration can be
    ↪  modified dynamically and persisted. Note that the original
    ↪ configuration item remains after transformation.
```

For v6.1.0 clusters upgraded (including online and offline upgrades) from earlier versions, note that:

```
* If the configuration items specified in the configuration file before the
    ↪ upgrade already exist, TiDB will automatically update the values of
    ↪ the configured items to those of the corresponding system variables
    ↪ during the upgrade process. In this way, after the upgrade, the
    ↪ system behavior is not affected by parameter optimization.
* The automatic update mentioned above occurs only once during the upgrade.
    ↪ After the upgrade, the deprecated configuration items are no longer
    ↪ effective.
```

This feature allows you to modify parameters dynamically, and validate and persist them, instead of restarting the system and interrupting services. This makes your daily maintenance easier.

User document

- Support killing queries or connections globally

  You can control the Global Kill feature using the `enable-global-kill` configuration (enabled by default).

  Before TiDB v6.1.0, when an operation consumes a lot of resources and causes cluster stability issues, you have to connect to the target TiDB instance and then run the `KILL TIDB ${id};` command to terminate the target connection and operation. In the case of many TiDB instances, this method is not easy to use and prone to wrong operations. Starting from v6.1.0, the `enable-global-kill` configuration is introduced and enabled by default. You can run the kill command in any TiDB instance to terminate a specified connection and operation, without worrying about incorrectly terminating other queries or sessions by mistake when there is a proxy between the client and TiDB. Currently, TiDB does not support using Ctrl+C to terminate queries or sessions.

  User document, #8854

- TiKV API V2 (experimental)

  Before v6.1.0, when TiKV is used as Raw Key Value storage, TiKV only provides basic Key Value read and write capability because it only stores the raw data passed in by the client.

  TiKV API V2 provides a new Raw Key Value storage format and access interface, including:

  - The data is stored in MVCC and the change timestamp of the data is recorded. This feature will lay the foundation for implementing Change Data Capture and incremental backup and restore.
  - Data is scoped according to different usage and supports co-existence of a single TiDB cluster, Transactional KV, RawKV applications.

Due to significant changes in the underlying storage format, after enabling API V2, you cannot roll back a TiKV cluster to a version earlier than v6.1.0. Downgrading TiKV might result in data corruption.

```
[User document](#api-version-new-in-v610), [#11745](https://github.com/tikv/
    ↪ tikv/issues/11745)
```

16.5.8.1.6 MySQL compatibility

- Support compatibility with user-level lock management with MySQL

  User-level locks are a user-named lock management system provided by MySQL through built-in functions. The locking functions can provide lock blocking, waiting, and other lock management capabilities. User-level locks are also widely used in ORM

frameworks, such as Rails, Elixir, and Ecto. Since v6.1.0, TiDB has supported MySQL-compatible user-level lock management, and supports `GET_LOCK`, `RELEASE_LOCK`, and `RELEASE_ALL_LOCKS` functions.

User document, #14994

16.5.8.1.7 Data migration

- The optimistic mode for merging and migrating sharded tables becomes GA

  DM adds a large number of scenario tests for tasks that merge and migrate data from sharded tables in the optimistic mode, which covers 90% of the daily use scenarios. Compared with the pessimistic mode, the optimistic mode is simpler and more efficient to use. It is recommended to use the optimistic mode preferably after you are familiar with the usage notes.

  User document

- DM WebUI supports starting a task according to the specified parameters

  When starting a migration task, you can specify a start time and a safe mode duration. This is especially useful when you create an incremental migration task with lots of sources, eliminating the need to specify the binlog start position specifically for each source.

  User document, #5442

16.5.8.1.8 TiDB data share subscription

- TiDB supports data sharing with various third-party data ecosystems

  - TiCDC supports sending TiDB incremental data to Kafka in the Avro format, allowing data sharing with third-parties, such as KSQL and Snowflake via Confluent.

    User document, #5338

  - TiCDC supports dispatching incremental data from TiDB to different Kafka topics by table, which, combined with the Canal-json format, allows sharing data directly with Flink.

    User document, #4423

  - TiCDC supports SASL GSSAPI authentication types and adds SASL authentication examples using Kafka.

    User document, #4423

- TiCDC supports replicating `charset=GBK` tables.

  User document, #4806

16.5.8.2 Compatibility changes

16.5.8.2.1 System variables

Variable name	Change type	Description
tidb_enable_list_partition	Modified	The default value is changed from OFF to ON.
tidb_mem_quota_query	Modified	This variable adds the GLOBAL scope, and the variable value persists to the cluster.

Variable name	Change type	Description
tidb_query_log_max_len	Modified	The variable scope is changed from INSTANCE to GLOBAL. The variable value persists to the cluster, and the value range is changed to [0, 1073741824].

Variable name	Change type	Description
`require_secure_transport`	Newly added	This setting was previously a `tidb.toml` option (`security.require-secure-transport`), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_committer_concurrency	Newly added	This setting was previously a `tidb.toml` option (`performance.committer-concurrency`), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_enable_auto_analyze	Newly added	This setting was previously a `tidb.toml` option (`run-auto-analyze`), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_enable_only_full_group_by_check ↪	Newly added	This variable controls the behavior when TiDB performs the ONLY_FULL_GROUP_BY ↪ check.

Variable name	Change type	Description
tidb_enable_outer_join_reorder ↪	Newly added	Since v6.1.0, the Join Reorder algorithm of TiDB supports Outer Join. This variable controls the support behavior, and the default value is ON.

Variable name	Change type	Description
tidb_enable_prepared_plan_cache	Newly added	This setting was previously a tidb. toml option (prepared-plan-cache.enabled), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_gc_max_wait_time ↪	Newly added	This variable is used to set the maximum time of GC safe point blocked by uncommitted transactions.

Variable name	Change type	Description
tidb_max_auto_analyze_time	Newly added	This variable is used to specify the maximum execution time of auto analyze.
tidb_max_tiflash_threads ↪	Newly added	This variable is used to set the maximum concurrency for TiFlash to execute a request.

Variable name	Change type	Description
tidb_memory_oom_action ↪	Newly added	This setting was previously a tidb ↪ . ↪ toml ↪ option (oom ↪ - ↪ action ↪), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_mem_quota_analyze ↪	Newly added	This variable controls the maximum memory usage when TiDB updates statistics, including manually executed ANALYZE ↪ ↪ TABLE ↪ by users and automatic analyze tasks in the TiDB background.
		3990

Variable name	Change type	Description
tidb_nontransactional_ignore_error ↪	Newly added	This variable specifies whether to return error immediately when an error occurs in a non-transactional DML statement.

Variable name	Change type	Description
tidb_prepared_plan_cache_memory_guard_ratio	Newly added	This setting was previously a tidb. toml option (prepared - plan - cache . memory - guard - ratio), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_prepared_plan_cache_size	Newly added	This setting was previously a `tidb.toml` option (`prepared-plan-cache.capacity`), but changed to a system variable starting from TiDB v6.1.0.

Variable name	Change type	Description
tidb_stats_cache_mem_quota ↪	Newly added	This variable sets the memory quota for the TiDB statistics cache.

16.5.8.2.2 Configuration file parameters

Configuration file	Configuration	Change type	Description
TiDB	`committer-concurrency`	Deleted	Replaced by the system variable `tidb_committer_concurrency`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`lower_case_table_names`	Deleted	Currently TiDB only supports `lower_case_table_name` =2. If another value is set, after the cluster is upgraded to v6.1.0, the value is lost.

Configuration file	Configuration item	Change type	Description
TiDB	`mem-quota-query`	Deleted	Replaced by the system variable `tidb_mem_quota_query`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`oom-action`	Deleted	Replaced by the system variable `tidb_mem_oom_action`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`prepared-plan-cache`	Deleted	Replaced by the system variable `tidb_prepared_plan_cache_size`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`prepared-plan-cache.enabled`	Deleted	Replaced by the system variable `tidb_enable_prepared_plan_cache`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`query` ↪ `-` ↪ `log` ↪ `-` ↪ `max` ↪ `-` ↪ `len` ↪	Deleted	Replaced by the system variable `tidb_query_log_max_len` ↪ . This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration item	Change type	Description
TiDB	`require-secure-transport`	Deleted	Replaced by the system variable `require_secure_transport`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration item	Change type	Description
TiDB	`run-auto-analyze`	Deleted	Replaced by the system variable `tidb_enable_auto_analyze`. This configuration item is no longer valid, if you want to modify the value, you need to modify the corresponding system variable.

Configuration file	Configuration	Change type	Description
TiDB	`enable-global-kill`	Newly added	Controls whether to enable the Global Kill (terminating queries or connections across instances) feature. When the value is `true`, both `KILL` and `KILL TIDB` statements can terminate queries or connections

Configuration file	Configuration	Change type	Description
TiDB	`enable-stats-cache-mem-quota`	Newly added	Controls whether to enable the memory quota for the statistics cache.
TiKV	`raft.raft-engine.enable`	Modified	The default value is changed from `FALSE` to `TRUE`.

Configuration file	Configuration	Change type	Description
TiKV	`region-max-keys`	Modified	The default value is changed from 1440000 to `region-split-keys / 2 * 3.`

Configuration file	Configuration	Change type	Description
TiKV	`region-max-size`	Modified	The default value is changed from 144 MB to `region-split-size / 2 * 3`
TiKV	`coprocessor.enable-region-bucket`	Newly added	Determines whether to divide a Region into smaller ranges called buckets.

Configuration file	Configuration	Change type	Description
TiKV	coprocessor.region-bucket-size	Newly added	The size of a bucket when enable is true.
TiKV	causal-ts.renew-batch-min-size	Newly added	The minimum number of locally cached timestamps.
TiKV	causal-ts.renew-interval	Newly added	The interval at which the locally cached timestamps are refreshed.

Configuration file	Configuration	Change type	Description
TiKV	`max-snapshot-file-raw-size`	Newly added	The snapshot file will split to multiple files when the snapshot file size exceeds this value.
TiKV	`raft-engine.memory-limit`	Newly added	Specifies the limit on the memory usage of Raft Engine.

Configuration file	Configuration	Change type	Description
TiKV	`storage.background-error-recovery-window`	Newly added	The maximum recovery time is allowed after RocksDB detects a recoverable background error.

Configuration file	Configuration	Change type	Description
TiKV	`storage.api-version`	Newly added	The storage format and interface version used by TiKV when TiKV serves as the raw key-value store.
PD	`scheduler.max-store-preparing-time`	Newly added	Controls the maximum waiting time for the store to go online.

Configuration file	Configuration	Change type	Description
TiCDC	`enable-tls`	Newly added	Whether to use TLS to connect to the downstream Kafka instance.

Configuration file	Configuration	Change type	Description
TiCDC	`sasl-gssapi-user`	Newly added	Used to support SASL/GSSAPI authentication for Kafka. For details, see Configure sink URI with `kafka`.
	`sasl-gssapi-password`		
	`sasl-gssapi-service-name`		
	`sasl-gssapi-realm`		
	`sasl-gssapi-key`		
	`sasl-gssapi-keytab-path`		
	`sasl-gssapi-kerberos-config-path`		

Configuration file	Configuration	Change type	Description
TiCDC	`avro-decimal-handling-mode`	Newly added	Determines the output details of Avro format.
TiCDC	`dispatchers.topic`	Newly added	Controls how TiCDC dispatches incremental data to different Kafka topics.

Configuration file	Configuration	Change type	Description
TiCDC	`dispatchers.partition`	Newly added	`dispatchers.partition` is an alias for `dispatchers.dispatcher.partition` Controls how TiCDC dispatches incremental data to Kafka partitions.
TiCDC	`schema-registry`	Newly added	Specifies the schema registry endpoint that stores Avro schema.

Configuration file	Configuration	Change type	Description
DM	`worker`	Deleted	This parameter in the `dmctl` ↪ is not recommended for use. Will provide a simpler implementation.
	↪ `start` ↪ `-` ↪ `relay` ↪ command		
DM	`relay` ↪ `-` ↪ `dir` ↪ in the source configuration file	Deleted	Replaced by the same configuration item in the worker configuration file.

Configuration file	Configuration item	Change type	Description
DM	`is-sharding` in the task configuration file	Deleted	Replaced by the `shard-mode` configuration item.
DM	`auto-fix-gtid` in the task configuration file	Deleted	Deprecated in v5.x and officially deleted in v6.1.0.

Configuration file	Configuration	Change type	Description
DM	`meta` ↪ `-` ↪ `dir` ↪ and `charset` ↪ in the source configuration file	Deleted	Deprecated in v5.x and officially deleted in v6.1.0.

16.5.8.2.3 Others

- Enable Prepared Plan Cache by default

  Prepared Plan Cache is enabled by default in new clusters to cache the execution plans for `Prepare` / `Execute` requests. In the subsequent execution, query plan optimization can be skipped and thus leads to a performance boost. Upgraded clusters inherit the configuration from the configuration file. New clusters use the new default values, which means Prepared Plan Cache is enabled by default and each session can cache 100 plans at most (`capacity=100`). For the memory consumption of this feature, see memory management of Prepared Plan Cache.

- Prior to TiDB v6.1.0, `SHOW ANALYZE STATUS` shows instance-level tasks and the task records are cleared after TiDB restarts. Since TiDB v6.1.0, `SHOW ANALYZE STATUS` ↪ shows cluster-level tasks, and the task records persist after the restart. When `tidb_analyze_version = 2`, the `Job_info` column adds the `analyze option` information.

- Damaged SST files in TiKV might cause the TiKV process to panic. Before TiDB v6.1.0, damaged SST files caused TiKV to panic immediately. Since TiDB v6.1.0, the TiKV process will panic 1 hour after SST files are damaged.

- The following TiKV configuration items support modifying values dynamically:

  - `raftstore.raft-entry-max-size`
  - `quota.foreground-cpu-time`

- `quota.foreground-write-bandwidth`
- `quota.foreground-read-bandwidth`
- `quota.max-delay-duration`
- `server.grpc-memory-pool-quota`
- `server.max-grpc-send-msg-len`
- `server.raft-msg-max-batch-size`

- In v6.1.0, some configuration file parameters are converted to system variables. For v6.1.0 clusters upgraded (including online and offline upgrades) from earlier versions, note that:

    - If the configuration items specified in the configuration file before the upgrade already exist, TiDB will automatically update the values of the configured items to those of the corresponding system variables during the upgrade process. In this way, after the upgrade, the system behavior does not change thanks to parameter optimization.
    - The automatic update mentioned above occurs only once during the upgrade. After the upgrade, the deprecated configuration items are no longer effective.

- The Dashboard page is removed from DM WebUI.

- When `dispatchers.topic` and `dispatchers.partition` are enabled, TiCDC cannot be downgraded to versions earlier than v6.1.0.

- TiCDC Changefeed using the Avro protocol cannot be downgraded to versions earlier than v6.1.0.

16.5.8.3 Improvements

- TiDB

    - Improve the performance of the `UnionScanRead` operator #32433
    - Improve the display of task types in the output of `EXPLAIN` (add the MPP task type) #33332
    - Support using `rand()` as the default value of a column #10377
    - Support using `uuid()` as the default value of a column #33870
    - Support modifying the character set of columns from `latin1` to `utf8`/`utf8mb4` #34008

- TiKV

    - Improve the old value hit rate of CDC when using in-memory pessimistic lock #12279
    - Improve the health check to detect unavailable Raftstore, so that the TiKV client can update Region Cache in time #12398
    - Support setting memory limit on Raft Engine #12255

- TiKV automatically detects and deletes the damaged SST files to improve the product availability #10578
- CDC supports RawKV #11965
- Support splitting a large snapshot file into multiple files #11595
- Move the snapshot garbage collection from Raftstore to background thread to prevent snapshot GC from blocking Raftstore message loops #11966
- Support dynamic setting of the the maximum message length (`max-grpc-send-msg-len`) and the maximum batch size of gPRC messages (`raft-msg-max-batch-size`) #12334
- Support executing online unsafe recovery plan through Raft #10483

- PD

  - Support time-to-live (TTL) for region labels #4694
  - Support Region Buckets #4668
  - Disable compiling swagger server by default #4932

- TiFlash

  - Optimize memory calculation for an aggregate operator so that a more efficient algorithm is used in the merge phase #4451

- Tools

  - Backup & Restore (BR)

    - Support backing up and restoring empty databases #33866

  - TiDB Lightning

    - Optimize Scatter Region to batch mode to improve the stability of the Scatter Region process #33618

  - TiCDC

    - TiCDC supports splitting large transactions during replication, which significantly reduces replication latency caused by large transactions #5280

16.5.8.4 Bug fixes

- TiDB

  - Fix the issue of possible panic that might occur when the `in` function processes the `bit` type data #33070
  - Fix the issue of wrong query result because the `UnionScan` operator cannot maintain the order #33175
  - Fix the issue that the Merge Join operator gets wrong results in certain cases #33042

- Fix the issue that the `index join` result might be wrong in the dynamic pruning mode #33231
- Fix the issue that data might not be garbage-collected when some partitions of a partitioned table is dropped #33620
- Fix the issue that some DDL statements might be stuck for a period after the PD node of a cluster is replaced #33908
- Fix the issue that the TiDB server might run out of memory when the `INFORMATION_SCHEMA.CLUSTER_SLOW_QUERY` table is queried. This issue can be triggered when you check slow queries on the Grafana dashboard #33893
- Fix the issue that the system variable `max_allowed_packet` does not take effect #31422
- Fix the issue of memory leak in the TopSQL module #34525 #34502
- Fix the issue that the Plan Cache might be wrong on the PointGet plan #32371
- Fix the issue that query result might be wrong when Plan Cache is started in the RC isolation level #34447

- TiKV

  - Fix the issue that the Raft log lag is increasing when a TiKV instance is taken offline #12161
  - Fix the issue that TiKV panics and destroys peers unexpectedly because the target Region to be merged is invalid #12232
  - Fix the issue that TiKV reports the `failed to load_latest_options` error when upgrading from v5.3.1 or v5.4.0 to v6.0.0 or later versions #12269
  - Fix the issue of OOM caused by appending Raft logs when the memory resource is insufficient #11379
  - Fix the issue of TiKV panic caused by the race between destroying peers and batch splitting Regions #12368
  - Fix the issue of TiKV memory usage spike in a short time after `stats_monitor` falls into a dead loop #12416
  - Fix the issue that TiKV reports the `invalid store ID 0` error when using Follower Read #12478

- PD

  - Fix the wrong status code of `not leader` #4797
  - Fix a bug of TSO fallback in some corner cases #4884
  - Fix the issue that a removed tombstone store appears again after the PD leader transfer #4941
  - Fix the issue that scheduling cannot start immediately after the PD leader transfer #4769

- TiDB Dashboard

  - Fix a bug that Top SQL cannot collect the CPU overhead of the SQL statements that were running before the Top SQL feature is enabled #33859

- TiFlash
    - Fix potential data inconsistency after a lot of INSERT and DELETE operations #4956
- Tools
    - TiCDC
        - Fix excessive memory usage by optimizing the way DDL schemas are buffered #1386
        - Fix data loss that occurs in special incremental scanning scenarios #5468
    - TiDB Data Migration (DM)
        - Fix the `start-time` time zone issue and change DM behavior from using the downstream time zone to using the upstream time zone #5271
        - Fix the issue that DM occupies more disk space after the task automatically resumes #3734 #5344
        - Fix the problem that checkpoint flush may cause the data of failed rows to be skipped #5279
        - Fix the issue that in some cases manually executing the filtered DDL in the downstream might cause task resumption failure #5272
        - Fix an issue that the uppercase table cannot be replicated when `case-sensitive: true` is not set #5255
        - Fix the DM worker panic issue that occurs when the primary key is not first in the index returned by the `SHOW CREATE TABLE` statement #5159
        - Fix the issue that CPU usage may increase and a large amount of log is printed when GTID is enabled or when the task is automatically resumed #5063
        - Fix the offline option and other usage issues in DM WebUI #4993
        - Fix the issue that incremental tasks fail to start when GTID is empty in the upstream #3731
        - Fix the issue that empty configurations may cause dm-master to panic #3732
    - TiDB Lightning
        - Fix the issue that the precheck does not check local disk resources and cluster availability #34213
        - Fix the issue of incorrect routing for schemas #33381
        - Fix the issue that the PD configuration is not restored correctly when TiDB Lightning panics #31733
        - Fix the issue of Local-backend import failure caused by out-of-bounds data in the `auto_increment` column #29737
        - Fix the issue of local backend import failure when the `auto_random` or `auto_increment` column is null #34208

16.6 v6.0

16.6.1 TiDB 6.0.0 Release Notes

Release date: April 7, 2022

TiDB version: 6.0.0-DMR

> **Note:**
>
> The TiDB 6.0.0-DMR documentation has been archived. PingCAP encourages you to use the latest LTS version of the TiDB database.

In 6.0.0-DMR, the key new features or improvements are as follows:

- Support placement rules in SQL to provide more flexible management for data placement.
- Add a consistency check between data and indexes at the kernel level, which improves system stability and robustness, with only very low resource overhead.
- Provide Top SQL, a self-serving database performance monitoring and diagnosis feature for non-experts.
- Support Continuous Profiling that collects cluster performance data all the time, reducing MTTR for technical experts.
- Cache hotspot small tables in memory, which greatly improves the access performance, improves the throughput and reduces access latency.
- Optimize in-memory pessimistic locking. Under the performance bottleneck caused by pessimistic locks, memory optimization for pessimistic locks can effectively reduce latency by 10% and increase QPS by 10%.
- Enhance prepared statements to share execution plans, which lessens CPU resource consumption and improves SQL execution efficiency.
- Improve the computing performance of the MPP engine by supporting pushing down more expressions and the general availability (GA) of the elastic thread pool.
- Add DM WebUI to facilitate managing a large number of migration tasks.
- Improve the stability and efficiency of TiCDC when replicating data in large clusters. TiCDC now supports replicating 100,000 tables simultaneously.
- Accelerate leader balancing after restarting TiKV nodes, which improves the speed of business recovery after a restart.
- Support canceling the automatic update of statistics, which reduces resource contention and limits the impact on SQL performance.
- Provide PingCAP Clinic, an automatic diagnosis service for TiDB clusters (Technical Preview version).
- Provide TiDB Enterprise Manager, an enterprise-level database management platform.

Also, as a core component of TiDB's HTAP solution, TiFlashTM is officially open source in this release. For details, see TiFlash repository.

16.6.1.1 Release strategy changes

Starting from TiDB v6.0.0, TiDB provides two types of releases:

- Long-Term Support Releases

  Long-Term Support (LTS) releases are released approximately every six months. An LTS release introduces new features and improvements, and accepts patch releases within its release lifecycle. For example, v6.1.0 will be an LTS release.

- Development Milestone Releases

  Development Milestone Releases (DMR) are released approximately every two months. A DMR introduces new features and improvements, but does not accept patch releases. It is not recommended for users to use DMR in production environments. For example, v6.0.0-DMR is a DMR.

TiDB v6.0.0 is a DMR, and its version is 6.0.0-DMR.

16.6.1.2 New features

16.6.1.2.1 SQL

- SQL-based placement rules for data

  TiDB is a distributed database with excellent scalability. Usually, data is deployed across multiple servers or even multiple data centers. Therefore, data scheduling management is one of the most important basic capabilities of TiDB. In most cases, users do not need to care about how to schedule and manage data. However, with the increasing application complexity, deployment changes caused by isolation and access latency have become new challenges for TiDB. Since v6.0.0, TiDB officially provides data scheduling and management capabilities based on SQL interfaces. It supports flexible scheduling and management in dimensions such as replica counts, role types, and placement locations for any data. TiDB also supports more flexible management for data placement in multi-service shared clusters and cross-AZ deployments.

  User document

- Support building TiFlash replicas by databases. To add TiFlash replicas for all tables in a database, you only need to use a single SQL statement, which greatly saves operation and maintenance costs.

  User document

16.6.1.2.2 Transaction

- Add a check for data index consistency at the kernel level

  Add a check for data index consistency when a transaction is executed, which improves system stability and robustness, with only very low resource overhead. You can control the check behavior using the `tidb_enable_mutation_checker` and `tidb_txn_assertion_level` variables. With the default configuration, the QPS drop is controlled within 2% in most scenarios. For the error description of the consistency check, see user document.

16.6.1.2.3 Observability

- Top SQL: Performance diagnosis for non-experts

  Top SQL is a self-serving database performance monitoring and diagnosis feature in TiDB Dashboard, for DBAs and App developers, which is now generally available in TiDB v6.0.

  Unlike existing diagnostic features for experts, Top SQL is designed for non-experts: you do not need to traverse thousands of monitoring charts to find correlations or understand TiDB internal mechanisms such as Raft Snapshot, RocksDB, MVCC, and TSO. To use Top SQL for analyzing database load quickly and improving App performance, only basic database knowledge (such as index, lock conflict, and execution plans) is needed.

  Top SQL is not enabled by default. When enabled, Top SQL provides you with the real-time CPU load of each TiKV or TiDB node. Therefore, you can spot SQL statements consuming high CPU loads at first glimpse, and quickly analyze the issues such as database hotspots and sudden load increases. For example, you can use Top SQL to pinpoint and diagnose an unusual query that consumes 90% CPU of a single TiKV node.

  User documentation

- Support Continuous Profiling

  TiDB Dashboard introduces the Continuous Profiling feature, which is now generally available in TiDB v6.0. Continuous profiling is not enabled by default. When enabled, the performance data of individual TiDB, TiKV, and PD instances will be collected all the time, with negligible overhead. With history performance data, technical experts can backtrack and pinpoint the root causes of issues like high memory consumption, even when the issues are difficult to reproduce. In this way, the mean time to recovery (MTTR) can be reduced.

  User document

16.6.1.2.4 Performance

- Cache hotspot small tables

  For user applications in scenarios where hotspot small tables are accessed, TiDB supports explicitly caching the hotspot tables in memory, which greatly improves the access performance, improves the throughput, and reduces access latency. This solution can effectively avoid introducing a third-party cache middleware, reduce the complexity of the architecture, and cut the cost of operation and maintenance. The solution is suitable for scenarios where small tables are frequently accessed but rarely updated, such as the configuration tables or exchange rate tables.

  User document, #25293

- In-memory pessimistic locking

  Since TiDB v6.0.0, in-memory pessimistic locking is enabled by default. After enabling this feature, pessimistic transaction locks are managed in memory. This avoids persisting pessimistic locks and the Raft replication of the lock information, and greatly reduces the overhead of managing pessimistic transaction locks. Under the performance bottleneck caused by pessimistic locks, memory optimization for pessimistic locks can effectively reduce latency by 10% and increase QPS by 10%.

  User document, #11452

- Optimization to get TSO at the Read Committed isolation level

  To reduce query latency, when read-write conflicts are rare, TiDB adds the `tidb_rc_read_check_ts` system variable at the Read Committed isolation level to get less unnecessary TSO. This variable is disabled by default. When the variable is enabled, this optimization avoids getting duplicated TSO to reduce latency in scenarios where there is no read-write conflict. However, in scenarios with frequent read-write conflicts, enabling this variable might cause a performance regression.

  User document, #33159

- Enhance prepared statements to share execution plans

  Reusing SQL execution plans can effectively reduce the time for parsing SQL statements, lessen CPU resource consumption, and improve SQL execution efficiency. One of the important methods of SQL tuning is to reuse SQL execution plans effectively. TiDB has supported sharing execution plans with prepared statements. However, when the prepared statements are closed, TiDB automatically clears the corresponding plan cache. After that, TiDB might unnecessarily parse the repeated SQL statements, affecting the execution efficiency. Since v6.0.0, TiDB supports controlling whether to ignore the `COM_STMT_CLOSE` command through the `tidb_ignore_prepared_cache_close_stmt` parameter (disabled by default). When the parameter is enabled, TiDB ignores the command of closing prepared statements and keeps the execution plan in the cache, improving the reuse rate of the execution plan.

  User document, #31056

- Improve query pushdown

  With its native architecture of separating computing from storage, TiDB supports filtering out invalid data by pushing down operators, which greatly reduces the data transmission between TiDB and TiKV and thereby improves the query efficiency. In v6.0.0, TiDB supports pushing down more expressions and the `BIT` data type to TiKV, improving the query efficiency when computing the expressions and data type.

  User document, #30738

- Optimization of hotspot index

  Writing monotonically increasing data in batches to the secondary index causes an index hotspot and affects the overall write throughput. Since v6.0.0, TiDB supports scattering the index hotspot using the `tidb_shard` function to improve the write performance. Currently, `tidb_shard` only takes effect on the unique secondary index. This application-friendly solution does not require modifying the original query conditions. You can use this solution in the scenarios of high write throughput, point queries, and batch point queries. Note that using the data that has been scattered by range queries in the application might cause a performance regression. Therefore, do not use this function in such cases without verification.

  User document, #31040

- Support dynamic pruning mode for partitioned tables in TiFlash MPP engine (experimental)

  In this mode, TiDB can read and compute the data on partitioned tables using the MPP engine of TiFlash, which greatly improves the query performance of partitioned tables.

  User document

- Improve the computing performance of the MPP engine

  - Support pushing down more functions and operators to the MPP engine
    - Logical functions: `IS`, `IS NOT`
    - String functions: `REGEXP()`, `NOT REGEXP()`
    - Mathematical functions: `GREATEST(int/real)`, `LEAST(int/real)`
    - Date functions: `DAYNAME()`, `DAYOFMONTH()`, `DAYOFWEEK()`, `DAYOFYEAR()`, `LAST_DAY()`, `MONTHNAME()`
    - Operators: Anti Left Outer Semi Join, Left Outer Semi Join

    User document

  - The elastic thread pool (enabled by default) becomes GA. This feature aims to improve CPU utilization.

    User document

16.6.1.2.5 Stability

- Enhance baseline capturing of execution plans

  Enhance the usability of baseline capturing of execution plans by adding a blocklist with such dimensions as table name, frequency, and user name. Introduce a new algorithm to optimize memory management for caching bindings. After baseline capturing is enabled, the system automatically creates bindings for most OLTP queries. Execution plans of bound statements are fixed, avoiding performance problems due to any change in the execution plans. Baseline capturing is applicable to scenarios such as major version upgrades and cluster migration, and helps reduce performance problems caused by regression of execution plans.

  User document, #32466

- Support TiKV quota limiter (experimental)

  If your machine deployed with TiKV has limited resources and the foreground is burdened by an excessively large amount of requests, background CPU resources are occupied by the foreground, causing TiKV performance unstable. In TiDB v6.0.0, you can use the quota-related configuration items to limit the resources used by the foreground, including CPU and read/write bandwidth. This greatly improves stability of clusters under long-term heavy workloads.

  User document, #12131

- Support the zstd compression algorithm in TiFlash

  TiFlash introduces two parameters, `profiles.default.dt_compression_method` and `profiles.default.dt_compression_level`, which allow users to select the optimal compression algorithm based on performance and capacity balance.

  User document

- Enable all I/O checks (Checksum) by default

  This feature was introduced in v5.4.0 as experimental. It enhances data accuracy and security without imposing an obvious impact on users' businesses.

  Warning: Newer version of data format cannot be downgraded in place to versions earlier than v5.4.0. During such a downgrade, you need to delete TiFlash replicas and replicate data after the downgrade. Alternatively, you can perform a downgrade by referring to dttool migrate.

  User document

- Improve thread utilization

  TiFlash introduces asynchronous gRPC and Min-TSO scheduling mechanisms. Such mechanisms ensure more efficient use of threads and avoid system crashes caused by excessive threads.

  User document

16.6.1.2.6 Data migration

TiDB Data Migration (DM)

- Add WebUI (experimental)

  With the WebUI, you can easily manage a large number of migration tasks. On the WebUI, you can:

  - View migration tasks on Dashboard
  - Manage migration tasks
  - Configure upstream settings
  - Query replication status
  - View master and worker information

  WebUI is still experimental and is still under development. Therefore, it is recommended only for trial. A known issue is that problems might occur if you use WebUI and dmctl to operate the same task. This issue will be resolved in later versions.

  User document

- Add an error handling mechanism

  More commands are introduced to address problems that interrupt a migration task. For example:

  - In case of a schema error, you can update the schema file by using the `--from-source/--from-target` parameter of the `binlog-schema update` command, instead of editing the schema file separately.
  - You can specify a binlog position to inject, replace, skip, or revert a DDL statement.

  User document

- Support full data storage to Amazon S3

  When DM performs all or full data migration tasks, sufficient hard disk space is required for storing full data from upstream. Compared with EBS, Amazon S3 has nearly infinite storage at lower costs. Now, DM supports configuring Amazon S3 as the dump directory. That means you can use S3 to store full data when you perform all or full data migration tasks.

  User document

- Support starting a migration task from specified time

  A new parameter `--start-time` is added to migration tasks. You can define time in the format of '2021-10-21 00:01:00' or '2021-10-21T00:01:00'.

  This feature is particularly useful in scenarios where you migrate and merge incremental data from shard mysql instances. Specifically, you do not need to set a binlog start

point for each source in an incremental migration task. Instead, you can create an incremental migration task quickly by using the `--start-time` parameter in `safe-mode`.

User document

TiDB Lightning

- Support configuring the maximum number of tolerable errors

  Added a configuration item `lightning.max-error`. The default value is 0. When the value is greater than 0, the max-error feature is enabled. If an error occurs in a row during encoding, a record containing this row is added to `lightning_task_info.type_error_v1` in the target TiDB and this row is ignored. When rows with errors exceed the threshold, TiDB Lightning exits immediately.

  Matching the `lightning.max-error` configuration, the `lightning.task-info-schema-name` configuration item records the name of the database that reports a data saving error.

  This feature does not cover all types of errors, for example, syntax errors are not applicable.

  User document

16.6.1.2.7 TiDB data share subscription

- Support replicating 100,000 tables simultaneously

  By optimizing the data processing flow, TiCDC reduces the resource consumption of processing incremental data for each table, which greatly improves the replication stability and efficiency when replicating data in large clusters. The result of an internal test shows that TiCDC can stably support replicating 100,000 tables simultaneously.

16.6.1.2.8 Deployment and maintenance

- Enable new collation rules by default

  Since v4.0, TiDB has supported new collation rules that behave the same way as MySQL in the case-insensitive, accent-insensitive, and padding rules. The new collation rules are controlled by the `new_collations_enabled_on_first_bootstrap` parameter, which was disabled by default. Since v6.0, TiDB enables the new collation rules by default. Note that this configuration takes effect only upon TiDB cluster initialization.

  User documentation

- Accelerate leader balancing after restarting TiKV nodes

  After a restart of TiKV nodes, the unevenly scattered leaders must be redistributed for load balance. In large-scale clusters, leader balancing time is positively correlated with the number of Regions. For example, the leader balancing of 100K Regions can take 20-30 minutes, which is prone to performance issues and stability risks due to uneven load. TiDB v6.0.0 provides a parameter to control the balancing concurrency and enlarges the default value to 4 times of the original, which greatly shortens the leader rebalancing time and accelerates the business recovery after a restart of the TiKV nodes.

  User documentation, #4610

- Support canceling the automatic update of statistics

  Statistics are one of the most important basic data that affect SQL performance. To ensure the completeness and timeliness of statistics, TiDB automatically updates object statistics periodically in the background. However, automatic statistics updates may result in resource contention, affecting SQL performance. To address this issue, you can manually cancel the automatic update of statistics since v6.0.

  User documentation

- PingCAP Clinic diagnostic service (Technical Preview version)

  PingCAP Clinic is a diagnostic service for TiDB clusters. This service helps troubleshoot cluster issues remotely and provides a quick check of cluster status locally. With PingCAP Clinic, you can ensure the stable operation of your TiDB cluster during its full life cycle, predict potential issues, reduce the probability of issues, and quickly troubleshoot cluster issues.

  When contacting PingCAP technical support for remote assistance to troubleshoot cluster issues, you can use the PingCAP Clinic service to collect and upload diagnostic data, thereby improving the troubleshooting efficiency.

  User documentation

- An enterprise-level database management platform, TiDB Enterprise Manager

  TiDB Enterprise Manager (TiEM) is an enterprise-level database management platform based on the TiDB database, which aims to help users manage TiDB clusters in self-hosted or public cloud environments.

  TiEM not only provides full lifecycle visual management for TiDB clusters, but also provides one-stop services: parameter management, version upgrades, cluster clone, active-standby cluster switching, data import and export, data replication, and data backup and restore services. TiEM can improve the efficiency of DevOps on TiDB and reduce the DevOps cost for enterprises.

  Currently, TiEM is provided in the TiDB Enterprise edition only. To get TiEM, contact us via the TiDB Enterprise page.

- Support customizing configurations of the monitoring components

  When you deploy a TiDB cluster using TiUP, TiUP automatically deploys monitoring components such as Prometheus, Grafana, and Alertmanager, and automatically adds new nodes into the monitoring scope after scale-out. You can customize the configurations of the monitoring components by adding configuration items to the `topology.yaml` file.

  User document

16.6.1.3 Compatibility changes

> **Note:**
>
> When upgrading from an earlier TiDB version to v6.0.0, if you want to know the compatibility change notes of all intermediate versions, you can check the Release Notes of the corresponding version.

16.6.1.3.1 System variables

Variable name	Change type	Description
`placement_checks`	Deleted	Controls whether the DDL statement validates the placement rules specified by Placement Rules in SQL. Replaced by `tidb_placement_mode` .
`tidb_enable_alter_placement`	Deleted	Controls whether to enable placement rules in SQL.

Variable name	Change type	Description
`tidb_mem_quota_hashjoin` ↪ `tidb_mem_quota_indexlookupjoin` ↪ `tidb_mem_quota_indexlookupreader` ↪ `tidb_mem_quota_mergejoin` ↪ `tidb_mem_quota_sort` ↪ `tidb_mem_quota_topn` ↪	Deleted	Since v5.0, these variables are replaced by `tidb_mem_quota_query` ↪ and removed from the system variables document. To ensure compatibility, these variables were kept in source code. Since TiDB 6.0.0, these variables are removed

Variable name	Change type	Description
tidb_enable_mutation_checker ↪	Newly added	Controls whether to enable the mutation checker. The default value is ON. For existing clusters that upgrade from versions earlier than v6.0.0, the mutation checker is disabled by default.

Variable name	Change type	Description
tidb_ignore_prepared_cache_close_stmt ↪	Newly added	Controls whether to ignore the command that closes Prepared Statement. The default value is `OFF`.
tidb_mem_quota_binding_cache ↪	Newly added	Sets the memory usage threshold for the cache holding `binding` ↪ . The default value is `67108864` ↪ (64 MiB).

Variable name	Change type	Description
tidb_placement_mode ↪	Newly added	Controls whether DDL statements ignore the placement rules specified by Placement Rules in SQL. The default value is `strict` ↪ , which means that DDL statements do not ignore placement rules.

| tidb_rc_read_check_ts | Newly added |

Optimizes read statement latency within a transaction. If read/write conflicts are more severe, turning this variable on will add additional overhead and latency, causing regressions in performance. The default value is `off`.

This variable is not yet compatible with replica-read. If a read request has `tidb_rc_read_check_ts` on, it might not be able to use replica-read. Do not turn on both variables at the same time.

| | `tidb_sysdate_is_now` | Newly added | Controls whether the `SYSDATE` function can be replaced by the `NOW` function. This configuration item has the same effect as the MySQL option `sysdate-is-now`. The default value is `OFF`. | | `tidb_table_cache_lease` | Newly added | Controls the lease time of table cache, in seconds. The default value is `3` ↪ . | | `tidb_top_sql_max_meta_count` | Newly added | Controls the maximum number of SQL statement types collected by Top SQL per minute. The default value is `5000`. | | `tidb_top_sql_max_time_series_count` | Newly added | Controls how many SQL statements that contribute the most to the load (that is, top N) can be recorded by Top SQL per minute. The default value is `100`. | | `tidb_txn_assertion_level` | Newly added | Controls the assertion level. The assertion is a consistency check between data and indexes, which checks whether a key being written exists in the transaction commit process. By default, the check enables most of the check items, with almost no impact on performance. For existing clusters that upgrade from versions earlier than v6.0.0, the check is disabled by default. |

16.6.1.3.2 Configuration file parameters

Configuration file	Configuration	Change type	Description
Configuration file	Configuration	Change type	Description
TiDB	`stmt` ↪ `-` ↪ `summary` ↪ `.` ↪ `enable` ↪ `stmt` ↪ `-` ↪ `summary` ↪ `.` ↪ `enable` ↪ `-` ↪ `internal` ↪ `-` ↪ `query` ↪ `stmt` ↪ `-` ↪ `summary` ↪ `.` ↪ `history` ↪ `-` ↪ `size` ↪ `stmt` ↪ `-` ↪ `summary` ↪ `.` ↪ `max` ↪ `-` ↪ `sql` ↪ `-` ↪ `length` ↪ `stmt` ↪ `-` ↪ `summary` ↪ `.` ↪ `max` ↪ `-` ↪ `stmt` ↪ `-` ↪ `count` ↪ `stmt`	Deleted	Configuration related to the statement summary tables. All these configuration items are removed. You need to use SQL variables to control the statement summary tables.

Configuration file	Configuration	Change type	Description
TiDB	`new_collations_enabled_on_first_bootstrap`	Modified	Controls whether to enable support for the new collation. Since v6.0, the default value is changed from `false` to `true`. This configuration item only takes effect when the cluster is initialized for the first time. After the first

Configuration file	Configuration item	Change type	Description
TiKV	backup ↪ . ↪ num ↪ - ↪ threads ↪	Modified	The value range is modified to [1, ↪ CPU ↪].
TiKV	raftstore ↪ . ↪ apply ↪ - ↪ max ↪ - ↪ batch ↪ - ↪ size ↪	Modified	The maximum value is changed to 10240 ↪ .

| TiKV | raftstore.raft-max-size-per-msg | Modified |

The minimum value is changed from 0 to larger than 0.

The maximum value is set to 3GB.

The unit is changed from MB to KB|MB|GB.

| | TiKV | raftstore.store-max-batch-size | Modified | The maximum value is set to 10240. | | TiKV | readpool.unified.max-thread-count | Modified | The adjustable range is changed to [min-thread-count, MAX(4, CPU)]. | | TiKV | rocksdb.enable- ↪ pipelined-write | Modified | The default value is changed from true to false. When this configuration is enabled, the previous Pipelined Write is used. When this configuration is disabled, the new Pipelined Commit mechanism is used. | | TiKV | rocksdb.max- ↪ background-flushes | Modified |

When the number of CPU cores is 10, the default value is 3.

When the number of CPU cores is 8, the default value is 2.

| | TiKV | rocksdb.max-background-jobs | Modified |

When the number of CPU cores is 10, the default value is 9.

When the number of CPU cores is 8, the default value is 7.

| | TiFlash | profiles.default.dt_enable_logical_split | Modified | Determines whether the segment of DeltaTree Storage Engine uses logical split. The default value is changed from true to false. | | TiFlash | profiles.default.enable_elastic_threadpool ↪ | Modified | Controls whether to enable the elastic thread pool. The default value is changed from false to true. | | TiFlash | storage.format_version | Modified | Controls the data validation feature of TiFlash. The default value is changed from 2 to 3.When format_version is set to 3, consistency check is performed on the read operations for all TiFlash data to avoid incorrect read due to hardware failure.Note that the new format version cannot be downgraded in place to versions earlier than v5.4. | | TiDB | pessimistic ↪ -txn.pessimistic-auto-commit | Newly added | Determines the transaction mode that the auto-commit transaction uses when the pessimistic transaction mode is globally enabled (tidb_txn_mode='pessimistic'). | | TiKV | pessimistic-txn.in-memory | Newly added | Controls whether to enable the in-memory pessimistic lock. With this feature enabled, pessimistic transactions store pessimistic locks in TiKV memory as much as possible, instead of writing pessimistic locks to disks or replicating to other replicas. This improves the performance of pessimistic transactions; however, there is a low probability that a pessimistic lock will be lost, which might cause the pessimistic transaction to fail to commit. The default value is true. | | TiKV | quota | Newly added | Add configuration items related to Quota Limiter, which limit the resources occupied by frontend requests. Quota Limiter is an experimental feature and is disabled by default. New quota-related configuration items are foreground-cpu-time, foreground-write-bandwidth, foreground-read-bandwidth ↪ , and max-delay-duration. | | TiFlash | profiles.default.dt_compression_method | Newly added | Specifies the compression algorithm for TiFlash. The optional values are LZ4, zstd and LZ4HC, all case insensitive. The default value is LZ4. | | TiFlash | profiles ↪ .default.dt_compression_level | Newly added | Specifies the compression level of TiFlash. The default value is 1. | | DM | loaders.<name>.import-mode | Newly added | The import mode during the full import phase. Since v6.0, DM uses TiDB Lightning's TiDB-backend mode to import data during the full import phase; the previous Loader component is no longer used. This is an internal replacement and has no obvious impact on daily operations.The default value is set to sql, which means using tidb-backend mode. In some rare cases, tidb-backend might not be fully compatible. You can fall back to Loader mode by configuring this parameter to loader. | | DM | loaders.<name>.on-duplicate | Newly added | Specifies the methods to resolve conflicts during the full import phase. The default value is replace, which means using the new data to replace the existing data. | | TiCDC | dial-timeout | Newly added | The timeout in establishing a connection with the downstream Kafka. The default value is 10s. | | TiCDC | read-timeout | Newly added | The timeout in getting a response returned by the downstream Kafka. The default value is 10s. | | TiCDC | write-timeout | Newly added | The timeout in sending a request to the downstream Kafka. The default value is 10s. |

16.6.1.3.3 Others

- The data placement policy has the following compatibility changes:

    - Binding is not supported. The direct placement option is removed from the syntax.
    - The `CREATE PLACEMENT POLICY` and `ALTER PLACEMENT POLICY` statements no longer support the `VOTERS` and `VOTER_CONSTRAINTS` placement options.
    - TiDB migration tools (TiDB Binlog, TiCDC, and BR) are now compatible with placement rules. The placement option is moved to a special comment in TiDB Binlog.
    - The `information_schema.placement_rules` system table is renamed to `information_schema.placement_policies`. This table now only displays information about placement policies.
    - The `placement_checks` system variable is replaced by `tidb_placement_mode`.
    - It is prohibited to add partitions with placement rules to tables that have TiFlash replicas.
    - Remove the `TIDB_DIRECT_PLACEMENT` column from the `INFORMATION_SCHEMA` table.

- The `status` value of SQL plan management (SPM) binding is modified:

    - Remove `using`.
    - Add `enabled` (available) to replace `using`.
    - Add `disabled` (unavailable).

- DM modifies the OpenAPI interface

    - Because of internal mechanism changes, the interface related to task management cannot be compatible with the previous experimental version. You need to refer to the new DM OpenAPI documentation for adaptation.

- DM changes the methods to resolve conflicts during the full import phase

    - A `loader.<name>.on-duplicate` parameter is added. The default value is `replace`, which means using the new data to replace the existing data. If you want to keep the previous behavior, you can set the value to `error`. This parameter only controls the behavior during the full import phase.

- To use DM, you should use the corresponding version of `dmctl`

    - Due to internal mechanism changes, after upgrading DM to v6.0.0, you should also upgrade `dmctl` to v6.0.0.

- For v5.4 and earlier versions, TiDB allows incorrect values for some noop system variables. Starting from v6.0.0, TiDB disallows setting incorrect values for system variables. #31538

16.6.1.4 Improvements

- TiDB

- Clear the placement rule settings of a table automatically after restoring the table using the `FLASHBACK` or `RECOVER` statement #31668
- Add a performance overview dashboard to show core performance metrics on typical critical paths, making metrics analysis on TiDB easier #31676
- Support using the `REPLACE` keyword in the `LOAD DATA LOCAL INFILE` statement #24515
- Support partition pruning for the built-in `IN` expression in Range partition tables #26739
- Improve query efficiency by eliminating potentially redundant Exchange operations in MPP aggregation queries #31762
- Improve compatibility with MySQL by allowing duplicate partition names in the `TRUNCATE PARTITION` and `DROP PARTITION` statements #31681
- Support showing the `CREATE_TIME` information in the results of the `ADMIN SHOW DDL JOBS` statement #23494
- Support a new built-in function `CHARSET()` #3931
- Support filtering a baseline capturing blocklist by usernames #32558
- Support using wildcards in a baseline capturing blocklist #32714
- Optimize the results of the `ADMIN SHOW DDL JOBS` and `SHOW TABLE STATUS` statements by displaying the time according to the current `time_zone` #26642
- Support pushing down the `DAYNAME()` and `MONTHNAME()` functions to TiFlash #32594
- Support pushing down the `REGEXP` function to TiFlash #32637
- Support pushing down the `DAYOFMONTH()` and `LAST_DAY()` functions to TiFlash #33012
- Support pushing down the `DAYOFWEEK()` and `DAYOFYEAR()` functions to TiFlash #33130
- Support pushing down the `IS_TRUE`, `IS_FALSE`, and `IS_TRUE_WITH_NULL` functions to TiFlash #33047
- Support pushing down the `GREATEST` and `LEAST` functions to TiFlash #32787
- Support tracking the execution of the `UnionScan` operator #32631
- Support using the PointGet plan for queries that read the `_tidb_rowid` column #31543
- Support showing the original partition name in the output of the `EXPLAIN` statement without converting the name to lowercase #32719
- Enable partition pruning for RANGE COLUMNS partitionings on IN conditions and string type columns #32626
- Return an error message when a system variable is set to NULL #32850
- Remove Broadcast Join from the non-MPP mode #31465
- Support executing MPP plans on partitioned tables in dynamic pruning mode #32347
- Support pushing down predicates for common table expressions (CTEs) #28163
- Simplify the configurations of `Statement Summary` and `Capture Plan Baselines` to be available on a global basis only #30557
- Update gopsutil to v3.21.12 to address alarms reported when building binary on macOS 12 #31607

- TiKV
    - Improve the sampling accuracy of the Raftstore for batches with many key ranges #12327
    - Add the correct "Content-Type" for `debug/pprof/profile` to make the Profile more easily identified #11521
    - Renew the lease time of the leader infinitely when the Raftstore has heartbeats or handles read requests, which helps reduce latency jitter #11579
    - Choose the store with the least cost when switching the leader, which helps improve performance stability #10602
    - Fetch Raft logs asynchronously to reduce the performance jitter caused by blocking the Raftstore #11320
    - Support the `QUARTER` function in vector calculation #5751
    - Support pushing down the `BIT` data type to TiKV #30738
    - Support pushing down the `MOD` function and the `SYSDATE` function to TiKV #11916
    - Reduce the TiCDC recovery time by reducing the number of the Regions that require the Resolve Locks step #11993
    - Support dynamically modifying `raftstore.raft-max-inflight-msgs` #11865
    - Support `EXTRA_PHYSICAL_TABLE_ID_COL_ID` to enable dynamic pruning mode #11888
    - Support calculation in buckets #11759
    - Encode the keys of RawKV API V2 as `user-key` + `memcomparable-padding` + `timestamp` #11965
    - Encode the values of RawKV API V2 as `user-value` + `ttl` + `ValueMeta` and encode `delete` in `ValueMeta` #11965
    - Support dynamically modifying `raftstore.raft-max-size-per-msg` #12017
    - Support monitoring multi-k8s in Grafana #12014
    - Transfer the leadership to CDC observer to reduce latency jitter #12111
    - Support dynamically modifying `raftstore.apply_max_batch_size` and `raftstore.store_max_batch_size` #11982
    - RawKV V2 returns the latest version upon receiving the `raw_get` or `raw_scan` request #11965
    - Support the RCCheckTS consistency reads #12097
    - Support dynamically modifying `storage.scheduler-worker-pool-size`(the thread count of the Scheduler pool) #12067
    - Control the use of CPU and bandwidth by using the global foreground flow controller to improve the performance stability of TiKV #11855
    - Support dynamically modifying `readpool.unified.max-thread-count` (the thread count of the UnifyReadPool) #11781
    - Use the TiKV internal pipeline to replace the RocksDB pipeline and deprecate the `rocksdb.enable-multibatch-write` parameter #12059
- PD

- Support automatically selecting the fastest object for transfer when evicting the leader, which helps speed up the eviction process #4229
- Forbid deleting a voter from a 2-replica Raft group in case the Region becomes unavailable #4564
- Speed up the scheduling of the balance leader #4652

- TiFlash

  - Forbid the logical splitting of TiFlash files (by adjusting the default value of `profiles.default.dt_enable_logical_split` to `false`. See user document for details) and improve the space usage efficiency of the TiFlash columnar storage so that the space occupation of a table synchronized to TiFlash is similar to the space occupation of the table in TiKV
  - Optimize the cluster management and replica replication mechanism for TiFlash by integrating the previous cluster management module into TiDB, which accelerates replica creation for small tables #29924

- Tools

  - Backup & Restore (BR)

    - Improve the speed of restoring the backup data. In the simulation test when BR restores 16 TB data to a TiKV cluster with 15 nodes (each node has 16 CPU cores), the throughput reaches 2.66 GiB/s. #27036
    - Support importing and exporting placement rules. Add a `--with-tidb-placement-mode` parameter to control whether to ignore the placement rules when importing data. #32290

  - TiCDC

    - Add a `Lag analyze` panel in Grafana #4891
    - Support placement rules #4846
    - Synchronize HTTP API handling #1710
    - Add the exponential backoff mechanism for restarting a changefeed #3329
    - Set the default isolation level of MySQL sink to read-committed to reduce deadlocks in MySQL #3589
    - Validate changefeed parameters upon creation and refine error messages #1716 #1718 #1719 #4472
    - Expose configuration parameters of the Kafka producer to make them configurable in TiCDC #4385

  - TiDB Data Migration (DM)

    - Support starting a task when upstream table schemas are inconsistent and in optimistic mode #3629 #3708 #3786
    - Support creating a task in the `stopped` state #4484
    - Support Syncer using the working directory of the DM-worker rather than `/tmp` to write internal files, and cleaning the directory after the task is stopped #4107

    - Precheck has improved. Some important checks are no longer skipped. #3608
  - TiDB Lightning
    - Add more retryable error types #31376
    - Support the base64 format password string #31194
    - Standardize error codes and error outputs #32239

16.6.1.5 Bug fixes

- TiDB
  - Fix a bug that TiDB fails to create tables with placement rules when `SCHEDULE = majority_in_primary`, and `PrimaryRegion` and `Regions` are of the same value #31271
  - Fix the `invalid transaction` error when executing a query using index lookup join #30468
  - Fix a bug that `show grants` returns incorrect results when two or more privileges are granted #30855
  - Fix a bug that `INSERT INTO t1 SET timestamp_col = DEFAULT` would set the timestamp to the zero timestamp for the field defaulted to `CURRENT_TIMESTAMP` #29926
  - Fix errors reported in reading the results by avoiding encoding the maximum value and minimum non-null value of the string type #31721
  - Fix load data panic if the data is broken at an escape character #31589
  - Fix the issue that the `greatest` or `least` function with collation gets a wrong result #31789
  - Fix a bug that the date_add and date_sub functions might return incorrect data types #31809
  - Fix possible panic when inserting data to virtually generated columns using an insert statement #31735
  - Fix a bug that no error is reported when duplicate columns are present in the created list partition #31784
  - Fix wrong results returned when `select for update union select` uses incorrect snapshots #31530
  - Fix the potential issue that Regions might be unevenly distributed after a restore operation is finished #31034
  - Fix a bug that COERCIBILITY is wrong for the `json` type #31541
  - Fix wrong collation of the `json` type when this type is processed using builtin-func #31320
  - Fix a bug that PD rules are not deleted when the count of TiFlash replicas is set to 0 #32190
  - Fix the issue that `alter column set default` wrongly updates the table schema #31074
  - Fix the issue that `date_format` in TiDB handles `'\n'` in a MySQL-incompatible way #32232

- Fix a bug that errors may occur when updating partitioned tables using join #31629
- Fix wrong range calculation results for Nulleq function on Enum values #32428
- Fix possible panic in `upper()` and `lower()` functions #32488
- Fix time zone problems encountered when changing the other type columns to timestamp type columns #29585
- Fix TiDB OOM when exporting data using ChunkRPC #31981 #30880
- Fix a bug that sub SELECT LIMIT does not work as expected in dynamic partition pruning mode #32516
- Fix wrong or inconsistent format of bit default value in the `INFORMATION_SCHEMA` ↪ `.COLUMNS` table #32655
- Fix a bug that partition table pruning might not work for listing partition tables after server restart #32416
- Fix a bug that `add column` might use a wrong default timestamp after `SET` ↪ `timestamp` is executed #31968
- Fix a bug that connecting to a TiDB passwordless account from MySQL 5.5 or 5.6 client may fail #32334
- Fix wrong results when reading partitioned tables in dynamic mode in transactions #29851
- Fix a bug that TiDB may dispatch duplicate tasks to TiFlash #32814
- Fix wrong results returned when the input of the `timdiff` function contains a millisecond #31680
- Fix wrong results when explicitly reading partitions and using the IndexJoin plan #32007
- Fix a bug that renaming a column fails when changing its column type concurrently #31075
- Fix a bug that the formula for calculating net cost for TiFlash plans is not aligned with TiKV plans #30103
- Fix a bug that `KILL TIDB` cannot take effect immediately on idle connections #24031
- Fix possible wrong results when querying a table with generated columns #33038
- Fix wrong results of deleting data of multiple tables using `left join` #31321
- Fix a bug that the `SUBTIME` function returns a wrong result in case of overflow #31868
- Fix a bug that the `selection` operator cannot be pushed down when an aggregation query contains the `having` condition #33166
- Fix a bug that CTE might be blocked when a query reports errors #31302
- Fix a bug that excessive length of varbinary or varchar columns when creating tables in non-strict mode might result in errors #30328
- Fix the wrong number of followers in `information_schema.placement_policies` when no follower is specified #31702
- Fix the issue that TiDB allows to specify column prefix length as 0 when an index is created #31972
- Fix the issue that TiDB allows partition names ending with spaces #31535
- Correct the error message of the `RENAME TABLE` statement #29893

- TiKV
    - Fix the panic issue caused by deleting snapshot files when the peer status is `Applying` #11746
    - Fix the issue of QPS drop when flow control is enabled and `level0_slowdown_trigger` is set explicitly #11424
    - Fix the issue that destroying a peer might cause high latency #10210
    - Fix a bug that TiKV cannot delete a range of data (which means the internal command `unsafe_destroy_range` is executed) when the GC worker is busy #11903
    - Fix a bug that TiKV panics when the data in `StoreMeta` is accidentally deleted in some corner cases #11852
    - Fix a bug that TiKV panics when performing profiling on an ARM platform #10658
    - Fix a bug that TiKV might panic if it has been running for 2 years or more #11940
    - Fix the compilation issue on the ARM64 architecture caused by missing SSE instruction set #12034
    - Fix the issue that deleting an uninitialized replica might cause an old replica to be recreated #10533
    - Fix a bug that stale messages cause TiKV to panic #12023
    - Fix the issue that undefined behavior (UB) might occur in TsSet conversions #12070
    - Fix a bug that replica reads might violate the linearizability #12109
    - Fix the potential panic issue that occurs when TiKV performs profiling on Ubuntu 18.04 #9765
    - Fix the issue that tikv-ctl returns an incorrect result due to its wrong string match #12329
    - Fix the issue of intermittent packet loss and out of memory (OOM) caused by the overflow of memory metrics #12160
    - Fix the potential issue of mistakenly reporting TiKV panics when exiting TiKV #12231
- PD
    - Fix the issue that PD generates the operator with meaningless steps of Joint Consensus #4362
    - Fix a bug that the TSO revoking process might get stuck when closing the PD client #4549
    - Fix the issue that the Region scatterer scheduling lost some peers #4565
    - Fix the issue that `Duration` fields of `dr-autosync` cannot be dynamically configured #4651
- TiFlash
    - Fix the TiFlash panic issue that occurs when the memory limit is enabled #3902

- Fix the issue that expired data is recycled slowly #4146
- Fix the potential issue of TiFlash panic when `Snapshot` is applied simultaneously with multiple DDL operations #4072
- Fix the potential query error after adding columns under heavy read workload #3967
- Fix the issue that the `SQRT` function with a negative argument returns `NaN` instead of `Null` #3598
- Fix the issue that casting `INT` to `DECIMAL` might cause overflow #3920
- Fix the issue that the result of `IN` is incorrect in multi-value expressions #4016
- Fix the issue that the date format identifies `'\n'` as an invalid separator #4036
- Fix the issue that the learner-read process takes too much time under high concurrency scenarios #3555
- Fix the wrong result that occurs when casting `DATETIME` to `DECIMAL` #4151
- Fix the issue of memory leak that occurs when a query is canceled #4098
- Fix a bug that enabling the elastic thread pool might cause a memory leak #4098
- Fix a bug that canceled MPP queries might cause tasks to hang forever when the local tunnel is enabled #4229
- Fix a bug that the failure of the HashJoin build side might cause MPP queries to hang forever #4195
- Fix a bug that MPP tasks might leak threads forever #4238

- Tools

    - Backup & Restore (BR)

        - Fix a bug that BR gets stuck when the restore operation meets some unrecoverable errors #33200
        - Fix a bug that causes the restore operation to fail when the encryption information is lost during backup retry #32423

    - TiCDC

        - Fix a bug that MySQL sink generates duplicated `replace` SQL statements when `batch-replace-enable` is disabled #4501
        - Fix a bug that a TiCDC node exits abnormally when a PD leader is killed #4248
        - Fix the error `Unknown system variable 'transaction_isolation'` for some MySQL versions #4504
        - Fix the TiCDC panic issue that might occur when `Canal-JSON` incorrectly handles `string` #4635
        - Fix a bug that sequence is incorrectly replicated in some cases #4563
        - Fix the TiCDC panic issue that might occur because `Canal-JSON` does not support nil #4736
        - Fix the wrong data mapping for avro codec of type `Enum/Set` and `TinyText` ↪ `/MediumText/Text/LongText` #4454
        - Fix a bug that Avro converts a `NOT NULL` column to a nullable field #4818
        - Fix an issue that TiCDC cannot exit #4699

- TiDB Data Migration (DM)
  - Fix the issue that syncer metrics are updated only when querying the status #4281
  - Fix the issue that execution errors of the update statement in safemode may cause the DM-worker panic #4317
  - Fix a bug that long varchars report an error `Column length too big` #4637
  - Fix the conflict issue caused by multiple DM-workers writing data from the same upstream #3737
  - Fix the issue that hundreds of "checkpoint has no change, skip sync flush checkpoint" print in the log and the replication is very slow #4619
  - Fix the DML loss issue when merging shards and replicating incremental data from upstream in the pessimistic mode #5002
- TiDB Lightning
  - Fix a bug that TiDB Lightning may not delete the metadata schema when some import tasks do not contain source files #28144
  - Fix the panic that occurs when the table names in the source file and in the target cluster are different #31771
  - Fix the checksum error "GC life time is shorter than transaction duration" #32733
  - Fix the issue that TiDB Lightning gets stuck when it fails to check empty tables #31797
- Dumpling
  - Fix the issue that the displayed progress is not accurate when running `dumpling --sql $query` #30532
  - Fix the issue that Amazon S3 cannot correctly calculate the size of compressed data #30534
- TiDB Binlog
  - Fix the issue that TiDB Binlog might be skipped when large upstream write transactions are replicated to Kafka #1136

16.7 v5.4

16.7.1 TiDB 5.4.3 Release Notes

Release date: October 13, 2022

TiDB version: 5.4.3

16.7.1.1 Improvements

- TiKV

- Support configuring the RocksDB write stall settings to a value smaller than the flow control threshold #13467
- Support configuring the `unreachable_backoff` item to avoid Raftstore broadcasting too many messages after one peer becomes unreachable #13054

- Tools

  - TiDB Lightning

    - Optimize Scatter Region to batch mode to improve the stability of the Scatter Region process #33618

  - TiCDC

    - Reduce performance overhead caused by runtime context switching in multi-Region scenarios #5610

16.7.1.2 Bug fixes

- TiDB

  - Fix the incorrect output of `SHOW CREATE PLACEMENT POLICY` #37526
  - Fix the issue that some DDL statements might be stuck for a period after the PD node of a cluster is replaced #33908
  - Fix the issue that `KILL TIDB` cannot take effect immediately on idle connections #24031
  - Fix the issue that incorrect results are returned in the `DATA_TYPE` and `COLUMN_TYPE` columns when querying the `INFORMSTION_SCHEMA.COLUMNS` system table #36496
  - Fix the issue that when TiDB Binlog is enabled, executing the `ALTER SEQUENCE` ↪ statement might cause a wrong metadata version and cause Drainer to exit #36276
  - Fix the issue that the `UNION` operator might return unexpected empty result #36903
  - Fix the wrong result that occurs when enabling dynamic mode in partitioned tables for TiFlash #37254
  - Fix the issue that `INL_HASH_JOIN` might hang when used with `LIMIT` #35638
  - Fix the issue that TiDB might return the `invalid memory address or nil` ↪ `pointer dereference` error when executing the `SHOW WARNINGS` statement #31569
  - Fix the `invalid transaction` error that occurs when performing Stale Read in the RC isolation level #30872
  - Fix the issue that the `EXPLAIN ANALYZE` statement with DML executors might return result before the transaction commit finishes #37373
  - Fix the issue of the `data and columnID count not match` error that occurs when inserting duplicated values with TiDB Binlog enabled #33608

- Fix the issue that in the static partition prune mode, SQL statements with an aggregate condition might return wrong result when the table is empty #35295
- Fix the issue that TiDB might panic when executing the `UPDATE` statement #32311
- Fix the issue of wrong query result because the `UnionScan` operator cannot maintain the order #33175
- Fix the issue that the UPDATE statements incorrectly eliminate the projection in some cases, which causes the `Can't find column` error #37568
- Fix the issue that partitioned tables cannot fully use indexes to scan data in some cases #33966
- Fix the issue that the `EXECUTE` might throw an unexpected error in specific scenarios #37187
- Fix the issue that TiDB might return wrong results when using a `BIT` type index with prepared plan cache enabled #33067

- TiKV

  - Fix the issue of continuous SQL execution errors in the cluster after the PD leader is switched or PD is restarted #12934
    - Cause: This issue is caused by a TiKV bug that TiKV does not retry sending heartbeat information to PD client after heartbeat requests fail, until TiKV reconnects to PD client. As a result, the Region information on the failed TiKV node becomes outdated, and TiDB cannot get the latest Region information, which causes SQL execution errors.
    - Affected versions: v5.3.2 and v5.4.2. This issue has been fixed in v5.3.3 and v5.4.3. If you are using v5.4.2, you can upgrade your cluster to v5.4.3.
    - Workaround: In addition to upgrade, you can also restart the TiKV nodes that cannot send Region heartbeat to PD, until there is no Region heartbeat to send.
  - Fix the issue that causes permission denied error when TiKV gets an error from the web identity provider and fails back to the default provider #13122
  - Fix the issue that the PD client might cause deadlocks #13191
  - Fix the issue that Regions might be overlapped if Raftstore is busy #13160

- PD

  - Fix the issue that PD cannot correctly handle dashboard proxy requests #5321
  - Fix the issue that a removed tombstone store appears again after the PD leader transfer #4941
  - Fix the issue that the TiFlash learner replica might not be created #5401

- TiFlash

  - Fix the issue that the `format` function might return a `Data truncated` error #4891

- Fix the issue that TiFlash might crash due to an error in parallel aggregation #5356
- Fix the panic that occurs after creating the primary index with a column containing the `NULL` value #5859

- Tools

  - TiDB Lightning

    - Fix the issue that an auto-increment column of the `BIGINT` type might be out of range #27397
    - Fix the issue that de-duplication might cause TiDB Lightning to panic in extreme cases #34163
    - Fix the issue that TiDB Lightning does not support columns starting with slash, number, or non-ascii characters in Parquet files #36980
    - Fix the issue that TiDB Lightning fails to connect to TiDB when TiDB uses an IPv6 host #35880

  - TiDB Data Migration (DM)

    - Fix the issue that DM Worker might get stuck when getting DB Conn #3733
    - Fix the issue that DM reports the `Specified key was too long` error #5315
    - Fix the issue that latin1 data might be corrupted during replication #7028
    - Fix the issue that DM fails to start when TiDB uses an IPv6 host #6249
    - Fix the issue of possible data race in `query-status` #4811
    - Fix goroutine leak when relay meets an error #6193

  - TiCDC

    - Fix the TiCDC panic issue when you set `enable-old-value = false` #6198

  - Backup & Restore (BR)

    - Fix the issue that might lead to backup and restoration failure if special characters exist in the authorization key of external storage #37469
    - Fix the issue that the regions are not balanced because the concurrency is set too large during the restoration #37549

  - Dumpling

    - Fix the issue that GetDSN does not support IPv6 #36112

16.7.2 TiDB 5.4.2 Release Notes

Release Date: July 8, 2022

TiDB version: 5.4.2

Warning:

It is not recommended to use v5.4.2, because this version has a known bug. For details, see #12934. This bug has been fixed in v5.4.3. It is recommended to use v5.4.3.

16.7.2.1 Improvements

- TiDB
  - Avoid sending requests to unhealthy TiKV nodes to improve availability #34906
- TiKV
  - Reload TLS certificate automatically for each update to improve availability #12546
  - Improve the health check to detect unavailable Raftstore, so that the TiKV client can update Region Cache in time #12398
  - Transfer the leadership to CDC observer to reduce latency jitter #12111
- PD
  - Disable compiling swagger server by default #4932
- Tools
  - TiDB Lightning
    - Optimize Scatter Region to batch mode to improve the stability of the Scatter Region process #33618

16.7.2.2 Bug Fixes

- TiDB
  - Fix the issue of wrong TableDual plans cached in binary protocol #34690 #34678
  - Fix the issue of incorrectly inferred null flag of the TiFlash `firstrow` aggregate function in the EqualAll case #34584
  - Fix the issue that the planner generates wrong 2-phase aggregate plan for TiFlash #34682
  - Fix the planner wrong behaviors that occur when `tidb_opt_agg_push_down` and `tidb_enforce_mpp` are enabled #34465

- Fix the wrong memory-usage value used when Plan Cache is evicted #34613
- Fix the issue that the column list does not work in the `LOAD DATA` statement #35198
- Avoid reporting `WriteConflict` errors in pessimistic transactions #11612
- Fix the issue that the prewrite requests are not idempotency when Region errors and network issues occur #34875
- Fix the issue that the async commit transactions being rolled back might not meet atomicity #33641
- Previously, when a network connectivity issue occurred, TiDB did not always correctly free the resources held by the disconnected session. This issue has been fixed so that open transactions can be rolled back and other associated resources can be released. #34722
- Fix the issue that the `references invalid table` error might be incorrectly reported when TiDB queries views with CTE #33965
- Fix the panic issue caused by the `fatal error: concurrent map read and map write` error #35340

- TiKV

  - Fix the panic issue caused by analyzing statistics when `max_sample_size` is set to `0` #11192
  - Fix the potential issue of mistakenly reporting TiKV panics when exiting TiKV #12231
  - Fix the panic issue that might occur when the source peer catches up logs by snapshot in the Region merge process #12663
  - Fix the panic issue that might occur when a peer is being split and destroyed at the same time #12825
  - Fix the issue of frequent PD client reconnection that occurs when the PD client meets an error #12345
  - Fix the issue of time parsing error that occurs when the `DATETIME` values contain a fraction and `Z` #12739
  - Fix the issue that TiKV panics when performing type conversion for an empty string #12673
  - Fix the possible duplicate commit records in pessimistic transactions when async commit is enabled #12615
  - Fix the issue that TiKV reports the `invalid store ID 0` error when using Follower Read #12478
  - Fix the issue of TiKV panic caused by the race between destroying peers and batch splitting Regions #12368
  - Fix the issue that tikv-ctl returns an incorrect result due to its wrong string match #12329
  - Fix the issue of failing to start TiKV on AUFS #12543

- PD

  - Fix the wrong status code of `not leader` #4797

- Fix the PD panic that occurs when a hot region has no leader #5005
- Fix the issue that scheduling cannot start immediately after the PD leader transfer #4769
- Fix a bug of TSO fallback in some corner cases #4884

- TiFlash
  - Fix the issue that TiFlash crashes after dropping a column of a table with clustered indexes in some situations #5154
  - Fix potential data inconsistency after a lot of INSERT and DELETE operations #4956
  - Fix wrong decimal comparison results in corner cases #4512

- Tools
  - Backup & Restore (BR)
    - Fix a bug that BR reports `ErrRestoreTableIDMismatch` in RawKV mode #35279
    - Fix a bug that BR does not retry when an error occurs in saving files #34865
    - Fix a panic issue when BR is running #34956
    - Fix the issue that BR cannot handle S3 internal errors #34350
    - Fix a bug that BR gets stuck when the restore operation meets some unrecoverable errors #33200
  - TiCDC
    - Fix data loss that occurs in special incremental scanning scenarios #5468
    - Fix a bug that the redo log manager flushes logs before writing logs #5486
    - Fix a bug that the resolved ts moves too fast when some tables are not maintained by the redo writer #5486
    - Fix the issue that file name conflicts may cause data loss #5486
    - Fix replication interruption that occurs when Region leader is missing and the retry exceeds the limit #5230
    - Fix the bug that MySQL Sink may save a wrong checkpointTs #5107
    - Fix a bug that may cause goroutine leak in the HTTP server #5303
    - Fix the issue that changes in meta Region can lead to latency increase #4756 #4762
  - TiDB Data Migration (DM)
    - Fix the issue that DM occupies more disk space after a task automatically resumes #5344
    - Fix the issue that the uppercase table cannot be replicated when `case-sensitive: true` is not set #5255

16.7.3 TiDB 5.4.1 Release Notes

Release Date: May 13, 2022

TiDB version: 5.4.1

16.7.3.1 Compatibility changes

TiDB v5.4.1 does not introduce any compatibility changes in product design. But note that bug fixes in this release might result in compatibility changes, too. For more information, see Bug Fixes.

16.7.3.2 Improvements

- TiDB
  - Support using the PointGet plan for queries that read the `_tidb_rowid` column #31543
  - Add more logs and metrics for the `Apply` operator to show whether it is parallel #33887
  - Improve the `TopN` pruning logic for Analyze Version 2 used for collecting statistics #34256
  - Support displaying multiple Kubernetes clusters in the Grafana dashboard #32593
- TiKV
  - Support displaying multiple Kubernetes clusters in the Grafana dashboard #12104
- PD
  - Support displaying multiple Kubernetes clusters in the Grafana dashboard #4673
- TiFlash
  - Support displaying multiple Kubernetes clusters in the Grafana dashboard #4129
- Tools
  - TiCDC
    - Support multiple Kubernetes clusters in Grafana dashboards #4665
    - Expose configuration parameters of the Kafka producer to make them configurable in TiCDC #4385
  - TiDB Data Migration (DM)
    - Support Syncer using the working directory of the DM-worker rather than `/tmp` to write internal files, and cleaning the directory after the task is stopped #4107

16.7.3.3 Bug Fixes

- TiDB
    - Fix the issue that `date_format` in TiDB handles `'\n'` in a MySQL-incompatible way #32232
    - Fix the issue that TiDB writes wrong data due to the wrong encoding of the `ENUM` or `SET` column #32302
    - Fix the issue that the Merge Join operator gets wrong results in certain cases #33042
    - Fix the issue that TiDB gets a wrong result when a correlated subquery returns a constant #32089
    - Fix the issue that TiDB gets the wrong result when using TiFlash to scan tables with empty range although TiFlash does not support reading tables with empty range yet #33083
    - Fix the issue that the `MAX` or `MIN` function on the `ENUM` or `SET` column returns a wrong result when the new collation is enabled in TiDB #31638
    - Fix a bug that CTE might be blocked when a query reports errors #31302
    - Fix wrong range calculation results for Nulleq function on Enum values #32428
    - Fix TiDB OOM when exporting data using ChunkRPC #31981 #30880
    - Fix a bug that `tidb_super_read_only` is not automatically enabled when `tidb_restricted_read_only` is enabled #31745
    - Fix the issue that the `greatest` or `least` function with collation gets a wrong result #31789
    - Fix load data panic if the data is broken at an escape character #31589
    - Fix the `invalid transaction` error when executing a query using index lookup join #30468
    - Fix wrong results of deleting data of multiple tables using `left join` #31321
    - Fix a bug that TiDB may dispatch duplicate tasks to TiFlash #32814
    - Fix the issue that granting the `all` privilege might fail in clusters that are upgraded from v4.0 #33588
    - Fix the session panic that occurs when executing the prepared statement after table schema change with the MySQL binary protocol #33509
    - Fix the issue that executing SQL statements that have the `compress()` expression with `tidb_enable_vectorized_expression` enabled will fail #33397
    - Fix the issue of high CPU usage by the `reArrangeFallback` function #30353
    - Fix the issue that the table attributes are not indexed when a new partition is added and the issue that the table range information is not updated when the partition changes #33929
    - Fix a bug that the `TopN` statistical information of a table during the initialization is not correctly sorted #34216
    - Fix the error that occurs when reading from the `INFORMATION_SCHEMA.` ↪ `ATTRIBUTES` table by skipping the unidentifiable table attributes #33665
    - Fix a bug that even if `@@tidb_enable_parallel_apply` is set, the `Apply` operator is not paralleled when an `order` property exists #34237

- Fix a bug that `'0000-00-00 00:00:00'` can be inserted into a `datetime` column when `sql_mode` is set to `NO_ZERO_DATE` #34099
- Fix the issue that the TiDB server might run out of memory when the `INFORMATION_SCHEMA.CLUSTER_SLOW_QUERY` table is queried. This issue can be triggered when you check slow queries on the Grafana dashboard #33893
- Fix a bug that in the `NOWAIT` statement, a transaction being executed does not return immediately when encountering a lock #32754
- Fix a bug that causes a failure when creating a table with the `GBK` charset and `gbk_bin` collation #31308
- Fix a bug that when `enable-new-charset` is `on`, creating a `GBK` charset table with collation fails with the "Unknown character set" error #31297

- TiKV

  - Fix the issue that TiKV panics and destroys peers unexpectedly because the target Region to be merged is invalid #12232
  - Fix a bug that stale messages cause TiKV to panic #12023
  - Fix the issue of intermittent packet loss and out of memory (OOM) caused by the overflow of memory metrics #12160
  - Fix the potential panic issue that occurs when TiKV performs profiling on Ubuntu 18.04 #9765
  - Fix a bug that replica reads might violate the linearizability #12109
  - Fix the TiKV panic issue that occurs when the target peer is replaced with the peer that is destroyed without being initialized when merging a Region #12048
  - Fix a bug that TiKV might panic if it has been running for 2 years or more #11940
  - Reduce the TiCDC recovery time by reducing the number of the Regions that require the Resolve Locks step #11993
  - Fix the panic issue caused by deleting snapshot files when the peer status is `Applying` #11746
  - Fix the issue that destroying a peer might cause high latency #10210
  - Fix the panic issue caused by invalid assertion in resource metering #12234
  - Fix the issue that slow score calculation is inaccurate in some corner cases #12254
  - Fix the OOM issue caused by the `resolved_ts` module and add more metrics #12159
  - Fix the issue that successfully committed optimistic transactions may report the `Write Conflict` error when the network is poor #34066
  - Fix the TiKV panic issue that occurs when replica read is enabled on a poor network #12046

- PD

  - Fix the issue that `Duration` fields of `dr-autosync` cannot be dynamically configured #4651

    - Fix the issue that when there exists a Store with large capacity (2T for example), fully allocated small Stores cannot be detected, which results in no balance operator being generated #4805
    - Fix the issue that the label distribution has residual labels in the metrics #4825

- TiFlash

    - Fix the panic issue that occurs when TLS is enabled #4196
    - Fix possible metadata corruption caused by Region merge on a lagging Region peer #4437
    - Fix the issue that a query containing `JOIN` might be hung if an error occurs #4195
    - Fix a bug that MPP tasks might leak threads forever #4238
    - Fix the overflow that occurs when casting `FLOAT` to `DECIMAL` #3998
    - Fix the issue that expired data is recycled slowly #4146
    - Fix a bug that canceled MPP queries might cause tasks to hang forever when the local tunnel is enabled #4229
    - Fix the issue of memory leak that occurs when a query is canceled #4098
    - Fix the wrong result that occurs when casting `DATETIME` to `DECIMAL` #4151
    - Fix the potential issue of TiFlash panic when `Snapshot` is applied simultaneously with multiple DDL operations #4072
    - Fix the bug that invalid storage directory configurations lead to unexpected behaviors #4093
    - Fix the bug that some exceptions are not handled properly #4101
    - Fix the issue that casting `INT` to `DECIMAL` might cause overflow #3920
    - Fix the issue that the result of `IN` is incorrect in multi-value expressions #4016
    - Fix the issue that the date format identifies `'\n'` as an invalid separator #4036
    - Fix the potential query error after adding columns under heavy read workload #3967
    - Fix the panic issue that occurs when the memory limit is enabled #3902
    - Fix potential data corruption in DTFiles #4778
    - Fix potential errors when querying on a table with many delete operations #4747
    - Fix a bug that TiFlash reports many "Keepalive watchdog fired" errors randomly #4192
    - Fix a bug that data not matching any region range remains on a TiFlash node #4414
    - Fix a bug that empty segments cannot be merged after GC #4511

- Tools

    - Backup & Restore (BR)

        - Fix a bug that causes the restore operation to fail when the encryption information is lost during backup retry #32423
        - Fix the issue that BR fails to back up RawKV #32607
        - Fix duplicate primary keys when inserting a record into a table after incremental restoration #33596

- Fix a bug that BR incremental restore returns errors mistakenly due to DDL jobs with empty query #33322
- Fix the potential issue that Regions might be unevenly distributed after a restore operation is finished #31034
- Fix the issue that BR does not retry enough times when Regions are not consistent during restoration #33419
- Fix the issue that BR might panic occasionally when merging small files is enabled #33801
- Fix the issue that schedulers do not resume after BR or TiDB Lightning exits abnormally #33546

- TiCDC
  - Fix incorrect metrics caused by owner changes #4774
  - Fix the TiCDC panic issue that might occur because `Canal-JSON` does not support nil #4736
  - Fix a stability problem in workerpool used by Unified Sorter #4447
  - Fix a bug that sequence is incorrectly replicated in some cases #4563
  - Fix the TiCDC panic issue that might occur when `Canal-JSON` incorrectly handles `string` #4635
  - Fix a bug that a TiCDC node exits abnormally when a PD leader is killed #4248
  - Fix a bug that MySQL sink generates duplicated `replace` SQL statements when `batch-replace-enable` is disabled #4501
  - Fix the DML construct error caused by the `rename tables` DDL #5059
  - Fix the issue that in rare cases replication can be stuck if the owner is changed and the new scheduler is enabled (disabled by default) #4963
  - Fix the issue that the error ErrProcessorDuplicateOperations is reported when the new scheduler is enabled (disabled by default) #4769
  - Fix the issue that TiCDC fails to start when the first PD set in `--pd` is not available after TLS is enabled #4777
  - Fix the issue that the checkpoint metrics are missing when tables are being scheduled #4714
- TiDB Lightning
  - Fix the checksum error "GC life time is shorter than transaction duration" #32733
  - Fix the issue that TiDB Lightning gets stuck when it fails to check empty tables #31797
  - Fix a bug that TiDB Lightning may not delete the metadata schema when some import tasks do not contain source files #28144
  - Fix the issue that the precheck does not check local disk resources and cluster availability #34213
- TiDB Data Migration (DM)
  - Fix the issue that hundreds of "checkpoint has no change, skip sync flush checkpoint" print in the log and the replication is very slow #4619

- Fix a bug that long varchars report an error `Column length too big` #4637
- Fix the issue that execution errors of the update statement in safemode may cause the DM-worker panic #4317
- Fix the issue that in some cases manually executing the filtered DDL in the downstream might cause task resumption failure #5272
- Fix a bug that no data is returned for the `query-status` command when binlog is not enabled in the upstream #5121
- Fix the DM worker panic issue that occurs when the primary key is not first in the index returned by the `SHOW CREATE TABLE` statement #5159
- Fix the issue that CPU usage may increase and a large amount of log is printed when GTID is enabled or when the task is automatically resumed #5063

16.7.4 TiDB 5.4 Release Notes

Release date: February 15, 2022

TiDB version: 5.4.0

In v5.4, the key new features or improvements are as follows:

- Support the GBK character set
- Support using Index Merge to access data, which merges the filtering results of indexes on multiple columns
- Support reading stale data using a session variable
- Support persisting the configuration for collecting statistics
- Support using Raft Engine as the log storage engine of TiKV (experimental)
- Optimize the impact of backup on the cluster
- Support using Azure Blob storage as the backup storage
- Continuously improve the stability and performance of TiFlash and the MPP engine
- Add a switch in TiDB Lightning to determine whether to allow importing to an existing table with data
- Optimize the Continuous Profiling feature (experimental)
- TiSpark supports user identification and authentication

16.7.4.1 Compatibility changes

> **Note:**
>
> When upgrading from an earlier TiDB version to v5.4.0, if you want to know the compatibility change notes of all intermediate versions, you can check the Release Notes of the corresponding version.

16.7.4.1.1 System variables

Variable name	Change type	Description
tidb_enable_column_tracking ↪	Newly added	Controls whether to allow TiDB to collect PREDICATE ↪ COLUMNS. The default value is OFF.

Variable name	Change type	Description
tidb_enable_paging ↪	Newly added	Controls whether to use the method of paging to send coprocessor requests in IndexLookUp operator. The default value is OFF. For read queries that use IndexLookup and Limit and that Limit cannot be pushed down to IndexScan, there might be high latency for the read queries and high CPU usage for TiKV's unified ↪ read ↪ pool. In such cases, because the Limit operator only requires a small set of data, if you set tidb_enable_paging ↪ to ON, TiDB processes less data, which reduces query

Variable name	Change type	Description
tidb_enable_top_sql	Newly added	Controls whether to enable the Top SQL feature. The default value is OFF.
tidb_persist_analyze_options	Newly added	Controls whether to enable the ANALYZE configuration persistence feature. The default value is ON.
tidb_read_staleness	Newly added	Controls the range of historical data that can be read in the current session. The default value is 0.
tidb_regard_null_as_point	Newly added	Controls whether the optimizer can use a query condition including null equivalence as a prefix condition for index access.

Variable name	Change type	Description
`tidb_stats_load_sync_wait` ↪	Newly added	Controls whether to enable the synchronously loading statistics feature. The default value `0` means that the feature is disabled and that the statistics is asynchronously loaded. When the feature is enabled, this variable controls the maximum time that SQL optimization can wait for synchronously loading statistics before timeout.

Variable name	Change type	Description
tidb_stats_load_pseudo_timeout ↪	Newly added	Controls when synchronously loading statistics reaches timeout, whether SQL fails (OFF) or falls back to using pseudo statistics (ON). The default value is OFF.
tidb_backoff_lock_fast ↪	Modified	The default value is changed from 100 to 10.

| tidb_enable_index_merge | Modified | The default value is changed from OFF to ON.

If you upgrade a TiDB cluster from versions earlier than v4.0.0 to v5.4.0 or later, this variable is OFF by default.

If you upgrade a TiDB cluster from v4.0.0 or later to v5.4.0 or later, this variable remains the same as before the upgrade.

For the newly created TiDB clusters of v5.4.0 and later, this variable is ON by default.

| | tidb_store_limit | Modified | Before v5.4.0, this variable can be configured at instance level and globally. Starting from v5.4.0, this variable only supports global configuration. |

16.7.4.1.2 Configuration file parameters

Configuration file	Configuration	Change type	Description
TiDB	stats-load-concurrency	Newly added	Controls the maximum number of columns that the TiDB synchronously loading statistics feature can process concurrently. The default value is 5.
TiDB	stats-load-queue-size	Newly added	Controls the maximum number of column requests that the TiDB synchronously loading statistics feature can cache. The default value is1000.
TiKV	snap-generator-pool-size	Newly added	The size of the snap-generator thread pool. The default value is 2.

Configuration file	Configuration	Change type	Description
TiKV	`log.file.` ↪ `max-` ↪ `size`, `log.file.` ↪ `max-` ↪ `days`, `log.file.` ↪ `max-` ↪ `backups`	Newly added	For details, see TiKV Configuration File - `log.file`.

Configuration file	Configuration	Change type	Description
TiKV	`raft-engine`	Newly added	Includes `enable`, `dir`, `batch-compression-threshold`, `bytes-per-sync`, `target-file-size`, `purge-threshold`, `recovery-mode`, `recovery-read-block-size`, `recovery-read-block-size`, and `recovery-threads`. For details, see TiKV Configuration File - `raft-engine`.

Configuration file	Configuration	Change type	Description
TiKV	`backup.` ↪ `enable-` ↪ `auto-` ↪ `tune`	Newly added	In v5.3.0, the default value is `false`. Since v5.4.0, the default value is changed to `true`. This parameter controls whether to limit the resources used by backup tasks to reduce the impact on the cluster when the cluster resource utilization is high. In the default configuration, the speed of backup tasks might slow down.

Configuration file	Configuration	Change type	Description
TiKV	`log-level`, `log-format`, `log-file`, `log-rotation-size`	Modified	The names of TiKV log parameters are replaced with the names that are consistent with TiDB log parameters, which are `log.level`, `log.format`, `log.file.filename`, and `log.enable-timestamp`. If you only set the old parameters, and their values are set to non-default values, the old parameters remain compatible with the new parameters. If both old and new parameters are set, the new parameters take effect. For details, see TiKV Configuration File -

Configuration file	Configuration	Change type	Description
TiKV	`log-` ↪ `rotation` ↪ `-` ↪ `timespan` ↪	Deleted	The timespan between log rotations. After this timespan passes, a log file is rotated, which means a timestamp is appended to the file name of the current log file, and a new log file is created.
TiKV	`allow-` ↪ `remove-` ↪ `leader`	Deleted	Determines whether to allow deleting the main switch.
TiKV	`raft-msg-` ↪ `flush-` ↪ `interval` ↪	Deleted	Determines the interval at which Raft messages are sent in batches. The Raft messages are sent in batches at every interval specified by this configuration item.

Configuration file	Configuration	Change type	Description
PD	`log.level`	Modified	The default value is changed from "INFO" to "info", guaranteed to be case-insensitive.
TiFlash	`profile.default.enable_elastic_threadpool`	Newly added	Determines whether to enable or disable the elastic thread pool function. Enabling this configuration item can significantly improve TiFlash CPU utilization in high concurrency scenarios. The default value is `false`.

Configuration file	Configuration	Change type	Description
TiFlash	`storage.format_version`	Newly added	Specifies the version of DTFile. The default value is 2, under which hashes are embedded in the data file. You can also set the value to 3. When it is 3, the data file contains metadata and token data checksum, and supports multiple hash algorithms.
TiFlash	`logger.count`	Modified	The default value is changed to 10.
TiFlash	`status.metrics_port`	Modified	The default value is changed to 8234.
TiFlash	`raftstore.apply-pool-size`	Newly added	The allowable number of threads in the pool that flushes Raft data to storage. The default value is 4.

Configuration file	Configuration	Change type	Description
TiFlash	raftstore. ↪ store- ↪ pool- ↪ size	Newly added	The allowable number of threads that process Raft, which is the size of the Raftstore thread pool. The default value is 4.
TiDB Data Migration (DM)	collation_compatible ↪	Newly added	The mode to sync the default collation in CREATE SQL statements. The value options are "loose" (by default) and "strict".
TiCDC	max- ↪ message ↪ -bytes	Modified	Change the default value of max- ↪ message ↪ -bytes in Kafka sink to 104857601 (10MB)

Configuration file	Configuration	Change type	Description
TiCDC	`partition-num`	Modified	Change the default value of `partition-num` in Kafka Sink from `4` to `3`. It makes TiCDC send messages to Kafaka partitions more evenly.
TiDB Lightning	`meta-schema-name`	Modified	Specifies the schema name for the metadata in the target TiDB. From v5.4.0, this schema is created only if you have enabled parallel import (the corresponding parameter is `tikv-importer.incremental-import = true`).

Configuration file	Configuration	Change type	Description
TiDB Lightning	`task-info-schema-name`	Newly added	Specifies the name of the database where duplicated data is stored when TiDB Lightning detects conflicts. By default, the value is "lightning_task_info". Specify this parameter only if you have enabled the "duplicate-resolution" feature.
TiDB Lightning	`incremental-import`	Newly added	Determines whether to allow importing data to tables where data already exists. The default value is `false`.

16.7.4.1.3 Others

- An interface is added between TiDB and PD. When you use the `information_schema.TIDB_HOT_REGIONS_HISTORY` system table, TiDB needs to use PD in the corresponding version.
- TiDB Server, PD Server, and TiKV Server start using a unified naming method for the log-related parameters to manage log names, output formats, and the rules for rotation and expiration. For details, see TiKV configuration file - log.

- Since v5.4.0, if you create a SQL binding for an execution plan that has been cached via Plan Cache, the binding invalidates the plan already cached for the corresponding query. The new binding does not affect execution plans cached before v5.4.0.
- In v5.3 and earlier versions, TiDB Data Migration (DM) documentation is independent of TiDB documentation. Since v5.4, DM documentation is integrated into TiDB documentation with the same version. You can directly read DM documentation without accessing the DM documentation site.
- Remove the experimental feature of Point-in-time recovery (PITR) along with cdclog. Since v5.4.0, cdclog-based PITR and cdclog are no longer supported.
- Make the behavior of setting system variables to the "DEFAULT" more MySQL-compatible #29680
- Set the system variable `lc_time_names` to read-only #30084
- Set the scope of `tidb_store_limit` from INSTANCE or GLOBAL to GLOBAL #30756
- Forbid converting the integer type column to the time type column when the column contains zero #25728
- Fix the issue that no error is reported for the `Inf` or `NAN` value when inserting floating-point values #30148
- Fix the issue that the `REPLACE` statement incorrectly changes other rows when the auto ID is out of range #30301

16.7.4.2 New features

16.7.4.2.1 SQL

- **TiDB supports the GBK character set since v5.4.0**

  Before v5.4.0, TiDB supports `ascii`, `binary`, `latin1`, `utf8`, and `utf8mb4` character sets.

  To better support Chinese users, TiDB supports the GBK character set since v5.4.0. After enabling the `new_collations_enabled_on_first_bootstrap` option in the TiDB configuration file when initializing a TiDB cluster for the first time, the TiDB GBK character set supports both `gbk_bin` and `gbk_chinese_ci` collations.

  When using the GBK character set, you need to pay attention to the compatibility restrictions. For details, see Character Set and Collation - GBK.

16.7.4.2.2 Security

- **TiSpark supports user authentication and authorization**

  Since TiSpark 2.5.0, TiSpark supports both database user authentication and read-/write authorization at a database or table level. After enabling this feature, you can prevent the business from running unauthorized batch tasks such as draws to obtain data, which improves the stability and data security of online clusters.

This feature is disabled by default. When it is enabled, if a user operating through TiSpark does not have the needed permissions, the user gets an exception from TiSpark.

User document

- **TiUP supports generating an initial password for the root user**

  An `--init` parameter is introduced to the command for starting a cluster. With this parameter, in a TiDB cluster deployed using TiUP, TiUP generates an initial strong password for the database root user. This avoids security risks in using a root user with an empty password and ensures the security of databases.

  User document

16.7.4.2.3 Performance

- **Continue improving the stability and performance of the columnar storage engine TiFlash and the computing engine MPP**

  - Support pusing down more functions to the MPP engine:
    - String functions: `LPAD()`, `RPAD()`, `STRCMP()`
    - Date functions: `ADDDATE(string, real)`, `DATE_ADD(string, real)`, `DATE_SUB(string, real)`, `SUBDATE(string, real)`, `QUARTER()`
  - Introduce the elastic thread pool feature to improve resource utilization (experimental)
  - Improve the efficiency of converting data from row-based storage format to column-based storage format when replicating data from TiKV, which brings 50% improvement in the overall performance of data replication
  - Improve TiFlash performance and stability by tuning the default values of some configuration items. In an HTAP hybrid load, the performance of simple queries on a single table improves up to 20%.

  User documents: Supported push-down calculations, Configure the tiflash.toml file

- **Read historical data within a specified time range through a session variable**

  TiDB is a multi-replica distributed database based on the Raft consensus algorithm. In face of high-concurrency and high-throughput application scenarios, TiDB can scale out its read performance through follower replicas, separating read and write requests.

  For different application scenarios, TiDB provides two modes of follower read: strongly consistent read and weakly consistent history read. The strongly consistent read mode is suitable for application scenarios that require real-time data. However, in this mode, because of the data replication latency between leaders and followers and the reduced throughput, the read request might have high latency, especially for geo-distributed deployments.

For the application scenarios that have less strict requirements on real-time data, the history read mode is recommended. This mode can reduce latency and improve throughput. TiDB currently supports reading historical data by the following methods: Use SQL statements to read data from a time point in the past, or start a read-only transaction based on a time point in the past. Both methods support reading the historical data of a specific point in time or within a specified time range. For details, refer to Read Historical Data Using the `AS OF TIMESTAMP` Clause.

Since v5.4.0, TiDB improves the usability of the history read mode by supporting reading historical data within a specified time range through a session variable. This mode serves low-latency, high-throughput read requests in quasi-real-time scenarios. You can set the variable as follows:

```
set @@tidb_replica_read=leader_and_follower
set @@tidb_read_staleness="-5"
```

By this setting, TiDB can select the nearest leader or follower node and read the latest historical data within 5 seconds.

User document

- **GA for Index Merge**

  Index Merge is introduced in TiDB v4.0 as an experimental feature for SQL optimization. This method greatly accelerates condition filtering when a query requires scanning of multiple columns of data. Take the following query as an example. In the `WHERE` statement, if the filtering conditions connected by `OR` have their respective indexes on columns *key1* and *key2*, the Index Merge feature filters the respective indexes at the same time, merges the query results, and returns the merged result.

  ```
  SELECT * FROM table WHERE key1 <= 100 OR key2 = 200;
  ```

  Before TiDB v4.0, a query on a table supports using only one index for filtering at one time. If you want to query multiple columns of data, you can enable Index Merge to get the exact query result in a short time by using the indexes in individual columns. Index Merge avoids unnecessary full table scans and does not require establishing a large number of composite indexes.

  In v5.4.0, Index Merge becomes GA. However, you still need to pay attention to the following restrictions:

  - Index Merge supports only disjunctive normal form (X1 X2 …Xn). That is, this feature only works when filtering conditions in a `WHERE` clause are connected by `OR`.
  - For newly deployed TiDB clusters of v5.4.0 or later, this feature is enabled by default. For v5.4.0 or later TiDB clusters upgraded from earlier versions, this feature inherits the setting before the upgrade and you can change the setting as required (for TiDB clusters earlier than v4.0, this feature does not exist and is disabled by default).

User document

- **Support Raft Engine (experimental)**

  Support using Raft Engine as the log storage engine in TiKV. Compared with RocksDB, Raft Engine can reduce TiKV I/O write traffic by up to 40% and CPU usage by 10%, while improving foreground throughput by about 5% and reducing tail latency by 20% under certain loads. In addition, Raft Engine improves the efficiency of log recycling and fixes the issue of log accumulation in extreme conditions.

  Raft Engine is still an experimental feature and is disabled by default. Note that the data format of Raft Engine in v5.4.0 is not compatible with previous versions. Before upgrading the cluster, you need to make sure that Raft Engine on all TiKV nodes is disabled. It is recommended to use Raft Engine only in v5.4.0 or a later version.

  User document

- **Support collecting statistics on `PREDICATE COLUMNS` (experimental)**

  In most cases, when executing SQL statements, the optimizer only uses statistics of some columns (such as columns in the `WHERE`, `JOIN`, `ORDER BY`, and `GROUP BY` statements). These used columns are called `PREDICATE COLUMNS`.

  Since v5.4.0, you can set the value of the `tidb_enable_column_tracking` system variable to `ON` to enable TiDB to collect `PREDICATE COLUMNS`.

  After the setting, TiDB writes the `PREDICATE COLUMNS` information to the `mysql.column_stats_usage` system table every 100 * `stats-lease`. When the query pattern of your business is stable, you can use the `ANALYZE TABLE TableName PREDICATE COLUMNS` syntax to collect statistics on the `PREDICATE COLUMNS` columns only, which can greatly reduce the overhead of collecting statistics.

  User document

- **Support synchronously loading statistics (experimental)**

  Since v5.4.0, TiDB introduces the synchronously loading statistics feature. This feature is disabled by default. After enabling the feature, TiDB can synchronously load large-sized statistics (such as histograms, TopN, and Count-Min Sketch statistics) into memory when you execute SQL statements, which improves the completeness of statistics for SQL optimization.

  User document

16.7.4.2.4 Stability

- **Support persisting ANALYZE configurations**

  Statistics are one type of the basic information that the optimizer refers to when generating execution plans. The accuracy of the statistics directly affects whether the generated execution plans are reasonable. To ensure the accuracy of the statistics,

sometimes it is necessary to set different collection configurations for different tables, partitions, and indexes.

Since v5.4.0, TiDB supports persisting some `ANALYZE` configurations. With this feature, the existing configurations can be easily reused for future statistics collection.

The `ANALYZE` configuration persistence feature is enabled by default (the system variable `tidb_analyze_version` is `2` and `tidb_persist_analyze_options` is `ON` by default). You can use this feature to record the persistence configurations specified in the `ANALYZE` statement when executing the statement manually. Once recorded, the next time TiDB automatically updates statistics or you manually collect statistics without specifying these configurations, TiDB will collect statistics according to the recorded configurations.

User document

16.7.4.2.5 High availability and disaster recovery

- **Reduce the impact of backup tasks on the cluster**

  Backup & Restore (BR) introduces the auto-tune feature (enabled by default). This feature can reduce the impact of backup tasks on the cluster by monitoring the cluster resource usage and adjusting the number of threads used by the backup tasks. In some cases, if you increase the cluster hardware resource for backup and enable the auto-tune feature, it can limit the impact of backup tasks on the cluster to 10% or less.

User document

- **Support Azure Blob Storage as a target storage for backup**

Backup & Restore (BR) supports Azure Blob Storage as a remote backup storage. If you deploy TiDB in Azure Cloud, now you can back up the cluster data to the Azure Blob Storage service.

```
[User document](#back-up-and-restore-data-on-azure-blob-storage-using-br)
```

16.7.4.2.6 Data migration

- **TiDB Lightning introduces a new feature to determine whether to allow importing data to tables with data**

  TiDB Lightning introduces a configuration item`incremental-import`. It determines whether to allow importing data to tables with data. The default value is `false`. When using the parallel import mode, you must set the configuration to `true`.

  User document

- **TiDB Lightning introduces the schema name that stores the meta information for parallel import**

  TiDB Lightning introduces the `meta-schema-name` configuration item. In parallel import mode, this parameter specifies the schema name that stores the meta information for each TiDB Lightning instance in the target cluster. By default, the value is "lightning_metadata". The value set for this parameter must be the same for each TiDB Lightning instance that participates in the same parallel import; otherwise, the correctness of the imported data cannot be ensured.

  User document

- **TiDB Lightning introduces duplicate resolution**

  In Local-backend mode, TiDB Lightning outputs duplicated data before the data import is completed, and then removes that duplicated data from the database. You can resolve the duplicated data after the import is completed and select suitable data to insert according to application rules. It is recommended to clean upstream data sources based on duplicated data to avoid data inconsistency caused by duplicated data encountered in the subsequent incremental data migration phase.

  User document

- **Optimize the usage of relay log in TiDB Data Migration (DM)**

  - Recover the `enable-relay` switch in the `source` configuration.
  - Support dynamically enabling and disabling relay log using the `start-relay` and `stop-relay` commands.
  - Bind the status of relay log to `source`. `source` keeps its original status of being enabled or disabled after it is migrated to any DM-worker.
  - Move the storage path of relay log to the DM-worker configuration file.

  User document

- **Optimize the processing of collation in DM**

  Add the `collation_compatible` configuration item. The value options are `loose` (default) and `strict`:

  - If your application does not have strict requirements on collation, and the collation of query results can be different between the upstream and downstream, you can use the default `loose` mode to avoid reporting errors.
  - If your application has strict requirements on collation, and the collation must be consistent between the upstream and downstream, you can use the `strict` mode. However, if the downstream does not support the upstream's default collation, the data replication might report errors.

User document

- **Optimize `transfer source` in DM to support running replication tasks smoothly**

  When the DM-worker nodes have an unbalanced load, the `transfer source` command can be used to manually transfer the configuration of a `source` to another load. After the optimization, the `transfer source` command simplifies the manual operation. You can smoothly transfer the source instead of pausing all related tasks, because DM completes other operations internally.

- **DM OpenAPI becomes generally available (GA)**

  DM supports daily management via API, including adding data sources and managing tasks. In v5.4.0, DM OpenAPI becomes GA.

  User document

16.7.4.2.7 Diagnostic efficiency

- **Top SQL (experimental feature)**

  A new experimental feature, Top SQL (disabled by default), is introduced to help you easily find source-consuming queries.

  User document

16.7.4.2.8 TiDB data share subscription

- **Optimize the impact of TiCDC on clusters**

  Significantly reduces the impact on the performance of TiDB clusters when you use TiCDC. In the test environment, the performance impact of TiCDC on TiDB can be reduced to less than 5%.

16.7.4.2.9 Deployment and maintenance

- **Enhance Continuous Profiling (experimental)**

  - More components supported: Besides TiDB, PD, and TiKV, TiDB v5.4.0 also supports CPU profiling of TiFlash.
  - More forms of profiling display: Supports showing CPU profiling and Goroutine results on flame charts.
  - More deployment environments supported: Continuous Profiling can also be used for clusters deployed using TiDB Operator.

Continuous Profiling is disabled by default and can be enabled on TiDB Dashboard.

```
Continuous Profiling is applicable to clusters deployed or upgraded using
    ↪ TiUP of v1.9.0 or later or TiDB Operator of v1.3.0 or later.

[User document](#tidb-dashboard-instance-profiling---continuous-profiling)
```

16.7.4.3 Improvements

- TiDB
    - Support the `ADMIN {SESSION | INSTANCE | GLOBAL} PLAN_CACHE` syntax to clear the cached query plan #30370
- TiKV
    - Coprocessor supports paging API to process requests in a stream-like way #11448
    - Support `read-through-lock` so that read operations do not need to wait for secondary locks to be resolved #11402
    - Add a disk protection mechanism to avoid panic caused by disk space drainage #10537
    - Support archiving and rotating logs #11651
    - Reduce the system call by the Raft client and increase CPU efficiency #11309
    - Coprocessor supports pushing down substring to TiKV #11495
    - Improve the scan performance by skip reading locks in the Read Committed isolation level #11485
    - Reduce the default thread pool size used by backup operations and limit the use of thread pool when the stress is high #11000
    - Support dynamically adjusting the sizes of the Apply thread pool and the Store thread pool #11159
    - Support configuring the size of the `snap-generator` thread pool #11247
    - Optimize the issue of global lock race that occurs when there are many files with frequent reads and writes #250
- PD
    - Record the historic hotspot information by default #25281
    - Add signature for the HTTP component to identify the request source #4490
    - Update TiDB Dashboard to v2021.12.31 #4257
- TiFlash
    - Optimize the communication of local operators
    - Increase the non-temporary thread count of gRPC to avoid the frequent creation or destruction of threads

- Tools
    - Backup & Restore (BR)
        - Add a validity check for the key when BR performs encrypted backup #29794
    - TiCDC
        - Reduce the count of "EventFeed retry rate limited" logs #4006
        - Reduce the replication latency when replicating many tables #3900
        - Reduce the time for the KV client to recover when a TiKV store is down #3191
    - TiDB Data Migration (DM)
        - Lower the usage rate of CPU when the relay is enabled #2214
    - TiDB Lightning
        - Use optimistic transactions by default to write data to improve performance in TiDB-backend mode #30953
    - Dumpling
        - Improve compatibility when Dumpling checks the database version #29500
        - Add the default collation when dumping `CREATE DATABASE` and `CREATE TABLE` #3420

16.7.4.4 Bug fixes

- TiDB
    - Fix the issue of the `tidb_analyze_version` value change that occurs when upgrading the cluster from v4.x to v5.x #25422
    - Fix the issue of the wrong result that occurs when using different collations in a subquery #30748
    - Fix the issue that the result of `concat(ifnull(time(3))` in TiDB is different from that in MySQL #29498
    - Fix the issue of potential data index inconsistency in optimistic transaction mode #30410
    - Fix the issue that the query execution plan of IndexMerge is wrong when an expression cannot be pushed down to TiKV #30200
    - Fix the issue that concurrent column type change causes inconsistency between the schema and the data #31048
    - Fix the issue that the IndexMerge query result is wrong when there is a subquery #30913
    - Fix the panic issue that occurs when the FetchSize is set too large in the client #30896
    - Fix the issue that LEFT JOIN might be mistakenly converted to INNER JOIN #20510

- Fix the issue that panic might occur when the `CASE-WHEN` expression and collation are used together #30245
- Fix the issue of wrong query result that occurs when the `IN` value contains a binary constant #31261
- Fix the issue of wrong query result that occurs when CTE has a subquery #31255
- Fix the issue that executing the `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` statement gets panic #28078
- Fix the issue that INDEX HASH JOIN returns the `send on closed channel` error #31129

- TiKV

  - Fix the issue that the MVCC deletion records are not cleared by GC #11217
  - Fix the issue that retrying prewrite requests in the pessimistic transaction mode might cause the risk of data inconsistency in rare cases #11187
  - Fix the issue that GC scan causes memory overflow #11410
  - Fix the issue that RocksDB flush or compaction causes panic when the disk capacity is full #11224

- PD

  - Fix the issue that Region statistics are not affected by `flow-round-by-digit` #4295
  - Fix the issue that the scheduling operator cannot fail fast because the target store is down #3353
  - Fix the issue that Regions on offline stores cannot be merged #4119
  - Fix the issue that the cold hotspot data cannot be deleted from the hotspot statistics #4390

- TiFlash

  - Fix the issue that TiFlash might panic when an MPP query is stopped
  - Fix the issue that queries with the `where <string>` clause return wrong results
  - Fix the potential issue of data inconsistency that might occur when setting the column type of an integer primary key to a larger range
  - Fix the issue that when an input time is earlier than 1970-01-01 00:00:01 UTC, the behavior of `unix_timestamp` is inconsistent with that of TiDB or MySQL
  - Fix the issue that TiFlash might return the `EstablishMPPConnection` error after it is restarted
  - Fix the issue that the `CastStringAsDecimal` behavior is inconsistent in TiFlash and in TiDB/TiKV
  - Fix the issue that the `DB::Exception: Encode type of coprocessor response is not CHBlock` error is returned in the query result
  - Fix the issue that the `castStringAsReal` behavior is inconsistent in TiFlash and in TiDB/TiKV

- Fix the issue that the returned result of the `date_add_string_xxx` function in TiFlash is inconsistent with that in MySQL

- Tools
  - Backup & Restore (BR)
    - Fix the potential issue that Region distribution might be uneven after a restore operation is finished #30425
    - Fix the issue that `'/'` cannot be specified in endpoint when `minio` is used as the backup storage #30104
    - Fix the issue that system tables cannot be restored because concurrently backing up system tables makes the table name fail to update #29710
  - TiCDC
    - Fix the issue that replication cannot be performed when `min.insync.replicas` is smaller than `replication-factor` #3994
    - Fix the issue that the `cached region` monitoring metric is negative #4300
    - Fix the issue that `mq sink write row` does not have monitoring data #3431
    - Fix the compatibility issue of `sql mode` #3810
    - Fix the potential panic issue that occurs when a replication task is removed #3128
    - Fix the issue of panic and data inconsistency that occurs when outputting the default column value #3929
    - Fix the issue that default values cannot be replicated #3793
    - Fix the potential issue that the deadlock causes a replication task to get stuck #4055
    - Fix the issue that no log is output when the disk is fully written #3362
    - Fix the issue that special comments in DDL statements cause the replication task to stop #3755
    - Fix the issue that the service cannot be started because of a timezone issue in the RHEL release #3584
    - Fix the issue of potential data loss caused by inaccurate checkpoint #3545
    - Fix the OOM issue in the container environment #1798
    - Fix the issue of replication stop caused by the incorrect configuration of `config.Metadata.Timeout` #3352
  - TiDB Data Migration (DM)
    - Fix the issue that the `CREATE VIEW` statement interrupts data replication #4173
    - Fix the issue the schema needs to be reset after a DDL statement is skipped #4177
    - Fix the issue that the table checkpoint is not updated in time after a DDL statement is skipped #4184
    - Fix a compatibility issue of the TiDB version with the Parser version #4298
    - Fix the issue that syncer metrics are updated only when querying the status #4281

- TiDB Lightning
  - Fix the issue of wrong import result that occurs when TiDB Lightning does not have the privilege to access the `mysql.tidb` table #31088
  - Fix the issue that some checks are skipped when TiDB Lightning is restarted #30772
  - Fix the issue that TiDB Lightning fails to report the error when the S3 path does not exist #30674
- TiDB Binlog
  - Fix the issue that Drainer fails because it is incompatible with the `CREATE PLACEMENT POLICY` statement #1118

16.8 v5.3

16.8.1 TiDB 5.3.4 Release Notes

Release date: November 24, 2022

TiDB version: 5.3.4

16.8.1.1 Improvements

- TiKV
  - Reload TLS certificate automatically for each update to improve availability #12546

16.8.1.2 Bug fixes

- TiDB
  - Fix the issue that the Region cache is not cleaned up in time when the Region is merged #37141
  - Fix the issue that TiDB writes wrong data due to the wrong encoding of the `ENUM` or `SET` column #32302
  - Fix the issue that database-level privileges are incorrectly cleaned up #38363
  - Fix the issue that the `grantor` field is missing in the `mysql.tables_priv` table #38293
  - Fix the issue that `KILL TIDB` cannot take effect immediately on idle connections #24031
  - Fix the issue that the return type of `date_add` and `date_sub` is different between TiDB and MySQL #36394, #27573
  - Fix the incorrect `INSERT_METHOD` value when Parser restores table options #38368

- Fix the issue that authentication fails when a MySQL client of v5.1 or earlier connects to the TiDB server #29725
- Fix wrong results of `GREATEST` and `LEAST` when passing in unsigned `BIGINT` arguments #30101
- Fix the issue that the result of `concat(ifnull(time(3))` in TiDB is different from that in MySQL #29498
- Fix the issue that the `avg()` function returns `ERROR 1105 (HY000): other error for mpp stream: Could not convert to the target type - - value is out of range.` when queried from TiFlash #29952
- Fix the issue that `ERROR 1105 (HY000): close of nil channel` is returned when using `HashJoinExec` #30289
- Fix the issue that TiKV and TiFlash return different results when querying logical operations #37258
- Fix the issue that the `EXPLAIN ANALYZE` statement with DML executors might return result before the transaction commit finishes #37373
- Fix the issue that Region cache is not cleared properly after merging many Regions #37174
- Fix the issue that the `EXECUTE` statement might throw an unexpected error in specific scenarios #37187
- Fix the issue that `GROUP CONCAT` with `ORDER BY` might fail when the `ORDER BY` clause contains a correlated subquery #18216
- Fix wrong results returned when length and width are incorrectly set for Decimal and Real when using plan cache #29565

- PD
  - Fix the issue that PD cannot correctly handle dashboard proxy requests #5321
  - Fix the issue that the TiFlash learner replica might not be created in specific scenarios #5401
  - Fix inaccurate Stream timeout and accelerate leader switchover #5207

- TiFlash
  - Fix the issue that logical operators return wrong results when the argument type is UInt8 #6127
  - Fix the issue that TiFlash bootstrap fails when `0.0` is used as the default value for integers, for example, `` `i` int(11)NOT NULL DEFAULT '0.0' `` #3157

- Tools
  - Dumpling
    - Fix the issue that Dumpling cannot dump data when the `--compress` option and the S3 output directory are set simultaneously #30534
  - TiCDC
    - Fix the issue that changefeed state is incorrect because a MySQL-related error is not reported to the owner in time #6698

16.8.2 TiDB 5.3.3 Release Note

Release date: September 14, 2022

TiDB version: 5.3.3

16.8.2.1 Bug fix

- TiKV
    - Fix the issue of continuous SQL execution errors in the cluster after the PD leader is switched or PD is restarted.
        - Cause: This issue is caused by a TiKV bug that TiKV does not retry sending heartbeat information to PD client after heartbeat requests fail, until TiKV reconnects to PD client. As a result, the Region information on the failed TiKV node becomes outdated, and TiDB cannot get the latest Region information, which causes SQL execution errors.
        - Affected versions: v5.3.2 and v5.4.2. This issue has been fixed in v5.3.3. If you are using v5.3.2, you can upgrade your cluster to v5.3.3.
        - Workaround: In addition to upgrade, you can also restart the TiKV nodes that cannot send Region heartbeat to PD, until there is no Region heartbeat to send.

      For bug details, see #12934.

16.8.3 TiDB 5.3.2 Release Notes

Release Date: June 29, 2022

TiDB version: 5.3.2

> **Warning:**
>
> It is not recommended to use v5.3.2, because this version has a known bug. For details, see #12934. This bug has been fixed in v5.3.3. It is recommended to use v5.3.3.

16.8.3.1 Compatibility Changes

- TiDB
    - Fix the issue that the `REPLACE` statement incorrectly changes other rows when the auto ID is out of range #29483

- PD
    - Disable compiling swagger server by default #4932

16.8.3.2 Improvements

- TiKV
    - Reduce the system call by the Raft client and increase CPU efficiency #11309
    - Improve the health check to detect unavailable Raftstore, so that the TiKV client can update Region Cache in time #12398
    - Transfer the leadership to CDC observer to reduce latency jitter #12111
    - Add more metrics for the garbage collection module of Raft logs to locate performance problems in the module #11374
- Tools
    - TiDB Data Migration (DM)
        - Support Syncer using the working directory of the DM-worker rather than `/tmp` to write internal files, and cleaning the directory after the task is stopped #4107
    - TiDB Lightning
        - Optimize Scatter Region to batch mode to improve the stability of the Scatter Region process #33618

16.8.3.3 Bug Fixes

- TiDB
    - Fix the issue that Amazon S3 cannot correctly calculate the size of compressed data #30534
    - Fix the issue of potential data index inconsistency in optimistic transaction mode #30410
    - Fix the issue that a SQL operation is canceled when its JSON type column joins its `CHAR` type column #29401
    - Previously, when a network connectivity issue occurred, TiDB did not always correctly free the resources held by the disconnected session. This issue has been fixed so that open transactions can be rolled back and other associated resources can be released. #34722
    - Fix the issue of the `data and columnID count not match` error that occurs when inserting duplicated values with TiDB Binlog enabled #33608
    - Fix the issue that query result might be wrong when Plan Cache is started in the RC isolation level #34447

- Fix the session panic that occurs when executing the prepared statement after table schema change with the MySQL binary protocol #33509
- Fix the issue that the table attributes are not indexed when a new partition is added and the issue that the table range information is not updated when the partition changes #33929
- Fix the issue that the TiDB server might run out of memory when the `INFORMATION_SCHEMA.CLUSTER_SLOW_QUERY` table is queried. This issue can be triggered when you check slow queries on the Grafana dashboard #33893
- Fix the issue that some DDL statements might be stuck for a period after the PD node of a cluster is replaced #33908
- Fix the issue that granting the `all` privilege might fail in clusters that are upgraded from v4.0 #33588
- Fix wrong results of deleting data of multiple tables using `left join` #31321
- Fix a bug that TiDB may dispatch duplicate tasks to TiFlash #32814
- Fix the issue that the background HTTP service of TiDB might not exit successfully and makes the cluster in an abnormal state #30571
- Fix the panic issue caused by the `fatal error: concurrent map read and map write` error #35340

- TiKV

  - Fix the issue of frequent PD client reconnection that occurs when the PD client meets an error #12345
  - Fix the issue of time parsing error that occurs when the `DATETIME` values contain a fraction and `Z` #12739
  - Fix the issue that TiKV panics when performing type conversion for an empty string #12673
  - Fix the possible duplicate commit records in pessimistic transactions when async commit is enabled #12615
  - Fix the bug that TiKV reports the `invalid store ID 0` error when using Follower Read #12478
  - Fix the issue of TiKV panic caused by the race between destroying peers and batch splitting Regions #12368
  - Fix the issue that successfully committed optimistic transactions may report the `Write Conflict` error when the network is poor #34066
  - Fix the issue that TiKV panics and destroys peers unexpectedly when the target Region to be merged is invalid #12232
  - Fix a bug that stale messages cause TiKV to panic #12023
  - Fix the issue of intermittent packet loss and out of memory (OOM) caused by the overflow of memory metrics #12160
  - Fix the potential panic issue that occurs when TiKV performs profiling on Ubuntu 18.04 #9765
  - Fix the issue that tikv-ctl returns an incorrect result due to its wrong string match #12329
  - Fix a bug that replica reads might violate the linearizability #12109

- Fix the TiKV panic issue that occurs when the target peer is replaced with the peer that is destroyed without being initialized when merging a Region #12048
- Fix a bug that TiKV might panic if it has been running for 2 years or more #11940

- PD

  - Fix the PD panic that occurs when a hot region has no leader #5005
  - Fix the issue that scheduling cannot start immediately after the PD leader transfer #4769
  - Fix the issue that a removed tombstone store appears again after the PD leader transfer #4941
  - Fix a bug of TSO fallback in some corner cases #4884
  - Fix the issue that when there exists a Store with large capacity (2T for example), fully allocated small Stores cannot be detected, which results in no balance operator being generated #4805
  - Fix the issue that schedulers do not work when `SchedulerMaxWaitingOperator` is set to `1` #4946
  - Fix the issue that the label distribution has residual labels in the metrics #4825

- TiFlash

  - Fix the bug that invalid storage directory configurations lead to unexpected behaviors #4093
  - Fix `TiFlash_schema_error` reported when `NOT NULL` columns are added #4596
  - Fix repeated crashes caused by the `commit state jump backward` errors #2576
  - Fix potential data inconsistency after a lot of INSERT and DELETE operations #4956
  - Fix a bug that canceled MPP queries might cause tasks to hang forever when the local tunnel is enabled #4229
  - Fix false reports of inconsistent TiFlash versions when TiFlash uses remote read #3713
  - Fix a bug that an MPP query might fail due to random gRPC keepalive timeout #4662
  - Fix a bug that an MPP query might hang forever if there are retries in the exchange receiver #3444
  - Fix the wrong result that occurs when casting `DATETIME` to `DECIMAL` #4151
  - Fix the overflow that occurs when casting `FLOAT` to `DECIMAL` #3998
  - Fix the potential `index out of bounds` error if calling `json_length` with empty string #2705
  - Fix wrong decimal comparison results in corner cases #4512
  - Fix bug that MPP query may hang forever if query failed in join build stage #4195
  - Fix possible wrong results when a query contains the `where <string>` clause #3447

- Fix the issue that the `CastStringAsReal` behavior is inconsistent in TiFlash and in TiDB or TiKV #3475
- Fix incorrect `microsecond` when casting string to datetime #3556
- Fix potential errors when querying on a table with many delete operations #4747
- Fix a bug that TiFlash reports many "Keepalive watchdog fired" errors randomly #4192
- Fix a bug that data not matching any region range remains on a TiFlash node #4414
- Fix a bug that MPP tasks might leak threads forever #4238
- Fix a bug that empty segments cannot be merged after GC #4511
- Fix the panic issue that occurs when TLS is enabled #4196
- Fix the issue that expired data is recycled slowly #4146
- Fix the bug that invalid storage directory configurations lead to unexpected behaviors #4093
- Fix the bug that some exceptions are not handled properly #4101
- Fix the potential query error after adding columns under heavy read workload #3967
- Fix the bug that the `STR_TO_DATE()` function incorrectly handles leading zeros when parsing microseconds #3557
- Fix the issue that TiFlash might return the `EstablishMPPConnection` error after it is restarted #3615

- Tools

  - Backup & Restore (BR)

    - Fix duplicate primary keys when inserting a record into a table after incremental restoration #33596
    - Fix the issue that schedulers do not resume after BR or TiDB Lightning exits abnormally #33546
    - Fix a bug that BR incremental restore returns errors mistakenly due to DDL jobs with empty query #33322
    - Fix the issue that BR does not retry enough times when Regions are not consistent during restoration #33419
    - Fix a bug that BR gets stuck when the restore operation meets some unrecoverable errors #33200
    - Fix the issue that BR fails to back up RawKV #32607
    - Fix the issue that BR cannot handle S3 internal errors #34350

  - TiCDC

    - Fix incorrect metrics caused by owner changes #4774
    - Fix the bug that the redo log manager flushes logs before writing logs #5486
    - Fix the bug that the resolved ts moves too fast when some tables are not maintained by the redo writer #5486
    - Add the UUID suffix to the redo log file name to fix the issue that file name conflicts may cause data loss #5486

- Fix the bug that MySQL Sink may save a wrong checkpointTs #5107
- Fix the issue that TiCDC clusters may panic after upgrade #5266
- Fix the issue that changefeed gets stuck when tables are repeatedly scheduled in the same node #4464
- Fix the issue that TiCDC fails to start when the first PD set in `--pd` is not available after TLS is enabled #4777
- Fix a bug that querying status through open API may be blocked when the PD node is abnormal #4778
- Fix a stability problem in workerpool used by Unified Sorter #4447
- Fix a bug that sequence is incorrectly replicated in some cases #4563

- TiDB Data Migration (DM)
  - Fix the issue that DM occupies more disk space after the task automatically resumes #3734 #5344
  - Fix an issue that the uppercase table cannot be replicated when `case-sensitive: true` is not set #5255
  - Fix the issue that in some cases manually executing the filtered DDL in the downstream might cause task resumption failure #5272
  - Fix the DM worker panic issue that occurs when the primary key is not first in the index returned by the `SHOW CREATE TABLE` statement #5159
  - Fix the issue that CPU usage may increase and a large amount of log is printed when GTID is enabled or when the task is automatically resumed #5063
  - Fix the issue that the relay log may be disabled after the DM-master reboots #4803
- TiDB Lightning
  - Fix the issue of Local-backend import failure caused by out-of-bounds data in the `auto_increment` column #27937
  - Fix the issue that the precheck does not check local disk resources and cluster availability #34213
  - Fix the checksum error "GC life time is shorter than transaction duration" #32733

16.8.4 TiDB 5.3.1 Release Notes

Release Date: March 3, 2022

TiDB version: 5.3.1

16.8.4.1 Compatibility changes

- Tools
  - TiDB Lightning
    - Change the default value of `regionMaxKeyCount` from 1_440_000 to 1_280_000, to avoid too many empty Regions after data import #30018

16.8.4.2 Improvements

- TiDB
  - Optimize the mapping logic of user login mode to make the logging more MySQL-compatible #30450
- TiKV
  - Reduce the TiCDC recovery time by reducing the number of the Regions that require the Resolve Locks step #11993
  - Speed up the Garbage Collection (GC) process by increasing the write batch size when performing GC to Raft logs #11404
  - Update the proc filesystem (procfs) to v0.12.0 #11702
- PD
  - Optimize the content format of the `DR_STATE` file #4341
- Tools
  - TiCDC
    - Expose configuration parameters of the Kafka producer to make them configurable in TiCDC #4385
    - Add a pre-cleanup process upon TiCDC startup if S3 is used as backend storage #3878
    - The TiCDC client works when no certificate name is specified #3627
    - Manage sink checkpoints per table to avoid unexpected advance of checkpoint timestamps #3545
    - Add the exponential backoff mechanism for restarting a changefeed. #3329
    - Change the default value of Kafka Sink `partition-num` to 3 so that TiCDC distributes messages across Kafka partitions more evenly #3337
    - Reduce the count of "EventFeed retry rate limited" logs #4006
    - Set the default value of `max-message-bytes` to 10M #4041
    - Add more Prometheus and Grafana monitoring metrics and alerts, including `no owner alert`, `mounter row`, `table sink total row`, and `buffer sink total row` #4054 #1606
    - Reduce the time for the KV client to recover when a TiKV store is down #3191
  - TiDB Lightning
    - Refine the output message of the precheck to make it more user-friendly when the local disk space check fails #30395

16.8.4.3 Bug fixes

- TiDB
    - Fix the issue that `date_format` in TiDB handles `'\n'` in a MySQL-incompatible way #32232
    - Fix the issue that `alter column set default` wrongly updates the table schema #31074
    - Fix a bug that `tidb_super_read_only` is not automatically enabled when `tidb_restricted_read_only` is enabled #31745
    - Fix the issue that the `greatest` or `least` function with collation gets a wrong result #31789
    - Fix the MPP task list empty error when executing a query #31636
    - Fix wrong results of index join caused by an innerWorker panic #31494
    - Fix wrong query results after changing the column type from `FLOAT` to `DOUBLE` #31372
    - Fix the `invalid transaction` error when executing a query using index lookup join #30468
    - Fix wrong query results due to the optimization of `Order By` #30271
    - Fix the issue that the configurations of `MaxDays` and `MaxBackups` do not take effect on the slow log #25716
    - Fix the issue that executing the `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` statement gets panic #28078
- TiKV
    - Fix the panic issue caused by deleting snapshot files when the peer status is `Applying` #11746
    - Fix the issue of QPS drop when flow control is enabled and `level0_slowdown_trigger` is set explicitly #11424
    - Fix the panic issue that occurs when the cgroup controller is not mounted #11569
    - Fix the issue that the latency of Resolved TS increases after TiKV stops operating #11351
    - Fix a bug that TiKV cannot delete a range of data (`unsafe_destroy_range` cannot be executed) when the GC worker is busy #11903
    - Fix the issue that destroying a peer might cause high latency #10210
    - Fix a bug that the `any_value` function returns a wrong result when regions are empty #11735
    - Fix the issue that deleting an uninitialized replica might cause an old replica to be recreated #10533
    - Fix the metadata corruption issue when `Prepare Merge` is triggered after a new election is finished but the isolated peer is not informed #11526
    - Fix the deadlock issue that happens occasionally when coroutines run too fast #11549
    - Fix the issue that a down TiKV node causes the resolved timestamp to lag #11351

- Fix the issue that batch messages are too large in Raft client implementation #9714
- Fix a panic issue that occurs when Region merge, ConfChange, and Snapshot happen at the same time in extreme conditions #11475
- Fix the issue that TiKV cannot detect the memory lock when TiKV performs a reverse table scan #11440
- Fix the issue that RocksDB flush or compaction causes panic when the disk capacity is full #11224
- Fix a bug that tikv-ctl cannot return the correct Region-related information #11393
- Fix the issue that the average latency of the by-instance gRPC requests is inaccurate in TiKV metrics #11299

- PD

    - Fix a bug that the scheduling process has the unnecessary JointConsensus steps in certain cases #4362
    - Fix a bug that the scheduling cannot be executed when demoting a voter directly #4444
    - Fix a data race issue that occurs when updating the configuration of the replication mode of replicas #4325
    - Fix a bug that the Read lock is not released in certain cases #4354
    - Fix the issue that the cold hotspot data cannot be deleted from the hotspot statistics #4390

- TiFlash

    - Fix the issue that `cast(arg as decimal(x,y))` returns a wrong result when the input argument `arg` overflows the range of `decimal(x,y)`
    - Fix the TiFlash crash issue that occurs when `max_memory_usage` and `max_memory_usage_for_all_queries` are enabled
    - Fix the issue that `cast(string as real)` returns a wrong result
    - Fix the issue that `cast(string as decimal)` returns a wrong result
    - Fix potential data inconsistency after altering a primary key column to a larger int data type
    - Fix the bug that when `in` has multiple arguments in the statements like `select (arg0, arg1)in (x,y)`, `in` returns a wrong result
    - Fix the issue that TiFlash might panic when an MPP query is stopped
    - Fix the issue that `str_to_date` returns a wrong result when the input argument has leading zeros
    - Fix the issue that the query returns a wrong result when the filter is in the `where <string>` format
    - Fix the issue that `cast(string as datetime)` returns a wrong result when the input argument `string` is in the `%Y-%m-%d\n%H:%i:%s` format

- Tools

- Backup & Restore (BR)
  - Fix the potential issue that Regions might be unevenly distributed after a restore operation is finished #31034
- TiCDC
  - Fix a bug that long varchars report an error `Column length too big` #4637
  - Fix a bug that a TiCDC node exits abnormally when a PD leader is killed #4248
  - Fix the issue that execution errors of the update statement in safemode may cause the DM-worker panic #4317
  - Fix the issue that cached region metric of the TiKV client may be negative #4300
  - Fix the bug that HTTP API panics when the required processor information does not exist #3840
  - Fix a bug that redo logs are not cleaned up when removing a paused changefeed #4740
  - Fix OOM in container environments #1798
  - Fix a bug that stopping a loading task results in unexpected transfer of the task #3771
  - Fix the issue that wrong progress is returned for the `query-status` command on loader #3252
  - Fix the issue that HTTP API fails to work if there are TiCDC nodes of different versions in a cluster #3483
  - Fix the issue that TiCDC exits abnormally when the S3 storage is configured with TiCDC Redo Log #3523
  - Fix the issue that default values cannot be replicated #3793
  - Fix a bug that MySQL sink generates duplicated `replace` SQL statements if `batch-replace-enable` is disabled #4501
  - Fix the issue that syncer metrics are updated only when querying the status #4281
  - Fix the issue that `mq sink write row` does not have monitoring data #3431
  - Fix the issue that replication cannot be performed when `min.insync.` ↪ `replicas` is smaller than `replication-factor` #3994
  - Fix the issue that `mq sink write row` does not have monitoring data #3431
  - Fix the potential panic issue that occurs when a replication task is removed #3128
  - Fix the potential issue that the deadlock causes a replication task to get stuck #4055
  - Fix the TiCDC panic issue that occurs when manually cleaning the task status in etcd #2980
  - Fix the issue that special comments in DDL statements cause the replication task to stop #3755
  - Fix the issue of replication stop caused by the incorrect configuration of `config.Metadata.Timeout` #3352
  - Fix the issue that the service cannot be started because of a timezone issue

in some RHEL releases #3584
- Fix the issue that `stopped` changefeeds resume automatically after a cluster upgrade #3473
- Fix the issue that default values cannot be replicated #3793
- Fix the issue of overly frequent warnings caused by MySQL sink deadlock #2706
- Fix the bug that the `enable-old-value` configuration item is not automatically set to `true` on the Canal protocol #3676
- Fix the issue that Avro sink does not support parsing JSON type columns #3624
- Fix the negative value error in the changefeed checkpoint lag #3010

- TiDB Data Migration (DM)
  - Fix a bug that the relay status in the DM-master is wrong after restarting the DM-master and DM-worker in a particular order #3478
  - Fix a bug that the DM-worker fails to boot up after a restart #3344
  - Fix a bug that a DM task fails if running a PARTITION DDL takes too long time #3854
  - Fix a bug that DM may report `invalid sequence` when upstream is MySQL 8.0 #3847
  - Fix a bug of data loss when DM does finer grained retry #3487
  - Fix the issue that the `CREATE VIEW` statement interrupts data replication #4173
  - Fix the issue the schema needs to be reset after a DDL statement is skipped #4177
- TiDB Lightning
  - Fix the bug that TiDB Lightning may not delete the metadata schema when some import tasks do not contain source files #28144
  - Fix the bug that TiDB Lightning returns an error if the storage URL prefix is "gs://xxx", instead of "gcs://xxx" #32742
  - Fix the issue that setting –log-file="-" does not print any log to stdout #29876
  - Fix the issue that TiDB Lightning does not report errors when the S3 storage path does not exist #30709

16.8.5 TiDB 5.3 Release Notes

Release date: November 30, 2021

TiDB version: 5.3.0

In v5.3, the key new features or improvements are as follows:

- Introduce temporary tables to simplify your application logic and improve performance
- Support setting attributes for tables and partitions

- Support creating users with the least privileges on TiDB Dashboard to enhance system security
- Optimize the timestamp processing flow in TiDB to improve the overall performance
- Enhance the performance of TiDB Data Migration (DM) so that data is migrated from MySQL to TiDB with lower latency
- Support parallel import using multiple TiDB Lightning instances to improve the efficiency of full data migration
- Support saving and restoring the on-site information of a cluster with a single SQL statement, which helps improve the efficiency of troubleshooting issues relating to execution plans
- Support the continuous profiling experimental feature to improve the observability of database performance
- Continue optimizing the storage and computing engines to improve the system performance and stability
- Reduce the write latency of TiKV by separating I/O operations from Raftstore thread pool (disabled by default)

16.8.5.1 Compatibility changes

> **Note:**
>
> When upgrading from an earlier TiDB version to v5.3.0, if you want to know the compatibility change notes of all intermediate versions, you can check the Release Notes of the corresponding version.

16.8.5.1.1 System variables

Variable name	Change type	Description
`tidb_enable_noop_functions`	Modified	Temporary tables are now supported by TiDB so `CREATE TEMPORARY TABLE` and `DROP TEMPORARY TABLE` no longer require enabling `tidb_enable_noop_functions`.

Variable name	Change type	Description
tidb_enable_pseudo_for_outdated_stats ↪	Newly added	Controls the behavior of the optimizer when the statistics on a table expire. The default value is ON. When the number of modified rows in the table is greater than 80% of the total rows (This ratio can be adjusted by the configuration pseudo-estimate-ratio), the optimizer considers that the statistics other than the total number of rows are no longer reliable and use pseudo statistics instead. When you set the value as OFF, even if the statistics expire, the optimizer still uses them.

Variable name	Change type	Description
`tidb_enable_tso_follower_proxy` ↪	Newly added	Determines whether to enable or disable the TSO Follower Proxy feature. The default value is `OFF`, which means the TSO Follower Proxy feature is disabled. At this time, TiDB only gets TSO from PD leader. When this feature is enabled, TiDB evenly sends the requests to all PD nodes when acquiring TSO. The PD follower then forwards the TSO requests to reduce the CPU pressure of PD leader.

Variable name	Change type	Description
tidb_tso_client_batch_max_wait_time	Newly added	Sets the maximum waiting time for a batch saving operation when TiDB requests TSO from PD. The default value is 0, which means no additional waiting.
tidb_tmp_table_max_size	Newly added	Limits the maximum size of a single temporary table. If the temporary table exceeds this size, an error will occur.

16.8.5.1.2 Configuration file parameters

Configuration file	Configuration item	Change type	Description
TiDB	prepared-plan-cache.capacity	Modified	Controls the number of cached statements. The default value is changed from 100 to 1000.

Configuration file	Configuration item	Change type	Description
TiKV	storage. ↪ reserve ↪ -space	Modified	Controls space reserved for disk protection when TiKV is started. Starting from v5.3.0, 80% of the reserved space is used as the extra disk space required for operations and maintenance when the disk space is insufficient, and the other 20% is used to store temporary files.
TiKV	memory- ↪ usage- ↪ limit	Modified	This configuration item is new in TiDB v5.3.0 and its value is calculated based on storage.block-cache.capacity.

Configuration file	Configuration item	Change type	Description
TiKV	raftstore. ↪ store- ↪ io-pool ↪ -size	Newly added	The allowable number of threads that process Raft I/O tasks, which is the size of the StoreWriter thread pool. When you modify the size of this thread pool, refer to Performance tuning for TiKV thread pools.

Configuration file	Configuration item	Change type	Description
TiKV	`raftstore.` ↪ `raft-` ↪ `write-` ↪ `size-` ↪ `limit`	Newly added	Determines the threshold at which Raft data is written into the disk. If the data size is larger than the value of this configuration item, the data is written to the disk. When the value of `raftstore.` ↪ `store-` ↪ `io-pool` ↪ `-size` is `0`, this configuration item does not take effect.

Configuration file	Configuration item	Change type	Description
TiKV	`raftstore.` ↪ `raft-` ↪ `msg-` ↪ `flush-` ↪ `interval` ↪	Newly added	Determines the interval at which Raft messages are sent in batches. The Raft messages in batches are sent at every interval specified by this configuration item. When the value of `raftstore.` ↪ `store-` ↪ `io-pool` ↪ `-size` is `0`, this configuration item does not take effect.
TiKV	`raftstore.` ↪ `raft-` ↪ `reject-` ↪ `transfer` ↪ `-leader` ↪ `-` ↪ `duration` ↪	Deleted	Determines the smallest duration that a Leader is transferred to a newly added node.

Configuration file	Configuration item	Change type	Description
PD	`log.file.` ↪ `max-` ↪ `days`	Modified	Controls the maximum number of days that logs are retained for. The default value is changed from `1` to `0`.
PD	`log.file.` ↪ `max-` ↪ `backups`	Modified	Controls the maximum number of logs that are retained for. The default value is changed from `7` to `0`.

Configuration file	Configuration item	Change type	Description
PD	`patrol-region-interval`	Modified	Controls the running frequency at which replicaChecker checks the health state of a Region. The smaller this value is, the faster replicaChecker runs. Normally, you do not need to adjust this parameter. The default value is changed from `100ms` to `10ms`.

Configuration file	Configuration item	Change type	Description
PD	`max-snapshot-count`	Modified	Controls the maximum number of snapshots that a single store receives or sends at the same time. PD schedulers depend on this configuration to prevent the resources used for normal traffic from being preempted. The default value is changed from `3` to `64`.

Configuration file	Configuration item	Change type	Description
PD	max-pending-peer-count	Modified	Controls the maximum number of pending peers in a single store. PD schedulers depend on this configuration to prevent too many Regions with outdated logs from being generated on some nodes. The default value is changed from 16 to 64.
TiD Lightning	meta-schema-name	Newly added	The schema name where the meta information for each TiDB Lightning instance is stored in the target cluster. The default value is "lightning_metadata".

16.8.5.1.3 Others

- Temporary tables:
  - If you have created local temporary tables in a TiDB cluster earlier than v5.3.0, these tables are actually ordinary tables, and handled as ordinary tables after the cluster is upgraded to v5.3.0 or a later version. If you have created global temporary tables in a TiDB cluster of v5.3.0 or a later version, when the cluster is downgraded to a version earlier than v5.3.0, these tables are handled as ordinary tables and cause a data error.
  - Since v5.3.0, TiCDC and BR support global temporary tables. If you use TiCDC and BR of a version earlier than v5.3.0 to replicate global temporary tables to the downstream, a table definition error occurs.
  - The following clusters are expected to be v5.3.0 or later; otherwise, data error is reported when you create a global temporary table:
    - the cluster to be imported using TiDB migration tools
    - the cluster restored using TiDB migration tools
    - the downstream cluster in a replication task using TiDB migration tools
  - For the compatibility information of temporary tables, refer to Compatibility with MySQL temporary tables and Compatibility restrictions with other TiDB features.

- For releases earlier than v5.3.0, TiDB reports an error when a system variable is set to an illegal value. For v5.3.0 and later releases, TiDB returns success with a warning such as "|Warning | 1292 | Truncated incorrect xxx: 'xx' " when a system variable is set to an illegal value.

- Fix the issue that the `SHOW VIEW` permission is not required to execute `SHOW CREATE VIEW`. Now you are expected to have the `SHOW VIEW` permission to execute the `SHOW CREATE VIEW` statement.

- The system variable `sql_auto_is_null` is added to the noop functions. When `tidb_enable_noop_functions = 0/OFF`, modifying this variable value causes an error.

- The `GRANT ALL ON performance_schema.*` syntax is no longer permitted. If you execute this statement in TiDB, an error occurs.

- Fix the issue that auto-analyze is unexpectedly triggered outside the specified time period when new indexes are added before v5.3.0. In v5.3.0, after you set the time period through the `tidb_auto_analyze_start_time` and `tidb_auto_analyze_end_time` variables, auto-analyze is triggered only during this time period.

- The default storage directory for plugins is changed from `""` to `/data/deploy/plugin`.

- The DM code is migrated to the folder "dm" in TiCDC code repository. Now DM follows TiDB in version numbers. Next to v2.0.x, the new DM version is v5.3.0, and you can upgrade from v2.0.x to v5.3.0 without any risk.

- The default deployed version of Prometheus is upgraded from v2.8.1 to v2.27.1, which is released in May 2021. This version provides more features and fixes a security issue. Compared with Prometheus v2.8.1, alert time representation in v2.27.1 is changed from Unix timestamp to UTC. For details, refer to Prometheus commit for more details.

16.8.5.2 New features

16.8.5.2.1 SQL

- **Use SQL interface to set placement rules for data (experimental feature)**

  Support the `[CREATE | ALTER] PLACEMENT POLICY` syntax that provides a SQL interface to set placement rules for data. Using this feature, you can specify tables and partitions to be scheduled to specific regions, data centers, racks, hosts, or replica count rules. This meets your application demands for lower cost and higher flexibility. The typical user scenarios are as follows:

  - Merge multiple databases of different applications to reduce the cost on database maintenance, and achieve application resource isolation through the rule configuration
  - Increase replica count for important data to improve the application availability and data reliability
  - Store new data into SSDs and store old data into HHDs to lower the cost on data archiving and storage
  - Schedule the leaders of hotspot data to high-performance TiKV instances
  - Separate cold data to lower-cost storage mediums to improve cost efficiency

  User document, #18030

- **Temporary tables**

  Support the `CREATE [GLOBAL] TEMPORARY TABLE` statement to create temporary tables. Using this feature, you can easily manage the temporary data generated in the calculation process of an application. Temporary data is stored in memory and you can use the `tidb_tmp_table_max_size` variable to limit the size of a temporary table. TiDB supports the following types of temporary tables:

  - Global temporary tables
    - Visible to all sessions in the cluster, and table schemas are persistent.
    - Provides transaction-level data isolation. The temporary data is effective only in the transaction. After the transaction finishes, the data is automatically dropped.
  - Local temporary tables
    - Visible only to the current session, and tables schemas are not persistent.

- Supports duplicated table names. You do not need to design complicated naming rules for your application.
- Provides session-level data isolation, which enables you to design a simpler application logic. After the transaction finishes, the temporary tables are dropped.

User document, #24169

- **Support the `FOR UPDATE OF TABLES` syntax**

  For a SQL statement that joins multiple tables, TiDB supports acquiring pessimistic locks on the rows correlated to the tables that are included in `OF TABLES`.

  User document, #28689

- **Table attributes**

  Support the `ALTER TABLE [PARTITION] ATTRIBUTES` statement that allows you to set attributes for a table or partition. Currently, TiDB only supports setting the `merge_option` attribute. By adding this attribute, you can explicitly control the Region merge behavior.

  User scenarios: When you perform the `SPLIT TABLE` operation, if no data is inserted after a certain period of time (controlled by the PD parameter `split-merge-interval` ↪), the empty Regions are automatically merged by default. In this case, you can set the table attribute to `merge_option=deny` to avoid the automatic merging of Regions.

  User document, #3839

16.8.5.2.2 Security

- **Support creating users with the least privileges on TiDB Dashboard**

  The account system of TiDB Dashboard is consistent with that of TiDB SQL. Users accessing TiDB Dashboard are authenticated and authorized based on TiDB SQL users' privileges. Therefore, TiDB Dashboard requires limited privileges, or merely the read-only privilege. You can configure users to access TiDB Dashboard based on the principle of least privilege, thus avoiding access of high-privileged users.

  It is recommended that you create a least-privileged SQL user to access and sign in with TiDB Dashboard. This avoids access of high-privileged users and improves security.

  User document

16.8.5.2.3 Performance

- **Optimize the timestamp processing flow of PD**

  TiDB optimizes its timestamp processing flow and reduces the timestamp processing load of PD by enabling PD Follower Proxy and modifying the batch waiting time required when the PD client requests TSO in batches. This helps improve the overall scalability of the system.

- Support enabling or disabling PD Follower Proxy through the system variable `tidb_enable_tso_follower_proxy`. Suppose that the TSO requests load of PD is too high. In this case, enabling PD follower proxy can batch forward the TSO requests collected during the request cycle on followers to the leader nodes. This solution can effectively reduce the number of direct interactions between clients and leaders, reduce the pressure of the load on leaders, and improve the overall performance of TiDB.

> **Note:**
> When the number of clients is small and the PD leader CPU load is not full, it is NOT recommended to enable PD Follower Proxy.

- Support using the system variable `tidb_tso_client_batch_max_wait_time` to set the maximum waiting time needed for the PD client to batch request TSO. The unit of this time is milliseconds. In case that PD has a high TSO requests load, you can reduce the load and improve the throughput by increasing the waiting time to get a larger batch size.

> **Note:**
> When the TSO request load is not high, it is NOT recommended to modify this variable value.

User document, #3149

16.8.5.2.4 Stability

- **Support Online Unsafe Recovery after some stores are permanently damaged (experimental feature)**

  Support the `pd-ctl unsafe remove-failed-stores` command that performs online data unsafe recovery. Suppose that the majority of data replicas encounter issues like permanent damage (such as disk damage), and these issues cause the data ranges in an application to be unreadable or unwritable. In this case, you can use the Online Unsafe Recovery feature implemented in PD to recover the data, so that the data is readable or writable again.

  It is recommended to perform the feature-related operations with the support of the TiDB team.

  User document, #10483

16.8.5.2.5 Data migration

- **DM replication performance enhanced**

  Supports the following features to ensure lower-latency data replication from MySQL to TiDB:

  - Compact multiple updates on a single row into one statement
  - Merge batch updates of multiple rows into one statement

- **Add DM OpenAPI to better maintain DM clusters (experimental feature)**

  DM provides the OpenAPI feature for querying and operating the DM cluster. It is similar to the feature of dmctl tools.

  Currently, DM OpenAPI is an experimental feature and disabled by default. It is not recommended to use it in a production environment.

  User document

- **TiDB Lightning Parallel Import**

  TiDB Lightning provides parallel import capability to extend the original feature. It allows you to deploy multiple Lightning instances at the same time to import single tables or multiple tables to downstream TiDB in parallel. Without changing the way customers use it, it greatly improves the data migration ability, allowing you to migrate data in a more real-time way to further process, integrate and analyze them. It improves the efficiency of enterprise data management.

  In our test, using 10 TiDB Lightning instances, a total of 20 TiB MySQL data can be imported to TiDB within 8 hours. The performance of multiple table import is also improved. A single TiDB Lightning instance can support importing at 250 GiB/h, and the overall migration is 8 times faster than the original performance.

  User document

- **TiDB Lightning Prechecks**

  TiDB Lightning provides the ability to check the configuration before running a migration task. It is enabled by default. This feature automatically performs some routine checks for disk space and execution configuration. The main purpose is to ensure that the whole subsequent import process goes smoothly.

  User document

- **TiDB Lightning supports importing files of GBK character set**

  You can specify the character set of the source data file. TiDB Lightning will convert the source file from the specified character set to UTF-8 encoding during the import process.

  User document

- **Sync-diff-inspector improvement**

  - Improve the comparison speed from 375 MB/s to 700 MB/s
  - Reduce the memory consumption of TiDB nodes by nearly half during comparison
  - Optimize the user interface and display the progress bar during comparison

  User document

16.8.5.2.6 Diagnostic efficiency

- **Save and restore the on-site information of a cluster**

  When you locate and troubleshoot the issues of a TiDB cluster, you often need to provide information on the system and the query plan. To help you get the information and troubleshoot cluster issues in a more convenient and efficient way, the `PLAN REPLAYER` command is introduced in TiDB v5.3.0. This command enables you to easily save and restore the on-site information of a cluster, improves the efficiency of troubleshooting, and helps you more easily archive the issues for management.

  The features of `PLAN REPLAYER` are as follows:

  - Exports the information of a TiDB cluster at an on-site troubleshooting to a ZIP-formatted file for storage.
  - Imports into a cluster the ZIP-formatted file exported from another TiDB cluster. This file contains the information of the latter TiDB cluster at an on-site troubleshooting.

  User document, #26325

16.8.5.2.7 TiDB data share subscription

- **TiCDC Eventually Consistent Replication**

  TiCDC provides the eventually consistent replication capability in disaster scenarios. When a disaster occurs in the primary TiDB cluster and the service cannot be resumed in a short period of time, TiCDC needs to provide the ability to ensure the consistency of data in the secondary cluster. Meanwhile, TiCDC needs to allow the business to quickly switch the traffic to the secondary cluster to avoid the database being unavailable for a long time and affecting the business.

  This feature supports TiCDC to replicate incremental data from a TiDB cluster to the secondary relational database TiDB/Aurora/MySQL/MariaDB. In case the primary cluster crashes, TiCDC can recover the secondary cluster to a certain snapshot in the primary cluster within 5 minutes, given the condition that before disaster the replication status of TiCDC is normal and replication lag is small. It allows data loss of less than 30 minutes, that is, RTO <= 5min, and RPO <= 30min.

  User document

- **TiCDC supports the HTTP protocol OpenAPI for managing TiCDC tasks**

  Since TiDB v5.3.0, TiCDC OpenAPI becomes a General Availability (GA) feature. You can query and operate TiCDC clusters using OpenAPI in the production environment.

16.8.5.2.8 Deployment and maintenance

- **Continuous Profiling (experimental feature)**

  TiDB Dashboard supports the Continuous Profiling feature, which stores instance performance analysis results automatically in real time when TiDB clusters are running. You can check the performance analysis result in a flame graph, which is more observable and shortens troubleshooting time.

  This feature is disabled by default and needs to be enabled on the **Continuous Profile** page of TiDB Dashboard.

  This feature is only available for clusters upgraded or installed using TiUP v1.7.0 or above.

  User document

16.8.5.3 Telemetry

TiDB adds the information to the telemetry report about whether or not the TEMPORARY TABLE feature is used. This does not include table names or table data.

To learn more about telemetry and how to disable this behavior, refer to Telemetry.

16.8.5.4 Removed feature

Starting from TiCDC v5.3.0, the cyclic replication feature between TiDB clusters (an experimental feature in v5.0.0) has been removed. If you have already used this feature to replicate data before upgrading TiCDC, the related data is not affected after the upgrade.

16.8.5.5 Improvements

- TiDB

  - Show the affected SQL statements in the debug log when the coprocessor encounters a lock, which is helpful in diagnosing problems #27718
  - Support showing the size of the backup and restore data when backing up and restoring data in the SQL logical layer #27247
  - Improve the default collection logic of ANALYZE when `tidb_analyze_version` is `2`, which accelerates collection and reduces resource overhead
  - Introduce the `ANALYZE TABLE table_name COLUMNS col_1, col_2, ... , col_n` syntax. The syntax allows collecting statistics only on a portion of the columns in wide tables, which improves the speed of statistics collection

- TiKV
    - Enhance disk space protection to improve storage stability

      To solve the issue that TiKV might panic in case of a disk fully-written error, TiKV introduces a two-level threshold defense mechanism to protect the disk remaining space from being exhausted by excess traffic. Additionally, the mechanism provides the ability to reclaim space when the threshold is triggered. When the remaining space threshold is triggered, some write operations will fail and TiKV will return a disk full error as well as a list of disk full nodes. In this case, to recover the space and restore the service, you can execute `Drop/Truncate Table` or scale out the nodes.
    - Simplify the algorithm of L0 flow control #10879
    - Improve the error log report in the raft client module #10944
    - Improve logging threads to avoid them becoming a performance bottleneck #10841
    - Add more statistics types of write queries #10507
    - Reduce the write latency by separating I/O operations from Raftstore thread pool (disabled by default). For more information about tuning, see Tune TiKV Thread Pool Performance #10540
- PD
    - Add more types of write queries to QPS dimensions in the hotspot scheduler #3869
    - Support dynamically adjusting the retry limit of the Balance Region scheduler to improve the performance of the scheduler #3744
    - Update TiDB Dashboard to v2021.10.08.1 #4070
    - Support that the evict leader scheduler can schedule Regions with unhealthy peers #4093
    - Speed up the exit process of schedulers #4146
- TiFlash
    - Improve the execution efficiency of the TableScan operator greatly
    - Improve the execution efficiency of the Exchange operator
    - Reduce write amplification and memory usage during GC of the storage engine (experimental feature)
    - Improve the stability and availability of TiFlash when TiFlash restarts, which reduces possible query failures following the restart
    - Support pushing down multiple new String and Time functions to the MPP engine
        - String functions: LIKE pattern, FORMAT(), LOWER(), LTRIM(), RTRIM(), SUBSTRING_INDEX(), TRIM(), UCASE(), UPPER()

    - Mathematical functions: ROUND (decimal, int)
    - Date and time functions: HOUR(), MICROSECOND(), MINUTE(), SECOND(), SYSDATE()
    - Type conversion function: CAST(time, real)
    - Aggregation functions: GROUP_CONCAT(), SUM(enum)
  - Support 512-bit SIMD
  - Enhance the cleanup algorithm for outdated data to reduce disk usage and read files more efficiently
  - Fix the issue that dashboard does not display memory or CPU information in some non-Linux systems
  - Unify the naming style of TiFlash log files (keep the naming style consistent with that of TiKV) and support dynamic modification of logger.count and logger.size
  - Improve the data validation capability of column-based files (checksums, experimental feature)

- Tools

  - TiCDC

    - Reduce the default value of the Kafka sink configuration item `MaxMessageBytes` ↪ from 64 MB to 1 MB to fix the issue that large messages are rejected by the Kafka Broker #3104
    - Reduce memory usage in the replication pipeline #2553#3037 #2726
    - Optimize monitoring items and alert rules to improve observability of synchronous links, memory GC, and stock data scanning processes #2735 #1606 #3000 #2985 #2156
    - When the sync task status is normal, no more historical error messages are displayed to avoid misleading users #2242

16.8.5.6 Bug Fixes

- TiDB

  - Fix an error that occurs during execution caused by the wrong execution plan. The wrong execution plan is caused by the shallow copy of schema columns when pushing down the aggregation operators on partitioned tables #27797 #26554
  - Fix the issue that `plan cache` cannot detect changes of unsigned flags #28254
  - Fix the wrong partition pruning when the partition function is out of range #28233
  - Fix the issue that planner might cache invalid plans for `join` in some cases #28087
  - Fix wrong `IndexLookUpJoin` when hash column type is `enum` #27893
  - Fix a batch client bug that recycling idle connection might block sending requests in some rare cases #27688

- Fix the TiDB Lightning panic issue when it fails to perform checksum on a target cluster #27686
- Fix wrong results of the `date_add` and `date_sub` functions in some cases #27232
- Fix wrong results of the `hour` function in vectorized expression #28643
- Fix the authenticating issue when connecting to MySQL 5.1 or an older client version #27855
- Fix the issue that auto analyze might be triggered out of the specified time when a new index is added #28698
- Fix a bug that setting any session variable invalidates `tidb_snapshot` #28683
- Fix a bug that BR is not working for clusters with many missing-peer Regions #27534
- Fix the unexpected error like `tidb_cast to Int32 is not supported` when the unsupported `cast` is pushed down to TiFlash #23907
- Fix the issue that `DECIMAL overflow` is missing in the `%s value is out of range in '%s'` error message #27964
- Fix a bug that the availability detection of MPP node does not work in some corner cases #3118
- Fix the `DATA RACE` issue when assigning `MPP task ID` #27952
- Fix the `INDEX OUT OF RANGE` error for a MPP query after deleting an empty `dual table` #28250
- Fix the issue of false positive error log `invalid cop task execution summaries length` for MPP queries #1791
- Fix the issue of error log `cannot found column in Schema column` for MPP queries #28149
- Fix the issue that TiDB might panic when TiFlash is shuting down #28096
- Remove the support for insecure 3DES (Triple Data Encryption Algorithm) based TLS cipher suites #27859
- Fix the issue that Lightning connects to offline TiKV nodes during pre-check and causes import failures #27826
- Fix the issue that pre-check cost too much time when importing many files to tables #27605
- Fix the issue that rewriting expressions makes `between` infer wrong collation #27146
- Fix the issue that `group_concat` function did not consider the collation #27429
- Fix the result wrong that occurs when the argument of the `extract` function is a negative duration #27236
- Fix the issue that creating partition fails if `NO_UNSIGNED_SUBTRACTION` is set #26765
- Avoid expressions with side effects in column pruning and aggregation pushdown #27106
- Remove useless gRPC logs #24190
- Limit the valid decimal length to fix precision-related issues #3091
- Fix the issue of a wrong way to check for overflow in `plus` expression #26977
- Fix the issue of `data too long` error when dumping statistics from the table with `new collation` data #27024

- Fix the issue that the retried transactions' statements are not included in `TIDB_TRX` #28670
- Fix the wrong default value of the `plugin_dir` configuration #28084
- Fix the issue that the `CONVERT_TZ` function returns `NULL` when it is given a named timezone and a UTC offset #8311
- Fix the issue that `CREATE SCHEMA` does not use the character set specified by `character_set_server` and `collation_server` for new schemas if none are provided as part of the statement #27214

- TiKV

  - Fix the issue of unavailable TiKV caused by Raftstore deadlock when migrating Regions. The workaround is to disable the scheduling and restart the unavailable TiKV #10909
  - Fix the issue that CDC adds scan retries frequently due to the Congest error #11082
  - Fix the issue that the Raft connection is broken when the channel is full #11047
  - Fix the issue that batch messages are too large in Raft client implementation #9714
  - Fix the issue that some coroutines leak in `resolved_ts` #10965
  - Fix a panic issue that occurs to the coprocessor when the size of response exceeds 4 GiB #9012
  - Fix the issue that snapshot Garbage Collection (GC) misses GC snapshot files when snapshot files cannot be garbage collected #10813
  - Fix a panic issue caused by timeout when processing Coprocessor requests #10852
  - Fix a memory leak caused by monitoring data of statistics threads #11195
  - Fix a panic issue caused by getting the cgroup information from some platforms #10980
  - Fix the issue of poor scan performance because MVCC Deletion versions are not dropped by compaction filter GC #11248

- PD

  - Fix the issue that PD incorrectly delete the peers with data and in pending status because the number of peers exceeds the number of configured peers #4045
  - Fix the issue that PD does not fix down peers in time #4077
  - Fix the issue that the scatter range scheduler cannot schedule empty Regions #4118
  - Fix the issue that the key manager cost too much CPU #4071
  - Fix the data race issue that might occur when setting configurations of hot Region scheduler #4159
  - Fix slow leader election caused by stucked Region syncer #3936

- TiFlash

  - Fix the issue of inaccurate TiFlash Store Size statistics

- Fix the issue that TiFlash fails to start up on some platforms due to the absence of library `nsl`
- Block the infinite wait of `wait index` when writing pressure is heavy (a default timeout of 5 minutes is added), which prevents TiFlash from waiting too long for data replication to provide services
- Fix the slow and no result issues of the log search when the log volume is large
- Fix the issue that only the most recent logs can be searched when searching old historical logs
- Fix the possible wrong result when a new collation is enabled
- Fix the possible parsing errors when an SQL statement contains extremely long nested expressions
- Fix the possible `Block schema mismatch` error of the Exchange operator
- Fix the possible `Can't compare` error when comparing Decimal types
- Fix the `3rd arguments of function substringUTF8 must be constants` error of the `left/substring` function

- Tools

    - TiCDC

        - Fix the issue that TiCDC replication task might terminate when the upstream TiDB instance unexpectedly exits #3061
        - Fix the issue that TiCDC process might panic when TiKV sends duplicate requests to the same Region #2386
        - Fix unnecessary CPU consumption when verifying downstream TiDB/MySQL availability #3073
        - Fix the issue that the volume of Kafka messages generated by TiCDC is not constrained by `max-message-size` #2962
        - Fix the issue that TiCDC sync task might pause when an error occurs during writing a Kafka message #2978
        - Fix the issue that some partitioned tables without valid indexes might be ignored when `force-replicate` is enabled #2834
        - Fix the issue that scanning stock data might fail due to TiKV performing GC when scanning stock data takes too long #2470
        - Fix a possible panic issue when encoding some types of columns into Open Protocol format #2758
        - Fix a possible panic issue when encoding some types of columns into Avro format #2648

    - TiDB Binlog

        - Fix the issue that when most tables are filtered out, checkpoint cannot be updated under some special load #1075

16.9 v5.2

16.9.1 TiDB 5.2.4 Release Notes

Release Date: April 26, 2022

TiDB version: 5.2.4

16.9.1.1 Compatibility change(s)

- TiDB
    - Change the default value of the system variable `tidb_analyze_version` from `2` to `1` #31748
- TiKV
    - Add `raft-log-compact-sync-interval` to control the time interval (`"2s"` by default) to compact unnecessary Raft logs #11404
    - Change the default value of `raft-log-gc-tick-interval` from `"10s"` to `"3s"` #11404
    - When `storage.flow-control.enable` is set to `true`, the value of `storage.flow-control.hard-pending-compaction-bytes-limit` overwrites that of `rocksdb.(defaultcf|writecf|lockcf).hard-pending-compaction-bytes-limit` #11424
- Tools
    - TiDB Lightning
        - Change the default value of `regionMaxKeyCount` from 1_440_000 to 1_280_000, to avoid too many empty Regions after data import #30018

16.9.1.2 Improvements

- TiKV
    - Transfer the leadership to CDC observer to reduce latency jitter #12111
    - Reduce the TiCDC recovery time by reducing the number of the Regions that require the Resolve Locks step #11993
    - Update the proc filesystem (procfs) to v0.12.0 #11702
    - Speed up the Garbage Collection (GC) process by increasing the write batch size when performing GC to Raft logs #11404
    - Increase the speed of inserting SST files by moving the verification process to the `Import` thread pool from the `Apply` thread pool #11239

- Tools
    - TiCDC
        - Change the default value of Kafka Sink `partition-num` to 3 so that TiCDC distributes messages across Kafka partitions more evenly #3337
        - Reduce the time for the KV client to recover when a TiKV store is down #3191
        - Add a `Lag analyze` panel in Grafana #4891
        - Expose configuration parameters of the Kafka producer to make them configurable in TiCDC #4385
        - Add the exponential backoff mechanism for restarting a changefeed #3329
        - Reduce the count of "EventFeed retry rate limited" logs #4006
        - Set the default value of `max-message-bytes` to 10M #4041
        - Add more Prometheus and Grafana monitoring metrics and alerts, including `no owner alert`, `mounter row`, `table sink total row`, and `buffer sink total row` #4054 #1606
        - Support multiple Kubernetes clusters in Grafana dashboards #4665
        - Add catch-up ETA (Estimated Time of Arrival) to the `changefeed checkpoint` monitoring metric #5232

16.9.1.3 Bug fixes

- TiDB
    - Fix wrong range calculation results for Nulleq function on Enum values #32428
    - Fix the issue that INDEX HASH JOIN returns the `send on closed channel` error #31129
    - Fix the issue that concurrent column type change causes inconsistency between the schema and the data #31048
    - Fix the issue of potential data index inconsistency in optimistic transaction mode #30410
    - Fix the issue that a SQL operation is canceled when its JSON type column joins its `CHAR` type column #29401
    - Fix the issue that window functions might return different results when using a transaction or not #29947
    - Fix the issue that the `Column 'col_name' in field list is ambiguous` error is reported unexpectedly when a SQL statement contains natural join #25041
    - Fix the issue that the length information is wrong when casting `Decimal` to `String` #29417
    - Fix the issue that the `GREATEST` function returns inconsistent results due to different values of `tidb_enable_vectorized_expression` (set to `on` or `off`) #29434
    - Fix wrong results of deleting data of multiple tables using `left join` #31321
    - Fix a bug that TiDB may dispatch duplicate tasks to TiFlash #32814
    - Fix the MPP task list empty error when executing a query #31636

- Fix wrong results of index join caused by an innerWorker panic #31494
- Fix the issue that executing the `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` statement gets panic #28078
- Fix wrong query results due to the optimization of `Order By` #30271
- Fix the wrong result that might occur when performing `JOIN` on `ENUM` type columns #27831
- Fix the panic when using the `CASE WHEN` function on the `ENUM` data type #29357
- Fix wrong results of the `microsecond` function in vectorized expressions #29244
- Fix the issue that the window function causes TiDB to panic instead of reporting an error #30326
- Fix the issue that the Merge Join operator gets wrong results in certain cases #33042
- Fix the issue that TiDB gets a wrong result when a correlated subquery returns a constant #32089
- Fix the issue that TiDB writes wrong data due to the wrong encoding of the `ENUM` or `SET` column #32302
- Fix the issue that the `MAX` or `MIN` function on the `ENUM` or `SET` column returns a wrong result when the new collation is enabled in TiDB #31638
- Fix the issue that the IndexHashJoin operator does not exit successfully #31062
- Fix the issue that TiDB might read wrong data when a table has a virtual column #30965
- Fix the issue that the setting of the log level does not take effect on the slow query log #30309
- Fix the issue that partitioned tables cannot fully use indexes to scan data in some cases #33966
- Fix the issue that the background HTTP service of TiDB might not exit successfully and makes the cluster in an abnormal state #30571
- Fix the issue that TiDB might unexpectedly output many logs of failed authentication #29709
- Fix the issue that the system variable `max_allowed_packet` does not take effect #31422
- Fix the issue that the `REPLACE` statement incorrectly changes other rows when the auto ID is out of range #29483
- Fix the issue that the slow query log cannot output log normally and might consume too much memory #32656
- Fix the issue that the result of NATURAL JOIN might include unexpected columns #24981
- Fix the issue that using `ORDER BY` and `LIMIT` together in one statement might output wrong results if a prefix-column index is used to query data #29711
- Fix the issue that the DOUBLE type auto-increment column might be changed when the optimistic transaction retries #29892
- Fix the issue that the STR_TO_DATE function cannot handle the preceding zero of the microsecond part correctly #30078
- Fix the issue that TiDB gets the wrong result when using TiFlash to scan tables with empty range although TiFlash does not support reading tables with empty

range yet #33083

- TiKV
    - Fix a bug that stale messages cause TiKV to panic #12023
    - Fix the issue of intermittent packet loss and out of memory (OOM) caused by the overflow of memory metrics #12160
    - Fix the potential panic issue that occurs when TiKV performs profiling on Ubuntu 18.04 #9765
    - Fix the issue that tikv-ctl returns an incorrect result due to its wrong string match #12329
    - Fix a bug that replica reads might violate the linearizability #12109
    - Fix a bug that TiKV might panic if it has been running for 2 years or more #11940
    - Fix the issue of QPS drop when flow control is enabled and `level0_slowdown_trigger` ↪ is set explicitly #11424
    - Fix the panic issue that occurs when the cgroup controller is not mounted #11569
    - Fix possible metadata corruption caused by Region merge on a lagging Region peer #11526
    - Fix the issue that the latency of Resolved TS increases after TiKV stops operating #11351
    - Fix a panic issue that occurs when Region merge, ConfChange, and Snapshot happen at the same time in extreme conditions #11475
    - Fix a bug that tikv-ctl cannot return the correct Region-related information #11393
    - Fix the issue of negative sign when the decimal divide result is zero #29586
    - Fix the issue that retrying prewrite requests in the pessimistic transaction mode might cause the risk of data inconsistency in rare cases #11187
    - Fix a memory leak caused by monitoring data of statistics threads #11195
    - Fix the issue that the average latency of the by-instance gRPC requests is inaccurate in TiKV metrics #11299
    - Fix the panic issue caused by deleting snapshot files when the peer status is `Applying` #11746
    - Fix a bug that TiKV cannot delete a range of data (which means the internal command `unsafe_destroy_range` is executed) when the GC worker is busy #11903
    - Fix the issue that deleting an uninitialized replica might cause an old replica to be recreated #10533
    - Fix the issue that TiKV cannot detect the memory lock when TiKV performs a reverse table scan #11440
    - Fix the deadlock issue that happens occasionally when coroutines run too fast #11549
    - Fix the issue that destroying a peer might cause high latency #10210
    - Fix the issue that TiKV panics and destroys peers unexpectedly because the target Region to be merged is invalid #12232

- Fix the TiKV panic issue that occurs when the target peer is replaced with the peer that is destroyed without being initialized when merging a Region #12048
- Fix the TiKV panic issue that occurs when applying snapshot is aborted #11618
- Fix a bug that TiKV cannot correctly calculate the number of snapshots being sent when the operator execution fails #11341

- PD

    - Fix the issue that the Region scatterer scheduling lost some peers #4565
    - Fix the issue that the cold hotspot data cannot be deleted from the hotspot statistics #4390

- TiFlash

    - Fix a bug that MPP tasks might leak threads forever #4238
    - Fix the issue that the result of `IN` is incorrect in multi-value expressions #4016
    - Fix the issue that the date format identifies `'\n'` as an invalid separator #4036
    - Fix the potential query error after adding columns under heavy read workload #3967
    - Fix the bug that invalid storage directory configurations lead to unexpected behaviors #4093
    - Fix the bug that some exceptions are not handled properly #4101
    - Fix the bug that the `STR_TO_DATE()` function incorrectly handles leading zeros when parsing microseconds #3557
    - Fix the issue that casting `INT` to `DECIMAL` might cause overflow #3920
    - Fix the wrong result that occurs when casting `DATETIME` to `DECIMAL` #4151
    - Fix the overflow that occurs when casting `FLOAT` to `DECIMAL` #3998
    - Fix the issue that the `CastStringAsReal` behavior is inconsistent in TiFlash and in TiDB or TiKV #3475
    - Fix the issue that the `CastStringAsDecimal` behavior is inconsistent in TiFlash and in TiDB or TiKV #3619
    - Fix the issue that TiFlash might return the `EstablishMPPConnection` error after it is restarted #3615
    - Fix the issue that obsolete data cannot be reclaimed after setting the number of TiFlash replicas to 0 #3659
    - Fix potential data inconsistency when widening the primary key column with the primary key being `handle` #3569
    - Fix possible parsing errors when an SQL statement contains extremely long nested expressions #3354
    - Fix possible wrong results when a query contains the `where <string>` clause #3447
    - Fix possible wrong results when `new_collations_enabled_on_first_bootstrap` ↪ is enabled #3388, #3391
    - Fix the panic issue that occurs when TLS is enabled #4196
    - Fix the panic issue that occurs when the memory limit is enabled #3902

- Fix the issue that TiFlash crashes occasionally when an MPP query is stopped #3401
- Fix the unexpected error of `Unexpected type of column: Nullable(Nothing)` #3351
- Fix possible metadata corruption caused by Region merge on a lagging Region peer #4437
- Fix the issue that a query containing `JOIN` might be hung if an error occurs #4195
- Fix possible wrong results returned for MPP queries due to incorrect execution plans #3389

- Tools
  - Backup & Restore (BR)
    - Fix the issue that BR fails to back up RawKV #32607
  - TiCDC
    - Fix the issue that default values cannot be replicated #3793
    - Fix a bug that sequence is incorrectly replicated in some cases #4563
    - Fix a bug that a TiCDC node exits abnormally when a PD leader is killed #4248
    - Fix a bug that MySQL sink generates duplicated `replace` SQL statements when `batch-replace-enable` is disabled #4501
    - Fix the issue of panic and data inconsistency that occurs when outputting the default column value #3929
    - Fix the issue that `mq sink write row` does not have monitoring data #3431
    - Fix the issue that replication cannot be performed when `min.insync.` ↪ `replicas` is smaller than `replication-factor` #3994
    - Fix the potential panic issue that occurs when a replication task is removed #3128
    - Fix the bug that HTTP API panics when the required processor information does not exist #3840
    - Fix the issue of potential data loss caused by inaccurate checkpoint #3545
    - Fix the potential issue that the deadlock causes a replication task to get stuck #4055
    - Fix the TiCDC panic issue that occurs when manually cleaning the task status in etcd #2980
    - Fix the issue that special comments in DDL statements cause the replication task to stop #3755
    - Fix the issue of replication stop caused by the incorrect configuration of `config.Metadata.Timeout` #3352
    - Fix the issue that the service cannot be started because of a timezone issue in the RHEL release #3584
    - Fix the issue that `stopped` changefeeds resume automatically after a cluster upgrade #3473

- Fix the issue of overly frequent warnings caused by MySQL sink deadlock #2706
- Fix the bug that the `enable-old-value` configuration item is not automatically set to `true` on the Canal protocol #3676
- Fix the issue that Avro sink does not support parsing JSON type columns #3624
- Fix the negative value error in the changefeed checkpoint lag #3010
- Fix the OOM issue in the container environment #1798
- Fix the memory leak issue after processing DDLs #3174
- Fix the issue that changefeed gets stuck when tables are repeatedly scheduled in the same node #4464
- Fix a bug that querying status through open API may be blocked when the PD node is abnormal #4778
- Fix incorrect metrics caused by owner changes #4774
- Fix a stability problem in workerpool used by Unified Sorter #4447
- Fix the issue that the `cached region` monitoring metric is negative #4300

- TiDB Lightning
  - Fix the issue of wrong import result that occurs when TiDB Lightning does not have the privilege to access the `mysql.tidb` table #31088
  - Fix the checksum error "GC life time is shorter than transaction duration" #32733
  - Fix a bug that TiDB Lightning may not delete the metadata schema when some import tasks do not contain source files #28144
  - Fix the issue that TiDB Lightning does not report errors when the S3 storage path does not exist #28031 #30709
  - Fix an error that occurs when iterating more than 1000 keys on GCS #30377

16.9.2 TiDB 5.2.3 Release Note

Release date: December 3, 2021

TiDB version: 5.2.3

16.9.2.1 Bug fix

- TiKV
  - Fix the issue that the `GcKeys` task does not work when it is called by multiple keys. Caused by this issue, compaction filter GC might not drop the MVCC deletion information. #11217

16.9.3 TiDB 5.2.2 Release Notes

Release Date: October 29, 2021

TiDB version: 5.2.2

16.9.3.1 Improvements

- TiDB
    - Show the affected SQL statements in the debug log when the coprocessor encounters a lock, which is helpful in diagnosing problems #27718
    - Support showing the size of the backup and restore data when backing up and restoring data in the SQL logical layer #27247
- TiKV
    - Simplify the algorithm of L0 flow control #10879
    - Improve the error log report in the raft client module #10983
    - Improve logging threads to avoid them becoming a performance bottleneck #10841
    - Add more statistics types of write queries #10507
- PD
    - Add more types of write queries to QPS dimensions in the hotspot scheduler #3869
    - Support dynamically adjusting the retry limit of the balance region scheduler to improve the performance of the scheduler #3744
    - Update TiDB Dashboard to v2021.10.08.1 #4070
    - Support that the evict leader scheduler can schedule regions with unhealthy peers #4093
    - Speed up the exit process of schedulers #4146
- Tools
    - TiCDC
        - Reduce the default value of the Kafka sink configuration item `MaxMessageBytes` ↪ from 64 MB to 1 MB to fix the issue that large messages are rejected by the Kafka Broker #3104
        - Reduce memory usage in the relpication pipeline #2553#3037 #2726
        - Optimize monitoring items and alert rules to improve observability of synchronous links, memory GC, and stock data scanning processes #2735 #1606 #3000 #2985 #2156
        - When the sync task status is normal, no more historical error messages are displayed to avoid misleading users #2242

16.9.3.2 Bug Fixes

- TiDB
    - Fix the issue that plan-cache cannot detect changes of unsigned flags #28254
    - Fix the wrong partition pruning when the partition function is out of range #28233
    - Fix the issue that planner might cache invalid plans for `join` in some cases #28087
    - Fix wrong index hash join when hash column type is enum #27893
    - Fix a batch client bug that recycling idle connection might block sending requests in some rare cases #27688
    - Fix the TiDB Lightning panic issue when it fails to perform checksum on a target cluster #27686
    - Fix wrong results of the `date_add` and `date_sub` functions in some cases #27232
    - Fix wrong results of the `hour` function in vectorized expression #28643
    - Fix the authenticating issue when connecting to MySQL 5.1 or an older client version #27855
    - Fix the issue that auto analyze might be triggered out of the specified time when a new index is added #28698
    - Fix a bug that setting any session variable invalidates `tidb_snapshot` #28683
    - Fix a bug that BR is not working for clusters with many missing-peer regions #27534
    - Fix the unexpected error like `tidb_cast to Int32 is not supported` when the unsupported `cast` is pushed down to TiFlash #23907
    - Fix the issue that `DECIMAL overflow` is missing in the `%s value is out of range in '%s'` error message #27964
    - Fix a bug that the availability detection of MPP node does not work in some corner cases #3118
    - Fix the `DATA RACE` issue when assigning `MPP task ID` #27952
    - Fix the `INDEX OUT OF RANGE` error for a MPP query after deleting an empty `dual table`. #28250
    - Fix the issue of false positive error log `invalid cop task execution summaries length` for MPP queries #1791
    - Fix the issue of error log `cannot found column in Schema column` for MPP queries #28149
    - Fix the issue that TiDB might panic when TiFlash is shuting down #28096
    - Remove the support for insecure 3DES (Triple Data Encryption Algorithm) based TLS cipher suites #27859
    - Fix the issue that Lightning connects to offline TiKV nodes during pre-check and causes import failures #27826
    - Fix the issue that pre-check cost too much time when importing many files to tables #27605
    - Fix the issue that rewriting expressions makes `between` infer wrong collation #27146
    - Fix the issue that `group_concat` function did not consider the collation #27429

- Fix the result wrong that occurs when the argument of the `extract` function is a negative duration #27236
- Fix the issue that creating partition fails if `NO_UNSIGNED_SUBTRACTION` is set #26765
- Avoid expressions with side effects in column pruning and aggregation pushdown #27106
- Remove useless gRPC logs #24190
- Limit the valid decimal length to fix precision-related issues #3091
- Fix the issue of a wrong way to check for overflow in `plus` expression #26977
- Fix the issue of `data too long` error when dumping statistics from the table with `new collation` data #27024
- Fix the issue that the retried transactions' statements are not included in `TIDB_TRX` #28670

- TiKV

  - Fix the issue that CDC adds scan retries frequently due to the Congest error #11082
  - Fix the issue that the raft connection is broken when the channel is full #11047
  - Fix the issue that batch messages are too large in Raft client implementation #9714
  - Fix the issue that some coroutines leak in `resolved_ts` #10965
  - Fix a panic issue that occurs to the coprocessor when the size of response exceeds 4 GiB #9012
  - Fix the issue that snapshot Garbage Collection (GC) misses GC snapshot files when snapshot files cannot be garbage collected #10813
  - Fix a panic issue caused by timeout when processing Coprocessor requests #10852

- PD

  - Fix the issue that PD incorrectly delete the peers with data and in pending status because the number of peers exceeds the number of configured peers #4045
  - Fix the issue that PD does not fix down peers in time #4077
  - Fix the issue that the scatter range scheduler cannot schedule empty regions #4118
  - Fix the issue that the key manager cost too much CPU #4071
  - Fix the data race issue that might occur when setting configurations of hot region scheduler #4159
  - Fix slow leader election caused by stuck region syncer#3936

- TiFlash

  - Fix the issue that TiFlash fails to start up on some platforms due to the absence of library `nsl`

- Tools

- TiCDC
  - Fix the issue that TiCDC replication task might terminate when the upstream TiDB instance unexpectedly exits #3061
  - Fix the issue that TiCDC process might panic when TiKV sends duplicate requests to the same Region #2386
  - Fix unnecessary CPU consumption when verifying downstream TiDB/MySQL availability #3073
  - Fix the issue that the volume of Kafka messages generated by TiCDC is not constrained by `max-message-size` #2962
  - Fix the issue that TiCDC sync task might pause when an error occurs during writing a Kafka message #2978
  - Fix the issue that some partitioned tables without valid indexes might be ignored when `force-replicate` is enabled #2834
  - Fix the issue that scanning stock data might fail due to TiKV performing GC when scanning stock data takes too long #2470
  - Fix a possible panic issue when encoding some types of columns into Open Protocol format #2758
  - Fix a possible panic issue when encoding some types of columns into Avro format #2648
- TiDB Binlog
  - Fix the issue that when most tables are filtered out, checkpoint cannot be updated under some special load #1075

16.9.4 TiDB 5.2.1 Release Notes

Release date: September 9, 2021

TiDB version: 5.2.1

16.9.4.1 Bug fixes

- TiDB
  - Fix an error that occurs during execution caused by the wrong execution plan. The wrong execution plan is caused by the shallow copy of schema columns when pushing down the aggregation operators on partitioned tables. #27797 #26554
- TiKV
  - Fix the issue of unavailable TiKV caused by Raftstore deadlock when migrating Regions. The workaround is to disable the scheduling and restart the unavailable TiKV. #10909

16.9.5 TiDB 5.2 Release Notes

Release date: August 27, 2021

TiDB version: 5.2.0

> **Warning:**
>
> Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 5.2.x version.

In v5.2, the key new features and improvements are as follows:

- Support using several functions in expression indexes to greatly improve query performance
- Improve the accuracy of optimizer cardinality estimation to help to select optimal execution plans
- Announce the general availability (GA) for the Lock View feature to observe transaction locking events and troubleshoot deadlock problems
- Add the TiFlash I/O traffic limit feature to improve the stability of read and write for TiFlash
- TiKV introduces a new flow control mechanism to replace the previous RocksDB write stall mechanism to improve the stability of TiKV flow control
- Simplify the operation and maintenance of Data Migration (DM) to reduce the management cost.
- TiCDC supports HTTP protocol OpenAPI to manage TiCDC tasks. It provides a more user-friendly operation method for both Kubernetes and self-hosted environments. (Experimental feature)

16.9.5.1 Compatibility changes

> **Note:**
>
> When upgrading from an earlier TiDB version to v5.2, if you want to know the compatibility change notes of all intermediate versions, you can check the Release Note for the corresponding version.

16.9.5.1.1 System variables

Variable name	Change type	Description
default_authentication_plugin	Newly added	Sets the authentication method that the server advertises. The default value is mysql_native_password.
tidb_enable_auto_increment_in_generated	Newly added	Determines whether to include the AUTO_INCREMENT columns when creating a generated column or an expression index. The default value is OFF.
tidb_opt_enable_correlation_adjustment	Newly added	Controls whether the optimizer estimates the number of rows based on column order correlation. The default value is ON.

Variable name	Change type	Description
tidb_opt_limit_push_down_threshold ↪	Newly added	Sets the threshold that determines whether to push the Limit or TopN operator down to TiKV. The default value is 100.
tidb_stmt_summary_max_stmt_count ↪	Modified	Sets the maximum number of statements that the statement summary tables store in memory. The default value is changed from 200 to 3000.
tidb_enable_streaming ↪	Deprecated	The system variable enable-streaming ↪ is deprecated and it is not recommended to use it any more.

16.9.5.1.2 Configuration file parameters

Configuration file	Configuration item	Change type	Description
TiDB configuration file	pessimistic-txn.deadlock-history-collect-retryable	Newly added	Controls whether the INFORMATION_SCHEMA.DEADLOCKS table collects retryable deadlock error messages or not.
TiDB configuration file	security.auto-tls	Newly added	Determines whether to automatically generate the TLS certificates on startup. The default value is false.
TiDB configuration file	stmt-summary.max-stmt-count	Modified	Indicates the maximum number of SQL categories allowed to be saved in the statement summary tables. The default value is changed from 200 to 3000.

Configuration file	Configuration item	Change type	Description
TiDB configuration file	`experimental.allow-expression-index`	Deprecated	The `allow-expression-index` configuration in the TiDB configuration file is deprecated.
TiKV configuration file	`raftstore.cmd-batch`	Newly added	Controls whether to enable batch processing of the requests. When it is enabled, the write performance is significantly improved. The default value is `true`.

Configuration file	Configuration item	Change type	Description
TiKV configuration file	`raftstore.` ↪ `inspect` ↪ `-` ↪ `interval` ↪	Newly added	At a certain interval, TiKV inspects the latency of the Raftstore component. This configuration item specifies the interval of the inspection. If the latency exceeds this value, this inspection is marked as timeout. The default value is `500ms`.
TiKV configuration file	`raftstore.` ↪ `max-` ↪ `peer-` ↪ `down-` ↪ `duration` ↪	Modified	Indicates the longest inactive duration allowed for a peer. A peer with timeout is marked as `down`, and PD tries to delete it later. The default value is changed from `5m` to `10m`.

Configuration file	Configuration item	Change type	Description
TiKV configuration file	`server.raft-client-queue-size`	Newly added	Specifies the queue size of the Raft messages in TiKV. The default value is `8192`.
TiKV configuration file	`storage.flow-control.enable`	Newly added	Determines whether to enable the flow control mechanism. The default value is `true`.
TiKV configuration file	`storage.flow-control.memtables-threshold`	Newly added	When the number of kvDB memtables reaches this threshold, the flow control mechanism starts to work. The default value is `5`.
TiKV configuration file	`storage.flow-control.l0-files-threshold`	Newly added	When the number of kvDB L0 files reaches this threshold, the flow control mechanism starts to work. The default value is `9`.

Configuration file	Configuration item	Change type	Description
TiKV configuration file	storage. ↪ flow- ↪ control ↪ .soft- ↪ pending ↪ - ↪ compaction ↪ -bytes- ↪ limit	Newly added	When the pending compaction bytes in KvDB reach this threshold, the flow control mechanism starts to reject some write requests and reports the ServerIsBusy ↪ error. The default value is "192GB".
TiKV configuration file	storage. ↪ flow- ↪ control ↪ .hard- ↪ pending ↪ - ↪ compaction ↪ -bytes- ↪ limit	Newly added	When the pending compaction bytes in KvDB reach this threshold, the flow control mechanism rejects all write requests and reports the ServerIsBusy ↪ error. The default value is "1024GB".

16.9.5.1.3 Others

- Before the upgrade, check whether the value of the `tidb_evolve_plan_baselines` system variable is `ON`. If the value is `ON`, set it to `OFF`; otherwise, the upgrade will fail.
- For TiDB clusters upgraded from v4.0 to v5.2, the default value of `tidb_multi_statement_mode` ↪ changes from `WARN` to `OFF`.
- Before the upgrade, check the value of the TiDB configuration `feedback-probability` ↪ . If the value is not `0`, the "panic in the recoverable goroutine" error will occur after the upgrade, but this error does not affect the upgrade.
- TiDB is now compatible with MySQL 5.7's noop variable `innodb_default_row_format` ↪ . Setting this variable has no effect. #23541
- Starting from TiDB 5.2, to improve system security, it is recommended (but not mandatory) to encrypt the transport layer for connections from clients. TiDB provides the Auto TLS feature to automatically configure and enable encryption in TiDB. To use the Auto TLS feature, before the TiDB upgrade, set `security.auto-tls` in the TiDB configuration file to `true`.
- Support the `caching_sha2_password` authentication method to make migration from MySQL 8.0 easier and to improve security.

16.9.5.2 New features

16.9.5.2.1 SQL

- **Support using several functions in expression indexes**

  The expression index is a type of special index that can be created on an expression. After an expression index is created, TiDB supports expression-based queries, which greatly improves query performance.

  User document, #25150

- **Support the `translate` function in Oracle**

  The `translate` function replaces all occurrences of characters by other characters in a string. In TiDB, this function does not treat empty strings as `NULL` as Oracle does.

  User document

- **Support spilling HashAgg**

  Support spilling HashAgg into disk. When a SQL statement that includes an HashAgg operator causes out of memory (OOM), you can try to set the concurrency of this operator to `1` to trigger disk spill, which alleviates memory stress.

  User document, #25882

- **Improve the accuracy of optimizer cardinality estimation**

  - Improve the accuracy of TiDB's estimation of TopN/Limit. For example, for pagination queries on a large table that contain the `order by col limit x` condition, TiDB can more easily select the right index and reduce query response time.

- Improve the accuracy of out-of-range estimation. For example, even if the statistics for a day have not been updated, TiDB can accurately select the corresponding index for a query that contains `where date=Now()`.
- Introduce the `tidb_opt_limit_push_down_threshold` variable to control the optimizer's behavior of pushing down Limit/TopN, which resolves the issue that Limit/TopN cannot be pushed down in some situations due to wrong estimation.

User document, #26085

- **Improve index selection of the optimizer**

  Add pruning rules for index selection. Before using the statistics for comparison, TiDB uses these rules to narrow down the scope of possible indexes to be selected, which reduces the possibility of selecting non-optimal indexes.

  User document

16.9.5.2.2 Transaction

- **General availability (GA) for Lock View**

  The Lock View feature provides more information about lock conflicts and lock waits of pessimistic locks, which helps DBAs to observe transaction locking events and troubleshoot deadlock problems.

  In v5.2, the following enhancements are made to Lock View:

  - In addition to the SQL digest column in the Lock View-related tables, add a column to these tables that shows the corresponding normalized SQL text. You do not have to manually query the statement corresponding to a SQL digest.
  - Add the `TIDB_DECODE_SQL_DIGESTS` function to query the normalized SQL statements (a form without formats and arguments) corresponding to a set of SQL digests in the cluster. This simplifies the operation of querying the statements that have been historically executed by a transaction.
  - Add a column in the `DATA_LOCK_WAITS` and `DEADLOCKS` system tables to show the table name, row ID, index value, and other key information interpreted from a key. This simplifies the operations such as locating the table to which a key belongs and interpreting the key information.
  - Support collecting the information of retryable deadlock errors in the `DEADLOCKS` table, which makes it easier to troubleshoot issues caused by such errors. The error collection is disabled by default and can be enabled using the `pessimistic-txn.deadlock-history-collect-retryable` configuration.
  - Support distinguishing query-executing transactions from idle transactions on the `TIDB_TRX` system table. The `Normal` state is now divided into `Running` and `Idle` states.

User documents:

```
- View the pessimistic lock-waiting events that are occurring on all TiKV
    ↪ nodes in the cluster: [`DATA_LOCK_WAITS`](#data_lock_waits)
- View the deadlock errors recently occurred on a TiDB node: [`DEADLOCKS`](#
    ↪ deadlocks)
- View the executing transaction on a TiDB node: [`TIDB_TRX`](#tidb_trx)
```

- Optimize the user scenarios of adding indexes on tables with the `AUTO_RANDOM` or `SHARD_ROW_ID_BITS` attribute.

16.9.5.2.3 Stability

- **Add TiFlash I/O traffic limit**

  This new feature is suitable for cloud storage with disk bandwidth of a small and specific size. It is disabled by default.

  TiFlash I/O Rate Limiter provides a new mechanism to avoid excessive race for I/O resources between read and write tasks. It balances the responses to read and write tasks, and limits the rate automatically according to read/write workload.

  User document

- **Improve stability of TiKV flow control**

  TiKV introduces a new flow control mechanism to replace the previous RocksDB write stall mechanism. Compared with the write stall mechanism, this new mechanism reduces the impact on the stability of foreground write.

  In specific, when the stress of RocksDB compaction accumulates, flow control is performed at the TiKV scheduler layer instead of the RocksDB layer, to avoid the following issues:

  - Raftstore is stuck, which is caused by RocksDB write stall.
  - Raft election times out, and the node leader is transferred as a result.

  This new mechanism improves the flow control algorithm to mitigate QPS decrease when the write traffic is high.

  User document, #10137

- **Detect and recover automatically from impact caused by a single slow TiKV node in a cluster**

  TiKV introduces the slow node detection mechanism. This mechanism calculates a score by inspecting the rate of TiKV Raftstore, and then reports the score to PD through store heartbeats. Meanwhile, it adds the `evict-slow-store-scheduler` scheduler on PD to automatically evict the leader on a single slow TiKV node. In this way, the impact on the whole cluster is mitigated. At the same time, more alert items about slow nodes are introduced to help you quickly pinpoint and solve problems.

  User document, #10539

16.9.5.2.4 Data Migration

- **Simplify operations of Data Migration (DM)**

  DM v2.0.6 can automatically identify the change event (failover or plan change) of the data source using VIP, and can automatically connect to a new data source instance, to reduce data replication latency and simplify operation procedures.

- TiDB Lightning supports customized line terminators in the CSV data, and is compatible with the MySQL LOAD DATA CSV data formats. You can then use TiDB Lightning directly in your data flow architecture.

  #1297

16.9.5.2.5 TiDB data share subscription

TiCDC supports using the HTTP protocol (OpenAPI) to manage TiCDC tasks, which is a more user-friendly operation method for both Kubernetes and self-hosted environments. (Experimental feature)

#2411

16.9.5.2.6 Deployment and operations

Support running the `tiup playground` command on Mac computers with Apple M1 chips.

16.9.5.3 Feature Enhancements

- Tools
  - TiCDC
    - Add the binary MQ format designed for TiDB. It is more compact than the open protocols based on JSON #1621
    - Remove support for file sorter #2114
    - Support log rotation configurations #2182
  - TiDB Lightning
    - Support customized line terminators (except `\r` and `\n`) #1297
    - Support expression index and the index that depends on virtual generated columns #1407
  - Dumpling
    - Support backing up MySQL compatible databases but does not support `START TRANSACTION ... WITH CONSISTENT SNAPSHOT` or`SHOW CREATE TABLE` #311

16.9.5.4 Improvements

- TiDB
    - Support pushing down the built-in function `json_unquote()` to TiKV #24415
    - Support removing the `union` branch from the dual table #25614
    - Optimize the aggregate operator's cost factor #25241
    - Allow the MPP outer join to choose the build table based on the table row count #25142
    - Support balancing the MPP query workload among different TiFlash nodes based on Regions #24724
    - Support invalidating stale Regions in the cache after the MPP query is executed #24432
    - Improve the MySQL compatibility of the built-in function`str_to_date` for the format specifiers `%b/%M/%r/%T` #25767
    - Fix the issue that inconsistent binding caches might be created in multiple TiDB after recreating different bindings for the same query #26015
    - Fix the issue that the existing bindings cannot be loaded into cache after upgrade #23295
    - Support ordering the result of `SHOW BINDINGS` by (`original_sql`, `update_time`) #26139
    - Improve the logic of query optimization when bindings exist, and reduce optimization times of a query #26141
    - Support completing the garbage collection automatically for the bindings in the "deleted" status #26206
    - Support showing whether a binding is used for query optimization in the result of `EXPLAIN VERBOSE` #26930
    - Add a new status variation `last_plan_binding_update_time` to view the timestamp corresponding to the binding cache in the current TiDB instance #26340
    - Support reporting an error when starting binding evolution or running `admin evolve bindings` to ban the baseline evolution (currently disabled in the TiDB Self-Hosted version because it is an experimental feature) affecting other features #26333
- PD
    - Add more QPS dimensions for hot Region scheduling, and support adjusting the priority of the scheduling #3869
    - Support hot Region balance scheduling for the write hotspot of TiFlash #3900
- TiFlash
    - Add operators: `MOD / %`, `LIKE`
    - Add string functions: `ASCII()`, `COALESCE()`, `LENGTH()`, `POSITION()`, `TRIM()`
    - Add mathematical functions: `CONV()`, `CRC32()`, `DEGREES()`, `EXP()`, `LN()`, `LOG()`, `LOG10()`, `LOG2()`, `POW()`, `RADIANS()`, `ROUND(decimal)`, `SIN()`, `MOD()`

- Add date functions: `ADDDATE(string, real)`, `DATE_ADD(string, real)`, `DATE_ADD()`
- Add other functions: `INET_NTOA()`, `INET_ATON()`, `INET6_ATON`, `INET6_NTOA()`
- Support Shuffled Hash Join calculation and Shuffled Hash Aggregation calculation in the MPP mode when a new collation is enabled
- Optimize basic code to improve MPP performance
- Support casting the `STRING` type to the `DOUBLE` type
- Optimize the non-joined data in right outer join using multiple threads
- Support automatically invalidating stale Regions in MPP queries

- Tools

  - TiCDC

    - Add the concurrency limit to the incremental scan of kv client #1899
    - TiCDC can always pull the old value internally #2271
    - TiCDC can fail and exit fast when unrecoverable DML errors occur #1928
    - `resolve lock` cannot be run immediately after a Region is initialized #2235
    - Optimize workerpool to reduce the number of goroutines under high concurrency #2201

  - Dumpling

    - Support always splitting TiDB v3.x tables through `tidb_rowid` to save TiDB memory #301
    - Reduce access of Dumpling to the `information_schema` to improve stability #305

16.9.5.5 Bug Fixes

- TiDB

  - Fix the issue that an incorrect result is returned when using merge join on the `SET` type column #25669
  - Fix the data corruption issue in the `IN` expression's arguments #25591
  - Avoid the sessions of GC being affected by global variables #24976
  - Fix the panic issue that occurs when using `limit` in the window function queries #25344
  - Fix the wrong value returned when querying a partitioned table using `Limit` #24636
  - Fix the issue that `IFNULL` does not correctly take effect on the `ENUM` or `SET` type column #24944
  - Fix the wrong results caused by changing the `count` in the join subqueries to `first_row` #24865
  - Fix the query hang issue that occurs when `ParallelApply` is used under the `TopN` operator #24930

- Fix the issue that more results than expected are returned when executing SQL statements using multi-column prefix indexes #24356
- Fix the issue that the `<=>` operator cannot correctly take effect #24477
- Fix the data race issue of the parallel `Apply` operator #23280
- Fix the issue that the `index out of range` error is reported when sorting the IndexMerge results of the PartitionUnion operator #23919
- Fix the issue that setting the `tidb_snapshot` variable to an unexpectedly large value might damage the transaction isolation #25680
- Fix the issue that the ODBC-styled constant (for example, `{d '2020-01-01'}`) cannot be used as the expression #25531
- Fix the issue that `SELECT DISTINCT` converted to `Batch Get` causes incorrect results #25320
- Fix the issue that backing off queries from TiFlash to TiKV cannot be triggered #23665 #24421
- Fix the `index-out-of-range` error that occurs when checking `only_full_group_by` ↪ #23839)
- Fix the issue that the result of index join in correlated subqueries is wrong #25799

- TiKV

    - Fix the wrong `tikv_raftstore_hibernated_peer_state` metric #10330
    - Fix the wrong arguments type of the `json_unquote()` function in the coprocessor #10176
    - Skip clearing callback during graceful shutdown to avoid breaking ACID in some cases #10353 #10307
    - Fix a bug that the read index is shared for replica reads on a Leader #10347
    - Fix the wrong function that casts `DOUBLE` to `DOUBLE` #25200

- PD

    - Fix the issue that the expected scheduling cannot be generated due to scheduling conflicts among multiple schedulers #3807 #3778

- TiFlash

    - Fix the issue that TiFlash keeps restarting because of the split failure
    - Fix the potential issue that TiFlash cannot delete the delta data
    - Fix a bug that TiFlash adds wrong padding for non-binary characters in the `CAST` function
    - Fix the issue of incorrect results when handling aggregation queries with complex `GROUP BY` columns
    - Fix the TiFlash panic issue that occurs under heavy write pressure
    - Fix the panic that occurs when the right jon key is not nullable and the left join key is nullable
    - Fix the potential issue that the `read-index` requests take a long time
    - Fix the panic issue that occurs when the read load is heavy

    - Fix the panic issue that might occur when the `Date_Format` function is called with the `STRING` type argument and `NULL` values

- Tools

    - TiCDC

        - Fix a bug that TiCDC owner exits abnormally when refreshing the checkpoint #1902
        - Fix a bug that changefeed fails immediately after its successful creation #2113
        - Fix a bug that changefeed fails due to the invalid format of rules filter #1625
        - Fix the potential DDL loss issue when the TiCDC owner panics #1260
        - Fix the CLI compatibility issue with 4.0.x clusters on the default sort-engine option #2373
        - Fix a bug that changefeed might be reset unexpectedly when TiCDC gets the `ErrSchemaStorageTableMiss` error #2422
        - Fix a bug that changefeed cannot be removed when TiCDC gets the `ErrGCTTLExceeded` error #2391
        - Fix a bug that TiCDC fails to synchronize large tables to cdclog #1259 #2424
        - Fix a bug that multiple processors might write data to the same table when TiCDC is rescheduling the table #2230

    - Backup & Restore (BR)

        - Fix a bug that BR skips restoring all system tables during the restore #1197 #1201
        - Fix a bug that BR misses DDL operations when restoring cdclog #870

    - TiDB Lightning

        - Fix a bug that TiDB Lightning fails to parse the `DECIMAL` data type in Parquet file #1272
        - Fix a bug that TiDB Lightning reports the "Error 9007: Write conflict" error when restoring table schemas #1290
        - Fix a bug that TiDB Lightning fails to import data due to the overflow of int handle #1291
        - Fix a bug that TiDB Lightning might get a checksum mismatching error due to data loss in the local backend mode #1403
        - Fix the Lightning incompatibility issue with clustered index when TiDB Lightning is restoring table schemas #1362

    - Dumpling

        - Fix a bug that the data export fails because the Dumpling GC safepoint is set too late #290
        - Fix the Dumpling getting stuck issue when exporting table names from the upstream database in certain MySQL versions #322

16.10 v5.1

16.10.1 TiDB 5.1.5 Release Notes

Release date: December 28, 2022

TiDB version: 5.1.5

Quick access: Quick start | Production deployment

16.10.1.1 Compatibility changes

- PD
    - Disable compiling swagger server by default #4932

16.10.1.2 Bug fixes

- TiDB
    - Fix the issue that the window function causes TiDB to panic instead of reporting an error #30326
    - Fix the wrong result that occurs when enabling dynamic mode in partitioned tables for TiFlash #37254
    - Fix wrong results of `GREATEST` and `LEAST` when passing in unsigned `BIGINT` arguments #30101
    - Fix wrong results of deleting data of multiple tables using `left join` #31321
    - Fix the issue that the result of `concat(ifnull(time(3)))` in TiDB is different from that in MySQL #29498
    - Fix the issue that the SQL statements that contain `cast(integer as char) union string` return wrong results #29513
    - Fix the issue that `INL_HASH_JOIN` might hang when used with `LIMIT` #35638
    - Fix the wrong `ANY_VALUE` result that occurs when a Region returns empty data #30923
    - Fix wrong results of index join caused by an innerWorker panic #31494
    - Fix the issue that a SQL operation is canceled when its JSON type column joins its `CHAR` type column #29401
    - Fix the issue that the background HTTP service of TiDB might not exit successfully and makes the cluster in an abnormal state #30571
    - Fix the issue that concurrent column type change causes inconsistency between the schema and the data #31048
    - Fix the issue that `KILL TIDB` cannot take effect immediately on idle connections #24031
    - Fix the bug that setting any session variable will make `tidb_snapshot` fail to work #35515

- Fix the issue that the Region cache is not cleaned up in time when the Region is merged #37141
- Fix the panic issue caused by the connection array race in the KV client #33773
- Fix the issue that when TiDB Binlog is enabled, executing the `ALTER SEQUENCE` ↪ statement might cause a wrong metadata version and cause Drainer to exit #36276
- Fix the bug that TiDB may panic when querying statement summary tables #35340
- Fix the issue that TiDB gets the wrong result when using TiFlash to scan tables with empty range although TiFlash does not support reading tables with empty range yet #33083
- Fix the issue that the `avg()` function returns `ERROR 1105 (HY000): other` ↪ `error for mpp stream: Could not convert to the target type - -` ↪ `value is out of range.` when queried from TiFlash #29952
- Fix the issue that `ERROR 1105 (HY000): close of nil channel` is returned when using `HashJoinExec` #30289
- Fix the issue that TiKV and TiFlash return different results when querying logical operations #37258
- Fix the issue that the `EXECUTE` statement might throw an unexpected error in specific scenarios #37187
- Fix the planner wrong behaviors that occur when `tidb_opt_agg_push_down` and `tidb_enforce_mpp` are enabled #34465
- Fix a bug that TiDB might send coprocessor requests when executing the `SHOW` ↪ `COLUMNS` statement #36496
- Add warnings for `lock tables` and `unlock tables` when the `enable-table-` ↪ `lock` flag is not enabled #28967
- Fix the issue that range partitions allow multiple `MAXVALUE` partitions #36329

- TiKV

  - Fix the issue of time parsing error that occurs when the `DATETIME` values contain a fraction and `Z` #12739
  - Fix a bug that replica reads might violate the linearizability #12109
  - Fix a bug that Regions might be overlapped if Raftstore is busy #13160
  - Fix the TiKV panic issue that occurs when applying snapshot is aborted #11618
  - Fix a bug that TiKV might panic if it has been running for 2 years or more #11940
  - Fix the panic issue that might occur when the source peer catches up logs by snapshot in the Region merge process #12663
  - Fix the issue that TiKV panics when performing type conversion for an empty string #12673
  - Fix a bug that stale messages cause TiKV to panic #12023
  - Fix the panic issue that might occur when a peer is being split and destroyed at the same time #12825

- Fix the TiKV panic issue that occurs when the target peer is replaced with the peer that is destroyed without being initialized when merging a Region #12048
- Fix the issue that TiKV reports the `invalid store ID 0` error when using Follower Read #12478
- Fix the possible duplicate commit records in pessimistic transactions when async commit is enabled #12615
- Support configuring the `unreachable_backoff` item to avoid Raftstore broadcasting too many messages after one peer becomes unreachable #13054
- Fix the issue that successfully committed optimistic transactions may report the `Write Conflict` error when the network is poor #34066
- Fix the wrong expression of `Unified Read Pool CPU` in dashboard #13086

- PD

    - Fix the issue that a removed tombstone store appears again after the PD leader transfer #4941
    - Fix the issue that scheduling cannot start immediately after the PD leader transfer #4769
    - Fix the wrong status code of `not leader` #4797
    - Fix the issue that PD cannot correctly handle dashboard proxy requests #5321
    - Fix a bug of TSO fallback in some corner cases #4884
    - Fix the issue that the TiFlash learner replica might not be created in specific scenarios #5401
    - Fix the issue that the label distribution has residual labels in the metrics #4825
    - Fix the issue that when there exists a Store with large capacity (2T for example), fully allocated small Stores cannot be detected, which results in no balance operator being generated #4805
    - Fix the issue that schedulers do not work when `SchedulerMaxWaitingOperator` is set to `1` #4946

- TiFlash

    - Fix incorrect `microsecond` when casting string to datetime #3556
    - Fix the panic issue that occurs when TLS is enabled #4196
    - Fix a bug that TiFlash might crash due to an error in parallel aggregation #5356
    - Fix the issue that a query containing `JOIN` might be hung if an error occurs #4195
    - Fix the issue that the function `OR` returns wrong results #5849
    - Fix the bug that invalid storage directory configurations lead to unexpected behaviors #4093
    - Fix potential data inconsistency after a lot of INSERT and DELETE operations #4956
    - Fix a bug that data not matching any region range remains on a TiFlash node #4414
    - Fix the potential query error after adding columns under heavy read workload #3967

- Fix repeated crashes caused by the `commit state jump backward` errors #2576
- Fix potential errors when querying on a table with many delete operations #4747
- Fix the issue that the date format identifies `''` as an invalid separator #4036
- Fix the wrong result that occurs when casting `DATETIME` to `DECIMAL` #4151
- Fix the bug that some exceptions are not handled properly #4101
- Fix the issue that `Prepare Merge` might damage the metadata of the raft store and cause TiFlash to restart #3435
- Fix a bug that an MPP query might fail due to random gRPC keepalive timeout #4662
- Fix the issue that the result of `IN` is incorrect in multi-value expressions #4016
- Fix a bug that MPP tasks might leak threads forever #4238
- Fix the issue that expired data is recycled slowly #4146
- Fix the overflow that occurs when casting `FLOAT` to `DECIMAL` #3998
- Fix the issue that logical operators return wrong results when the argument type is UInt8 #6127
- Fix the potential `index out of bounds` error if calling `json_length` with empty string #2705
- Fix wrong decimal comparison results in corner cases #4512
- Fix `TiFlash_schema_error` reported when `NOT NULL` columns are added #4596
- Fix the issue that TiFlash bootstrap fails when `0.0` is used as the default value for integers, for example, `` `i` int(11)NOT NULL DEFAULT '0.0' `` #3157

- Tools

  - TiDB Binlog

    - Fix the issue that Drainer cannot send requests to Pump correctly when `compressor` is set to `zip` #1152

  - Backup & Restore (BR)

    - Fix the issue that system tables cannot be restored because concurrently backing up system tables makes the table name fail to update #29710

  - TiCDC

    - Fix data loss that occurs in special incremental scanning scenarios #5468
    - Fix the issue that there are no sorter metrics #5690
    - Fix excessive memory usage by optimizing the way DDL schemas are buffered #1386

16.10.2 TiDB 5.1.4 Release Notes

Release Date: February 22, 2022

TiDB version: 5.1.4

16.10.2.1 Compatibility changes

- TiDB
    - Change the default value of the system variable `tidb_analyze_version` from `2` to `1` #31748
    - Since v5.1.4, if TiKV is configured with `storage.enable-ttl = true`, the requests from TiDB are rejected, because the TTL feature of TiKV only supports the RawKV mode #27303
- Tools
    - TiCDC
        - Set the default value of `max-message-bytes` to 10M #4041

16.10.2.2 Improvements

- TiDB
    - Support partition pruning for the built-in `IN` expression in Range partition tables #26739
    - Improve the accuracy of tracking memory usage when `IndexJoin` is executed #28650
- TiKV
    - Update the proc filesystem (procfs) to v0.12.0 #11702
    - Improve the error log report in the Raft client #11959
    - Increase the speed of inserting SST files by moving the verification process to the `Import` thread pool from the `Apply` thread pool #11239
- PD
    - Speed up the exit process of schedulers #4146
- TiFlash
    - Support pushing down `ADDDATE()` and `DATE_ADD()` to TiFlash
    - Support pushing down `INET6_ATON()` and `INET6_NTOA()` to TiFlash
    - Support pushing down `INET_ATON()` and `INET_NTOA()` to TiFlash
    - Increase the max supported depth of an expression or a plan tree in a DAG request from `100` to `200`
- Tools

- TiCDC
  - Add the exponential backoff mechanism for restarting a changefeed. #3329
  - Reduce the replication latency when replicating many tables #3900
  - Add metrics for observing the remaining time of incremental scan #2985
  - Reduce the count of "EventFeed retry rate limited" logs #4006
  - Add more Prometheus and Grafana monitoring metrics and alerts, including `no owner alert`, `mounter row`, `table sink total row`, and `buffer sink total row` #4054 #1606
  - Optimize rate limiting control on TiKV reloads to reduce gPRC congestion during changefeed initialization #3110
  - Reduce the time for the KV client to recover when a TiKV store is down #3191

16.10.2.3 Bug fixes

- TiDB
  - Fix a memory leak bug that occurs when the system variable `tidb_analyze_version` is set to `2` #32499
  - Fix the issue that the `MaxDays` and `MaxBackups` configurations do not take effect for the slow log #25716
  - Fix the issue that executing the `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` statement gets panic #28078
  - Fix the wrong result that might occur when performing `JOIN` on `ENUM` type columns #27831
  - Fix the issue that INDEX HASH JOIN returns the `send on closed channel` error #31129
  - Fix the issue that using the `BatchCommands` API might block sending TiDB requests to TiKV in some rare cases #32500
  - Fix the issue of potential data index inconsistency in optimistic transaction mode #30410
  - Fix the issue that window functions might return different results when using a transaction or not #29947
  - Fix the issue that the length information is wrong when casting `Decimal` to `String` #29417
  - Fix the issue that the `GREATEST` function returns incorrect result that occurs when setting the `tidb_enable_vectorized_expression` vectorized expression to `off` #29434
  - Fix the issue that the optimizer might cache invalid plans for `join` in some cases #28087
  - Fix wrong results of the `microsecond` and `hour` functions in vectorized expressions #29244 #28643
  - Fix the TiDB panic when executing the `ALTER TABLE.. ADD INDEX` statement in some cases #27687

- Fix a bug that the availability detection of MPP node does not work in some corner cases #3118
- Fix the `DATA RACE` issue when assigning `MPP task ID` #27952
- Fix the `INDEX OUT OF RANGE` error for a MPP query after deleting an empty `dual table` #28250
- Fix the issue of false positive error log `invalid cop task execution summaries length` for MPP queries #1791
- Fix the issue that SET GLOBAL tidb_skip_isolation_level_check=1 doesn't affect new session settings #27897
- Fix the `index out of range` issue that occurs when `tiup bench` runs for a long time #26832

- TiKV

  - Fix a bug that TiKV cannot delete a range of data (`unsafe_destroy_range` cannot be executed) when the GC worker is busy #11903
  - Fix the issue that destroying a peer might cause high latency #10210
  - Fix a bug that the `any_value` function returns a wrong result when regions are empty #11735
  - Fix the issue that deleting an uninitialized replica might cause an old replica to be recreated #10533
  - Fix the metadata corruption issue when `Prepare Merge` is triggered after a new election is finished but the isolated peer is not informed #11526
  - Fix the deadlock issue that happens occasionally when coroutines run too fast #11549
  - Fix the potential deadlock and memory leak issues when profiling flame graphs #11108
  - Fix the rare data inconsistency issue when retrying a prewrite request in pessimistic transactions #11187
  - Fix a bug that the configuration `resource-metering.enabled` does not work #11235
  - Fix the issue that some coroutines leak in `resolved_ts` #10965
  - Fix the issue of reporting false "GC can not work" alert under low write flow #9910
  - Fix a bug that tikv-ctl cannot return the correct Region-related information #11393
  - Fix the issue that a down TiKV node causes the resolved timestamp to lag #11351
  - Fix a panic issue that occurs when Region merge, ConfChange, and Snapshot happen at the same time in extreme conditions #11475
  - Fix the issue that TiKV cannot detect the memory lock when TiKV performs a reverse table scan #11440
  - Fix the issue of negative sign when the decimal divide result is zero #29586
  - Fix a memory leak caused by the monitoring data of statistics threads #11195
  - Fix the issue of TiCDC panic that occurs when the downstream database is missing #11123

- Fix the issue that TiCDC adds scan retries frequently due to the Congest error #11082
- Fix the issue that batch messages are too large in Raft client implementation #9714
- Collapse some uncommon storage-related metrics in Grafana dashboard #11681

- PD

  - Fix a bug that the schedule generated by the region scatterer might decrease the number of peers #4565
  - Fix the issue that Region statistics are not affected by `flow-round-by-digit` #4295
  - Fix slow leader election caused by stucked region syncer #3936
  - Support that the evict leader scheduler can schedule regions with unhealthy peers #4093
  - Fix the issue that the cold hotspot data cannot be deleted from the hotspot statistics #4390
  - Fix a panic issue that occurs after the TiKV node is removed #4344
  - Fix the issue that the scheduling operator cannot fail fast because the target store is down #3353

- TiFlash

  - Fix the issue that the `str_to_date()` function incorrectly handles leading zeros when parsing microseconds
  - Fix the TiFlash crash problem when the memory limit is enabled
  - Fix the issue that when an input time is earlier than 1970-01-01 00:00:01 UTC, the behavior of `unix_timestamp` is inconsistent with that of TiDB or MySQL
  - Fix the potential data inconsistency caused by widening the primary key column when the primary key is handle
  - Fix the overflow bug and the issue of reporting `Can't compare` error when comparing data in the `DECIMAL` data type
  - Fix the unexpected error of `3rd arguments of function substringUTF8 must be constants.`
  - Fix the issue that TiFlash fails to start on platforms without the `nsl` library
  - Fix the overflow bug when casting data to the `DECIMAL` data type
  - Fix the issue that the `castStringAsReal` behavior is inconsistent in TiFlash and in TiDB/TiKV
  - Fix the issue that TiFlash might return the `EstablishMPPConnection` error after it is restarted
  - Fix the issue that obsolete data cannot be reclaimed after setting the number of TiFlash replicas to 0
  - Fix the issue that the `CastStringAsDecimal` behavior is inconsistent in TiFlash and in TiDB/TiKV
  - Fix the issue that queries with the `where <string>` clause return wrong results

- Fix the issue that TiFlash might panic when an MPP query is stopped
- Fix the unexpected error of `Unexpected type of column: Nullable(Nothing)`

- Tools
  - TiCDC
    - Fix a bug that MySQL sink generates duplicated `replace` SQL statements if `batch-replace-enable` is disabled #4501
    - Fix the issue that the `cached region` monitoring metric is negative #4300
    - Fix the issue that replication cannot be performed when `min.insync.replicas` is smaller than `replication-factor` #3994
    - Fix the potential panic issue that occurs when a replication task is removed #3128
    - Fix the issue of potential data loss caused by inaccurate checkpoint #3545
    - Fix the potential issue that the deadlock causes a replication task to get stuck #4055
    - Fix the issue that special comments in DDL statements cause the replication task to stop #3755
    - Fix a bug that EtcdWorker might hang the owner and processor #3750
    - Fix the issue that `stopped` changefeeds resume automatically after a cluster upgrade #3473
    - Fix the issue that default values cannot be replicated #3793
    - Fix data inconsistency caused by TiCDC default value padding exceptions #3918 #3929
    - Fix a bug that an owner gets stuck when a PD leader shuts down and transfers to a new node #3615
    - Fix the TiCDC panic issue that occurs when manually cleaning the task status in etcd #2980
    - Fix the issue that the service cannot be started because of a timezone issue in the RHEL release #3584
    - Fix the issue of overly frequent warnings caused by MySQL sink deadlock #2706
    - Fix the bug that the `enable-old-value` configuration item is not automatically set to `true` on the Canal protocol #3676
    - Fix the issue that Avro sink does not support parsing JSON type columns #3624
    - Fix the negative value error in the changefeed checkpoint lag #3010
    - Fix the OOM issue in the container environment #1798
    - Fix the TiCDC replication interruption issue when multiple TiKVs crash or during a forced restart #3288
    - Fix the memory leak issue after processing DDLs #3174
    - Fix the issue that changefeed does not fail fast enough when the ErrGCTTLExceeded error occurs #3111
    - Fix the issue that TiCDC replication task might terminate when the upstream TiDB instance unexpectedly exits #3061

- Fix the issue that TiCDC process might panic when TiKV sends duplicate requests to the same Region #2386
- Fix the issue that Kafka may send excessively large messages by setting the default value of `max-message-bytes` to `10M` #3081
- Fix the issue that TiCDC sync task might pause when an error occurs during writing a Kafka message #2978

- Backup & Restore (BR)
  - Fix the potential issue that Regions might be unevenly distributed after a restore operation is finished #30425 #31034
- TiDB Binlog
  - Fix the issue that DBaaS importing CSV fails with InvalidRange if CSV file size is about 256MB and `strict-format` is `true` #27763
- TiDB Lightning
  - Fix the issue that TiDB Lightning does not report errors when the S3 storage path does not exist #28031 #30709

16.10.3 TiDB 5.1.3 Release Note

Release date: December 3, 2021

TiDB version: 5.1.3

16.10.3.1 Bug fix

- TiKV
  - Fix the issue that the `GcKeys` task does not work when it is called by multiple keys. Caused by this issue, compaction filter GC might not drop the MVCC deletion information. #11217

16.10.4 TiDB 5.1.2 Release Notes

Release Date: September 27, 2021

TiDB version: 5.1.2

16.10.4.1 Compatibility changes

- TiDB
  - The following bug fixes change execution results, which might cause upgrade incompatibilities:

- Fix the issue that `greatest(datetime)union null` returns empty string #26532
- Fix the issue that the `having` clause might not work correctly #26496
- Fix the wrong execution results that occur when the collations around the `between` expression are different #27146
- Fix the wrong execution results that occur when the column in the `group_concat` function has a non-bin collation #27429
- Fix an issue that using a `count(distinct)` expression on multiple columns returns wrong result when the new collation is enabled #27091
- Fix the result wrong that occurs when the argument of the `extract` function is a negative duration #27236
- Fix the issue that inserting an invalid date does not report an error when the `SQL_MODE` is 'STRICT_TRANS_TABLES' #26762
- Fix the issue that using an invalid default date does not report an error when the `SQL_MODE` is 'NO_ZERO_IN_DATE' #26766

- Tools
  - TiCDC
    - Set the compatible version from `5.1.0-alpha` to `5.2.0-alpha` #2659

16.10.4.2 Improvements

- TiDB
  - Trigger auto-analyze by histogram row count and increase the accuracy of this trigger action #24237
- TiKV
  - Support dynamically modifying TiCDC configurations #10645
  - Reduce the size of Resolved TS message to save network bandwidth #2448
  - Limit the counts of peer stats in the heartbeat message reported by a single store #10621
- PD
  - Allow empty Regions to be scheduled and use a separate tolerance configuration in scatter range scheduler #4117
  - Improve the performance of synchronizing Region information between PDs #3933
  - Support dynamically adjusting the retry limit of a store based on the generated operator #3744
- TiFlash

- Support the `DATE()` function
- Add Grafana panels for write throughput per instance
- Optimize the performance of the `leader-read` process
- Accelerate the process of canceling MPP tasks

- Tools

  - TiCDC

    - Optimize memory management when the Unified Sorter is using memory to sort data #2553
    - Optimize workerpool for fewer goroutines when concurrency is high #2211
    - Reduce goroutine usage when a table's Region transfer away from a TiKV node #2284
    - Add a global gRPC connection pool and share gRPC connections among KV clients #2534
    - Prohibit operating TiCDC clusters across major and minor versions #2599

  - Dumpling

    - Support backing up MySQL-compatible databases that do not support `START TRANSACTION ... WITH CONSISTENT SNAPSHOT` and `SHOW CREATE TABLE` #309

16.10.4.3 Bug fixes

- TiDB

  - Fix the potential wrong results of index hash join when the hash column is the `ENUM` type #27893
  - Fix a batch client bug that recycle idle connection might block sending requests in some rare cases #27678
  - Fix the issue that the overflow check of the `FLOAT64` type is different with that of MySQL #23897
  - Fix the issue that TiDB returns an `unknow` error while it should return the `pd is timeout` error #26147
  - Fix the wrong character set and collation for the `case when` expression #26662
  - Fix the potential `can not found column in Schema column` error for MPP queries #28148
  - Fix a bug that TiDB might panic when TiFlash is shutting down #28096
  - Fix the issue of wrong range caused by using `enum like 'x%'` #27130
  - Fix the Common Table Expression (CTE) dead lock issue when used with IndexLookupJoin #27410
  - Fix a bug that retryable deadlocks are incorrectly recorded into the `INFORMATION_SCHEMA.DEADLOCKS` table #27400
  - Fix the issue that the `TABLESAMPLE` query result from partitioned tables is not sorted as expected #27349

- Remove the unused `/debug/sub-optimal-plan` HTTP API #27265
- Fix a bug that the query might return wrong results when the hash partitioned table deals with unsigned data #26569
- Fix a bug that creating partition fails if `NO_UNSIGNED_SUBTRACTION` is set #26765
- Fix the issue that the `distinct` flag is missing when `Apply` is converted to `Join` #26958
- Set a block duration for the newly recovered TiFlash node to avoid blocking queries during this time #26897
- Fix a bug that might occur when the CTE is referenced more than once #26212
- Fix a CTE bug when MergeJoin is used #25474
- Fix a bug that the `SELECT FOR UPDATE` statement does not correctly lock the data when a normal table joins a partitioned table #26251
- Fix the issue that the `SELECT FOR UPDATE` statement returns an error when a normal table joins a partitioned table #26250
- Fix the issue that `PointGet` does not use the lite version of resolving lock #26562

- TiKV

  - Fix a panic issue that occurs after TiKV is upgraded from v3.x to later versions #10902
  - Fix the potential disk full issue caused by corrupted snapshot files #10813
  - Make the slow log of TiKV coprocessor only consider the time spent on processing requests #10841
  - Drop log instead of blocking threads when the slogger thread is overloaded and the queue is filled up #10841
  - Fix a panic issue that occurs when processing Coprocessor requests times out #10852
  - Fix the TiKV panic issue that occurs when upgrading from a pre-5.0 version with Titan enabled #10842
  - Fix the issue that TiKV of a newer version cannot be rolled back to v5.0.x #10842
  - Fix the issue that TiKV might delete files before ingesting data to RocksDB #10438
  - Fix the parsing failure caused by the left pessimistic locks #26404

- PD

  - Fix the issue that PD does not fix the down peers in time #4077
  - Fix the issue that the replica count of the default placement rules stays constant after `replication.max-replicas` is updated #3886
  - Fix a bug that PD might panic when scaling out TiKV #3868
  - Fix a bug that the hot Region scheduler cannot work when the cluster has the evict leader scheduler #3697

- TiFlash

- Fix the issue of unexpected results when TiFlash fails to establish MPP connections
- Fix the potential issue of data inconsistency that occurs when TiFlash is deployed on multiple disks
- Fix a bug that MPP queries get wrong results when TiFlash server is under high load
- Fix a potential bug that MPP queries hang forever
- Fix the panic issue when operating store initialization and DDL simultaneously
- Fix a bug of incorrect results that occurs when queries contain filters like `CONSTANT` ↪ , `<`, `<=`, `>`, `>=`, or `COLUMN`
- Fix the potential panic issue when `Snapshot` is applied simultaneously with multiple DDL operations
- Fix the issue that the store size in metrics is inaccurate under heavy writing
- Fix the potential issue that TiFlash cannot garbage-collect the delta data after running for a long time
- Fix the issue of wrong results when the new collation is enabled
- Fix the potential panic issue that occurs when resolving locks
- Fix a potential bug that metrics display wrong values

- Tools
  - Backup & Restore (BR)
    - Fix the issue that the average speed is not accurate during data backup and restore #1405
  - Dumpling
    - Fix the issue that Dumpling is pending when `show table status` returns incorrect results in some MySQL versions (8.0.3 and 8.0.23) #322
    - Fix the CLI compatibility issue with 4.0.x clusters on the default `sort-` ↪ `engine` option #2373
  - TiCDC
    - Fix a bug that the JSON encoding might cause panic when processing a string type value that is `string` or `[]byte` #2758
    - Reduce gRPC window size to avoid OOM #2673
    - Fix a gRPC `keepalive` error under high memory pressure #2202
    - Fix a bug that an unsigned `tinyint` causes TiCDC to panic #2648
    - Fix an empty value issue in TiCDC Open Protocol. An empty value is no longer output when there is no change in one transaction. #2612
    - Fix a bug in DDL handling during manual restarts #2603
    - Fix the issue that `EtcdWorker`'s snapshot isolation might be wrongly violated when managing the metadata #2559
    - Fix a bug that multiple processors might write data to the same table when TiCDC is rescheduling the table #2230
    - Fix a bug that changefeed might be reset unexpectedly when TiCDC gets the `ErrSchemaStorageTableMiss` error #2422

- Fix a bug that changefeed cannot be removed when TiCDC gets the `ErrGCTTLExceeded` error #2391
- Fix a bug that TiCDC fails to synchronize large tables to cdclog #1259 #2424

16.10.5 TiDB 5.1.1 Release Notes

Release Date: July 30, 2021

TiDB version: 5.1.1

16.10.5.1 Compatibility changes

- TiDB
  - For TiDB clusters upgrade from v4.0 to v5.1, the default value of `tidb_multi_statement_mode` is `OFF`. It is recommended to use the multi-statement feature of your client library instead. See the documentation on `tidb_multi_statement_mode` for details. #25751
  - Change the default value of the `tidb_stmt_summary_max_stmt_count` variable from `200` to `3000` #25874
  - Require the `SUPER` privilege to access the `table_storage_stats` table #26352
  - Require the `SELECT` privilege on `mysql.user` to access the `information_schema.user_privileges` table to show other user's privileges #26311
  - Require the `CONFIG` privilege to access the `information_schema.cluster_hardware` table #26297
  - Require the `PROCESS` privilege to access the `information_schema.cluster_info` table #26297
  - Require the `PROCESS` privilege to access the `information_schema.cluster_load` table #26297
  - Require the `PROCESS` privilege to access the `information_schema.cluster_systeminfo` table #26297
  - Require the `PROCESS` privilege to access the `information_schema.cluster_log` table #26297
  - Require the `CONFIG` privilege to access the `information_schema.cluster_config` table #26150

16.10.5.2 Feature enhancements

- TiDB Dashboard
  - Support OIDC SSO. By setting the OIDC-compatible SSO services (such as Okta and Auth0), users can log into TiDB Dashboard without entering the SQL password. #3883
- TiFlash
  - Support the `HAVING()` function in DAG requests

16.10.5.3 Improvements

- TiDB
    - Announce the general availability (GA) of the Stale Read feature
    - Avoid allocation for `paramMarker` to speed up data insertion #26076
    - Support the stable result mode to make the query results more stable #25995
    - Support pushing down the built-in function `json_unquote()` to TiKV #26265
    - Support retrying MPP queries #26480
    - Change the `LOCK` record into the `PUT` record for the index keys using `point get` or `batch point get` for `UPDATE` reads #26225
    - Forbid creating views from stale queries #26200
    - Thoroughly push down the `COUNT(DISTINCT)` aggregation function in the MPP mode #26194
    - Check the availability of TiFlash before launching MPP queries #26192
    - Do not allow setting the read timestamp to a future time #25763
    - Print log warnings when aggregation functions cannot be pushed down in `EXPLAIN` statements #25737
    - Add the `statements_summary_evicted` table to record the evicted count information of a cluster #25587
    - Improve the MySQL compatibility of the built-in function `str_to_date` for the format specifiers `%b/%M/%r/%T` #25768
- TiKV
    - Make the prewrite requests as idempotent as possible to reduce the chance of undetermined errors #10586
    - Prevent the risk of stack overflow when handling many expired commands #10502
    - Avoid excessive commit request retrying by not using the Stale Read request's `start_ts` to update `max_ts` #10451
    - Handle read ready and write ready separately to reduce read latency #10592
    - Reduce the impact on data import speed when the I/O rate limiting is enabled #10390
    - Improve the load balance between Raft gRPC connections #10495
- Tools
    - TiCDC
        - Remove `file sorter` #2327
        - Improve the error message returned when a PD endpoint misses the certificate #1973
    - TiDB Lightning
        - Add a retry mechanism for restoring schemas #1294
    - Dumpling

- Always split tables using `_tidb_rowid` when the upstream is a TiDB v3.x cluster, which helps reduce TiDB's memory usage #295
- Reduce the frequency of accessing the database metadata to improve Dumpling's performance and stability #315

16.10.5.4 Bug fixes

- TiDB
    - Fix the data loss issue that might occur when changing the column type with `tidb_enable_amend_pessimistic_txn=on` #26203
    - Fix the issue that the behavior of the `last_day` function is incompatible in the SQL mode #26001
    - Fix the panic issue that might occur when `LIMIT` is on top of window functions #25344
    - Fix the issue that committing pessimistic transactions might cause write conflict #25964
    - Fix the issue that the result of index join in correlated subqueries is wrong #25799
    - Fix a bug that the successfully committed optimistic transactions might report commit errors #10468
    - Fix the issue that an incorrect result is returned when using merge join on the `SET` type column #25669
    - Fix a bug that the index keys in a pessimistic transaction might be repeatedly committed #26359
    - Fix the risk of integer overflow when the optimizer is locating partitions #26227
    - Fix the issue that invalid values might be written when casting `DATE` to timestamp #26292
    - Fix the issue that the Coprocessor Cache metrics are not displayed on Grafana #26338
    - Fix the issue of annoying logs caused by telemetry #25760 #25785
    - Fix a bug on the query range of prefix index #26029
    - Fix the issue that concurrently truncating the same partition hangs DDL executions #26229
    - Fix the issue of duplicate `ENUM` items #25955
    - Fix a bug that the CTE iterator is not correctly closed #26112
    - Fix the issue that the `LOAD DATA` statement might abnormally import non-utf8 data #25979
    - Fix the panic issue that might occur when using the window function on the unsigned integer columns #25956
    - Fix the issue that TiDB might panic when resolving async commit locks #25778
    - Fix the issue that Stale Read is not fully compatible with the `PREPARE` statements #25800
    - Fix the issue that the ODBC-styled constant (for example, `{d '2020-01-01'}`) cannot be used as the expression #25531

    - Fix an error that occurs when running TiDB alone #25555

- TiKV

    - Fix the issue that the duration calculation might panic on certain platforms #10569
    - Fix the issue that Load Base Split mistakenly uses the unencoded keys of `batch_get_command` #10542
    - Fix the issue that changing the `resolved-ts.advance-ts-interval` configuration dynamically cannot take effect immediately #10426
    - Fix the issue of follower metadata corruption in rare cases with more than 4 replicas #10225
    - Fix the panic issue that occurs when building a snapshot twice if encryption is enabled #9786 #10407
    - Fix the wrong `tikv_raftstore_hibernated_peer_state` metric #10330
    - Fix the wrong arguments type of the `json_unquote()` function in the coprocessor #10176
    - Fix a bug that the index keys in a pessimistic transaction might be repeatedly committed #10468
    - Fix the issue that the `ReadIndex` request returns stale result right after the leader is transferred #9351

- PD

    - Fix the issue the expected scheduling cannot be generated when the conflict occurs due to multiple schedulers running at the same time #3807 #3778
    - Fix the issue that the scheduler might appear again even if the scheduler is already deleted #2572

- TiFlash

    - Fix the potential panic issue that occurs when running table scan tasks
    - Fix a bug that TiFlash raises the error about `duplicated region` when handling DAQ requests
    - Fix the panic issue that occurs when the read load is heavy
    - Fix the potential panic issue that occurs when executing the `DateFormat` function
    - Fix the potential memory leak issue that occurs when executing MPP tasks
    - Fix the issue of unexpected results when executing the aggregation functions `COUNT` or `COUNT DISTINCT`
    - Fix a potential bug that TiFlash cannot restore data when deployed on multiple disks
    - Fix the issue that TiDB Dashboard cannot display the disk information of TiFlash correctly
    - Fix the potential panic issue that occurs when deconstructing `SharedQueryBlockInputStream` ↪
    - Fix the potential panic issue that occurs when deconstructing `MPPTask`

    - Fix the potential issue of data inconsistency after synchronizing data via snapshot

- Tools

    - TiCDC
        - Fix the support for the new collation feature #2301
        - Fix the issue that an unsynchronized access to a shared map at runtime might cause panic #2300
        - Fix the potential DDL loss issue that occurs when the owner crashes while executing the DDL statement #2290
        - Fix the issue of trying to resolve locks in TiDB prematurely #2188
        - Fix a bug that might cause data loss if a TiCDC node is killed immediately after a table migration #2033
        - Fix the handling logic of `changefeed update` on `--sort-dir` and `--start-ts` #1921
    - Backup & Restore (BR)
        - Fix the issue that the size of the data to restore is incorrectly calculated #1270
        - Fix the issue of missed DDL events that occurs when restoring from cdclog #870
    - TiDB Lightning
        - Fix the issue that TiDB fails to parse the `DECIMAL` type data in Parquet files #1275
        - Fix the issue of integer overflow when calculating key intervals #1291 #1290

16.10.6 TiDB 5.1 Release Notes

Release date: June 24, 2021

TiDB version: 5.1.0

In v5.1, the key new features or improvements are as follows:

- Support the Common Table Expression (CTE) feature of MySQL 8.0 to improve the readability and execution efficiency of SQL statements.
- Support changing column types online to improve code development flexibility.
- Introduce a new statistics type to improve query stability, which is enabled as an experimental feature by default.
- Support the dynamic privilege feature of MySQL 8.0 to implement more fine-grained control over certain operations.
- Support directly reading data from the local replica using the Stale Read feature to reduce read latency and improve query performance (Experimental).
- Add the Lock View feature to facilitate database administrators (DBAs) to observe transaction locking events and troubleshoot deadlock problems (Experimental).
- Add TiKV write rate limiter for background tasks to ensure that the latency of read and write requests is stable.

16.10.6.1 Compatibility changes

> **Note:**
>
> When upgrading from an earlier TiDB version to v5.1, if you want to know the compatibility change notes of all intermediate versions, you can check the Release Notes for the corresponding version.

16.10.6.1.1 System variables

Variable name	Change type	Description
`cte_max_recursion_depth`	Newly added	Controls the maximum recursion depth in Common Table Expressions.
`init_connect`	Newly added	Controls the initial connection to a TiDB server.
`tidb_analyze_version`	Newly added	Controls how TiDB collects statistics. The default value of this variable is 2. This is an experimental feature.

Variable name	Change type	Description
tidb_enable_enhanced_security ↪	Newly added	Indicates whether the TiDB server you are connected to has the Security Enhanced Mode (SEM) enabled. This variable setting cannot be changed without restarting the TiDB server.
tidb_enforce_mpp ↪	Newly added	Controls whether to ignore the optimizer's cost estimation and to forcibly use the MPP mode for query execution. The data type of this variable is `BOOL` and the default value is `false`.

Variable name	Change type	Description
`tidb_partition_prune_mode` ↪	Newly added	Specifies whether to enable dynamic pruning mode for partitioned tables. This feature is experimental. The default value of this variable is `static`, which means the dynamic pruning mode for partitioned tables is disabled by default.

16.10.6.1.2 Configuration file parameters

Configuration file	Configuration item	Change type	Description

Configuration file	Configuration item	Change type	Description
TiDB configuration file	`security.` ↪ `enable-` ↪ `sem`	Newly added	Controls whether to enable the Security Enhanced Mode (SEM). The default value of this configuration item is `false`, which means the SEM is disabled.
TiDB configuration file	`performance` ↪ `.` ↪ `committer` ↪ `-` ↪ `concurrency` ↪	Modified	Controls the concurrency number for requests related to commit operations in the commit phase of a single transaction. The default value is changed from `16` to `128`.

Configuration file	Configuration item	Change type	Description
TiDB configuration file	performance ↪ .tcp-no ↪ -delay	Newly added	Determines whether to enable TCP_NODELAY at the TCP layer. The default value is true, which means TCP_NODELAY is enabled.
TiDB configuration file	performance ↪ . ↪ enforce ↪ -mpp	Newly added	Controls whether TiDB ignores cost estimates of Optimizer at the instance level and enforces the MPP mode. The default value is false. This configuration item controls the initial value of the system variable tidb_enforce_mpp ↪ .

Configuration file	Configuration item	Change type	Description
TiDB configuration file	`pessimistic-txn.deadlock-history-capacity`	Newly added	Sets the maximum number of deadlock events that can be recorded in the `INFORMATION_SCHEMA.DEADLOCKS` table of a single TiDB server. The default value is `10`.
TiKV configuration file	`abort-on-panic`	Newly added	Sets whether the `abort` process allows the system to generate core dump files when TiKV panics. The default value is `false`, which means it is not allowed to generate core dump files.

Configuration file	Configuration item	Change type	Description
TiKV configuration file	`hibernate-regions`	Modified	The default value is changed from `false` to `true`. If a Region is idle for a long time, it is automatically set as hibernated.
TiKV configuration file	`old-value-cache-memory-quota`	Newly added	Sets the upper limit of memory usage by TiCDC old values. The default value is `512MB`.
TiKV configuration file	`sink-memory-quota`	Newly added	Sets the upper limit of memory usage by TiCDC data change events. The default value is `512MB`.

Configuration file	Configuration item	Change type	Description
TiKV configuration file	`incremental-scan-threads`	Newly added	Sets the number of threads for the task of incrementally scanning historical data. The default value is `4`, which means there are four threads for the task.
TiKV configuration file	`incremental-scan-concurrency`	Newly added	Sets the maximum number of concurrent executions for the tasks of incrementally scanning historical data. The default value is `6`, which means that six tasks can be concurrently executed at most.

Configuration file	Configuration item	Change type	Description
TiKV configuration file	soft-pending-compaction-bytes-limit	Modified	The soft limit on the pending compaction bytes. The default value is changed from "64GB" to "192GB".
TiKV configuration file	storage.io-rate-limit	Newly added	Controls the I/O rate of TiKV writes. The default value of storage.io-rate-limit.max-bytes-per-sec is "0MB".
TiKV configuration file	resolved-ts.enable	Newly added	Determines whether to maintain the resolved-ts for all Region leaders. The default value is true.
TiKV configuration file	resolved-ts.advance-ts-interval	Newly added	The interval at which the resolved-ts is forwarded. The default value is "1s". You can change the value dynamically.

Configuration file	Configuration item	Change type	Description
TiKV configuration file	`resolved-ts.scan-lock-pool-size`	Newly added	The number of threads that TiKV uses to scan the MVCC (multi-version concurrency control) lock data when initializing the `resolved-ts`. The default value is 2.

16.10.6.1.3 Others

- Before the upgrade, check the value of the TiDB configuration `feedback-probability`. If the value is not 0, the "panic in the recoverable goroutine" error will occur after the upgrade, but this error does not affect the upgrade.
- Upgrade the Go compiler version of TiDB from go1.13.7 to go1.16.4, which improves the TiDB performance. If you are a TiDB developer, upgrade your Go compiler version to ensure a smooth compilation.
- Avoid creating tables with clustered indexes in the cluster that uses TiDB Binlog during the TiDB rolling upgrade.
- Avoid executing statements like `alter table ... modify column` or `alter table ... change column` during the TiDB rolling upgrade.
- Since v5.1, setting the replica of system tables, when building TiFlash replicas for each table, is no longer supported. Before upgrading the cluster, you need to clear the relevant system table replicas; otherwise, the upgrade will fail.
- Deprecate the `--sort-dir` parameter in the `cdc cli changefeed` command of TiCDC. Instead, you can set `--sort-dir` in the `cdc server` command. #1795
- After upgrading to TiDB 5.1, if TiDB returns the "function READ ONLY has only noop implementation" error, you can let TiDB ignore this error by setting the value of `tidb_enable_noop_functions` to `ON`. This is because the `read_only` variable in MySQL does not yet take effect in TiDB (which is a 'noop' behavior in TiDB). Therefore, even if this variable is set in TiDB, you can still write data into the TiDB cluster.

16.10.6.2 New features

16.10.6.2.1 SQL

- Support the Common Table Expression (CTE) feature of MySQL 8.0.

  This feature empowers TiDB with the capability of querying hierarchical data recursively or non-recursively and meets the needs of using tree queries to implement application logics in multiple sectors such as human resources, manufacturing, financial markets, and education.

  In TiDB, you can apply the `WITH` statement to use Common Table Expressions. User document, #17472

- Support the dynamic privilege feature of MySQL 8.0.

  Dynamic privileges are used to limit the `SUPER` privilege and provide TiDB with more flexible privilege configuration for more fine-grained access control. For example, you can use dynamic privileges to create a user account that can only perform `BACKUP` and `RESTORE` operations.

  The supported dynamic privileges are as follows:

  - `BACKUP_ADMIN`
  - `RESTORE_ADMIN`
  - `ROLE_ADMIN`
  - `CONNECTION_ADMIN`
  - `SYSTEM_VARIABLES_ADMIN`

  You can also use plugins to add new privileges. To check out all supported privileges, execute the `SHOW PRIVILEGES` statement. User document

- Add a new configuration item for the Security Enhanced Mode (SEM), which divides the TiDB administrator privileges in a finer-grained way.

  The Security Enhanced Mode is disabled by default. To enable it, see the user document.

- Enhance the capability of changing column types online. Support changing the column type online using the `ALTER TABLE` statement, including but not limited to:

  - Changing `VARCHAR` to `BIGINT`
  - Modifying the `DECIMAL` precision
  - Compressing the length of `VARCHAR(10)` to `VARCHAR(5)`

  User document

- Introduce a new SQL syntax `AS OF TIMESTAMP` to perform Stale Read, a new experimental feature used to read historical data from a specified point in time or from a specified time range.

  User document, #21094

  The examples of `AS OF TIMESTAMP` are as follows:

```
SELECT * FROM t AS OF TIMESTAMP '2020-09-06 00:00:00';
START TRANSACTION READ ONLY AS OF TIMESTAMP '2020-09-06 00:00:00';
SET TRANSACTION READ ONLY as of timestamp '2020-09-06 00:00:00';
```

- Introduce a new statistics type `tidb_analyze_version = 2` (Experimental).

  `tidb_analyze_version` is set to 2 by default, which avoids the large errors that might occur in the large data volume caused by hash conflicts in Version 1 and maintains the estimation accuracy in most scenarios.

  User document

16.10.6.2.2 Transaction

- Support the Lock View feature (Experimental)

  The Lock View feature provides more information about lock conflicts and lock waits of pessimistic locks, which helps DBAs to observe transaction locking conditions and troubleshoot deadlock problems. #24199

  User document:

  - View the pessimistic locks and other locks that currently occur on all TiKV nodes in the clusters: `DATA_LOCK_WAITS`
  - View several deadlock errors that recently occurred on the TiDB nodes: `DEADLOCKS`
  - View the transaction information executed currently on the TiDB nodes: `TIDB_TRX`

16.10.6.2.3 Performance

- Stale read of data replicas (Experimental)

  Read local replicas data directly to reduce read latency and improve the query performance

  User document, #21094

- Enable the Hibernate Region feature by default.

  If a Region is in an inactive state for a long time, it is automatically set to a silent state, which reduces the system overhead of the heartbeat information between the Leader and the Follower.

  User document, #10266

16.10.6.2.4 Stability

- Solve the replication stability issue of TiCDC
  - Improve TiCDC memory usage to avoid OOM in the following scenarios
  - If large amounts of data is accumulated during the replication interruption, exceeding 1TB, the re-replication causes OOM problems.
  - Large amounts of data writes cause OOM problems in TiCDC.
  - Reduce the possibility of TiCDC replication interruption in the following scenarios:

    project#11
    - Replication interruption when the network is unstable
    - Replication interruption when some TiKV/PD/TiCDC nodes are down
- TiFlash storage memory control

  Optimize the speed and memory usage of Region snapshot generation and reduce the possibility of OOM
- Add a write rate limiter for TiKV background tasks (TiKV Write Rate Limiter)

  To ensure the duration stability of read and write requests, TiKV Write Rate Limiter smoothes the write traffic of TiKV background tasks such as GC and Compaction. The default value of TiKV background task write rate limiter is "0MB". It is recommended to set this value to the optimal I/O bandwidth of the disk, such as the maximum I/O bandwidth specified by the cloud disk manufacturer.

  User document, #9156
- Solve scheduling stability issues when multiple scaling tasks are performed at the same time

16.10.6.2.5 Telemetry

TiDB adds the running status of TiDB cluster requests in telemetry, including execution status and failure status.

To learn more about the information and how to disable this behavior, refer to Telemetry.

16.10.6.3 Improvements

- TiDB
  - Support the built-in function `VITESS_HASH()` #23915
  - Support pushing down data of the enumerated type to TiKV to improve performance when using enumerated types in `WHERE` clauses #23619

- Support the `RENAME USER` syntax #23648
- Optimize the calculation of Window Function to solve TiDB OOM problems when paging data with ROW_NUMBER() #23807
- Optimize the calculation of `UNION ALL` to solve the TiDB OOM problems when using `UNION ALL` to join a large number of `SELECT` statements #21441
- Optimize the dynamic pruning mode of partitioned tables to improve performance and stability #24150
- Fix the `Region is Unavailable` issue that occurs in multiple scenarios project#62
- Fix multiple `Region is Unavailable` issues that might occur in frequent scheduling situations
- Fix `Region is Unavailable` issue that might occur in some high stress write situations
- Avoid frequently reading the `mysql.stats_histograms` table if the cached statistics is up-to-date to avoid high CPU usage #24317

- TiKV

  - Use `zstd` to compress Region snapshots, preventing large space differences between nodes in case of heavy scheduling or scaling #10005
  - Solve OOM issues in multiple cases #10183
    - Add memory usage tracking for each module
    - Solve the OOM issue caused by oversized Raft entries cache
    - Solve the OOM issue caused by stacked GC tasks
    - Solve the OOM issue caused by fetching too many Raft entries from the Raft log to memory at one time
  - Split Regions more evenly to mitigate the issue that the growth of Region size exceeds the splitting speed when there are hotspot writes #9785

- TiFlash

  - Support `Union All`, `TopN`, and `Limit` functions
  - Support the Cartesian product including left outer join and semi anti join in MPP mode
  - Optimize lock operations to avoid that running DDL statements and read operations are blocked by each other
  - Optimize cleanup of expired data by TiFlash
  - Support further filtering of query filters on `timestamp` columns at the TiFlash storage level
  - Improve the startup and scalability speed of TiFlash when a large number of tables are in a cluster
  - Improve TiFlash compatibility when running on unknown CPUs

- PD

- Avoid unexpected statistics after adding the `scatter region` scheduler #3602
- Solve multiple scheduling issues in the scaling process
    - Optimize the generation process of replica snapshots to solve slow scheduling issues during scaling #3563 #10059 #10001
    - Solve slow scheduling issues caused by heartbeat pressure due to traffic changes #3693 #3739 #3728 #3751
    - Reduce the space discrepancies of large clusters due to scheduling, and optimize the scheduling formula to prevent the bursting issue (which is similar to heterogeneous space clusters) caused by large compression rate discrepancies #3592 #10005

- Tools
    - Backup & Restore (BR)
        - Support backing up and restoring system tables in the `mysql` schema #1143 #1078
        - Support the S3-compatible storage that is based on the virtual-host addressing mode #10243
        - Optimize the format of backupmeta to reduce memory usage #1171
    - TiCDC
        - Improve the descriptions of some log messages to be clearer and more useful for diagnosing problems #1759
        - Support the back pressure feature to allow the TiCDC scanning speed to sense the downstream processing capacity #10151
        - Reduce memory usage when TiCDC performs the initial scan #10133
        - Improve the cache hit rate for the TiCDC Old Value in pessimistic transactions #10089
    - Dumpling
        - Improve the logic of exporting data from TiDB v4.0 to avoid TiDB becoming out of memory (OOM) #273
        - Fix the issue that no error is output when a backup operation fails #280
    - TiDB Lightning
        - Improve data importing speed. The optimization results show that the speed of importing TPC-C data is increased by 30%, and the speed of importing large tables (2TB+) with more indexes (5 indexes) is increased by more than 50%. #753
        - Add a pre-check on the data to be imported and also on the target cluster before importing, and report errors to reject the import process if it does not meet the import requirements #999
        - Optimize the timing of checkpoint updates on the Local backend to improve performance of restarting from breakpoints#1080

16.10.6.4 Bug Fixes

- TiDB
    - Fix the issue that the execution result of project elimination might be wrong when the projection result is empty #23887
    - Fix the issue of wrong query results when a column contains `NULL` values in some cases #23891
    - Forbid generating MPP plans when the scan contains virtual columns #23886
    - Fix the wrong reuse of `PointGet` and `TableDual` in Plan Cache #23187 #23144 #23304 #23290
    - Fix the error that occurs when the optimizer builds the `IndexMerge` plan for clustered indexes #23906
    - Fix the type inference of the BIT-type errors #23832
    - Fix the issue that some optimizer hints do not take effect when the `PointGet` operator exists #23570
    - Fix the issue that DDL operations might fail when rolling back due to an error #23893
    - Fix the issue that the index range of the binary literal constant is incorrectly built #23672
    - Fix the potential wrong results of the `IN` clause in some cases #23889
    - Fix the wrong results of some string functions #23759
    - Users now need both `INSERT` and `DELETE` privileges on a table to perform `REPLACE` operations #23909
    - Users now need both `INSERT` and `DELETE` privileges on a table to perform `REPLACE` operations #24070
    - Fix the wrong `TableDual` plans caused by incorrectly comparing binaries and bytes#23846
    - Fix the panic issue caused by using the prefix index and index join in some cases #24547 #24716 #24717
    - Fix the issue that the prepared plan cache of `point get` is incorrectly used by the `point get` statement in the transaction #24741
    - Fix the issue of writing the wrong prefix index value when the collation is `ascii_bin` or `latin1_bin` #24569
    - Fix the issue that the ongoing transaction might be interrupted by the GC worker #24591
    - Fix a bug that the point query might get wrong on the clustered index when `new-collation` is enabled but `new-row-format` is disabled #24541
    - Refactor the conversion of partition keys for shuffle hash join #24490
    - Fix the panic issue that occurs when building the plan for queries that contain the `HAVING` clause #24045
    - Fix the issue that the column pruning improvement causes the `Apply` and `Join` operators' results to go wrong #23887
    - Fix a bug that the primary lock fallen back from async commit cannot be resolved #24384

- Fix a GC issue of statistics that might cause duplicated fm-sketch records #24357
- Avoid unnecessary pessimistic rollback when the pessimistic locking receives the `ErrKeyExists` error #23799
- Fix the issue that numeric literals cannot be recognized when the sql_mode contains `ANSI_QUOTES` #24429
- Forbid statements such as `INSERT INTO table PARTITION (<partitions>)... ↪ ON DUPLICATE KEY UPDATE` to read data from non-listed partitions #24746
- Fix the potential `index out of range` error when a SQL statement contains both `GROUP BY` and `UNION` #24281
- Fix the issue that the `CONCAT` function incorrectly handles the collation #24296
- Fix the issue that the `collation_server` global variable does not take effect in new sessions #24156

- TiKV

  - Fix the issue that the coprocessor fails to properly handle the signed or unsigned integer types in the `IN` expression #9821
  - Fix the issue of many empty Regions after batch ingesting SST files #964
  - Fix a bug that TiKV cannot start up after the file dictionary file is damaged #9886
  - Fix a TiCDC OOM issue caused by reading old values #9996 #9981
  - Fix the issue of empty value in the secondary index for the clustered primary key column when collation is `latin1_bin` #24548
  - Add the `abort-on-panic` configuration, which allows TiKV to generate the core dump file when panic occurs. Users still need to correctly configure the environment to enable core dump #10216
  - Fix the performance regression issue of `point get` queries that occurs when TiKV is not busy #10046

- PD

  - Fix the issue that the PD Leader re-election is slow when there are many stores #3697
  - Fix the panic issue that occurs when removing the evict leader scheduler from a non-existent store #3660
  - Fix the issue that the statistics are not updated after offline peers are merged #3611

- TiFlash

  - Fix the issue of incorrect results when casting the time type to the integer type
  - Fix a bug that the `receiver` cannot find corresponding tasks within 10 seconds
  - Fix the issue that there might be invalid iterators in `cancelMPPQuery`
  - Fix a bug that the behavior of the `bitwise` operator is different from that of TiDB

- Fix the alert issue caused by overlapping ranges when using the `prefix key`
- Fix the issue of incorrect results when casting the string type to the integer type
- Fix the issue that consecutive and fast writes might make TiFlash out of memory
- Fix the potential issue that the exception of null pointer might be raised during the table GC
- Fix the TiFlash panic issue that occurs when writing data to dropped tables
- Fix the issue that TiFlash might panic during BR restore
- Fix the issue of incorrect results when cloning shared delta index concurrently
- Fix the potential panic that occurs when the Compaction Filter feature is enabled
- Fix the issue that TiFlash cannot resolve the lock fallen back from async commit
- Fix the issue of incorrect results returned when the casted result of the `TIMEZONE` type contains the `TIMESTAMP` type
- Fix the TiFlash panic issue that occurs during Segment Split

- Tools
  - TiDB Lightning
    - Fix the issue of TiDB Lightning panic that occurs when generating KV data #1127
    - Fix a bug that the batch split Region fails due to the total key size exceeding the raft entry limit during the data import #969
    - Fix the issue that when importing CSV files, if the last line of the file does not contain line break characters (`\r\n`), an error will be reported #1133
    - Fix the issue that if a table to be imported includes an auto-increment column of the double type, the auto_increment value becomes abnormal #1178
  - Backup & Restore (BR)
    - Fix the issue of backup interruption caused by the failure of a few TiKV nodes #980
  - TiCDC
    - Fix the concurrency issue in Unified Sorter and filter the unhelpful error messages #1678
    - Fix a bug that the creation of redundant directories might interrupt the replication with MinIO #1463
    - Set the default value of the `explicit_defaults_for_timestamp` session variable to ON to make the MySQL 5.7 downstream keep the same behavior with the upstream TiDB #1585
    - Fix the issue that the incorrect handling of `io.EOF` might cause replication interruption #1633
    - Correct the TiKV CDC endpoint CPU metric in the TiCDC dashboard #1645
    - Increase `defaultBufferChanSize` to avoid replication blocking in some cases #1259
    - Fix the issue that the time zone information is lost in the Avro output #1712
    - Support cleaning up stale temporary files in Unified Sorter and forbid sharing the `sort-dir` directory #1742

- Fix a deadlock bug in the KV client that occurs when many stale Regions exist #1599
- Fix the wrong help information in the `--cert-allowed-cn` flag #1697
- Revert the update for `explicit_defaults_for_timestamp` which requires the SUPER privilege when replicating data to MySQL #1750
- Support the sink flow control to reduce the risk of memory overflow #1840
- Fix a bug that the replication task might stop when moving a table #1828
- Fix the issue that the TiKV GC safe point is blocked due to the stagnation of TiCDC changefeed checkpoint #1759

16.11 v5.0

16.11.1 TiDB 5.0.6 Release Notes

Release date: December 31, 2021

TiDB version: 5.0.6

16.11.1.1 Compatibility changes

- Tools
  - TiCDC
    - Change the output of the `cdc server` command error from stdout to stderr #3133
    - Set the default value of Kafka sink `max-message-bytes` to `10M` #3081
    - Change the default value of Kafka Sink `partition-num` to 3 so that TiCDC distributes messages across Kafka partitions more evenly #3337

16.11.1.2 Improvements

- TiDB
  - Show the affected SQL statements in the debug log when the coprocessor encounters a lock, which is helpful in diagnosing problems #27718
- TiKV
  - Increase the speed of inserting SST files by moving the verification process to the `Import` thread pool from the `Apply` thread pool #11239
  - Add more metrics for the garbage collection module of Raft logs to locate performance problems in the module #11374
  - Collapse some uncommon storage-related metrics in Grafana dashboard #11681

- PD
    - Speed up the exit process of schedulers #4146
    - Make the scheduling results of the `scatter-range-scheduler` scheduler more even by allowing the scheduler to schedule empty Regions and fix the configurations of the scheduler #4497
    - Support that the evict leader scheduler can schedule Regions with unhealthy peers #4093
- Tools
    - TiCDC
        - Optimize rate limiting control on TiKV reloads to reduce gPRC congestion during changefeed initialization #3110
        - Add a tick frequency limit to EtcdWorker to prevent frequent etcd writes from affecting PD services #3112
        - Add the default configuration for `config.Metadata.Timeout` in Kafka sink #3352
        - Set the default value of `max-message-bytes` to `10M`, to reduce the probability that Kafka messages cannot be sent #3081
        - Add more Prometheus and Grafana monitoring metrics and alerts, including `no owner alert`, `mounter row`, `table sink total row`, and `buffer sink total row` #4054 #1606
    - Backup & Restore (BR)
        - Retry BR tasks when encountering the PD request error or the TiKV I/O timeout error #27787
        - Improve the robustness of restoring #27421
    - TiDB Lightning
        - Support importing data into tables that have expression index or the index that depends on virtual generated columns #1404

16.11.1.3 Bug fixes

- TiDB
    - Fix the issue that optimistic transaction conflicts might cause transactions to block each other #11148
    - Fix the issue of false positive error log `invalid cop task execution summaries length` for MPP queries #1791
    - Fix the panic that might occur when DML and DDL statements are executed concurrently #30940
    - Fix the `privilege check fail` error when performing the `grant` and `revoke` operations to grant and revoke global level privileges #29675

- Fix the TiDB panic when executing the `ALTER TABLE.. ADD INDEX` statement in some cases #27687
- Fix the issue that the `enforce-mpp` configuration does not take effect in v5.0.4 #29252
- Fix the panic when using the `CASE WHEN` function on the `ENUM` data type #29357
- Fix wrong results of the `microsecond` function in vectorized expressions #29244
- Fix the issue of incomplete log information from the `auto analyze` result #29188
- Fix wrong results of the `hour` function in vectorized expression #28643
- Fix the unexpected error like `tidb_cast to Int32 is not supported` when the unsupported `cast` is pushed down to TiFlash #23907
- Fix a bug that the availability detection of MPP node does not work in some corner cases #3118
- Fix the `DATA RACE` issue when assigning `MPP task ID` #27952
- Fix the `INDEX OUT OF RANGE` error for a MPP query after deleting an empty `dual table` #28250
- Fix the TiDB panic when inserting invalid date values concurrently #25393
- Fix the unexpected `can not found column in Schema column` error for queries in the MPP mode #30980
- Fix the issue that TiDB might panic when TiFlash is shuting down #28096
- Fix the unexpected `index out of range` error when the planner is doing join reorder #24095
- Fix wrong results of the control functions (such as `IF` and `CASE WHEN`) when using the `ENUM` type data as parameters of such functions #23114
- Fix the wrong result of `CONCAT(IFNULL(TIME(3))` #29498
- Fix wrong results of `GREATEST` and `LEAST` when passing in unsigned `BIGINT` arguments #30101
- Fix the issue that a SQL operation is canceled when its JSON type column joins its `CHAR` type column #29401
- Fix the data inconsistency issue caused by incorrect usage of lazy existence check and untouched key optimization #30410
- Fix the issue that window functions might return different results when using a transaction or not #29947
- Fix the issue that the SQL statements that contain `cast(integer as char) union string` return wrong results #29513
- Fix the issue that the length information is wrong when casting `Decimal` to `String` #29417
- Fix the issue that the `Column 'col_name' in field list is ambiguous` error is reported unexpectedly when a SQL statement contains natural join #25041
- Fix the issue that the `GREATEST` function returns inconsistent results due to different values of `tidb_enable_vectorized_expression` (set to `on` or `off`) #29434
- Fix the issue that the planner might cache invalid plans for `join` in some cases #28087
- Fix the issue that the `index out of range [1] with length 1` error is reported when a SQL statement evaluates an aggregation result on the result of join in some cases #1978

- TiKV
    - Fix the issue that a down TiKV node causes the resolved timestamp to lag #11351
    - Fix the issue that batch messages are too large in Raft client implementation #9714
    - Fix a panic issue that occurs when Region merge, ConfChange, and Snapshot happen at the same time in extreme conditions #11475
    - Fix the issue that TiKV cannot detect the memory lock when TiKV perform a reverse table scan #11440
    - Fix the issue of negative sign when the decimal divide result is zero #29586
    - Fix the issue that the accumulation of GC tasks might cause TiKV to be OOM (out of memory) #11410
    - Fix the issue that the average latency of the by-instance gRPC requests is inaccurate in TiKV metrics #11299
    - Fix a memory leak caused by monitoring data of statistics threads #11195
    - Fix the issue of TiCDC panic that occurs when the downstream database is missing #11123
    - Fix the issue that TiCDC adds scan retries frequently due to the Congest error #11082
    - Fix the issue that the Raft connection is broken when the channel is full #11047
    - Fix the issue of TiKV panic that occurs when the files do not exist when TiDB Lightning imports data #10438
    - Fix the issue that TiDB cannot correctly identify whether the `Int64` types in `Max`/`Min` functions are a signed integer or not, which causes the wrong calculation result of `Max`/`Min` #10158
    - Fix the issue that the node of a TiKV replica is down after the node gets snapshots because TiKV cannot modify the metadata accurately #10225
    - Fix the leak issue of the backup thread pool #10287
    - Fix the issue of casting illegal strings into floating-point numbers #23322
- PD
    - Fix a panic issue that occurs after the TiKV node is removed #4344
    - Fix the issue that operator can get blocked due to down store #3353
    - Fix slow leader election caused by stucked Region syncer #3936
    - Fix the issue that the speed of removing peers is limited when repairing the down nodes #4090
    - Fix the issue that the hotspot cache cannot be cleared when the Region heartbeat is less than 60 seconds #4390
- TiFlash
    - Fix potential data inconsistency after altering a primary key column to a larger int data type
    - Fix the issue that TiFlash fails to start up on some platforms, such as ARM, due to the absence of the `libnsl.so` library

- Fix the issue that the `Store size` metric does not match the actual data size on a disk
- Fix the issue that TiFlash crashes due to a `Cannot open file` error
- Fix occasional crashes of TiFlash when an MPP query is killed
- Fix the unexpected error `3rd arguments of function substringUTF8 must be constants`
- Fix query failures caused by excessive `OR` conditions
- Fix the bug that results of `where <string>` are wrong
- Fix inconsistent behaviors of `CastStringAsDecimal` between TiFlash and TiDB/TiKV
- Fix query failures caused by the error `different types: expected Nullable(Int64), got Int64`
- Fix query failures caused by the error `Unexpected type of column: Nullable(Nothing)`
- Fix query failures caused by overflow when comparing data in the `DECIMAL` data type

- Tools

  - TiCDC

    - Fix the issue that some partitioned tables without valid indexes might be ignored when `force-replicate` is enabled #2834
    - Fix the issue that `cdc cli` silently truncates user parameters when receiving unexpected parameters, causing the user input parameters to be lost #2303
    - Fix the issue that TiCDC sync task might pause when an error occurs during writing a Kafka message #2978
    - Fix a possible panic issue when encoding some types of columns into Open Protocol format #2758
    - Fix the issue that Kafka may send excessively large messages by setting the default value of `max-message-bytes` to `10M` #3081
    - Fix the issue that TiCDC replication task might terminate when the upstream TiDB instance unexpectedly exits #3061
    - Fix the issue that TiCDC process might panic when TiKV sends duplicate requests to the same Region #2386
    - Fix the TiCDC replication interruption issue when multiple TiKVs crash or during a forced restart #3288
    - Fix the negative value error in the changefeed checkpoint lag #3010
    - Fix the issue of overly frequent warnings caused by MySQL sink deadlock #2706
    - Fix the issue that Avro sink does not support parsing JSON type columns #3624
    - Fix the bug that TiCDC reads the incorrect schema snapshot from TiKV when the TiKV owner restarts #2603
    - Fix the memory leak issue after processing DDLs #3174

- Fix the bug that the `enable-old-value` configuration item is not automatically set to `true` on the Canal protocol #3676
- Fix the timezone error that occurs when the `cdc server` command runs on some Red Hat Enterprise Linux releases (such as 6.8 and 6.9) #3584
- Fix the issue of the inaccurate `txn_batch_size` monitoring metric for Kafka sink #3431
- Fix the issue that `tikv_cdc_min_resolved_ts_no_change_for_1m` keeps alerting when there is no changefeed #11017
- Fix the TiCDC panic issue that occurs when manually cleaning the task status in etcd #2980
- Fix the issue that changefeed does not fail fast enough when the ErrGCTTLExceeded error occurs #3111
- Fix the issue that scanning stock data might fail due to TiKV performing GC when scanning stock data takes too long #2470
- Fix OOM in container environments #1798

- Backup & Restore (BR)
  - Fix a bug that the average speed is inaccurately calculated for backup and restore #1405
- Dumpling
  - Fix the bug that Dumpling becomes very slow when dumping tables with the composite primary key or unique key #29386

16.11.2 TiDB 5.0.5 Release Note

Release date: December 3, 2021

TiDB version: 5.0.5

16.11.2.1 Bug fix

- TiKV
  - Fix the issue that the `GcKeys` task does not work when it is called by multiple keys. Caused by this issue, compaction filter GC might not drop the MVCC deletion information. #11217

16.11.3 TiDB 5.0.4 Release Notes

Release Date: September 27, 2021

TiDB version: 5.0.4

16.11.3.1 Compatibility changes

- TiDB
    - Fix the issue that executing `SHOW VARIABLES` in a new session is slow. This fix reverts some changes made in #19341 and might cause compatibility issues. #24326
    - Change the default value of the `tidb_stmt_summary_max_stmt_count` variable from `200` to `3000` #25873
    - The following bug fixes change execution results, which might cause upgrade incompatibilities:
        - Fix the issue that TiDB returns wrong result when the children of `UNION` contain the `NULL` value #26559
        - Fix the issue that `greatest(datetime)union null` returns empty string #26532
        - Fix the issue that the behavior of the `last_day` function is incompatible in SQL mode #26000
        - Fix the issue that the `having` clause might not work correctly #26496
        - Fix the wrong execution results that occur when the collations around the `between` expression are different #27146
        - Fix the wrong execution results that occur when the column in the `group_concat` function has a non-bin collation #27429
        - Fix an issue that using a `count(distinct)` expression on multiple columns returns wrong result when the new collation is enabled #27091
        - Fix the result wrong that occurs when the argument of the `extract` function is a negative duration #27236
        - Fix the issue that inserting an invalid date does not report an error when the `SQL_MODE` is 'STRICT_TRANS_TABLES' #26762
        - Fix the issue that using an invalid default date does not report an error when the `SQL_MODE` is 'NO_ZERO_IN_DATE' #26766
        - Fix a bug on the query range of prefix index #26029
        - Fix the issue that the `LOAD DATA` statement might abnormally import non-utf8 data #25979
        - Fix the issue that `insert ignore on duplicate update` might insert wrong data when the secondary index has the same column with the primary key #25809
        - Fix the issue that `insert ignore duplicate update` might insert wrong data when a partitioned table has a clustered index #25846
        - Fix the issue that the query result might be wrong when the key is the `ENUM` type in point get or batch point get #24562
        - Fix the wrong result that occurs when dividing a `BIT`-type value #23479
        - Fix the issue that the results of `prepared` statements and direct queries might be inconsistent #22949
        - Fix the issue that the query result might be wrong when a `YEAR` type is compared with a string or an integer type #23262

16.11.3.2 Feature enhancements

- TiDB
    - Support setting `tidb_enforce_mpp=1` to ignore the optimizer estimation and forcibly use the MPP mode #26382
- TiKV
    - Support changing TiCDC configurations dynamically #10645
- PD
    - Add OIDC-based SSO support for TiDB Dashboard #3884
- TiFlash
    - Support the `HAVING()` function in DAG requests
    - Support the `DATE()` function
    - Add Grafana panels for write throughput per instance

16.11.3.3 Improvements

- TiDB
    - Trigger auto-analyze based on the histogram row count #24237
    - Stop sending requests to a TiFlash node for a period if the node has failed and restarted before #26757
    - Increase the `split region` upper limit to make `split table` and `presplit` more stable #26657
    - Support retry for MPP queries #26483
    - Check the availability of TiFlash before launching MPP queries #1807
    - Support the stable result mode to make the query result more stable #26084
    - Support the MySQL system variable `init_connect` and its associated features #18894
    - Thoroughly push down the `COUNT(DISTINCT)` aggregation function in the MPP mode #25861
    - Print log warnings when the aggregation function cannot be pushed down in `EXPLAIN` statements #25736
    - Add error labels for `TiFlashQueryTotalCounter` in Grafana dashboard #25327
    - Support getting the MVCC data of a clustered index table through a secondary index by HTTP API #24209
    - Optimize the memory allocation of `prepared` statement in parser #24371
- TiKV

- Handle read ready and write ready separately to reduce read latency #10475
- Reduce the size of Resolved TS messages to save network bandwidth #2448
- Drop log instead of blocking threads when the slogger thread is overloaded and the queue is filled up #10841
- Make the slow log of TiKV coprocessor only consider the time spent on processing requests #10841
- Make prewrite as idempotent as possible to reduce the chance of undetermined errors #10587
- Avoid the false "GC can not work" alert under low write flow #10662
- Make the database to be restored always match the original cluster size during backup. #10643
- Ensure that the panic output is flushed to the log #9955

- PD

  - Improve the performance of synchronizing Region information between PDs #3993

- Tools

  - Dumpling

    - Support backing up MySQL-compatible databases that do not support the `START TRANSACTION ... WITH CONSISTENT SNAPSHOT` or the `SHOW CREATE TABLE` syntax #309

  - TiCDC

    - Optimize memory management when Unified Sorter is using memory to sort #2553
    - Prohibit operating TiCDC clusters across major or minor versions #2598
    - Reduce the goroutine usage when a table's Regions are all transferred away from a TiKV node #2284
    - Remove `file sorter` #2326
    - Always pull the old values from TiKV and the output is adjusted according to `enable-old-value` #2301
    - Improve the error message returned when a PD endpoint misses the certificate #1973
    - Optimize workerpool for fewer goroutines when concurrency is high #2211
    - Add a global gRPC connection pool and share gRPC connections among KV clients #2533

16.11.3.4 Bug Fixes

- TiDB

- Fix the issue that TiDB might panic when querying a partitioned table and the partition key has the `IS NULL` condition #23802
- Fix the issue that the overflow check of the `FLOAT64` type is different with that of MySQL #23897
- Fix the wrong character set and collation for the `case when` expression #26662
- Fix the issue that committing pessimistic transactions might cause write conflicts #25964
- Fix a bug that the index keys in a pessimistic transaction might be repeatedly committed #26359 #10600
- Fix the issue that TiDB might panic when resolving the async commit locks #25778
- Fix a bug that a column might not be found when using `INDEX MERGE` #25045
- Fix a bug that `ALTER USER REQUIRE SSL` clears users' `authentication_string` #25225
- Fix a bug that the value of the `tidb_gc_scan_lock_mode` global variable on a new cluster shows "PHYSICAL" instead of the actual default mode "LEGACY" #25100
- Fix the bug that the `TIKV_REGION_PEERS` system table does not show the correct `DOWN` status #24879
- Fix the issue of memory leaks that occurs when HTTP API is used #24649
- Fix the issue that views do not support `DEFINER` #24414
- Fix the issue that `tidb-server --help` exits with the code `2` #24046
- Fix the issue that setting the global variable `dml_batch_size` does not take effect #24709
- Fix the issue that using `read_from_storage` and partitioned table at the same time causes an error #20372
- Fix the issue that TiDB panics when executing the projection operator #24264
- Fix the issue that statistics might cause queries to panic #24061
- Fix the issue that using the `approx_percentile` function on a `BIT` column might panic #23662
- Fix the issue that the metrics on the **Coprocessor Cache** panel in Grafana are wrong #26338
- Fix the issue that concurrently truncating the same partition causes DDL statements to stuck #26229
- Fix the issue of wrong query results that occurs when the session variable is used as the `GROUP BY` item #27106
- Fix the wrong implicit conversion between `VARCHAR` and timestamp when joining tables #25902
- Fix the wrong results in associated subquery statements #27233

- TiKV

  - Fix the potential disk full issue caused by corrupted snapshot files #10813
  - Fix the TiKV panic issue that occurs when upgrading from a pre-5.0 version with Titan enabled #10843

- Fix the issue that TiKV of a newer version cannot be rolled back to v5.0.x #10843
- Fix the TiKV panic issue that occurs when upgrading from a pre-5.0 version to a 5.0 version or later. If a cluster was upgraded from TiKV v3.x with Titan enabled before the upgrade, this cluster might encounter the issue. #10774
- Fix the parsing failure caused by the left pessimistic locks #26404
- Fix the panic that occurs when calculating duration on certain platforms #10571
- Fix the issue that the keys of `batch_get_command` in Load Base Split are unencoded #10542

- PD

    - Fix the issue that PD does not fix the down peers in time #4077
    - Fix the issue that the replica count of the default placement rules stays constant after `replication.max-replicas` is updated #3886
    - Fix a bug that PD might panic when scaling out TiKV #3868
    - Fix the scheduling conflict issue that occurs when multiple schedulers are running at same time #3807
    - Fix the issue that the scheduler might appear again even if it has been deleted #2572

- TiFlash

    - Fix the potential panic issue that occurs when running table scan tasks
    - Fix the potential memory leak issue that occurs when executing MPP tasks
    - Fix a bug that TiFlash raises the `duplicated region` error when handling DAQ requests
    - Fix the issue of unexpected results when executing the aggregation functions `COUNT` or `COUNT DISTINCT`
    - Fix the potential panic issue that occurs when executing MPP tasks
    - Fix a potential bug that TiFlash cannot restore data when deployed on multiple disks
    - Fix the potential panic issue that occurs when deconstructing `SharedQueryBlockInputStream` ↪
    - Fix the potential panic issue that occurs when deconstructing `MPPTask`
    - Fix the issue of unexpected results when TiFlash fails to establish MPP connections
    - Fix the potential panic issue that occurs when resolving locks
    - Fix the issue that the store size in metrics is inaccurate under heavy writing
    - Fix a bug of incorrect results that occurs when queries contain filters like `CONSTANT` ↪ , `<`, `<=`, `>`, `>=`, or `COLUMN`
    - Fix the potential issue that TiFlash cannot garbage-collect the delta data after running for a long time
    - Fix a potential bug that metrics display wrong values
    - Fix the potential issue of data inconsistency that occurs when TiFlash is deployed on multiple disks

- Tools
  - Dumpling
    - Fix the issue that the execution of `show table status` is stuck in MySQL 8.0.3 or a later version #322
  - TiCDC
    - Fix the issue of process panic that occurs when encoding the data types such as `mysql.TypeString`, `mysql.TypeVarString`, `mysql.TypeVarchar` into JSON #2758
    - Fix a data inconsistency issue that occurs because multiple processors might write data to the same table when this table is being re-scheduled #2417
    - Decrease the gRPC window size to avoid the OOM that occurs when TiCDC captures too many Regions #2724
    - Fix the error that the gRPC connection is frequently broken when the memory pressure is high #2202
    - Fix a bug that causes TiCDC to panic on the unsigned `TINYINT` type #2648
    - Fix the issue that TiCDC Open Protocol outputs an empty value when inserting a transaction and deleting data of the same row in the upstream #2612
    - Fix a bug that DDL handling fails when a changefeed starts at the finish TS of a schema change #2603
    - Fix the issue that irresponsive downstreams interrupt the replication task in old owner until the task times out #2295
    - Fix a bug in metadata management #2558
    - Fix the issue of data inconsistency that occurs after the TiCDC owner switch #2230
    - Fix the issue that outdated capture might appear in the output of the `capture list` command #2388
    - Fix the `ErrSchemaStorageTableMiss` error that occurs when the DDL Job duplication is encountered in the integrated test #2422
    - Fix the bug that a changefeed cannot be removed if the `ErrGCTTLExceeded` error occurs #2391
    - Fix a bug that replicating large tables to cdclog fails #1259 #2424
    - Fix the CLI backward compatibility issue #2373
    - Fix the issue of insecure concurrent access to the map in `SinkManager` #2299
    - Fix the issue of potential DDL loss when the owner crashes when executing DDL statements #1260
    - Fix the issue that the lock is resolved immediately after a Region is initialized #2188
    - Fix the issue of extra partition dispatching that occurs when adding a new partitioned table #2263
    - Fix the issue that TiCDC keeps warning on removed changefeeds #2156

16.11.4 TiDB 5.0.3 Release Notes

Release date: July 2, 2021

TiDB version: 5.0.3

16.11.4.1 Compatibility Changes

- TiDB
    - After a v4.0 cluster is upgraded to v5.0 or a later version (dev or v5.1), the default value of the `tidb_multi_statement_mode` variable changes from `WARN` to `OFF`
    - TiDB is now compatible with MySQL 5.7's noop variable `innodb_default_row_format` ↪ . Setting this variable will have no effect. #23541

16.11.4.2 Feature Enhancements

- Tools
    - TiCDC
        - Add an HTTP API to get the changefeed information and the health information of the node #1955
        - Add the SASL/SCRAM support for the kafka sink #1942
        - Make TiCDC support `--data-dir` at the server level #2070

16.11.4.3 Improvements

- TiDB
    - Support pushing down the `TopN` operator to TiFlash #25162
    - Support pushing down the built-in function `json_unquote()` to TiKV #24415
    - Support removing the union branch from the dual table #25614
    - Support pushing down the built-in function `replace()` to TiFlash #25565
    - Support pushing down the built-in functions `unix_timestamp()`, `concat()`, `year` ↪ `()`, `day()`, `datediff()`, `datesub()`, and `concat_ws()` to TiFlash #25564
    - Optimize the aggregate operator's cost factor #25241
    - Support pushing down the `Limit` operator to TiFlash #25159
    - Support pushing down the built-in function `str_to_date` to TiFlash #25148
    - Allow the MPP outer join to choose the build table based on the table row count #25142
    - Support pushing down the built-in functions `left()`, `right()`, and `abs()` to TiFlash #25133
    - Support pushing down the Broadcast Cartesian join to TiFlash #25106
    - Support pushing down the `Union All` operator to TiFlash #25051

- Support balancing the MPP query workload among different TiFlash nodes based on Regions #24724
- Support invalidating stale Regions in the cache after the MPP query is executed #24432
- Improve the MySQL compatibility of the built-in function `str_to_date` for the format specifiers `%b/%M/%r/%T` #25767

- TiKV

  - Limit the TiCDC sink's memory consumption #10305
  - Add the memory-bounded upper limit for the TiCDC old value cache #10313

- PD

  - Update TiDB Dashboard to v2021.06.15.1 #3798

- TiFlash

  - Support casting the `STRING` type to the `DOUBLE` type
  - Support the `STR_TO_DATE()` function
  - Optimize the non-joined data in right outer join using multiple threads
  - Support the Cartesian join
  - Support the `LEFT()` and `RIGHT()` functions
  - Support automatically invalidating stale Regions in MPP queries
  - Support the `ABS()` function

- Tools

  - TiCDC

    - Refine gRPC's reconnection logic and increase the KV client's throughput #1586 #1501 #1682 #1393 #1847 #1905 #1904
    - Make the sorter I/O errors more user-friendly

16.11.4.4 Bug Fixes

- TiDB

  - Fix the issue that an incorrect result is returned when using merge join on the `SET` type column #25669
  - Fix the data corruption issue in the `IN` expression's arguments #25591
  - Avoid the sessions of GC being affected by global variables #24976
  - Fix the panic issue that occurs when using `limit` in the window function queries #25344
  - Fix the wrong value returned when querying a partitioned table using `Limit` #24636

- Fix the issue that `IFNULL` does not correctly take effect on the `ENUM` or `SET` type column #24944
- Fix the wrong results caused by changing the `count` in the join subqueries to `first_row` #24865
- Fix the query hang issue that occurs when `ParallelApply` is used under the `TopN` operator #24930
- Fix the issue that more results than expected are returned when executing SQL statements using multi-column prefix indexes #24356
- Fix the issue that the `<=>` operator cannot correctly take effect #24477
- Fix the data race issue of the parallel `Apply` operator #23280
- Fix the issue that the `index out of range` error is reported when sorting the IndexMerge results of the PartitionUnion operator #23919
- Fix the issue that setting the `tidb_snapshot` variable to an unexpectedly large value might damage the transaction isolation #25680
- Fix the issue that the ODBC-styled constant (for example, `{d '2020-01-01'}`) cannot be used as the expression #25531
- Fix the issue that `SELECT DISTINCT` converted to `Batch Get` causes incorrect results #25320
- Fix the issue that backing off queries from TiFlash to TiKV cannot be triggered #23665 #24421
- Fix the `index-out-of-range` error that occurs when checking `only_full_group_by` ↪ #23839)
- Fix the issue that the result of index join in correlated subqueries is wrong #25799

- TiKV

    - Fix the wrong `tikv_raftstore_hibernated_peer_state` metric #10330
    - Fix the wrong arguments type of the `json_unquote()` function in the coprocessor #10176
    - Skip clearing callback during graceful shutdown to avoid breaking ACID in some cases #10353 #10307
    - Fix a bug that the read index is shared for replica reads on a Leader #10347
    - Fix the wrong function that casts `DOUBLE` to `DOUBLE` #25200

- PD

    - Fix the data race issue that occurs when loading TTL configurations after the scheduler is started #3771
    - Fix a bug that the `is_learner` field of the `TIKV_REGION_PEERS` table in TiDB is incorrect #3372 #24293
    - Fix the issue that when all TiKV nodes in a zone are offline or down, PD does not schedule replicas to other zones #3705
    - Fix the issue that PD might get panic after the scatter Region scheduler is added #3762

- TiFlash

- Fix the issue that TiFlash keeps restarting because of the split failure
- Fix the potential issue that TiFlash cannot delete the delta data
- Fix a bug that TiFlash adds wrong padding for non-binary characters in the `CAST` function
- Fix the issue of incorrect results when handling aggregation queries with complex `GROUP BY` columns
- Fix the TiFlash panic issue that occurs under heavy write pressure
- Fix the panic that occurs when the right jon key is not nullalbe and the left join key is nullable
- Fix the potential issue that the `read-index` requests take a long time
- Fix the panic issue that occurs when the read load is heavy
- Fix the panic issue that might occur when the `Date_Format` function is called with the `STRING` type argument and `NULL` values

- Tools

  - TiCDC

    - Fix the issue that TiCDC owner exits when refreshing the checkpoint #1902
    - Fix a bug that some MySQL connection might leak after MySQL sink meets the error and pauses #1946
    - Fix the panic issue that occurs when TiCDC fails to read `/proc/meminfo` #2024
    - Reduce TiCDC's runtime memory consumption #2012 #1958
    - Fix a bug that might cause TiCDC server panic due to the late calculation of resolved ts #1576
    - Fix the potential deadlock issue for the processor #2142

  - Backup & Restore (BR)

    - Fix a bug that all system tables are filtered during restore #1197 #1201
    - Fix the issue that Backup & Restore reports the error of "file already exists" when TDE is enabled during the restore #1179

  - TiDB Lightning

    - Fix the TiDB Lightning panic issue for some special data #1213
    - Fix the EOF error reported when TiDB Lightning splits the imported large CSV files #1133
    - Fix a bug that an excessively large base value is generated when TiDB Lightning imports tables with the `auto_increment` column of the `FLOAT` or `DOUBLE` type #1186
    - Fix the issue that TiDB fails to parse the `DECIMAL` type data in Parquet files #1277

16.11.5 TiDB 5.0.2 Release Notes

Release date: June 10, 2021

TiDB version: 5.0.2

16.11.5.1 Compatibility Changes

- Tools
  - TiCDC
    - Deprecate `--sort-dir` in the `cdc cli changefeed` command. Instead, users can set `--sort-dir` in the `cdc server` command. #1795

16.11.5.2 New Features

- TiKV
  - Enable the Hibernate Region feature by default #10266

16.11.5.3 Improvements

- TiDB
  - Avoid frequently reading the `mysql.stats_histograms` table if the cached statistics is up-to-date to avoid high CPU usage #24317
- TiKV
  - BR now supports the S3-compatible storage using the virtual-host addressing mode #10243
  - Support the back pressure for TiCDC's scan speed #10151
  - Reduce the memory usage of TiCDC's initial scan #10133
  - Improve the cache hit ratio of the TiCDC's Old Value feature in the pessimistic transaction #10089
  - Split Regions more evenly to mitigate the issue that the growth of Region size exceeds the splitting speed when there are hotspot writes #9785
- TiFlash
  - Optimize the table lock to prevent DDL jobs and data reads from blocking each other
  - Support casting the `INTEGER` or `REAL` type to `REAL` type
- Tools
  - TiCDC
    - Add monitoring metrics for the table memory consumption #1885
    - Optimize the memory and CPU usages during the sorting stage #1863
    - Delete some useless log information that might cause user confusion #1759

- Backup & Restore (BR)
  - Clarify some ambiguous error messages #1132
  - Support checking the cluster version of a backup #1091
  - Support backing up and restoring system tables in the `mysql` schema #1143 #1078
- Dumpling
  - Fix the issue that no error is output when a backup operation fails #280

16.11.5.4 Bug Fixes

- TiDB
  - Fix the panic issue caused by using the prefix index and index join in some cases #24547 #24716 #24717
  - Fix the issue that the prepared plan cache of `point get` is incorrectly used by the `point get` statement in the transaction #24741
  - Fix the issue of writing the wrong prefix index value when the collation is `ascii_bin` or `latin1_bin` #24569
  - Fix the issue that the ongoing transaction might be interrupted by the GC worker #24591
  - Fix a bug that the point query might get wrong on the clustered index when `new-collation` is enabled but `new-row-format` is disabled #24541
  - Refactor the conversion of partition keys for shuffle hash join #24490
  - Fix the panic issue that occurs when building the plan for queries that contain the `HAVING` clause #24045
  - Fix the issue that the column pruning improvement causes the `Apply` and `Join` operators' results to go wrong #23887
  - Fix a bug that the primary lock fallen back from async commit cannot be resolved #24384
  - Fix a GC issue of statistics that might cause duplicated fm-sketch records #24357
  - Avoid unnecessary pessimistic rollback when the pessimistic locking receives the `ErrKeyExists` error #23799
  - Fix the issue that numeric literals cannot be recognized when the sql_mode contains `ANSI_QUOTES` #24429
  - Forbid statements such as `INSERT INTO table PARTITION (<partitions>)... ↪ ON DUPLICATE KEY UPDATE` to read data from non-listed partitions #24746
  - Fix the potential `index out of range` error when a SQL statement contains both `GROUP BY` and `UNION` #24281
  - Fix the issue that the `CONCAT` function incorrectly handles the collation #24296
  - Fix the issue that the `collation_server` global variable does not take effect in new sessions #24156
- TiKV

    - Fix a TiCDC OOM issue caused by reading old values #9996 #9981
    - Fix the issue of empty value in the secondary index for the clustered primary key column when collation is `latin1_bin` #24548
    - Add the `abort-on-panic` configuration, which allows TiKV to generate the core dump file when panic occurs. Users still need to correctly configure the environment to enable core dump #10216
    - Fix the performance regression issue of `point get` queries that occurs when TiKV is not busy #10046

- PD

    - Fix the issue that the PD Leader re-election is slow when there are many stores #3697
    - Fix the panic issue that occurs when removing the evict leader scheduler from a non-existent store #3660
    - Fix the issue that the statistics are not updated after offline peers are merged #3611

- TiFlash

    - Fix the issue of incorrect results when cloning shared delta index concurrently
    - Fix the potential issue that TiFlash fails to restart with incomplete data
    - Fix the issue that old dm files are not removed automatically
    - Fix the potential panic that occurs when the Compaction Filter feature is enabled
    - Fix the potential issue that `ExchangeSender` sends duplicated data
    - Fix the issue that TiFlash cannot resolve the lock fallen back from async commit
    - Fix the issue of incorrect results returned when the casted result of the `TIMEZONE` type contains the `TIMESTAMP` type
    - Fix the TiFlash panic issue that occurs during Segment Split
    - Fix the issue that the execution information about the non-root MPP task is not accurate

- Tools

    - TiCDC

        - Fix the issue that the time zone information is lost in the Avro output #1712
        - Support cleaning up stale temporary files in Unified Sorter and forbid sharing the `sort-dir` directory #1742
        - Fix a deadlock bug in the KV client that occurs when many stale Regions exist #1599
        - Fix the wrong help information in the `--cert-allowed-cn` flag #1697
        - Revert the update for `explicit_defaults_for_timestamp` which requires the `SUPER` privilege when replicating data to MySQL #1750
        - Support the sink flow control to reduce the risk of memory overflow #1840
        - Fix a bug that the replication task might stop when moving a table #1828

- Fix the issue that the TiKV GC safe point is blocked due to the stagnation of TiCDC changefeed checkpoint #1759

- Backup & Restore (BR)
  - Fix the issue that the `DELETE` events are lost during the log restore #1063
  - Fix a bug that causes BR to send too many useless RPC requests to TiKV #1037
  - Fix the issue that no error is output when a backup operation fails #1043
- TiDB Lightning
  - Fix the issue of TiDB Lightning panic that occurs when generating KV data #1127
  - Fix the issue that TiDB Lightning in the TiDB-backend mode cannot load any data when the autocommit is disabled #1104
  - Fix a bug that the batch split Region fails due to the total key size exceeding the raft entry limit during the data import #969

16.11.6 TiDB 5.0.1 Release Notes

Release date: April 24, 2021

TiDB version: 5.0.1

16.11.6.1 Compatibility change

- The default value of the `committer-concurrency` configuration item is changed from `16` to `128`.

16.11.6.2 Improvements

- TiDB
  - Support the built-in function `VITESS_HASH()` #23915
- TiKV
  - Use `zstd` to compress the Region snapshot #10005
- PD
  - Modify the Region score calculator to better satisfy isomerous stores #3605
  - Avoid unexpected statistics after adding the `scatter region` scheduler #3602
- Tools
  - Backup & Restore (BR)
    - Remove some misleading information from the summary log #1009

16.11.6.3 Bug Fixes

- TiDB
    - Fix the issue that the execution result of project elimination might be wrong when the projection result is empty #24093
    - Fix the issue of wrong query results when a column contains `NULL` values in some cases #24063
    - Forbid generating MPP plans when the scan contains virtual columns #24058
    - Fix the wrong reuse of `PointGet` and `TableDual` in Plan Cache #24043
    - Fix the error that occurs when the optimizer builds the `IndexMerge` plan for clustered indexes #24042
    - Fix the type inference of the BIT-type errors #24027
    - Fix the issue that some optimizer hints do not take effect when the `PointGet` operator exists #23685
    - Fix the issue that DDL operations might fail when rolling back due to an error #24080
    - Fix the issue that the index range of the binary literal constant is incorrectly built #24041
    - Fix the potential wrong results of the `IN` clause in some cases #24023
    - Fix the wrong results of some string functions #23879
    - Users now need both `INSERT` and `DELETE` privileges on a table to perform `REPLACE` operations #23939
    - Fix the performance regression when executing the point query #24070
    - Fix the wrong `TableDual` plans caused by incorrectly comparing binaries and bytes #23918
- TiKV
    - Fix the issue that the coprocessor fails to properly handle the signed or unsigned integer types in the `IN` expression #10018
    - Fix the issue of many empty Regions after batch ingesting SST files #10015
    - Fix the potential panic that occurs when the input of `cast_string_as_time` is invalid UTF-8 bytes #9995
    - Fix a bug that TiKV cannot start up after the file dictionary file is damaged #9992
- TiFlash
    - Fix the issue that the storage engine fails to remove the data of some ranges
    - Fix the issue of incorrect results when casting the time type to the integer type
    - Fix a bug that the `receiver` cannot find corresponding tasks within 10 seconds
    - Fix the issue that there might be invalid iterators in `cancelMPPQuery`
    - Fix a bug that the behavior of the `bitwise` operator is different from that of TiDB

- Fix the alert issue caused by overlapping ranges when using the `prefix key`
- Fix the issue of incorrect results when casting the string type to the integer type
- Fix the issue that consecutive and fast writes might make TiFlash out of memory
- Fix the issue that duplicated column names will make TiFlash raise errors
- Fix the issue that TiFlash fails to parse MPP plans
- Fix the potential issue that the exception of null pointer might be raised during the table GC
- Fix the TiFlash panic issue that occurs when writing data to dropped tables
- Fix the issue that TiFlash might panic during BR restore

- Tools
  - TiDB Lightning
    - Fix the issue of the inaccurate table count in the progress log during the import #1005
  - Backup & Restore (BR)
    - Fix a bug that the actual backup speed exceeds the `--ratelimit` limit #1026
    - Fix the issue of backup interruption caused by the failure of a few TiKV nodes #1019
    - Fix the issue of the inaccurate table count in the progress log during TiDB Lightning's import #1005
  - TiCDC
    - Fix the concurrency issue in Unified Sorter and filter the unhelpful error messages #1678
    - Fix a bug that the creation of redundant directories might interrupt the replication with MinIO #1672
    - Set the default value of the `explicit_defaults_for_timestamp` session variable to `ON` to make the MySQL 5.7 downstream keep the same behavior with the upstream TiDB #1659
    - Fix the issue that the incorrect handling of `io.EOF` might cause replication interruption #1648
    - Correct the TiKV CDC endpoint CPU metric in the TiCDC dashboard #1645
    - Increase `defaultBufferChanSize` to avoid replication blocking in some cases #1632

16.11.7 What's New in TiDB 5.0

Release date: April 7, 2021

TiDB version: 5.0.0

In v5.0, PingCAP is dedicated to helping enterprises quickly build applications based on TiDB, freeing them from worries about database performance, performance jitter, security, high availability, disaster recovery, troubleshooting SQL performance, and so on.

In v5.0, the key new features or improvements are as follows:

- Introduce Massively Parallel Processing (MPP) architecture through TiFlash nodes, which shares the execution workloads of large join queries among TiFlash nodes. When the MPP mode is enabled, TiDB, based on cost, determines whether to use the MPP framework to perform the calculation. In the MPP mode, the join keys are redistributed through the `Exchange` operation while being calculated, which distributes the calculation pressure to each TiFlash node and speeds up the calculation. According to the benchmark, with the same cluster resource, TiDB 5.0 MPP shows 2 to 3 times of speedup over Greenplum 6.15.0 and Apache Spark 3.1.1, and some queries have 8 times better performance.
- Introduce the clustered index feature to improve database performance. For example, in the TPC-C tpmC test, the performance of TiDB, with clustered index enabled, improves by 39%.
- Enable the async commit feature to reduce the write latency. For example, in the 64-thread Sysbench test, the average latency of updating indexes, with async commit enabled, is reduced by 41.7%, from 12.04 ms to 7.01 ms.
- Reduce jitters. This is achieved by improving the optimizer stability and by limiting system tasks' usages of I/O, network, CPU, and memory resources. For example, in the 8-hour performance test, the standard deviation of TPC-C tpmC does not exceed 2%.
- Enhance system stability by improving scheduling and by keeping execution plans stable as much as possible.
- Introduces Raft Joint Consensus algorithm, which ensures the system availability during the Region membership change.
- Optimize `EXPLAIN` features and invisible index, which helps Database Administrators (DBAs) debug SQL statements more efficiently.
- Guarantee reliability for enterprise data. You can back up data from TiDB to Amazon S3 storage and Google Cloud GCS, or restore data from these cloud storage platforms.
- Improve performance of data import from or data export to Amazon S3 storage or TiDB/MySQL, which helps enterprises quickly build applications on the cloud. For example, in the TPC-C test, the performance of importing 1 TiB data improves by 40%, from 254 GiB/h to 366 GiB/h.

16.11.7.1 Compatibility changes

16.11.7.1.1 System variables

- Add the `tidb_executor_concurrency` system variable to control the concurrency of multiple operators. The previous `tidb_*_concurrency` settings (such as `tidb_projection_concurrency`) still take effect but with a warning when you use them.

- Add the `tidb_skip_ascii_check` system variable to specify whether to skip the ASCII validation check when the ASCII character set is written. This default value is `OFF`.

- Add the tidb_enable_strict_double_type_check system variable to determine whether the syntax like double(N) can be defined in the table schema. This default value is OFF.
- Change the default value of tidb_dml_batch_size from 20000 to 0. This means that batch DML statements are no longer used by default in LOAD/INSERT INTO SELECT ↪ .... Instead, large transactions are used to comply with the strict ACID semantics.

> **Note:**
> The scope of the variable is changed from session to global, and the default value is changed from 20000 to 0. If the application relies on the original default value, you need to use the set global statement to modify the variable to the original value after the upgrade.

- Control temporary tables' syntax compatibility using the tidb_enable_noop_functions ↪ system variable. When this variable value is OFF, the CREATE TEMPORARY TABLE syntax returns an error.
- Add the following system variables to directly control the garbage collection-related parameters:
  - tidb_gc_concurrency
  - tidb_gc_enable
  - tidb_gc_life_time
  - tidb_gc_run_interval
  - tidb_gc_scan_lock_mode
- Change the default value of enable-joint-consensus from false to true, which enables the Joint Consensus feature by default.
- Change the value of tidb_enable_amend_pessimistic_txn from 0 or 1 to ON or OFF.
- Change the default value of tidb_enable_clustered_index from OFF to INT_ONLY with the following new meanings:
  - ON: clustered index is enabled. Adding or deleting non-clustered indexes is supported.
  - OFF: clustered index is disabled. Adding or deleting non-clustered indexes is supported.
  - INT_ONLY: the default value. The behavior is consistent with that before v5.0. You can control whether to enable clustered index for the INT type together with alter-primary-key = false.

Note:

The `INT_ONLY` value of `tidb_enable_clustered_index` in 5.0 GA has the same meaning as the `OFF` value in 5.0 RC. After upgrading from a 5.0 RC cluster with the `OFF` setting to 5.0 GA, it will be displayed as `INT_ONLY`.

16.11.7.1.2 Configuration file parameters

- Add the `index-limit` configuration item for TiDB. Its value defaults to `64` and ranges between `[64,512]`. A MySQL table supports 64 indexes at most. If its value exceeds the default setting and more than 64 indexes are created for a table, when the table schema is re-imported into MySQL, an error will be reported.
- Add the `enable-enum-length-limit` configuration item for TiDB to be compatible and consistent with MySQL's ENUM/SET length (ENUM length < 255). The default value is `true`.
- Replace the `pessimistic-txn.enable` configuration item with the `tidb_txn_mode` environment variable.
- Replace the `performance.max-memory` configuration item with `performance.server-memory-quota`
- Replace the `tikv-client.copr-cache.enable` configuration item with `tikv-client.copr-cache.capacity-mb`. If the item's value is `0.0`, this feature is disabled. If the item's value is greater than `0.0`, this feature is enabled. Its default value is `1000.0`.
- Replace the `rocksdb.auto-tuned` configuration item with `rocksdb.rate-limiter-auto-tuned`.
- Delete the `raftstore.sync-log` configuration item. By default, written data is forcibly spilled to the disk. Before v5.0, you can explicitly disable `raftstore.sync-log`. Since v5.0, the configuration value is forcibly set to `true`.
- Change the default value of the `gc.enable-compaction-filter` configuration item from `false` to `true`.
- Change the default value of the `enable-cross-table-merge` configuration item from `false` to `true`.
- Change the default value of the `rate-limiter-auto-tuned` configuration item from `false` to `true`.

16.11.7.1.3 Others

- Before the upgrade, check the value of the TiDB configuration `feedback-probability`. If the value is not 0, the "panic in the recoverable goroutine" error will occur after the upgrade, but this error does not affect the upgrade.
- Forbid conversion between `VARCHAR` type and `CHAR` type during the column type change to avoid data correctness issues.

16.11.7.2 New features

16.11.7.2.1 SQL

List partitioning (**Experimental**)

User document

With the list partitioning feature, you can effectively query and maintain tables with a large amount of data.

With this feature enabled, partitions and how data is distributed among partitions are defined according to the `PARTITION BY LIST(expr)PARTITION part_name VALUES IN (...)` expression. The partitioned tables' data set supports at most 1024 distinct integer values. You can define the values using the `PARTITION ... VALUES IN (...)` clause.

To enable list partitioning, set the session variable `tidb_enable_list_partition` to `ON`.

List COLUMNS partitioning (**Experimental**)

User document

List COLUMNS partitioning is a variant of list partitioning. You can use multiple columns as partition keys. Besides the integer data type, you can also use the columns in the string, `DATE`, and `DATETIME` data types as partition columns.

To enable List COLUMNS partitioning, set the session variable `tidb_enable_list_partition` ↪ to `ON`.

Invisible indexes

User document, #9246

When you tune performance or select optimal indexes, you can set an index to be `Visible` ↪ or `Invisible` by using SQL statements. This setting can avoid performing resource-consuming operations, such as `DROP INDEX` and `ADD INDEX`.

To modify the visibility of an index, use the `ALTER INDEX` statement. After the modification, the optimizer decides whether to add this index to the index list based on the index visibility.

`EXCEPT` and `INTERSECT` operators

User document, #18031

The `INTERSECT` operator is a set operator, which returns the intersection of the result sets of two or more queries. To some extent, it is an alternative to the `Inner Join` operator.

The `EXCEPT` operator is a set operator, which combines the result sets of two queries and returns elements that are in the first query result but not in the second.

16.11.7.2.2 Transaction

User document, #18005

In the pessimistic transaction mode, if the tables involved in a transaction contain concurrent DDL operations or `SCHEMA VERSION` changes, the system automatically updates the transaction's `SCHEMA VERSION` to the latest to ensure the successful transaction commit, and to avoid that the client receives the `Information schema is changed` error when the transaction is interrupted by DDL operations or `SCHEMA VERSION` changes.

This feature is disabled by default. To enable the feature, modify the value of `tidb_enable_amend_pessimistic_txn` system variable. This feature is introduced in v4.0.7 and has the following issues fixed in v5.0:

- The compatibility issue that occurs when TiDB Binlog executes `Add Column` operations
- The data inconsistency issue that occurs when using the feature together with the unique index
- The data inconsistency issue that occurs when using the feature together with the added index

Currently, this feature still has the following incompatibility issues:

- Transaction's semantics might change when there are concurrent transactions
- Known compatibility issue that occurs when using the feature together with TiDB Binlog
- Incompatibility with `Change Column`

16.11.7.2.3 Character set and collation

- Support the `utf8mb4_unicode_ci` and `utf8_unicode_ci` collations. User document, #17596
- Support the case-insensitive comparison sort for collations

16.11.7.2.4 Security

User document, #18566

To meet security compliance requirements (such as *General Data Protection Regulation*, or GDPR), the system supports desensitizing information (such as ID and credit card number) in the output error messages and logs, which can avoid leaking sensitive information.

TiDB supports desensitizing the output log information. To enable this feature, use the following switches:

- The global variable `tidb_redact_log`. Its default value is `0`, which means that desensitization is disabled. To enable desensitization for tidb-server logs, set the variable value to `1`.
- The configuration item `security.redact-info-log`. Its default value is `false`, which means that desensitization is disabled. To enable desensitization for tikv-server logs, set the variable value to `true`.

- The configuration item `security.redact-info-log`. Its default value is `false`, which means that desensitization is disabled. To enable desensitization for pd-server logs, set the variable value to `true`.
- The configuration item `security.redact_info_log` for tiflash-server and `security.redact-info-log` for tiflash-learner. Their default values are both `false`, which means that desensitization is disabled. To enable desensitization for tiflash-server and tiflash-learner logs, set the values of both variables to `true`.

This feature is introduced in v5.0. To use the feature, enable the system variable and all configuration items above.

16.11.7.3 Performance optimization

16.11.7.3.1 MPP architecture

User document

TiDB introduces the MPP architecture through TiFlash nodes. This architecture allows multiple TiFlash nodes to share the execution workload of large join queries.

When the MPP mode is on, TiDB determines whether to send a query to the MPP engine for computation based on the calculation cost. In the MPP mode, TiDB distributes the computation of table joins to each running TiFlash node by redistributing the join key during data calculation (`Exchange` operation), and thus accelerates the calculation. Furthermore, with the aggregation computing feature that TiFlash has already supported, TiDB can pushdown the computation of a query to the TiFlash MPP cluster. Then the distributed environment can help accelerate the entire execution process and dramatically increase the speed of analytic queries.

In the TPC-H 100 benchmark test, TiFlash MPP delivers significant processing speed over analytic engines of traditional analytic databases and SQL on Hadoop. With this architecture, you can perform large-scale analytic queries directly on the latest transaction data, with a higher performance than traditional offline analytic solutions. According to the benchmark, with the same cluster resource, TiDB 5.0 MPP shows 2 to 3 times of speedup over Greenplum 6.15.0 and Apache Spark 3.1.1, and some queries have 8 times better performance.

Currently, the main features that the MPP mode does not support are as follows (For details, refer to Use TiFlash):

- Table partitioning
- Window Function
- Collation
- Some built-in functions
- Reading data from TiKV
- OOM spill
- Union
- Full Outer Join

16.11.7.3.2 Clustered index

User document, #4841

When you are designing table structures or analyzing database behaviors, it is recommended to use the clustered index feature if you find that some columns with primary keys are often grouped and sorted, queries on these columns often return a certain range of data or a small amount of data with different values, and the corresponding data does not cause read or write hotspot issues.

Clustered indexes, also known as *index-organized tables* in some database management systems, is a storage structure associated with the data of a table. When creating a clustered index, you can specify one or more columns from the table as the keys for the index. TiDB stores these keys in a specific structure, which allows TiDB to quickly and efficiently find the rows associated with the keys, thus improves the performance of querying and writing data.

When the clustered index feature is enabled, the TiDB performance improves significantly (for example in the TPC-C tpmC test, the performance of TiDB, with clustered index enabled, improves by 39%) in the following cases:

- When data is inserted, the clustered index reduces one write of the index data from the network.
- When a query with an equivalent condition only involves the primary key, the clustered index reduces one read of index data from the network.
- When a query with a range condition only involves the primary key, the clustered index reduces multiple reads of index data from the network.
- When a query with an equivalent or range condition involves the primary key prefix, the clustered index reduces multiple reads of index data from the network.

Each table can either use a clustered or non-clustered index to sort and store data. The differences of these two storage structures are as follows:

- When creating a clustered index, you can specify one or more columns in the table as the key value of the index. A clustered index sorts and stores the data of a table according to the key value. Each table can have only one clustered index. If a table has a clustered index, it is called a clustered index table. Otherwise, it is called a non-clustered index table.
- When you create a non-clustered index, the data in the table is stored in an unordered structure. You do not need to explicitly specify the key value of the non-clustered index, because TiDB automatically assigns a unique ROWID to each row of data. During a query, the ROWID is used to locate the corresponding row. Because there are at least two network I/O operations when you query or insert data, the performance is degraded compared with clustered indexes.

When table data is modified, the database system automatically maintains clustered indexes and non-clustered indexes for you.

All primary keys are created as non-clustered indexes by default. You can create a primary key as a clustered index or non-clustered index in either of the following two ways:

- Specify the keyword `CLUSTERED | NONCLUSTERED` in the statement when creating a table, then the system creates the table in the specified way. The syntax is as follows:

```
CREATE TABLE `t` (`a` VARCHAR(255), `b` INT, PRIMARY KEY (`a`, `b`)
    ↪ CLUSTERED);
```

Or

```
CREATE TABLE `t` (`a` VARCHAR(255) PRIMARY KEY CLUSTERED, `b` INT);
```

You can execute the statement `SHOW INDEX FROM tbl-name` to query whether a table has a clustered index.

- Configure the system variable `tidb_enable_clustered_index` to control the clustered index feature. Supported values are `ON`, `OFF`, and `INT_ONLY`.
    - `ON`: Indicates that the clustered index feature is enabled for all types of primary keys. Adding and dropping non-clustered indexes are supported.
    - `OFF`: Indicates that the clustered index feature is disabled for all types of primary keys. Adding and dropping non-clustered indexes are supported.
    - `INT_ONLY`: The default value. If the variable is set to `INT_ONLY` and `alter-primary-key` is set to `false`, the primary keys which consist of single integer columns are created as clustered indexes by default. The behavior is consistent with that of TiDB v5.0 and earlier versions.

If a `CREATE TABLE` statement contains the keyword `CLUSTERED | NONCLUSTERED`, the statement overrides the configuration of the system variable and the configuration item.

You are recommended to use the clustered index feature by specifying the keyword `CLUSTERED | NONCLUSTERED` in statements. In this way, it is more flexible for TiDB to use all data types of clustered and non-clustered indexes in the system at the same time as required.

It is not recommended to use `tidb_enable_clustered_index = INT_ONLY`, because `INT_ONLY` is temporarily used to make this feature compatible and will be deprecated in the future.

Limitations for the clustered index are as follows:

- Mutual conversion between clustered indexes and non-clustered indexes is not supported.
- Dropping clustered indexes is not supported.
- Adding, dropping, and altering clustered indexes using `ALTER TABLE` statements are not supported.

- Reorganizing and re-creating a clustered index is not supported.
- Enabling or disabling indexes is not supported, which means the invisible index feature is not effective for clustered indexes.
- Creating a `UNIQUE KEY` as a clustered index is not supported.
- Using the clustered index feature together with TiDB Binlog is not supported. After TiDB Binlog is enabled, TiDB only supports creating a single integer primary key as a clustered index. TiDB Binlog does not replicate data changes of existing tables with clustered indexes to the downstream.
- Using the clustered index feature together with the attributes `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` is not supported.
- If the cluster is upgraded to a later version then rolls back, you need to downgrade newly-added tables by exporting table data before the rollback and importing the data after the rollback. Other tables are not affected.

16.11.7.3.3 Async Commit

User document, #8316

The client of the database will wait for the database system to complete the transaction commit in two phases (2PC) synchronously. The transaction returns the result to the client after the first phase commit is successful, and the system executes the second phase commit operation in the background asynchronously to reduce the transaction commit latency. If the transaction write involves only one Region, the second phase is omitted directly, and the transaction becomes a one-phase commit.

After the Async Commit feature is enabled, with the same hardware and configuration, when Sysbench is set to test the Update index with 64 threads, the average latency decreases by 41.7% from 12.04ms to 7.01ms.

When Async Commit feature is enabled, to reduce one network interaction latency and improve the performance of data writes, database application developers are recommended to consider reducing the consistency of transactions from linear consistency to causal consistency. The SQL statement to enable causal consistency is `START TRANSACTION WITH CAUSAL CONSISTENCY`.

After the causal consistency is enabled, with the same hardware and configuration, when Sysbench is set to test oltp_write_only with 64 threads, the average latency decreased by 5.6% from 11.86ms to 11.19ms.

After the consistency of transactions is reduced from the linear consistency to causal consistency, if there is no interdependence between multiple transactions in the application, the transactions do not have a globally consistent order.

The Async Commit feature is enabled by default for newly created v5.0 clusters.

This feature is disabled by default for clusters upgraded from earlier versions to v5.0. You can enable this feature by executing the `set global tidb_enable_async_commit = ON;` and `set global tidb_enable_1pc = ON;` statements.

The limitation for the Async Commit feature is as follows:

- Direct downgrade is not supported.

16.11.7.3.4 Enable the Coprocessor cache feature by default

User document, #18028

In 5.0 GA, the Coprocessor cache feature is enabled by default. After this feature is enabled, to reduce the latency of reading data, TiDB caches the calculation results of the operators pushed down to tikv-server in tidb-server.

To disable the Coprocessor cache feature, you can modify the `capacity-mb` configuration item of `tikv-client.copr-cache` to `0.0`.

16.11.7.3.5 Improve the execution performance of `delete from table where id <? Limit ?` statement

#18028

The p99 performance of the `delete from table where id <? limit ?` statement is improved by 4 times.

16.11.7.3.6 Optimize load base split strategy to solve the performance problem that data cannot be split in some small table hotspot read scenarios

#18005

16.11.7.4 Improve stability

16.11.7.4.1 Optimize the performance jitter issue caused by imperfect scheduling

#18005

The TiDB scheduling process occupies resources such as I/O, network, CPU, and memory. If TiDB does not control the scheduled tasks, QPS and delay might cause performance jitter due to resource preemption.

After the following optimizations, in the 8-hour performance test, the standard deviation of TPC-C tpmC does not exceed 2%.

Introduce new scheduling calculation formulas to reduce unnecessary scheduling and performance jitter

When the node capacity is always near the waterline set in the system, or when the `store-limit` is set too large, to balance the capacity load, the system frequently schedules Regions to other nodes or even schedules Regions back to their original nodes. Because

scheduling occupies resources, such as I/O, network, CPU, and memory, and causes performance jitter, this type of scheduling is not necessary.

To mitigate this issue, PD introduces a new set of default scheduling calculation formulas. You can switch back to the old formulas by configuring `region-score-formula-version = v1`.

Enable the cross-table Region merge feature by default

User document

Before v5.0, TiDB disables the cross-table Region merge feature by default. Starting from v5.0, this feature is enabled by default to reduce the number of empty Regions and the overhead of network, memory, and CPU. You can disable this feature by modifying the `schedule.enable-cross-table-merge` configuration item.

Enable the system to automatically adjust the data compaction speed by default to balance the contention for I/O resources between background tasks and foreground reads and writes

User document

Before v5.0, to balance the contention for I/O resources between background tasks and foreground reads and writes, the feature that the system automatically adjusts the data compaction speed is disabled by default. Starting from v5.0, TiDB enables this feature by default and optimizes the algorithm so that the latency jitter is significantly reduced.

You can disable this feature by modifying the `rate-limiter-auto-tuned` configuration item.

Enable the GC Compaction Filter feature by default to reduce GC's consumption of CPU and I/O resources

User document, #18009

When TiDB performs garbage collection (GC) and data compaction, partitions occupy CPU and I/O resources. Overlapping data exists during the execution of these two tasks.

To reduce GC's consumption of CPU and I/O resources, the GC Compaction Filter feature combines these two tasks into one and executes them in the same task. This feature is enabled by default. You can disable it by configuring `gc.enable-compaction-filter = false`.

TiFlash limits the compression and data sorting's use of I/O resources (**experimental feature**)

This feature alleviates the contention for I/O resources between background tasks and foreground reads and writes.

This feature is disabled by default. You can enable this feature by modifying the `bg_task_io_rate_limit` configuration item.

Improve the performance of checking scheduling constraints and the performance of fixing the unhealthy Regions in a large cluster

16.11.7.4.2 Ensure that the execution plans are unchanged as much as possible to avoid performance jitter

User document

SQL Binding supports the `INSERT`, `REPLACE`, `UPDATE`, `DELETE` statements

When tuning performance or maintaining the database, if you find that the system performance is unstable due to unstable execution plans, you can select a manually optimized SQL statement according to your judgement or tested by `EXPLAIN ANALYZE`. You can bind the optimized SQL statement to the SQL statement to be executed in the application code to ensure stable performance.

When manually binding SQL statements using the SQL BINDING statement, you need to ensure that the optimized SQL statement has the same syntax as the original SQL statement.

You can view the manually or automatically bound execution plan information by running the `SHOW {GLOBAL | SESSION} BINDINGS` command. The output is the same as that of versions earlier than v5.0.

Automatically capture and bind execution plans

When upgrading TiDB, to avoid performance jitter, you can enable the baseline capturing feature to allow the system to automatically capture and bind the latest execution plan and store it in the system table. After TiDB is upgraded, you can export the bound execution plan by running the `SHOW GLOBAL BINDING` command and decide whether to delete these plans.

This feature is disbled by default. You can enable it by modifying the server or setting the `tidb_capture_plan_baselines` global system variable to `ON`. When this feature is enabled, the system fetches the SQL statements that appear at least twice from the Statement Summary every `bind-info-lease` (the default value is `3s`), and automatically captures and binds these SQL statements.

16.11.7.4.3 Improve stability of TiFlash queries

Add a system variable `tidb_allow_fallback_to_tikv` to fall back queries to TiKV when TiFlash fails. The default value is `OFF`.

16.11.7.4.4 Improve TiCDC stability and alleviate the OOM issue caused by replicating too much incremental data

User document, #1150

In TiCDC v4.0.9 or earlier versions, replicating too much data change might cause OOM. In v5.0, the Unified Sorter feature is enabled by default to mitigate OOM issues caused by the following scenarios:

- The data replication task in TiCDC is paused for a long time, during which a large amount of incremental data is accumulated and needs to be replicated.

- The data replication task is started from an early timestamp, so it becomes necessary to replicate a large amount of incremental data.

Unified Sorter is integrated with the `memory/file` sort-engine options of earlier versions. You do not need to manually configure the change.

Limitations:

- You need to provide sufficient disk capacity according to the amount of your incremental data. It is recommended to use SSDs with free capacity greater than 128 GB.

16.11.7.5 High availability and disaster recovery

16.11.7.5.1 Improve system availability during Region membership change

User document, #18079, #7587, #2860

In the process of Region membership changes, "adding a member" and "deleting a member" are two operations performed in two steps. If a failure occurs when the membership change finishes, the Regions will become unavailable and an error of foreground application is returned.

The introduced Raft Joint Consensus algorithm can improve the system availability during Region membership change. "adding a member" and "deleting a member" operations during the membership change are combined into one operation and sent to all members. During the change process, Regions are in an intermediate state. If any modified member fails, the system is still available.

This feature is enabled by default. You can disable it by running the `pd-ctl config set enable-joint-consensus` command to set the `enable-joint-consensus` value to `false`.

16.11.7.5.2 Optimize the memory management module to reduce system OOM risks

Track the memory usage of aggregate functions. This feature is enabled by default. When SQL statements with aggregate functions are executed, if the total memory usage of the current query exceeds the threshold set by `mem-quota-query`, the system automatically performs operations defined by `oom-action`.

16.11.7.5.3 Improve the system availability during network partition

16.11.7.6 Data migration

16.11.7.6.1 Migrate data from S3/Aurora to TiDB

TiDB data migration tools support using Amazon S3 (and other S3-compatible storage services) as the intermediate for data migration and initializing Aurora snapshot data directly into TiDB, providing more options for migrating data from Amazon S3/Aurora to TiDB.

To use this feature, refer to the following documents:

- Export data to Amazon S3 cloud storage, #8
- Migrate from Amazon Aurora MySQL Using TiDB Lightning, #266

16.11.7.6.2 Optimize the data import performance of TiDB Cloud

TiDB Lightning optimizes its data import performance specifically for AWS T1.standard configurations (or equivalent) of TiDB Cloud. Test results show that TiDB Lightning improves its speed of importing 1TB of TPC-C data into TiDB by 40%, from 254 GiB/h to 366 GiB/h.

16.11.7.7 Data sharing and subscription

16.11.7.7.1 Integrate TiDB to Kafka Connect (Confluent Platform) using TiCDC (experimental feature)

User document, #660

To support the business requirements of streaming TiDB data to other systems, this feature enables you to stream TiDB data to the systems such as Kafka, Hadoop, and Oracle.

The Kafka connectors protocol provided by the Confluent platform is widely used in the community, and it supports transferring data to either relational or non-relational databases in different protocols. By integrating TiCDC to Kafka Connect of the Confluent platform, TiDB extends the ability to stream TiDB data to other heterogeneous databases or systems.

16.11.7.8 Diagnostics

User document

During the troubleshooting of SQL performance issues, detailed diagnostic information is needed to determine the causes of performance issues. Before TiDB 5.0, the information collected by the `EXPLAIN` statements was not detailed enough. The root causes of the issues can only be determined based on log information, monitoring information, or even on guess, which might be inefficient.

In TiDB v5.0, the following improvements are made to help you troubleshoot performance issues more efficiently:

- Support using the `EXPLAIN ANALYZE` statement to analyze all DML statements to show the actual performance plans and the execution information of each operator. #18056

- Support using the `EXPLAIN FOR CONNECTION` statement to check the real-time status of all the SQL statements being executed. For example, you can use the statement to check the execution duration of each operator and the number of processed rows. #18233
- Provide more details about the operator execution in the output of the `EXPLAIN ANALYZE` statement, including the number of RPC requests sent by operators, the duration of resolving lock conflicts, network latency, the scanned volume of deleted data in RocksDB, and the hit rate of RocksDB caches. #18663
- Support automatically recording the detailed execution information of SQL statements in the slow log. The execution information in the slow log is consistent with the output information of the `EXPLAIN ANALYZE` statement, which includes the time consumed by each operator, the number of processed rows, and the number of sent RPC requests. #15009

16.11.7.9 Deployment and maintenance

16.11.7.9.1 Optimize the logic of cluster deployment operations, to help DBAs deploy a set of standard TiDB production cluster faster

User Document

In previous TiDB versions, DBAs using TiUP to deploy TiDB clusters find that the environment initialization is complicated, the checksum configuration is excessive, and the cluster topology file is difficult to edit. All of these issues lead to low deployment efficiency for DBAs. In TiDB v5.0, the TiDB deployment efficiency using TiUP is improved for DBAs through the following items:

- TiUP Cluster supports the `check topo.yaml` command to perform a more comprehensive one-click environment check and provide repair recommendations.
- TiUP Cluster supports the `check topo.yaml --apply` command to automatically repair environmental problems found during the environment check.
- TiUP Cluster supports the `template` command to get the cluster topology template file for DBAs to edit and support modifying the global node parameters.
- TiUP supports editing the `remote_config` parameter using the `edit-config` command to configure remote Prometheus.
- TiUP supports editing the `external_alertmanagers` parameter to configure different AlertManagers using the `edit-config` command.
- When editing the topology file using the `edit-config` subcommand in tiup-cluster, you can modify the data types of the configuration item values.

16.11.7.9.2 Improve upgrade stability

Before TiUP v1.4.0, during the upgrade of a TiDB cluster using tiup-cluster, the SQL responses of the cluster jitter for a long period of time, and during PD online rolling upgrades, the QPS of the cluster jitter between 10s to 30s.

TiUP v1.4.0 adjusts the logic and makes the following optimizations:

- During the upgrade of PD nodes, TiUP automatically checks the status of the restarted PD node, and then rolls to upgrade the next PD node after confirming that the status is ready.
- TiUP identifies the PD role automatically, first upgrades the PD nodes of the follower role, and finally upgrades the PD Leader node.

16.11.7.9.3 Optimize the upgrade time

Before TiUP v1.4.0, when DBAs upgrade TiDB clusters using tiup-cluster, for clusters with a large number of nodes, the total upgrade time is long and cannot meet the upgrade time window requirement for certain users.

Starting from v1.4.0, TiUP optimizes the following items:

- Supports fast offline upgrades using the `tiup cluster upgrade --offline` subcommand.
- Speeds up the Region Leader relocation for users using rolling upgrades during upgrades by default, so that reduces the time of rolling TiKV upgrades.
- Checks the status of the Region monitor using the `check` subcommand before running a rolling upgrade. Ensure that the cluster is in a normal state before the upgrade, thus reducing the probability of upgrade failures.

16.11.7.9.4 Support the breakpoint feature

Before TiUP v1.4.0, when DBAs upgrade TiDB clusters using tiup-cluster, if the execution of a command is interrupted, all the upgrade operations have to be performed again from the beginning.

TiUP v1.4.0 supports retrying failed operations from breakpoints using the tiup-cluster `replay` subcommand, to avoid re-executing all operations after an upgrade interruption.

16.11.7.9.5 Enhance the functionalities of maintenance and operations

TiUP v1.4.0 further enhances the functionalities for operating and maintaining TiDB clusters.

- Supports the upgrade or patch operation on the downtime TiDB and DM clusters to adapt to more usage scenarios.
- Adds the `--version` parameter to the `display` subcommand of tiup-cluster to get the cluster version.
- When only Prometheus is included in the node being scaled out, the operation of updating the monitoring configuration is not performed, to avoid scale-out failure due to the absence of the Prometheus node.
- Adds user input to the error message when the results of the input TiUP commands are incorrect, so that you can locate the cause of the problem more quickly.

16.11.7.10 Telemetry

TiDB adds cluster usage metrics in telemetry, such as the number of data tables, the number of queries, and whether new features are enabled.

To learn more about details and how to disable this behavior, refer to telemetry.

16.11.8 TiDB 5.0 RC Release Notes

Release date: January 12, 2021

TiDB version: 5.0.0-rc

TiDB v5.0.0-rc is the predecessor version of TiDB v5.0. In v5.0, PingCAP will be dedicated to helping enterprises quickly build applications based on TiDB, freeing them from worries about database performance, performance jitter, security, high availability, disaster recovery, troubleshooting SQL performance, and so on.

In v5.0, the key new features or improvements are as follows:

- Clustered index. When this feature is enabled, database performance is improved. For example, in the TPC-C tpmC test, TiDB's performance, with clustered index enabled, improves by 39%.
- Async commit. When this feature is enabled, the write latency is reduced. For example, in the Sysbench olpt-insert test, the write latency of TiDB, with async commit enabled, is reduced by 37.3%.
- Reduced jitters. This is achieved by improving the optimizer stability and by limiting system tasks' usages of I/O, network, CPU, and memory resources. For example, in the 72-hour performance test, the standard deviation of Sysbench TPS jitter is reduced from 11.09% to 3.36%.
- Raft Joint Consensus algorithm, which ensures the system availability during the Region membership change.
- Optimized `EXPLAIN` features and invisible index, which helps Database Administrators (DBAs) debug SQL statements more efficiently.
- Guaranteed reliability for enterprise data. You can back up data from TiDB to AWS S3 storage and Google Cloud GCS, or restore data from these cloud storage platforms.
- Improved performance of data import from or data export to AWS S3 storage or TiDB/MySQL, which helps enterprises quickly build applications on the cloud. For example, in the TPC-C test, the performance of importing 1 TiB data improves by 40%, from 254 GiB/h to 366 GiB/h.

16.11.8.1 SQL

16.11.8.1.1 Support clustered index (experimental)

When the clustered index feature is enabled, TiDB performance improves significantly (for example in the TPC-C tpmC test, TiDB's performance, with clustered index enabled, improves by 39%) in the following cases:

- When data is inserted, the clustered index reduces one write of the index data from the network.
- When a query with an equivalent condition only involves the primary key, the clustered index reduces one read of index data from the network.
- When a query with a range condition only involves the primary key, the clustered index reduces multiple reads of index data from the network.
- When a query with an equivalent or range condition involves the primary key prefix, the clustered index reduces multiple reads of index data from the network.

Clustered index defines the physical storage order of data in a table. The data in the table is sorted only according to the definition of the clustered index. Each table has only one clustered index.

Users can enable the clustered index feature by modifying the `tidb_enable_clustered_index` ↪ variable. When enabled, the feature takes effect only on newly created tables and applies to the primary key that has multiple columns or is non-integer types in a single column. If the primary key is an integer type in a single column, or if the table has no primary key, the data is sorted in the same way as before, without being affected by the clustered index.

For example, to check whether a table (`tbl_name`) has a clustered index, execute `select tidb_pk_type from information_schema.tables where table_name = '{` ↪ `tbl_name}'`.

- User document
- Related issue: #4841

16.11.8.1.2 Support invisible indexes

When users tune performance or select optimal indexes, they can set an index to be `Visible` or `Invisible` by using SQL statements. This setting can avoid performing resource-consuming operations, such as `DROP INDEX` and `ADD INDEX`.

To modify the visibility of an index, use the `ALTER INDEX` statement. After the modification, the optimizer decides whether to add this index to the index list based on the index visibility.

- User document
- Related issue: #9246

16.11.8.1.3 Support `EXCEPT` and `INTERSECT` operators

The `INTERSECT` operator is a set operator, which returns the intersection of the result sets of two or more queries. To some extent, it is an alternative to the `InnerJoin` operator.

The `EXCEPT` operator is a set operator, which combines the result sets of two queries and returns elements that are in the first query result but not in the second.

- User document
- Related issue: #18031

16.11.8.2 Transaction

16.11.8.2.1 Increase the success rate of executing pessimistic transactions

In the pessimistic transaction mode, if the tables involved in a transaction contain concurrent DDL operations or `SCHEMA VERSION` changes, the system automatically updates the transaction's `SCHEMA VERSION` to the latest to avoid the transaction being interrupted by DDL operations and to ensure the successful transaction commit. If the transaction is interrupted, the client receives the `Information schema is changed` error message.

- User document
- Related issue: #18005

16.11.8.3 Character set and collation

Support case-insensitive comparison sort for character sets.

- User document
- Related issue: #17596

16.11.8.4 Security

16.11.8.4.1 Support desensitizing error messages and log files

TiDB now supports desensitizing error messages and log files to avoid leaking sensitive information such as ID information and credit card number.

Users can enable the desensitization feature for different components:

- For the TiDB side, set the `tidb_redact_log=1` variable using SQL statements in tidb-server.
- For the TiKV side, set the `security.redact-info-log = true` configuration in tikv-server.
- For the PD side, set the `security.redact-info-log = true` configuration in pd-server. #2852 #3011
- For the TiFlash side, set the `security.redact_info_log = true` configuration in tiflash-server and set `security.redact-info-log = true` in tiflash-learner.

User document

Related issue: #18566

16.11.8.5 Performance improvements

16.11.8.5.1 Support async commit (experimental)

Enabling the async commit feature can significantly reduce the latency of transactions. For example, with this feature enabled, the latency of transactions in the Sysbench oltp-insert test is 37.3% lower than that when this feature is disabled.

Previously without the async commit feature, the statements being written were only returned to the client after the two-phase transaction commit finished. Now the async commit feature supports returning the result to the client after the first phase of the two-phase commit finishes. The second phase is then performed asynchronously in the background, thus reducing the latency of transaction commit.

However, when async commit is enabled, the external consistency of transactions can be guaranteed **only** when `tidb_guarantee_external_consistency = ON` is set. With async commit enabled, the performance might drop.

Users can enable this feature by setting the global variable `tidb_enable_async_commit = ON`.

- User document
- Related issue: #8316

16.11.8.5.2 Improve the optimizer's stability in index selection (experimental)

The optimizer's ability to always select a relatively suitable index greatly determines whether the latency of queries is stable. We have improved and refactored the statistics module to ensure that, for the same SQL statements, the optimizer does not select a different index from multiple candidate indexes each time due to missing or inaccurate statistics. The main improvements to help the optimizer select a relatively suitable index are as follows:

- Add more information to the statistics module, such as the multi-column NDV, the multi-column order dependency, and the multi-column function dependency.
- Refactor the statistics module.
    - Delete `TopN` values from `CMSKetch`.
    - Refactor the search logic of `TopN`.
    - Delete the `TopN` information from the histogram and create an index of the histogram for easy maintenance of Bucket NDV.

Related issue: #18065

16.11.8.5.3 Optimize performance jitter caused by imperfect scheduling or imperfect I/O flow control

The TiDB scheduling process occupies resources such as I/O, network, CPU, and memory. If TiDB does not control the scheduled tasks, QPS and delay might cause performance jitter due to resource preemption. After the following optimizations, in the 72-hour test, the standard deviation of Sysbench TPS jitter is reduced from 11.09% to 3.36%.

- Reduce the redundant scheduling issues caused by fluctuations of node capacity (always near the waterline) and caused by PD's `store-limit` configuration value set too large. This is achieved by introducing a new set of scheduling calculation formulas enabled via the `region-score-formula-version = v2` configuration item. #3269
- Enable the cross-Region merge feature by modifying `enable-cross-table-merge = true` to reduce the number of empty Regions. #3129
- Data compaction in the TiKV background occupies a lot of I/O resources. The system automatically adjusts the compaction rate to balance the contention for I/O resources between background tasks and foreground reads and writes. After enabling this feature via the `rate-limiter-auto-tuned` configuration item, the delay jitter is greatly reduced. #18011
- When TiKV performs garbage collection (GC) and data compaction, partitions occupy CPU and I/O resources. Overlapping data exists during the execution of these two tasks. To reduce I/O usage, the GC Compaction Filter feature combines these two tasks into one and executes them in the same task. This feature is still experimental and you can enable it via `gc.enable-compaction-filter = true`. #18009
- When TiFlash compresses or sorts data, it occupies a lot of I/O resources. The system alleviates contention for resources by limiting the compression and data sorting's use of I/O resources. This feature is still experimental and you can enable it via `bg_task_io_rate_limit`.

Related issue: #18005

16.11.8.5.4 Improve the stability of TiFlash in Real-time BI / Data Warehousing scenarios

- Limit the memory usage of DeltaIndex to avoid system out of memory (OOM) caused by excessive memory usage in the scenarios of huge data volume.
- Limit the I/O write traffic used by the background data sorting task to reduce the impact on the foreground tasks.
- Add new thread pools to queue coprocessor tasks, which avoids system OOM caused by excessive memory usage when processing coprocessors in high concurrency.

16.11.8.5.5 Other performance optimizations

- Improve the execution performance of `delete from table where id <?` statement. Its P99 performance improves by four times. #18028

- TiFlash supports concurrently reading and writing data in multiple local disks to improve performance.

16.11.8.6 High availability and disaster recovery

16.11.8.6.1 Improve system availability during Region membership change (experimental)

In the process of Region membership changes, "adding a member" and "deleting a member" are two operations performed in two steps. If a failure occurs when the membership change finishes, the Regions will become unavailable and an error of foreground application is returned. The introduced Raft Joint Consensus algorithm can improve the system availability during Region membership change. "adding a member" and "deleting a member" operations during the membership change are combined into one operation and sent to all members. During the change process, Regions are in an intermediate state. If any modified member fails, the system is still available. Users can enable this feature by modifying the membership variable by executing `pd-ctl config set enable-joint-consensus true`. #7587 #2860

- User document
- Related issue: #18079

16.11.8.6.2 Optimize the memory management module to reduce system OOM risks

- Reduce the memory consumption of caching statistics.
- Reduce the memory consumption of exporting data using the Dumpling tool.
- Reduced the memory consumption by storing the encrypted intermediate results of data to the disk.

16.11.8.7 Backup and restore

- The Backup & Restore tool (BR) supports backing up data to AWS S3 and Google Cloud GCS. (User document)
- The Backup & Restore tool (BR) supports restoring data from AWS S3 and Google Cloud GCS to TiDB. (User document)
- Related issue: #89

16.11.8.8 Data import and export

- TiDB Lightning supports importing Aurora snapshot data from AWS S3 storage to TiDB. (Related issue: #266)

- In the TPC-C test of importing 1 TiB of data into DBaaS T1.standard, the performance improves by 40%, from 254 GiB/h to 366 GiB/h.
- Dumpling supports exporting data from TiDB/MySQL to AWS S3 storage (experimental) (Related issue: #8, User document)

16.11.8.9 Diagnostics

16.11.8.9.1 Optimized `EXPLAIN` features with more collected information help users troubleshoot performance issues

When users troubleshoot SQL performance issues, they need detailed diagnostic information to determine the causes of performance issues. In previous TiDB versions, the information collected by the `EXPLAIN` statements was not detailed enough. DBAs performed troubleshooting only based on log information, monitoring information, or even on guess, which might be inefficient. The following improvements are made in TiDB v5.0 to help users troubleshoot performance issues more efficiently:

- `EXPLAIN ANALYZE` supports analyzing all DML statements and shows the actual performance plans and the execution information of each operator. #18056
- Users can use `EXPLAIN FOR CONNECTION` to analyze the status information of the SQL statements that are being executed. This information includes the execution duration of each operator and the number of processed rows. #18233
- More information is available in the output of `EXPLAIN ANALYZE`, including the number of RPC requests sent by operators, the duration of resolving lock conflicts, network latency, the scanned volume of deleted data in RocksDB, and the hit rate of RocksDB caches. #18663
- The detailed execution information of SQL statements is recorded in the slow log, which is consistent with the output information of `EXPLAIN ANALYZE`. This information includes the time consumed by each operator, the number of processed rows, and the number of sent RPC requests. #15009

User document

16.11.8.10 Deployment and maintenance

- Previously, when the configuration information of TiDB Ansible was imported to TiUP, TiUP put the user configuration in the `ansible-imported-configs` directory. When users later needed to edit the configuration using `tiup cluster edit-config`, the imported configuration was not displayed in the editor interface, which could be confusing for users. In TiDB v5.0, when TiDB Ansible configuration is imported, TiUP puts the configuration information both in `ansible-imported-configs` and in the editor interface. With this improvement, users can see the imported configuration when they are editing the cluster configuration.

- Enhanced `mirror` command that supports merging multiple mirrors into one, publishing components in the local mirror, and adding component owners in the local mirror. #814
    - For a large enterprise, especially for the financial industry, any change in the production environment is given careful consideration. It can be troublesome if each version requires users to use a CD for installation. In TiDB v5.0, the `merge` ↪ command of TiUP supports merging multiple installation packages into one, which makes the installation easier.
    - In v4.0, users had to start tiup-server to publish the self-built mirror, which was not convenient enough. In v5.0, users can publish the self-built mirror simply by using `tiup mirror set` to set the current mirror to the local mirror.

16.12 v4.0

16.12.1 TiDB 4.0.16 Release Notes

Release date: December 17, 2021

TiDB version: 4.0.16

16.12.1.1 Compatibility changes

- TiKV
    - Before v4.0.16, when TiDB converts an illegal UTF-8 string to a Real type, an error is reported directly. Starting from v4.0.16, TiDB processes the conversion according to the legal UTF-8 prefix in the string #11466
- Tools
    - TiCDC
        - Change the default value of Kafka Sink `max-message-bytes` to 1 MB to prevent TiCDC from sending too large messages to Kafka clusters #2962
        - Change the default value of Kafka Sink `partition-num` to 3 so that TiCDC distributes messages across Kafka partitions more evenly #3337

16.12.1.2 Improvements

- TiDB
    - Upgrade the Grafana version from 7.5.7 to 7.5.11
- TiKV

- Reduce disk space consumption by adopting the zstd algorithm to compress SST files when restoring data using Backup & Restore or importing data using Local-backend of TiDB Lightning #11469

- Tools

  - Backup & Restore (BR)

    - Improve the robustness of restoring #27421

  - TiCDC

    - Add a tick frequency limit to EtcdWorker to prevent frequent etcd writes from affecting PD services #3112
    - Optimize rate limiting control on TiKV reloads to reduce gPRC congestion during changefeed initialization #3110

16.12.1.3 Bug fixes

- TiDB

  - Fix the query panic caused by overflow in the statistics module when converting a range to points for cost estimation #23625
  - Fix wrong results of the control functions (such as `IF` and `CASE WHEN`) when using the `ENUM` type data as parameters of such functions #23114
  - Fix the issue that the `GREATEST` function returns inconsistent results due to different values of `tidb_enable_vectorized_expression` (`on` or `off`) #29434
  - Fix the panic when applying index join on prefix indexes in some cases #24547
  - Fix the issue that planner might cache invalid plans for `join` in some cases #28087
  - Fix a bug that TiDB cannot insert `null` into a non-null column when `sql_mode` is empty #11648
  - Fix the wrong result type of the `GREATEST` and `LEAST` functions #29019
  - Fix the `privilege check fail` error when performing the `grant` and `revoke` operations to grant and revoke global level privileges #29675
  - Fix the panic when using the `CASE WHEN` function on the `ENUM` data type #29357
  - Fix wrong results of the `microsecond` function in vectorized expressions #29244
  - Fix wrong results of the `hour` function in vectorized expression #28643
  - Fix the issue that optimistic transaction conflicts might cause transactions to block each other #11148
  - Fix the issue of incomplete log information from the `auto analyze` result #29188
  - Fix the issue that using an invalid default date does not report an error when the `SQL_MODE` is 'NO_ZERO_IN_DATE' #26766
  - Fix the issue that the Coprocessor Cache panel in Grafana does not display metrics. Now, Grafana displays the number of `hits/miss/evict` #26338
  - Fix the issue that concurrently truncating the same partition causes DDL statements to stuck #26229

    - Fix the issue that the length information is wrong when converting `Decimal` to `String` #29417
    - Fix the issue of an extra column in the query result when `NATURAL JOIN` is used to join multiple tables #29481
    - Fix the issue that `TopN` is wrongly pushed down to `indexPlan` when `IndexScan` uses a prefix index #29711
    - Fix the issue that retrying transactions with the auto-increment columns of `DOUBLE` type causes data corruption #29892

- TiKV

    - Fix a panic issue that occurs when Region merge, ConfChange, and Snapshot happen at the same time in extreme conditions #11475
    - Fix the issue of negative sign when the decimal divide result is zero #29586
    - Fix the issue that the average latency of the by-instance gRPC requests is inaccurate in TiKV metrics #11299
    - Fix the issue of TiCDC panic that occurs when the downstream database is missing #11123
    - Fix the issue that the Raft connection is broken when the channel is full #11047
    - Fix the issue that TiDB cannot correctly identify whether the `Int64` types in `Max`/`Min` functions are a signed integer or not, which causes the wrong calculation result of `Max`/`Min` #10158
    - Fix the issue that CDC adds scan retries frequently due to the Congest error #11082

- PD

    - Fix a panic issue that occurs after the TiKV node is removed #4344
    - Fix slow leader election caused by stucked region syncer #3936
    - Support that the evict leader scheduler can schedule regions with unhealthy peers #4093

- TiFlash

    - Fix the issue that TiFlash fails to start up on some platforms due to the absence of library `nsl`

- Tools

    - TiDB Binlog

        - Fix the bug that Drainer exits when transporting a transaction greater than 1 GB #28659

    - TiCDC

        - Fix the negative value error in the changefeed checkpoint lag #3010
        - Fix OOM in container environments #1798

- Fix the TiCDC replication interruption issue when multiple TiKVs crash or during a forced restart #3288
- Fix the memory leak issue after processing DDLs #3174
- Fix the issue that changefeed does not fail fast enough when the ErrGCTTLExceeded error occurs #3111
- Fix the issue that TiCDC replication task might terminate when the upstream TiDB instance unexpectedly exits #3061
- Fix the issue that TiCDC process might panic when TiKV sends duplicate requests to the same Region #2386
- Fix the issue that the volume of Kafka messages generated by TiCDC is not constrained by `max-message-size` #2962
- Fix the issue that `tikv_cdc_min_resolved_ts_no_change_for_1m` keeps alerting when there is no changefeed #11017
- Fix the issue that TiCDC sync task might pause when an error occurs during writing a Kafka message #2978
- Fix the issue that some partitioned tables without valid indexes might be ignored when `force-replicate` is enabled #2834
- Fix the memory leak issue when creating a new changefeed #2389
- Fix the issue that might cause inconsistent data due to Sink components advancing resolved ts early #3503
- Fix the issue that scanning stock data might fail due to TiKV performing GC when scanning stock data takes too long #2470
- Fix the issue that the changefeed update command does not recognize global command line parameters #2803

16.12.2 TiDB 4.0.15 Release Notes

Release Date: September 27, 2021

TiDB version: 4.0.15

16.12.2.1 Compatibility changes

- TiDB
  - Fix the issue that executing `SHOW VARIABLES` in a new session is slow. This fix reverts some changes made in #21045 and might cause compatibility issues. #24326
  - The following bug fixes change execution results, which might cause upgrade incompatibilities:
    - Fix the issue that `greatest(datetime)union null` returns empty string #26532
    - Fix the issue that the `having` clause might not work correctly #26496
    - Fix the wrong execution results that occur when the collations around the `between` expression are different #27146

- Fix the result wrong that occurs when the argument of the `extract` function is a negative duration #27236
- Fix the wrong execution results that occur when the column in the `group_concat` function has a non-bin collation #27429
- Fix the issue that column information is missed when converting the `Apply` operator to `Join` #27233
- Fix the issue of unexpected behavior when casting the invalid string to `DATE` #26762
- Fix a bug that the `count distinct` result on multiple columns is wrong when the new collation is enabled #27091

16.12.2.2 Feature enhancement

- TiKV
  - Support changing TiCDC configurations dynamically #10645

16.12.2.3 Improvements

- TiDB
  - Trigger auto-analyze based on the histogram row count #24237
- TiKV
  - Handle read ready and write ready separately to reduce read latency #10475
  - The slow log of TiKV coprocessor only considers the time spent on processing requests. #10841
  - Drop log instead of blocking threads when the slogger thread is overloaded and the queue is filled up #10841
  - Reduce the size of Resolved TS messages to save network bandwidth #2448
- PD
  - Improve the performance of synchronizing Region information between PDs #3932
- Tools
  - Backup & Restore (BR)
    - Split and scatter Regions concurrently to improve restore speed #1363
    - Retry BR tasks when encountering the PD request error or the TiKV I/O timeout error #27787
    - Reduce empty Regions when restoring many small tables to avoid affecting cluster operations after the restore #1374

    - Perform the `rebase auto id` operation while creating tables, which saves the separate `rebase auto id` DDL operation and speeds up restore #1424
- Dumpling
    - Filter the skipped databases before getting the table information to improve the filtering efficiency of `SHOW TABLE STATUS` #337
    - Use `SHOW FULL TABLES` to get table information for tables to be exported, because `SHOW TABLE STATUS` cannot work properly in some MySQL versions #322
    - Support backing up MySQL-compatible databases that do not support the `START TRANSACTION ... WITH CONSISTENT SNAPSHOT` or the `SHOW CREATE TABLE` syntax #309
    - Refine the Dumpling warning log to avoid the misleading information that a dump fails #340
- TiDB Lightning
    - Support importing data into tables that have expression index or the index that depends on virtual generated columns #1404
- TiCDC
    - Always pulls old values from TiKV internally to improve usability #2397
    - Reduce the goroutine usage when a table's Regions are all transferred away from a TiKV node #2284
    - Optimize workerpool for fewer goroutines when concurrency is high #2211
    - Execute DDL statements asynchronously to avoid affecting other changefeeds #2295
    - Add a global gRPC connection pool and share gRPC connections among KV clients #2531
    - Fail fast for unrecoverable DML errors #1724
    - Optimize memory management when the Unified Sorter is using memory to sort data #2553
    - Add Prometheus metrics for DDL executions #2595 #2669
    - Prohibit operating TiCDC clusters across major or minor versions #2601
    - Remove `file sorter` #2325
    - Clean up changefeed metrics when a changefeed is removed, and clean up processor metrics when a processor exits #2156
    - Optimize the lock-resolving algorithm after a Region is initialized #2188

16.12.2.4 Bug fixes

- TiDB
    - Fix a bug that collation is incorrectly set for binary literals when building ranges #23672

- Fix the "index out of range" error that occurs when a query includes both `GROUP BY` and `UNION` #26553
- Fix the issue that TiDB might fail to send requests if TiKV has tombstone stores #23676 #24648
- Remove the undocumented `/debug/sub-optimal-plan` HTTP API #27264
- Fix the issue of wrong character set and collation for the `case when` expression #26662

- TiKV
  - Fix the issue that BR reports the "file already exists" error when TDE is enabled during data restore #1179
  - Fix the potential disk full issue caused by corrupted snapshot files #10813
  - Fix the issue that TiKV deletes stale Regions too frequently #10680
  - Fix the issue that TiKV frequently reconnects the PD client #9690
  - Check stale file information from the encryption file dictionary #9115

- PD
  - Fix the issue that PD does not fix the down peers in time #4077
  - Fix a bug that PD might panic when scaling out TiKV #3868

- TiFlash
  - Fix the potential issue of data inconsistency that occurs when TiFlash is deployed on multiple disks
  - Fix a bug of incorrect results that occurs when queries contain filters like `CONSTANT <, <=, >, >=, or COLUMN`
  - Fix the issue that the store size in metrics is inaccurate under heavy writing
  - Fix a potential bug that TiFlash cannot restore data when deployed on multiple disks
  - Fix the potential issue that TiFlash cannot garbage-collect the delta data after running for a long time

- Tools
  - Backup & Restore (BR)
    - Fix a bug that the average speed is inaccurately calculated for backup and restore #1405
  - TiCDC
    - Fix the `ErrSchemaStorageTableMiss` error that occurs when the DDL Job duplication is encountered in the integrated test #2422
    - Fix a bug that a changefeed cannot be removed if the `ErrGCTTLExceeded` error occurs #2391

- Fix the issue that outdated capture might appear in the output of the `capture list` command #2388
- Fix the deadlock issue in the TiCDC processor #2017
- Fix a data inconsistency issue that occurs because multiple processors might write data to the same table when this table is being re-scheduled #2230
- Fix a bug that the `EtcdWorker` snapshot isolation is violated in metadata management #2557
- Fix the issue that the changefeed cannot be stopped due to the DDL sink error #2552
- Fix the issue of TiCDC Open Protocol: TiCDC outputs an empty value when there is no change in a transaction #2612
- Fix a bug that causes TiCDC to panic on the unsigned `TINYINT` type #2648
- Decrease the gRPC window size to avoid the OOM that occurs when TiCDC captures too many Regions #2202
- Fix the OOM issue that occurs when TiCDC captures too many Regions #2673
- Fix the issue of process panic that occurs when encoding the data types such as `mysql.TypeString`, `mysql.TypeVarString`, `mysql.TypeVarchar` into JSON #2758
- Fix the a memory leak issue that might occur when creating a new changefeed #2389
- Fix a bug that DDL handling fails when a changefeed starts at the finish TS of a schema change #2603
- Fix the issue of potential DDL loss when the owner crashes when executing DDL statements #1260
- Fix the issue of insecure concurrent access to the map in `SinkManager` #2298

16.12.3 TiDB 4.0.14 Release Notes

Release date: July 27, 2021

TiDB version: 4.0.14

16.12.3.1 Compatibility changes

- TiDB
  - Change the default value of `tidb_multi_statement_mode` from `WARN` to `OFF` in v4.0. It is recommended to use the multi-statement feature of your client library instead. See the documentation on `tidb_multi_statement_mode` for details. #25749
  - Upgrade Grafana dashboard from v6.1.16 to v7.5.7 to solve two security vulnerabilities. See the Grafana blog post for details.
  - Change the default value of the `tidb_stmt_summary_max_stmt_count` variable from 200 to 3000 #25872

- TiKV
    - Change the default value of `merge-check-tick-interval` from `10` to `2` to speed up the Region merge process #9676

16.12.3.2 Feature enhancements

- TiKV
    - Add a metric `pending` to monitor the number of pending PD heartbeats, which helps locate the issue of slow PD threads #10008
    - Support using the virtual-host addressing mode to make BR support the S3-compatible storage #10242
- TiDB Dashboard
    - Support OIDC SSO. By setting the OIDC-compatible SSO services (such as Okta and Auth0), users can log into TiDB Dashboard without entering the SQL password. #960
    - Add the **Debug API** UI, which is an alternative method to the command line to call several common TiDB and PD internal APIs for advanced debugging #927

16.12.3.3 Improvements

- TiDB
    - Change the `LOCK` record into the `PUT` record for the index keys using `point get` or `batch point get` for `UPDATE` reads #26223
    - Support the MySQL system variable `init_connect` and its associated features #26031
    - Support the stable result mode to make the query results more stable #26003
    - Support pushing down the built-in function `json_unquote()` to TiKV #25721
    - Make the SQL Plan Management (SPM) not affected by the character set #23295
- TiKV
    - Shutdown the status server first to make sure that the client can correctly check the shutdown status #10504
    - Always respond to stale peers to make sure that these peers are cleared quicker #10400
    - Limit the TiCDC sink's memory consumption #10147
    - When a Region is too large, use the even split to speed up the split process #10275
- PD

    - Reduce the conflicts among multiple schedulers that run at the same time #3858 #3854

- TiDB Dashboard

    - Update TiDB Dashboard to v2021.07.17.1 #3882
    - Support sharing the current session as a read-only session to avoid further modification to it #960

- Tools

    - Backup & Restore (BR)
        - Speed up restore by merging small backup files #655
    - Dumpling
        - Always split tables using `_tidb_rowid` when the upstream is a TiDB v3.x cluster, which helps reduce TiDB's memory usage #306
    - TiCDC
        - Improve the error message returned when a PD endpoint misses the certificate #1973
        - Make the sorter I/O errors more user-friendly #1976
        - Add a concurrency limit on the Region incremental scan in the KV client to reduce the pressure of TiKV #1926
        - Add metrics for the table memory consumption #1884
        - Add `capture-session-ttl` to the TiCDC server configuration #2169

16.12.3.4 Bug fixes

- TiDB

    - Fix the issue that the `SELECT` result is incompatible with MySQL when joining a subquery with a `WHERE` clause evaluated to `false` #24865
    - Fix the calculation error of the `ifnull` function that occurs when the argument is the `ENUM` or `SET` type #24944
    - Fix the wrong aggregate pruning in some cases #25202
    - Fix the incorrect result of the merge join operation that might occur when the column is the `SET` type #25669
    - Fix the issue that TiDB returns wrong results for cartesian join #25591
    - Fix the panic issue that occurs when `SELECT ... FOR UPDATE` works on a join operation and the join uses a partitioned table #20028
    - Fix the issue that the cached `prepared` plan is incorrectly used for `point get` #24741
    - Fix the issue that the `LOAD DATA` statement can abnormally import non-utf8 data #25979

- Fix a potential memory leak issue that occurs when accessing the statistics via an HTTP API #24650
- Fix a security issue that occurs when executing the `ALTER USER` statement #25225
- Fix a bug that the `TIKV_REGION_PEERS` table cannot correctly handle the `DOWN` status #24879
- Fix the issue that invalid strings are not truncated when parsing `DateTime` #22231
- Fix the issue that the `select into outfile` statement might have no result when the column type is `YEAR` #22159
- Fix the issue that the query result might be wrong when `NULL` is in the `UNION` subquery #26532
- Fix the issue that the projection operator in execution might cause panic in some cases #26534

- TiKV

  - Fix the issue that the duration calculation might panic on certain platforms #related-issue
  - Fix the wrong function that casts `DOUBLE` to `DOUBLE` #25200
  - Fix the issue that the panic log might be lost when using the async logger #8998
  - Fix the panic issue that occurs when building a snapshot twice if encryption is enabled #9786 #10407
  - Fix the wrong arguments type of the `json_unquote()` function in the coprocessor #10176
  - Fix the issues of suspicious warnings during shutdown and the non-deterministic response from Raftstore #10353 #10307
  - Fix the issue of backup threads leak #10287
  - Fix the issue that Region split might panic and corrupt the metadata if the split process is too slow and Region merge is on-going #8456 #8783
  - Fix the issue that the Region heartbeats prevent TiKV from splitting large Regions in some situations #10111
  - Fix the wrong statistics caused by the format inconsistency of CM Sketch between TiKV and TiDB #25638
  - Fix the wrong statistics of the `apply wait duration` metric #9893
  - Fix the "Missing Blob" error after using `delete_files_in_range` in Titan #10232

- PD

  - Fix a bug that the scheduler might reappear after executing the delete operation #2572
  - Fix the data race issue that might occur when the scheduler is started before the temporary configuration is loaded #3771
  - Fix a PD panic issue that might occur during the Region scattering operation #3761
  - Fix the issue that the priority of some operators is not set correctly #3703

- Fix a PD panic issue that might occur when deleting the `evict-leader` scheduler from a non-existent store #3660
- Fix the issue that the PD Leader re-election is slow when there are many stores #3697

- TiDB Dashboard

  - Fix the issue that the **Profiling** UI cannot profile all TiDB instances #944
  - Fix the issue that the **Statements** UI does not display "Plan Count" #939
  - Fix the issue that the **Slow Query** UI might display the "unknown field" error after cluster upgrade #902

- TiFlash

  - Fix the potential panic issue that occurs when compiling DAG requests
  - Fix the panic issue that occurs when the read load is heavy
  - Fix the issue that TiFlash keeps restarting because of the split failure in column storage
  - Fix a potential bug that TiFlash cannot delete the delta data
  - Fix the incorrect results that occur when cloning the shared delta index concurrently
  - Fix a bug that TiFlash fails to restart in the case of incomplete data
  - Fix the issue that the old dm files cannot be removed automatically
  - Fix the panic issue that occurs when executing the `SUBSTRING` function with specific arguments
  - Fix the issue of incorrect results when casting the `INTEGER` type to the `TIME` type

- Tools

  - Backup & Restore (BR)

    - Fix the issue that the data restore from the `mysql` schema might fail #1142

  - TiDB Lightning

    - Fix the issue that TiDB Lightning fails to parse the `DECIMAL` type data in Parquet files #1276
    - Fix the EOF error reported when TiDB Lightning splits the imported large CSV files #1133
    - Fix a bug that an excessively large base value is generated when TiDB Lightning imports tables with the `auto_increment` column of the `FLOAT` or `DOUBLE` type #1185
    - Fix the issue of TiDB Lightning panic that occurs when generating KV data larger than 4 GB #1128

  - Dumpling

    - When using Dumpling to export data to the S3 storage, the `s3:ListBucket` ↪ permission is no longer required on the entire bucket. The permission is required only on the data source prefix. #898

- TiCDC
    - Fix the issue of extra partition dispatching after adding new table partitions #2205
    - Fix the panic issue that occurs when TiCDC fails to read `/proc/meminfo` #2023
    - Reduce TiCDC's runtime memory consumption #2011 #1957
    - Fix a bug that some MySQL connection might leak after MySQL sink meets the error and pauses #1945
    - Fix the issue that TiCDC changefeed cannot be created when start TS is less than current TS minus GC TTL #1839
    - Reduce memory `malloc` in sort heap to avoid too much CPU overhead #1853
    - Fix a bug that the replication task might stop when moving a table #1827

16.12.4 TiDB 4.0.13 Release Notes

Release date: May 28, 2021

TiDB version: 4.0.13

16.12.4.1 New Features

- TiDB
    - Support changing an `AUTO_INCREMENT` column to an `AUTO_RANDOM` one #24608
    - Add the `infoschema.client_errors_summary` tables to help users keep track of the errors that have been returned to clients #23267

16.12.4.2 Improvements

- TiDB
    - Avoid frequently reading the `mysql.stats_histograms` table if the cached statistics is up-to-date to avoid high CPU usage #24352
- TiKV
    - Make the calculation process of `store used size` more precise #9904
    - Set more Regions in the `EpochNotMatch` message to reduce Region misses #9731
    - Speed up freeing the memory accumulated in the long-running cluster #10035
- PD
    - Optimize the metrics of TSO processing time to help users determine whether the TSO processing time at the PD side is too long #3524

    - Update the dashboard version to v2021.03.12.1 #3469

- TiFlash

    - Automatically clean archived data to free up disk space

- Tools

    - Backup & Restore (BR)

        - Support backing up user tables created in the `mysql` schema #1077
        - Update `checkVersion` to check the cluster data and the backup data #1090
        - Tolerate a small number of TiKV node failures during backup #1062

    - TiCDC

        - Implement the processor flow control to avoid memory overflow (OOM) #1751
        - Support cleaning up stale temporary files in Unified Sorter and prevent multiple `cdc server` instances from sharing the same `sort-dir` directory #1741
        - Add the HTTP handler for the failpoint #1732

16.12.4.3 Bug Fixes

- TiDB

    - Fix the panic issue that occurs when the `UPDATE` statement with a subquery updates the generated column #24658
    - Fix the issue that causes duplicate query results when using the multi-column index for data reads #24634
    - Fix the issue that causes wrong query result when using the `BIT` type constant as the divisor in the DIV expression #24266
    - Fix the issue that the `NO_ZERO_IN_DATE` SQL mode does not take effect for the default column value set in DDL statements #24185
    - Fix an issue which causes wrong query results when using `UNION` between a `BIT` type column and an `INTEGER` type column #24026
    - Fix the issue that the `TableDual` plans are mistakenly created when comparing the `BINARY` type and the `CHAR` type #23917
    - Fix the issue that the `insert ignore on duplicate` statement might unexpectedly delete table records #23825
    - Fix the issue that the Audit plugin causes TiDB panic #23819
    - Fix the issue that the `HashJoin` operator incorrectly processes the collation #23812
    - Fix the issue of disconnection that occurs when `batch_point_get` incorrectly handles abnormal values in the pessimistic transaction #23778

- Fix the issue of inconsistent indexes that occurs when the `tidb_row_format_version` ↪ configuration value is set to `1` and the `enable_new_collation` value is set to `true` #23772
- Fix a bug that occurs when comparing the `INTEGER` type column with the `STRING` constant value #23705
- Fix the error that occurs when the `BIT` type column is passed into the `approx_percent` function #23702
- Fix a bug that causes TiDB to mistakenly report the `TiKV server timeout` error when executing TiFlash batch requests #23700
- Fix the issue that the `IndexJoin` operator returns wrong results on the prefix column index #23691
- Fix the issue which causes wrong query results because the collation on the `BINARY` type column is not properly handled #23598
- Fix the issue of query panic that occurs when the `UPDATE` statement contains the join query with the `HAVING` clause #23575
- Fix the issue that causes TiFlash to return wrong results when using the `NULL` constant in the comparison expression #23474
- Fix the issue of wrong results when comparing the `YEAR` type column with the `STRING` constant #23335
- Fix the issue that `group_concat` panics when `session.group_concat_max_len` is set too small #23257
- Fix the issue of wrong query results that occurs when using the `BETWEEN` expression for the `TIME` type column #23233
- Fix the issue of privilege check in the `DELETE` statements #23215
- Fix the issue that no error is reported when inserting invalid strings to the `DECIMAL` type column #23196
- Fix the issue of parsing error occurred when inserting data to the `DECIMAL` type columns #23152
- Fix the issue that the `USE_INDEX_MERGE` hint does not take effect #22924
- Fix a bug that the query returns wrong results when using `ENUM` or `SET` columns in the `WHERE` clause as an filter #22814
- Fix a bug that the query returns wrong results when using the clustered index and the new collation at the same time #21408
- Fix the panic that occurs when executing `ANALYZE` with `enable_new_collation` enabled #21299
- Fix the issue that SQL views does not correctly handle the default roles associated with the SQL DEFINER #24531
- Fix the issue that cancelling DDL jobs gets stuck #24445
- Fix the issue that the `concat` function incorrectly handles the collation #24300
- Fix a bug that the query returns wrong results when the `SELECT` field has an `IN` subquery and the subquery's outer side contains `NULL` tuples #24022
- Fix a bug that TiFlash is chosen wrongly by the optimizer when `TableScan` is in descending order #23974
- Fix a bug that the `point_get` plan returns the column name that is inconsistent with that of MySQL #23970

- Fix the issue that executing the `show table status` statement on a database with a upper-cased name returns wrong results #23958
- Fix a bug that the users who do not have the `INSERT` and `DELETE` privileges on a table at the same time can perform the `REPLACE` operation #23938
- Fix the issue that the results of the `concat`/`make_set`/`insert` expressions are wrong because the collation is incorrectly handled #23878
- Fix the panic that occurs when executing a query on the table that has `RANGE` partitions #23689
- Fix the issue: In the cluster of an earlier version, if the `tidb_enable_table_partition` ↪ variable is set to `false`, the tables that contain partitions are handled as non-partitioned tables. Executing `batch point get` queries on this table, when the cluster is upgraded to a later version, causes connection panic. #23682
- Fix the issue that when TiDB is configured to listen on TCP and UNIX sockets, the remote hosts over the TCP connection are not correctly validated for connection #23513
- Fix a bug that the non-default collation causes wrong query results #22923
- Fix a bug that the **Coprocessor Cache** panel of Grafana does not work #22617
- Fix the error that occurs when the optimizer accesses the statistic cache #22565

- TiKV

  - Fix a bug that TiKV cannot start if the `file_dict` file is not fully written into the disk that has been full #9963
  - Limit TiCDC's scan speed at 128MB/s by default #9983
  - Reduce the memory usage of TiCDC's initial scan #10133
  - Support the back pressure for TiCDC's scan speed #10142
  - Fix a potential OOM issue by avoiding unnecessary reads to get TiCDC old values #10031
  - Fix a TiCDC OOM issue caused by reading old values #10197
  - Add a timeout mechanism for S3 storages to avoid the client hanging without responses #10132

- TiFlash

  - Fix the issue that number of `delta-merge-tasks` is not reported to Prometheus
  - Fix the TiFlash panic issue that occurs during `Segment Split`
  - Fix the issue that the `Region write Duration (write blocks)` panel in Grafana is shown in a wrong place
  - Fix the potential issue that the storage engine fails to remove data
  - Fix the issue of incorrect results when casting the `TIME` type to the `INTEGER` type
  - Fix a bug that the behavior of the `bitwise` operator is different from that of TiDB
  - Fix the issue of incorrect results when casting the `STRING` type to the `INTEGER` type
  - Fix the issue that consecutive and fast writes might make TiFlash out of memory

- Fix the potential issue that the exception of null pointer might be raised during the table GC
- Fix the TiFlash panic issue that occurs when writing data to dropped tables
- Fix the TiFlash panic issue that occurs during BR restore
- Fix a bug that the weights of some characters are wrong when using the general CI collation
- Fix the potential issue that data will be lost in tombstoned tables
- Fix the issue of incorrect results when comparing the string which contains zero bytes
- Fix the issue that the logical function returns wrong results if the input column contains null constants
- Fix the issue that the logical function only accepts the numeric type
- Fix the issue of incorrect results that occurs when the timestamp value is `1970-01-01` and the timezone offset is negative
- Fix the issue that hash value of `Decimal256` is not stable

- Tools

    - TiCDC

        - Fix the deadlock issue caused by the flow control when the sorter's input channel has been blocked #1779
        - Fix the issue that the TiKV GC safe point is blocked due to the stagnation of TiCDC changefeed checkpoint #1756
        - Revert the update in `explicit_defaults_for_timestamp` which requires the `SUPER` privilege when replicating data to MySQL #1749

    - TiDB Lightning

        - Fix a bug that TiDB Lightning's TiDB-backend cannot load any data when autocommit is disabled

16.12.5 TiDB 4.0.12 Release Notes

Release date: April 2, 2021

TiDB version: 4.0.12

16.12.5.1 New Features

- TiFlash

    - Add tools to check the status of `tiflash replica` for online rolling updates

16.12.5.2 Improvements

- TiDB
    - Refine the output information of the `EXPLAIN` statement for the `batch cop` mode #23164
    - Add the warning information for expressions that cannot be pushed to the storage layer in the output of the `EXPLAIN` statement #23020
    - Migrate a part of the DDL package code from `Execute`/`ExecRestricted` to the safe API (2) #22935
    - Migrate a part of the DDL package code from `Execute`/`ExecRestricted` to the safe API (1) #22929
    - Add `optimization-time` and `wait-TS-time` into the slow log #22918
    - Support querying `partition_id` from the `infoschema.partitions` table #22489
    - Add `last_plan_from_binding` to help the users know whether a SQL statement's execution plan is matched with the hints in the binding #21430
    - Scatter truncated tables without the `pre-split` option #22872
    - Add three format specifiers for the `str_to_date` expression #22812
    - Record the `PREPARE` execution failure as `Failed Query OPM` in the metrics monitor #22672
    - Do not report errors for the `PREPARE` execution if `tidb_snapshot` is set #22641
- TiKV
    - Prevent a large number of reconnections in a short period of time #9879
    - Optimize the write operations and Batch Get in the scenarios of many tombstones #9729
    - Change the default value of `leader-transfer-max-log-lag` to `128` to increase the success rate of leader transfer #9605
- PD
    - Update the Region cache only when `pending-peers` or `down-peers` changes, which reduces the pressure of updating heartbeats #3471
    - Prevent the Regions in `split-cache` from becoming the target of merge #3459
- TiFlash
    - Optimize the configuration file and remove useless items
    - Reduce the size of TiFlash binary files
    - Use an adaptive aggressive GC strategy to reduce memory usage
- Tools
    - TiCDC

    - Add a double confirmation when users create or resume the changefeed with the `start-ts` or `checkpoint-ts` 1 day before the current timestamp #1497
    - Add Grafana panels for the Old Value feature #1571
  - Backup & Restore (BR)
    - Log the `HTTP_PROXY` and `HTTPS_PROXY` environmental variables #827
    - Improve the backup performance when there are many tables #745
    - Report errors if the service safe point check fails #826
    - Add the `cluster_version` and `br_version` information in `backupmeta` #803
    - Add retry for external storage errors to increase the success rate of backup #851
    - Reduce memory usage during backup #886
  - TiDB Lightning
    - Check the TiDB cluster version before running TiDB Lightning to avoid unexpected errors #787
    - Fail fast when TiDB Lightning meets the `cancel` error #867
    - Add `tikv-importer.engine-mem-cache-size` and `tikv-importer.local-writer-mem-cache-size` configuration items to balance between memory usage and performance #866
    - Run `batch split region` in parallel for TiDB Lightning's Local-backend to increase the import speed #868
    - When using TiDB Lightning to import data from a S3 storage, TiDB Lightning no longer requires the `s3:ListBucket` permission #919
    - When resuming from a checkpoint, TiDB Lightning keeps using the original engine #924

16.12.5.3 Bug Fixes

- TiDB
  - Fix the issue that the `get` variable expression goes wrong when the session variable is hexadecimal literals#23372
  - Fix the issue that wrong collation is used when creating the fast execution plan for the `Enum` or `Set` type #23292
  - Fix the possible wrong result of the `nullif` expression when it is used with `is-null` #23279
  - Fix the issue that the auto-analysis is triggered outside its time range #23219
  - Fix the issue that the `CAST` function might ignore errors for the `point get` plan #23211
  - Fix a bug that prevents SPM from taking effect when `CurrentDB` is empty #23209
  - Fix the issue of possible wrong table filters for the IndexMerge plan #23165
  - Fix the issue of unexpected `NotNullFlag` in the returning types of the `NULL` constant #23135
  - Fix a bug that collation might not be handled by the text type #23092

- Fix the issue that the range partition might incorrectly handle the `IN` expression #23074
- Fix the issue that after marking a TiKV store as tombstone, starting new TiKV stores with different StoreIDs with the same IP address and port keeps returning the `StoreNotMatch` error #23071
- Do not adjust the `INT` type when it is `NULL` and compared with `YEAR` #22844
- Fix the issue of lost connection when loading data on tables with the `auto_random` column #22736
- Fix the issue of DDL hangover when the DDL operation meets panic in the cancelling path #23297
- Fix the wrong key range of index scan when comparing the `YEAR` column with `NULL` #23104
- Fix the issue that a successfully created view is failed to use #23083

- TiKV

  - Fix the issue that the `IN` expression does not properly handle unsigned/signed integers #9850
  - Fix the issue that the ingest operation is not re-entrant #9779
  - Fix the issue that the space is missed when converting JSON to string in TiKV coprocessor #9666

- PD

  - Fix a bug that the isolation level is wrong when the store lacks the label #3474

- TiFlash

  - Fix the issue of incorrect execution results when the default value of the `binary` type column contains leading or tailing zero bytes
  - Fix a bug that TiFlash fails to synchronize schema if the name of the database contains special characters
  - Fix the issue of incorrect results when handling the `IN` expression with decimal values
  - Fix a bug that the metric for the opened file count shown in Grafana is high
  - Fix a bug that TiFlash does not support the `Timestamp` literal
  - Fix the potential not responding issue while handling the `FROM_UNIXTIME` expression
  - Fix the issue of incorrect results when casting string as integer
  - Fix a bug that the `like` function might return wrong results

- Tools

  - TiCDC

    - Fix a disorder issue of the `resolved ts` event #1464

- Fix a data loss issue caused by wrong table scheduling due to the network problem #1508
- Fix a bug of untimely release of resources after a processor is stopped #1547
- Fix a bug that the transaction counter is not correctly updated, which might cause database connection leak #1524
- Fix the issue that multiple owners can co-exist when PD has jitter, which might lead to table missing #1540

- Backup & Restore (BR)
  - Fix a bug that `WalkDir` for the s3 storage returns `nil` if the target path is bucket name #733
  - Fix a bug that the `status` port is not served with TLS #839
- TiDB Lightning
  - Fix the error that TiKV Importer might ignore that the file has already existed #848
  - Fix a bug that the TiDB Lightning might use the wrong timestamp and read the wrong data #850
  - Fix a bug that TiDB Lightning's unexpected exit might cause damaged checkpoint file #889
  - Fix the issue of possible data error that occurs because the `cancel` error is ignored #874

16.12.6 TiDB 4.0.11 Release Notes

Release date: February 26, 2021

TiDB version: 4.0.11

16.12.6.1 New Features

- TiDB
  - Support the `utf8_unicode_ci` and `utf8mb4_unicode_ci` collations #22558
- TiKV
  - Support the `utf8mb4_unicode_ci` collation #9577
  - Support the `cast_year_as_time` collation #9299
- TiFlash
  - Add a Coprocessor thread pool to queue Coprocessor requests for execution, which avoids out of memory (OOM) in some cases, and add the `cop_pool_size` ↪ and `batch_cop_pool_size` configuration items with the default values of `NumOfPhysicalCores * 2`

16.12.6.2 Improvements

- TiDB
  - Reorder inner joins that are simplified from outer joins #22402
  - Support multiple clusters in Grafana dashboards #22534
  - Add a workaround for the issue of multiple statements #22468
  - Divide the metrics of slow query into `internal` and `general` #22405
  - Add interface for `utf8_unicode_ci` and `utf8mb4_unicode_ci` collations #22099
- TiKV
  - Add metrics of server information for DBaaS #9591
  - Support multiple clusters in Grafana dashboards #9572
  - Report RocksDB metrics to TiDB #9316
  - Record the suspension time for Coprocessor tasks #9277
  - Add thresholds of key counts and key size for Load Base Split #9354
  - Check whether the file exists before data import #9544
  - Improve Fast Tune panels #9180
- PD
  - Support multiple clusters in Grafana dashboards #3398
- TiFlash
  - Optimize the performance of the `date_format` function
  - Optimize the memory consumption of handling ingest SST
  - Optimize the retrying logic in Batch Coprocessor to reduce the probability of Region error
- Tools
  - TiCDC
    - Add the version information in the `capture` metadata and add the CLI version of a `changefeed` in the `changefeed` metadata #1342
  - TiDB Lightning
    - Create tables in parallel to improve import performance #502
    - Skip splitting Regions to improve import performance if the engine's total size is smaller than the Region size #524
    - Add a import progress bar and optimize the accuracy of restore progress #506

16.12.6.3 Bug Fixes

- TiDB
    - Fix the issue of abnormal `unicode_ci` constant propagation #22614
    - Fix the issue that might cause wrong collation and coercibility #22602
    - Fix the issue that might cause wrong collation results #22599
    - Fix the issue of constant substitution for different collations #22582
    - Fix a bug that the `like` function might return wrong result when using collation #22531
    - Fix the issue of incorrect `duration` type inference in `least` and `greatest` functions #22580
    - Fix a bug that occurs when the `like` function handles a single character wildcard (`_`) followed by a multiple character wildcard (`%`) #22575
    - Fix the type inference error of the TiDB's built-in functions (`least` and `greatest`) #22562
    - Fix a bug that makes the `like` function get the wrong result if the pattern string is a unicode string #22529
    - Fix a bug that the point get query does not get the snapshot data when the `@@tidb_snapshot` variable is set #22527
    - Fix the potential panic that occurs when generating hints from joins #22518
    - Fix the issue that strings are incorrectly converted to the `BIT` type #22420
    - Fix the `index out of range` error that occurs when inserting values to the `tidb_rowid` column #22359
    - Fix a bug that the cached plan is incorrectly used #22353
    - Fix the runtime panic in the `WEIGHT_STRING` function when the length of the binary/char string is too large #22332
    - Forbid using the generated column when the number of function parameters is invalid #22174
    - Correctly set the process information before building the execution plan #22148
    - Fix the issue of inaccurate runtime statistics of `IndexLookUp` #22136
    - Add cache for the memory usage information when the cluster is deployed in a container #22116
    - Fix the issue of the decoding plan errors #22022
    - Report errors for using invalid window specifications #21976
    - Report errors when the `PREPARE` statement is nested with `EXECUTE`, `DEALLOCATE` or `PREPARE` #21972
    - Fix the issue that no error is reported when the `INSERT IGNORE` statement is used on a non-existing partition #21971
    - Unify the encoding of `EXPLAIN` results and slow log #21964
    - Fix the issue of unknown columns in join when using the aggregate operator #21957
    - Fix the wrong type inference in the `ceiling` function #21936
    - Fix the issue that the `Double` type column ignores its decimal #21916
    - Fix the issue that the correlated aggregation is calculated in subqueries #21877

- Report errors for the JSON object with key length >= 65536 #21870
- Fix the issue that the `dyname` function is incompatible with MySQL #21850
- Fix the issue that the `to_base64` function returns `NULL` when the input data is too long #21813
- Fix the failure of comparing multiple fields in the subquery #21808
- Fix the issue that occurs when comparing the float type in JSON #21785
- Fix the issue that occurs when comparing the types of JSON objects #21718
- Fix the issue that the coercibility value of the `cast` function is incorrectly set #21714
- Fix an unexpected panic when using the `IF` function #21711
- Fix the issue that the `NULL` result returned from JSON search is incompatible with MySQL #21700
- Fix the issue that occurs when checking the `only_full_group_by` mode using `ORDER BY` and `HAVING` #21697
- Fix the issue that the units of `Day` and `Time` are incompatible with MySQL #21676
- Fix the issue that the default values of `LEAD` and `LAG` cannot adapt to the field type #21665
- Perform a check to ensure that the `LOAD DATA` statement can only load data into base tables #21638
- Fix the issue that occurs when `addtime` and `subtime` functions handle invalid arguments #21635
- Change the round rule for approximate values to "round to the nearest even number" #21628
- Fix the issue that `WEEK()` does not recognize `@@GLOBAL.default_week_format` until it has been explicitly read #21623

- TiKV

    - Fix the issue that TiKV is failed to build with `PROST=1` #9604
    - Fix the unmatched memory diagnostics #9589
    - Fix the issue that the end key of a partial RawKV-restore range is inclusive #9583
    - Fix the issue of TiKV panic that occurs when loading the old value of a key of a rolled-back transaction during TiCDC's incremental scan #9569
    - Fix the configuration glitch of old values when changefeeds with different settings connect to one Region #9565
    - Fix a crash issue that occurs when running a TiKV cluster on a machine with a network interface that lacks the MAC address (introduced in v4.0.9) #9516
    - Fix the issue of TiKV OOM when backing up a huge Region #9448
    - Fix the issue that `region-split-check-diff` cannot be customized #9530
    - Fix the issue of TiKV panic when the system time goes back #9542

- PD

    - Fix the issue that member health metrics are incorrectly displayed #3368
    - Forbid removing the tombstone store that still has peers #3352

- Fix the issue that the store limit cannot be persisted #3403
- Fix the limit constriction of the scatter range scheduler #3401

- TiFlash

  - Fix a bug that the `min/max` result is wrong for the decimal type
  - Fix a bug that TiFlash might crash when reading data
  - Fix the issue that some data written after DDL operations might be lost after data compaction
  - Fix the issue that TiFlash incorrectly handles decimal constants in Coprocessor
  - Fix the potential crash during the learner read process
  - Fix the inconsistent behaviors of division by `0` or `NULL` between TiDB and TiFlash

- Tools

  - TiCDC

    - Fix a bug that the TiCDC service might unexpectedly exit when `ErrTaskStatusNotExists` and the closing of `capture` session occur at the same time #1240
    - Fix the old value switch issue that a `changefeed` might be affected by another `changefeed` #1347
    - Fix a bug that the TiCDC service might hang when processing a new `changefeed` with the invalid `sort-engine` parameter #1309
    - Fix the issue of panic that occurs when getting the debugging information on non-owner nodes #1349
    - Fix the issue that the `ticdc_processor_num_of_tables` and `ticdc_processor_table_res`
      ↪ metrics are not properly updated when adding or removing tables #1351
    - Fix the issue of potential data loss if a processor crashes when adding a table #1363
    - Fix a bug that the owner might lead to abnormal TiCDC server exits during table migrations #1352
    - Fix a bug that TiCDC does not exit in time after the service GC safepoint is lost #1367
    - Fix a bug that the KV client might skip creating the event feed #1336
    - Fix a bug that the atomicity of transactions is broken when the transactions are replicated to the downstream #1375

  - Backup & Restore (BR)

    - Fix the issue that TiKV might be caused to generate a big Region after BR restores the backup #702
    - Fix the issue that BR restores a table's Auto ID even if the table does not have Auto ID #720

  - TiDB Lightning

    - Fix a bug that `column count mismatch` might be triggered when using the TiDB-backend #535

- Fix a bug that TiDB-backend panics if the column count of the source file and the column count of the target table mismatch #528
- Fix a bug that TiKV might unexpectedly panic during TiDB Lightning's data import #554

16.12.7 TiDB 4.0.10 Release Notes

Release date: January 15, 2021

TiDB version: 4.0.10

16.12.7.1 New Features

- PD
  - Add the `enable-redact-log` configuration item to redact user data from logs #3266
- TiFlash
  - Add the `security.redact_info_log` configuration item to redact user data from logs

16.12.7.2 Improvements

- TiDB
  - Make the size limit of a key-value entry in transaction configurable using `txn-entry-size-limit` #21843
- PD
  - Optimize the `store-state-filter` metrics to show more information #3100
  - Upgrade the `go.etcd.io/bbolt` dependency to v1.3.5 #3331
- Tools
  - TiCDC
    - Enable the unified sorter feature by default #1230
  - Dumpling
    - Support checking unrecognized arguments and printing the current progress during dumping #228
  - TiDB Lightning
    - Support retrying the error that occurs when reading from S3 #533

16.12.7.3 Bug Fixes

- TiDB
    - Fix a concurrency bug that might cause the batch client timeout #22336
    - Fix the issue of duplicate bindings caused by concurrent baseline capture #22295
    - Make the baseline capture bound to the SQL statement work when the log level is `'debug'` #22293
    - Correctly release GC locks when Region merge occurs #22267
    - Return correct values for user variables of the `datetime` type #22143
    - Fix the issue of using index merge when there are multiple table filters #22124
    - Fix the `wrong precision` issue in TiFlash caused by the `prepare` plan cache #21960
    - Fix the issue of incorrect results caused by schema change #21596
    - Avoid unnecessary column flag changes in `ALTER TABLE` #21474
    - Set the database name for table aliases of query blocks used in optimizer hints #21380
    - Generate the proper optimizer hint for `IndexHashJoin` and `IndexMergeJoin` #21020
- TiKV
    - Fix the wrong mapping between ready and peer #9409
    - Fix the issue that some logs are not redacted when `security.redact-info-log` is set to `true` #9314
- PD
    - Fix the issue that the ID allocation is not monotonic #3308 #3323
    - Fix the issue that the PD client might be blocked in some cases #3285
- TiFlash
    - Fix the issue that TiFlash fails to start because TiFlash fails to process the TiDB schema of an old version
    - Fix the issue that TiFlash fails to start due to incorrect handling of `cpu_time` on the RedHat system
    - Fix the issue that TiFlash fails to start when `path_realtime_mode` is set to `true`
    - Fix an issue of incorrect results when calling the `substr` function with three parameters
    - Fix the issue that TiFlash does not support changing the `Enum` type even if the change is lossless
- Tools
    - TiCDC

- Fix a bug that outdated metadata might cause the newly created changefeed abnormal #1184
- Fix the issue of creating the receiver on the closed notifier #1199
- Fix a bug that the TiCDC owner might consume too much memory in the etcd watch client #1227
- Fix the issue that `max-batch-size` does not take effect #1253
- Fix the issue of cleaning up stale tasks before the capture information is constructed #1280
- Fix the issue that the recycling of db conn is block because `rollback` is not called in MySQL sink #1285

- Dumpling
  - Avoid TiDB out of memory (OOM) by setting the default behavior of `tidb_mem_quota_query` #233
- Backup & Restore (BR)
  - Fix the issue that BR v4.0.9 cannot restore the files backed up using BR v4.0.8 on GCS #688
  - Fix the issue that BR panics when the GCS storage URL has no prefix #673
  - Disable backup statistics by default to avoid BR OOM #693
- TiDB Binlog
  - Fix the issue that when the `AMEND TRANSACTION` feature is enabled, Drainer might choose the incorrect schema version to generate SQL statements #1033
- TiDB Lightning
  - Fix a bug that the Region is not split because the Region key is incorrectly encoded #531
  - Fix the issue that the failure of `CREATE TABLE` might be lost when multiple tables are created #530
  - Fix the issue of `column count mismatch` when using the TiDB-backend #535

16.12.8 TiDB 4.0.9 Release Notes

Release date: December 21, 2020

TiDB version: 4.0.9

16.12.8.1 Compatibility Changes

- TiDB
  - Deprecate the `enable-streaming` configuration item #21055
- TiKV

- Reduce I/O and mutex contention when encryption at rest is enabled. The change is backwardly incompatible. If users need to downgrade the cluster to a version earlier than v4.0.9, `security.encryption.enable-file-dictionary-log` must be disabled and TiKV must be restarted before the downgrade. #9195

16.12.8.2 New Features

- TiFlash
  - Support storing the latest data of the storage engine on multiple disks (experimental)
- TiDB Dashboard
  - Support displaying and sorting by all fields in the **SQL Statements** page #749
  - Support zooming and panning the topology graph #772
  - Support displaying the disk usage information in the **SQL Statements** and **Slow Queries** pages #777
  - Support exporting list data in the **SQL Statements** and **Slow Queries** pages #778
  - Support customizing the Prometheus address #808
  - Add a page for cluster statistics #815
  - Add more time-related fields in the **Slow Queries** details #810

16.12.8.3 Improvements

- TiDB
  - Avoid the (index) merge join in a heuristical way when converting equal conditions to other conditions #21146
  - Differentiate the types of user variables #21107
  - Support setting the `GOGC` variable in the configuration file #20922
  - Make the dumped binary time (`Timestamp` and `Datetime`) more compatible with MySQL #21135
  - Provide an error message for statements that use the `LOCK IN SHARE MODE` syntax #21005
  - Avoid outputting unnecessary warnings or errors when folding constants in shortcut-able expressions #21040
  - Raise an error when preparing the `LOAD DATA` statement #21199
  - Ignore the attribute of the integer zero-fill size when changing the integer column types #20986
  - Add the executor-related runtime information of DML statements in the result of `EXPLAIN ANALYZE` #21066
  - Disallow multiple updates on the primary key in a singe SQL statements #21113

    - Add a monitoring metric for the connection idle time #21301
    - Temporarily enable the slow log when the `runtime/trace` tool is running #20578

- TiKV

    - Add the tag to trace the source of the `split` command #8936
    - Support dynamically changing the `pessimistic-txn.pipelined` configuration #9100
    - Reduce the impact on performance when running Backup & Restore and TiDB Lightning #9098
    - Add monitoring metrics for the ingesting SST errors #9096
    - Prevent the leader from being hibernated when some peers still need to replicate logs #9093
    - Increase the success rate of the pipelined pessimistic locking #9086
    - Change the default value of `apply-max-batch-size` and `store-max-batch-size` to `1024` #9020
    - Add the `max-background-flushes` configuration item #8947
    - Disable `force-consistency-checks` by default to improve performance #9029
    - Offload the queries on the Region size from `pd heartbeat worker` to `split check worker` #9185

- PD

    - Check the TiKV cluster version when a TiKV stores become `Tombstone`, which prevents users from enabling incompatible features during the process of downgrade or upgrade #3213
    - Disallow the TiKV store of a lower version to change from `Tombstone` back to `Up` #3206

- TiDB Dashboard

    - Keep expanding when "Expand" is clicked for SQL statements #775
    - Open detail pages in new windows for **SQL Statements** and **Slow Queries** #816
    - Improve descriptions for time-related fields in **Slow Queries** details #817
    - Display detailed error messages #794

- TiFlash

    - Reduce the latency of replica reads
    - Refine TiFlash's error messages
    - Limit the memory usage of cache data when the data volume is huge
    - Add a monitoring metric for the number of coprocessor tasks being handled

- Tools

- Backup & Restore (BR)
  - Disallow the ambiguous `--checksum false` argument in the command line, which does not correctly disable checksum. Only `--checksum=false` is accepted. #588
  - Support changing the PD configuration temporarily so that PD can recover the original configuration after BR accidentally exists #596
  - Support analyzing tables after restore #622
  - Retry for the `read index not ready` and `proposal in merging mode` errors #626
- TiCDC
  - Add an alert for enabling TiKV's Hibernate Region feature #1120
  - Reduce memory usage in the schema storage #1127
  - Add the feature of unified sorter, which accelerates replication when the data size of the incremental scan is large (experimental) #1122
  - Support configuring the maximum message size and the maximum message batch in the TiCDC Open Protocol message (only for Kafka sink) #1079
- Dumpling
  - Retry dumping data on failed chunks #182
  - Support configuring both the `-F` and `-r` arguments at the same time #177
  - Exclude system databases in `--filter` by default #194
  - Support the `--transactional-consistency` parameter and support rebuilding MySQL connections during retry #199
  - Support using the `-c,--compress` parameter to specify the compression algorithm used by Dumpling. An empty string means no compression. #202
- TiDB Lightning
  - Filter out all system schemas by default #459
  - Support setting a default value for the auto-random primary key for the Local-backend or Importer-backend #457
  - Use range properties to make the range split more precise in Local-backend #422
  - Support a human-readable format (such as "2.5 GiB") in `tikv-importer` ↪ `.region-split-size`, `mydumper.read-block-size`, `mydumper.batch-` ↪ `size`, and `mydumper.max-region-size` #471
- TiDB Binlog
  - Exit the Drainer process with the non-zero code if the upstream PD is down or if applying DDL or DML statements to the downstream fails #1012

16.12.8.4 Bug Fixes

- TiDB

- Fix the issue of incorrect results when using a prefix index with the `OR` condition #21287
- Fix a bug that might cause panic when automatic retry is enabled #21285
- Fix a bug that occurs when checking partition definition according to column type #21273
- Fix a bug that the value type of the partition expression is not consistent with the partition column type #21136
- Fix a bug that the hash-type partition does not check whether the partition name is unique #21257
- Fix the wrong results returned after inserting a value of the non-`INT` type into the hash partitioned table #21238
- Fix the unexpected error when using index join in the `INSERT` statement in some cases #21249
- Fix the issue that the `BigInt` unsigned column value in the `CASE WHEN` operator is incorrectly converted to the `BigInt` signed value #21236
- Fix a bug that index hash join and index merge join do not consider collation #21219
- Fix a bug that the partitioned table does not consider collation in the `CREATE TABLE` and `SELECT` syntax #21181
- Fix the issue that the query result of `slow_query` might miss some rows #21211
- Fix the issue that `DELETE` might not delete data correctly when the database name is not in a pure lower representation #21206
- Fix a bug that causes schema change after DML operations #21050
- Fix the bug that the coalesced column cannot be queried when using join #21021
- Fix the wrong results of some semi-join queries #21019
- Fix the issue that the table lock does not take effect on the `UPDATE` statement #21002
- Fix the issue of stack overflow that occurs when building the recursive view #21001
- Fix the unexpected result returned when performing index merge join operations on outer join #20954
- Fix the issue that sometimes a transaction that has an undetermined result might be treated as failed #20925
- Fix the issue that `EXPLAIN FOR CONNECTION` cannot show the last query plan #21315
- Fix the issue that when Index Merge is used in a transaction with the Read Committed isolation level, the result might be incorrect #21253
- Fix the auto-ID allocation failure caused by the transaction retry after the write conflict #21079
- Fix the issue that JSON data cannot be correctly imported to TiDB using `LOAD DATA` #21074
- Fix the issue that the default value of newly added `Enum`-type columns is incorrect #20998
- Fix the issue that the `adddate` function inserts invalid characters #21176
- Fix the issue that the wrong `PointGet` plan generated in some situations causes

wrong results #21244
- Ignore the conversion of daylight saving time in the `ADD_DATE` function to be compatible with MySQL #20888
- Fix a bug that prevents inserting strings with trailing spaces that exceed `varchar` or `char`'s length constraint #21282
- Fix a bug that does not converting the integer from `[1, 69]` to `[2001, 2069]` or from `[70, 99]` to `[1970, 1999]` when comparing `int` with `year` #21283
- Fix the panic caused by the overflowing result of the `sum()` function when calculating the `Double` type field #21272
- Fix a bug that `DELETE` fails to add lock on the unique key #20705
- Fix a bug that snapshot reads hits the lock cache #21539
- Fix an issue of potential memory leak after reading a lot of data in a long-lived transaction #21129
- Fix the issue that omitting the table alias in a subquery will have a syntax error returned #20367
- Fix the issue that when the argument of the `IN` function in a query is the time type, the query might return an incorrect result #21290

- TiKV

    - Fix the issue that Coprocessor might return wrong results when there are more than 255 columns #9131
    - Fix the issue that Region Merge might cause data loss during network partition #9108
    - Fix the issue that the `ANALYZE` statement might cause panic when using the `latin1` character set #9082
    - Fix the wrong results returned when converting the numeric type to the time type #9031
    - Fix a bug that TiDB Lightning fails to ingest SST files to TiKV with the Importer-backend or Local-backend when Transparent Data Encryption (TDE) is enabled #8995
    - Fix the invalid `advertise-status-addr` value (`0.0.0.0`) #9036
    - Fix the issue that an error is returned indicating that a key exists when this key is locked and deleted in a committed transaction #8930
    - Fix the issue that the RocksDB cache mapping error causes data corruption #9029
    - Fix a bug that Follower Read might return stale data after the leader is transferred #9240
    - Fix the issue that stale old values might be read in the pessimistic lock #9282
    - Fix a bug that replica read might get stale data after the leader transfer #9240
    - Fix the issue of TiKV crash that occurs when receiving `SIGPROF` after profiling #9229

- PD

    - Fix the issue that the leader roles specified using placement rules do not take effect in some cases #3208

- Fix the issue that the `trace-region-flow` value is unexpectedly set to `false` #3120
- Fix a bug that the service safepoint with infinite Time To Live (TTL) does not work #3143

- TiDB Dashboard

  - Fix a display issue of time in the Chinese language #755
  - Fix a bug that the browser compatibility notice does not work #776
  - Fix the issue that the transaction `start_ts` is incorrectly displayed in some scenarios #793
  - Fix the issue that some SQL texts are incorrectly formatted #805

- TiFlash

  - Fix the issue that `INFORMATION_SCHEMA.CLUSTER_HARDWARE` might contain the information of disks that are not in use
  - Fix the issue that the estimate on memory usage of Delta Cache is smaller than the actual usage
  - Fix the memory leak caused by thread information statistics

- Tools

  - Backup & Restore (BR)
    - Fix the failure caused by special characters in S3 secret access keys #617
  - TiCDC
    - Fix the issue that multiple owners might exist when the owner campaign key is deleted #1104
    - Fix a bug that TiCDC might fail to continue replicating data when a TiKV node crashes or recovers from a crash. This bug only exists in v4.0.8. #1198
    - Fix the issue that the metadata is repeatedly flushed to etcd before a table is initialized #1191
    - Fix an issue of replication interruption caused by early GC or the latency of updating `TableInfo` when the schema storage caches TiDB tables #1114
    - Fix the issue that the schema storage costs too much memory when DDL operations are frequent #1127
    - Fix the goroutine leak when a changefeed is paused or stopped #1075
    - Increase the maximum retry timeout to 600 seconds in Kafka producer to prevent replication interruption caused by the service or network jitter in the downstream Kafka #1118
    - Fix a bug that the Kafka batch size does not take effect #1112
    - Fix a bug that some tables' row change might be lost when the network between TiCDC and PD has jitter and when there are paused changefeeds being resumed at the same time #1213

- Fix a bug that the TiCDC process might exit when the network between TiCDC and PD is not stable #1218
- Use a singleton PD client in TiCDC and fix a bug that TiCDC closes PD client by accident which causes replication block #1217
- Fix a bug that the TiCDC owner might consume too much memory in the etcd watch client #1224

- Dumpling
  - Fix the issue that Dumpling might get blocked when its connection to the MySQL database server is closed #190
- TiDB Lightning
  - Fix the issue that keys are encoded using the wrong field information #437
  - Fix the issue that GC life time TTL does not take effect #448
  - Fix the issue that causes panic when manually stops the running TiDB Lightning in the Local-backend mode #484

16.12.9 TiDB 4.0.8 Release Notes

Release date: October 30, 2020

TiDB version: 4.0.8

16.12.9.1 New Features

- TiDB
  - Support the new aggregate function `APPROX_PERCENTILE` #20197
- TiFlash
  - Support pushing down `CAST` functions
- Tools
  - TiCDC
    - Support snapshot-level consistent replication #932

16.12.9.2 Improvements

- TiDB
  - Prioritize low-selectivity indexes in the greedy search procedure of `Selectivity()` #20154
  - Record more RPC runtime information in Coprocessor runtime statistics #19264

- Speed up parsing the slow log to improve query performance #20556
- Wait for timeout execution plans during the plan binding stage to record more debug information when the SQL optimizer is verifying potential new plans #20530
- Add the execution retry time in the slow log and the slow query result #20495 #20494
- Add the `table_storage_stats` system table #20431
- Add the RPC runtime statistical information for the `INSERT/UPDATE/REPLACE` statement #20430
- Add the operator information in the result of `EXPLAIN FOR CONNECTION` #20384
- Adjust the TiDB error log to the `DEBUG` level for the client connection/disconnection activities #20321
- Add monitoring metrics for Coprocessor Cache #20293
- Add the runtime information of pessimistic lock keys #20199
- Add two extra sections of time consumption information in the runtime information and `trace` span #20187
- Add the runtime information of transaction commit in the slow log #20185
- Disable the index merge join #20599
- Add the ISO 8601 and timezone supports for temporal string literals #20670

- TiKV

  - Add the **Fast-Tune** panel page to assist performance diagnostics #8804
  - Add the `security.redact-info-log` configuration item, which redacts user data from logs #8746
  - Reformat the metafile of error codes #8877
  - Enable dynamically changing the `pessimistic-txn.pipelined` configuration #8853
  - Enable the memory profiling features by default #8801

- PD

  - Generate the metafile of errors #3090
  - Add the additional information for the operator #3009

- TiFlash

  - Add monitoring metrics of Raft logs
  - Add monitoring metrics of memory usage for `cop` tasks
  - Make the `min/max` index more accurate when data is deleted
  - Improve query performance in the case of a small data volume
  - Add the `errors.toml` file to support the standard error code

- Tools

  - Backup and Restore (BR)

    - Speed up the restore process by pipelining `split` and `ingest` #427

    - Support manually restoring PD schedulers #530
    - Use `pause` schedulers instead of `remove` schedulers #551
  - TiCDC
    - Print statistics in MySQL sink periodically #1023
  - Dumpling
    - Support dumpling data directly to S3 storages #155
    - Support dumping views #158
    - Support dumping the table that only contains generated columns #166
  - TiDB Lightning
    - Support multi-byte CSV delimiters and separators #406
    - Speed up the restore process by disabling some PD schedulers #408
    - Use the GC-TTL API for checksum GC safepoint in the v4.0 cluster to avoid the GC error #396

16.12.9.3 Bug Fixes

- TiDB
  - Fix the unexpected panic that occurs when using partitioned tables #20565
  - Fix the wrong result of outer join when filtering the outer side using index merge join #20427
  - Fix the issue that the `NULL` value is returned when converting data to the `BIT` type if the data is too long #20363
  - Fix the corrupted default value for the `BIT` type column #20340
  - Fix the overflow error that might occur when converting the `BIT` type to the `INT64` type #20312
  - Fix the possible wrong result of the propagate column optimization for the hybrid type column #20297
  - Fix the panic that might occur when storing outdated plans from the plan cache #20246
  - Fix the bug that the returned result is mistakenly truncated if `FROM_UNIXTIME` and `UNION ALL` are used together #20240
  - Fix the issue that wrong results might be returned when the `Enum` type value is converted to the `Float` type #20235
  - Fix the possible panic of `RegionStore.accessStore` #20210
  - Fix the wrong result returned when sorting the maximum unsigned integer in `BatchPointGet` #20205
  - Fix the bug that the coercibilities of `Enum` and `Set` are wrong #20364
  - Fix an issue of ambiguous `YEAR` conversion #20292
  - Fix the issue of wrong reported result that occurs when the **KV duration** panel contains `store0` #20260

- Fix the issue that the `Float` type data is mistakenly inserted regardless of the `out of range` error #20252
- Fix the bug that the generated column does not handle bad `NULL` values #20216
- Fix the inaccurate error information for the `YEAR` type data that is out of range #20170
- Fix the unexpected `invalid auto-id` error that might occur during the pessimistic transaction retry #20134
- Fix the issue that the constraint is not checked when using `ALTER TABLE` to change the `Enum`/`Set` type #20046
- Fix the wrong runtime information of `cop` tasks recorded when multiple operators are used for concurrency #19947
- Fix the issue that read-only system variables cannot be explicitly selected as the session variables #19944
- Fix the issue that the duplicate `ORDER BY` condition might cause sub-optimal execution plans #20333
- Fix the issue that the generated metric profile might fail if the font size exceeds the maximum allowable value #20637

- TiKV

  - Fix the bug that the mutex conflict in encryption causes pd-worker to process heartbeats slowly #8869
  - Fix the issue that the memory profile is mistakenly generated #8790
  - Fix the failure to back up databases on GCS when the storage class is specified #8763
  - Fix the bug that a learner cannot find a leader when the Region is restarted or newly split #8864

- PD

  - Fix a bug that Key Visualizer of TiDB Dashboard might cause PD panic in some cases #3096
  - Fix the bug that PD might panic if a PD store is down for more than 10 minutes #3069

- TiFlash

  - Fix the issue of wrong timestamp in the log message
  - Fix the issue that during the multi-disk TiFlash deployment, the wrong capacity causes the creation of TiFlash replicas to fail
  - Fix the bug that TiFlash might throw errors about broken data files after restart
  - Fix the issue that broken files might be left on disk after TiFlash crashes
  - Fix the bug that it might take a long time to wait for index during learner reads if the proxy cannot catch up with the latest Raft lease information
  - Fix the bug that the proxy writes too much Region state information to the key-value engine while replaying the outdated Raft log

- Tools
    - Backup and Restore (BR)
        - Fix the `send on closed channel` panic during restore #559
    - TiCDC
        - Fix the unexpected exit caused by the failure to update the GC safepoint #979
        - Fix the issue that the task status is unexpectedly flushed because of the incorrect mod revision cache #1017
    - TiDB Lightning
        - Fix the issue of wrong column information #420
        - Fix the infinity loop that occurs when retrying to get Region information in the local mode #418

16.12.10 TiDB 4.0.7 Release Notes

Release date: September 29, 2020

TiDB version: 4.0.7

16.12.10.1 New Features

- PD
    - Add the `GetAllMembers` function in the PD client to get PD member information #2980
- TiDB Dashboard
    - Support generating the metrics relationship graph #760

16.12.10.2 Improvements

- TiDB
    - Add more runtime information for the `join` operator #20093
    - Add the hit ratio information of coprocessor cache in `EXPLAIN ANALYZE` #19972
    - Support pushing down the `ROUND` function to TiFlash #19967
    - Add the default value of `CMSketch` for `ANALYZE` #19927
    - Refine error message desensitization #20004
    - Accept connections from clients using connectors from MySQL 8.0 #19959
- TiKV

    - Support the JSON log format #8382

- PD

    - Count schedule operators when they are finished rather than added #2983
    - Set the `make-up-replica` operator to high priority #2977

- TiFlash

    - Improve the error handling of the Region meta change that occurs during reads

- Tools

    - TiCDC
        - Support translating more execution-efficient SQL statements in MySQL sink when the old value feature is enabled #955
    - Backup & Restore (BR)
        - Add connection retry when the connection is broken during backup #508
    - TiDB Lightning
        - Support dynamically updating the log level via the HTTP interface #393

16.12.10.3 Bug Fixes

- TiDB

    - Fix a vectorization bug from `and`/`or`/`COALESCE` caused by shortcut #20092
    - Fix the issue that plan digests are the same when the cop task stores are of different types #20076
    - Fix the wrong behavior of the `!= any()` function #20062
    - Fix the query error that occurs when the `slow-log` file does not exist #20051
    - Fix the issue that Region requests continue to retry when the context is canceled #20031
    - Fix the issue that querying the time type of the `cluster_slow_query` table in streaming request might result in an error #19943
    - Fix the issue that DML statements using `case when` might cause schema change #20095
    - Fix the issue that the `prev_stmt` information in slow log is not desensitized #20048
    - Fix the issue that tidb-server does not release the table lock when it exits abnormally #20020
    - Fix the incorrect error message that occurs when inserting data of the `ENUM` and `SET` type #19950
    - Fix the wrong behavior of the `IsTrue` function in some situations #19903

- Fix the issue that the `CLUSTER_INFO` system table might not work normally after PD is scaled in or out #20026
- Avoid unnecessary warnings or errors when folding constants in `control` expressions #19910
- Update the method of updating statistics to avoid Out of Memory (OOM) #20013

- TiKV
  - Fix the issue of unavailable Status API when TLS handshake fails #8649
  - Fix the potential undefined behaviors #7782
  - Fix the possible panic caused by generating snapshots when executing `UnsafeDestroyRange` #8681

- PD
  - Fix the bug that PD might panic if some Regions have no Leader when `balance-region` is enabled #2994
  - Fix the statistical deviation of Region size and Region keys after Region merge #2985
  - Fix the incorrect hotspot statistics #2991
  - Fix the issue that there is no `nil` check in `redirectSchedulerDelete` #2974

- TiFlash
  - Fix the wrong result of right outer join

- Tools
  - Backup & Restore (BR)
    - Fix a bug that causes the TiDB configuration to change after the restore process #509
  - Dumpling
    - Fix the issue that Dumpling fails to parse metadata when some variables are `NULL` #150

16.12.11 TiDB 4.0.6 Release Notes

Release date: September 15, 2020

TiDB version: 4.0.6

16.12.11.1 New Features

- TiFlash
  - Support outer join in TiFlash broadcast join
- TiDB Dashboard
  - Add Query Editor and execution UI (experimental) #713
  - Support store location topology visualization #719
  - Add cluster configuration UI (experimental) #733
  - Support sharing the current session #741
  - Support displaying the number of execution plans in SQL Statement list #746
- Tools
  - TiCDC (GA since v4.0.6)

16.12.11.2 Improvements

- TiDB
  - Replace error codes and messages with standard errors #19888
  - Improve the write performance of partitioned table #19649
  - Record more RPC runtime information in `Cop Runtime` statistics #19264
  - Forbid creating tables in `metrics_schema` and `performance_schema` #19792
  - Support adjusting the concurrency of the union executor #19886
  - Support out join in broadcast join #19664
  - Add SQL digest for the process list #19829
  - Switch to the pessimistic transaction mode for autocommit statement retry #19796
  - Support the `%r` and `%T` data format in `Str_to_date()` #19693
  - Enable `SELECT INTO OUTFILE` to require the file privilege #19577
  - Support the `stddev_pop` function #19541
  - Add the `TiDB-Runtime` dashboard #19396
  - Improve compatibility for the `ALTER TABLE` algorithms #19364
  - Encode `insert/delete/update` plans in the slow log `plan` field #19269
- TiKV
  - Reduce QPS drop when `DropTable` or `TruncateTable` is being executed #8627
  - Support generating metafile of error codes #8619
  - Add performance statistics for cf scan details #8618
  - Add the `rocksdb perf context` panel in the Grafana default template #8467

- PD
    - Update TiDB Dashboard to v2020.09.08.1 #2928
    - Add more metrics for Region and store heartbeat #2891
    - Change back to the original way to control the low space threshold #2875
    - Support standard error codes
        - #2918 #2911 #2913 #2915 #2912
        - #2907 #2906 #2903 #2806 #2900 #2902
- TiFlash
    - Add Grafana panels for data replication (`apply Region snapshots` and `ingest SST files`)
    - Add Grafana panels for `write stall`
    - Add `dt_segment_force_merge_delta_rows` and `dt_segment_force_merge_delta_deletes` to adjust the threshold of `write stall`
    - Support setting `raftstore.snap-handle-pool-size` to `0` in TiFlash-Proxy to disable applying Region snapshot by multi-thread to reduce memory consumption during data replication
    - Support CN check on `https_port` and `metrics_port`
- Tools
    - TiCDC
        - Skip resolved lock during puller initialization #910
        - Reduce PD write frequency #937
    - Backup & Restore (BR)
        - Add real time cost in summary log #486
    - Dumpling
        - Support outputting `INSERT` with column names #135
        - Unify the `--filesize` and `--statement-size` definitions with those of mydumper #142
    - TiDB Lightning
        - Split and ingest Regions in more precise sizes #369
    - TiDB Binlog
        - Support setting GC time in `go time` package format #996

16.12.11.3 Bug Fixes

- TiDB
    - Fix an issue of collecting the `tikv_cop_wait` time in metric profile #19881

- Fix the wrong result of `SHOW GRANTS` #19834
- Fix the incorrect query result of `!= ALL (subq)` #19831
- Fix a bug of converting the `enum` and `set` types #19778
- Add a privilege check for `SHOW STATS_META` and `SHOW STATS_BUCKET` #19760
- Fix the error of unmatched column lengths caused by `builtinGreatestStringSig` ↪ and `builtinLeastStringSig` #19758
- If unnecessary errors or warnings occur, the vectorized control expresions fall back to their scalar execution #19749
- Fix the error of the `Apply` operator when the type of the correlation column is `Bit` #19692
- Fix the issue that occurs when the user queries `processlist` and `cluster_log` in MySQL 8.0 client #19690
- Fix the issue that plans of the same type have different plan digests #19684
- Forbid changing the column type from `Decimal` to `Int` #19682
- Fix the issue that `SELECT ... INTO OUTFILE` returns the runtime error #19672
- Fix the incorrect implementation of `builtinRealIsFalseSig` #19670
- Fix the issue that the partition expression check misses the parentheses expression #19614
- Fix a query error when there is an `Apply` operator upon `HashJoin` #19611
- Fix an incorrect result of vectorization that casts `Real` as `Time` #19594
- Fix the bug that the `SHOW GRANTS` statement shows grants for non-existent users #19588
- Fix a query error when there is an `Apply` executor upon `IndexLookupJoin` #19566
- Fix the wrong results when converting `Apply` to `HashJoin` on a partitioned table #19546
- Fix incorrect results when there is an `IndexLookUp` executor on the inner side of an `Apply` #19508
- Fix an unexpected panic when using view #19491
- Fix the incorrect result of the `anti-semi-join` query #19477
- Fix the bug that the `TopN` statistics is not deleted when the statistics is dropped #19465
- Fix a wrong result caused by mistaken usage of batch point get #19460
- Fix the bug that a column cannot be found in `indexLookupJoin` with a virtual generated column #19439
- Fix an error that different plans of the `select` and `update` queries compare datum #19403
- Fix a data race for TiFlash work index in Region cache #19362
- Fix the bug that the `logarithm` function does not show a warning #19291
- Fix an unexpected error that occurs when TiDB persists data to disks #19272
- Support using a single partitioned table on the inner side of index join #19197
- Fix the wrong hash key value generated for decimal #19188
- Fix the issue that TiDB returns a `no regions` error when table endKey and Region endKey are the same #19895
- Fix the unexpected success of alter partition #19891
- Fix the wrong value of the default maximum packet length allowed for pushed

down expressions #19876
- Fix a wrong behavior for the `Max`/`Min` functions on the `ENUM`/`SET` columns #19869
- Fix the read failure from the `tiflash_segments` and `tiflash_tables` system tables when some TiFlash nodes are offline #19748
- Fix a wrong result of the `Count(col)` aggregation function #19628
- Fix a runtime error of the `TRUNCATE` operation #19445
- Fix the issue that `PREPARE statement FROM @Var` will fail when `Var` contains uppercase characters #19378
- Fix the bug that schema charset modification in an uppercase schema will cause panic #19302
- Fix the inconsistency of plans between `information_schema.statements_summary` ↪ and `explain`, when the information contains `tikv/tiflash` #19159
- Fix the error in tests that the file does not exist for `select into outfile` #19725
- Fix the issue that `INFORMATION_SCHEMA.CLUSTER_HARDWARE` does not have raid device information #19457
- Make the `add index` operation that has a generated column with the `case-when` expression can exit normally when it encounters a parse error #19395
- Fix the bug that the DDL operation takes too long to retry #19488
- Make statements like `alter table db.t1 add constraint fk foreign key (` ↪ `c2)references t2(c1)` execute without first executing `use db` #19471
- Change the dispatch error from the `Error` to the `Info` message in the server log file #19454

- TiKV

  - Fix the estimation error for a non-index column when collation is enabled #8620
  - Fix the issue that Green GC might miss locks during the process of Region transfer #8460
  - Fix a panic issue that occurs when TiKV runs very slowly during Raft membership change #8497
  - Fix the deadlock issue that occurs between the PD client thread and other threads when calling PD sync requests #8612
  - Upgrade jemalloc to v5.2.1 to address the issue of memory allocation in huge page #8463
  - Fix the issue that the unified thread pool hangs for long-running queries #8427

- PD

  - Add the `initial-cluster-token` configuration to prevent different clusters from communicating with each other during bootstrap #2922
  - Fix the unit of store limit rate when the mode is `auto` #2826
  - Fix the issue that some schedulers persist configuration without solving errors #2818
  - Fix the empty HTTP response in scheduler #2871 #2874

- TiFlash

- Fix the issue that after renaming the primary key column in previous versions, TiFlash might not start after upgrading to v4.0.4/v4.0.5
- Fix the exceptions that occur after modifying the column's `nullable` attribute
- Fix the crash caused by computing a table's replication status
- Fix the issue that TiFlash is not available for data reads after users applied unsupported DDL operations
- Fix the exceptions caused by unsupported collations which are treated as `utf8mb4_bin`
- Fix the issue that the QPS panel for the TiFlash coprocessor executor always displays `0` in Grafana
- Fix the wrong result of the `FROM_UNIXTIME` function when input is `NULL`

- Tools

  - TiCDC

    - Fix the issue that TiCDC leaks memory in some cases #942
    - Fix the issue that TiCDC might panic in Kafka sink #912
    - Fix the issue that CommitTs or ResolvedTs (CRTs) might be less than `resolvedTs` in puller #927
    - Fix the issue that `changefeed` might be blocked by MySQL driver #936
    - Fix the incorrect Resolved Ts interval of TiCDC #8573

  - Backup & Restore (BR)

    - Fix a panic that might occur during checksum #479
    - Fix a panic that might occur after the change of PD Leader #496

  - Dumpling

    - Fix the issue that the `NULL` value for the binary type is not handled properly #137

  - TiDB Lightning

    - Fix the issue that all failed operations of writes and ingests are mistakenly displayed as successful #381
    - Fix the issue that some checkpoint updates might not be written to the database before TiDB Lightning exits #386

16.12.12 TiDB 4.0.5 Release Notes

Release date: August 31, 2020

TiDB version: 4.0.5

16.12.12.1 Compatibility Changes

- TiDB

- Change `drop partition` and `truncate partition`'s job arguments to support the ID array of multiple partitions #18930
- Add the delete-only state for checking `add partition` replicas #18865

16.12.12.2 New Features

- TiKV
  - Define error code for errors #8387
- TiFlash
  - Support the unified log format with TiDB
- Tools
  - TiCDC
    - Support Kafka SSL connection #764
    - Support outputting the old value #708
    - Add the column flags #796
    - Support outputting the DDL statements and table schema of the previous version #799

16.12.12.3 Improvements

- TiDB
  - Optimize the performance of `DecodePlan` for big union queries #18941
  - Reduce the number of GC lock scans when the `Region cache miss` error occurs #18876
  - Ease the impact of statistical feedback on cluster performance #18772
  - Support canceling operations before the RPC response is returned #18580
  - Add the HTTP API to generate the TiDB metric profile #18531
  - Support scattering partitioned tables #17863
  - Add detailed memory usage of each instance in Grafana #18679
  - Show the detailed runtime information of the `BatchPointGet` operator in the result of `EXPLAIN` #18892
  - Show the detailed runtime information of the `PointGet` operator in the result of `EXPLAIN` #18817
  - Warn the potential deadlock for `Consume` in `remove()` #18395
  - Refine the behaviors of `StrToInt` and `StrToFloat` and support converting JSON to the `date`, `time`, and `timestamp` types #18159
  - Support limiting the memory usage of the `TableReader` operator #18392
  - Avoid too many times of backoff when retrying the `batch cop` request #18999

    - Improve compatibility for `ALTER TABLE` algorithms #19270
    - Make the single partitioned table support `IndexJoin` on the inner side #19151
    - Support searching the log file even when the log includes invalid lines #18579

- PD

    - Support scattering Regions in stores with special engines (such as TiFlash) #2706
    - Support the Region HTTP API to prioritize Region scheduling of a given key range #2687
    - Improve the leader distribution after Region scattering #2684
    - Add more tests and logs for the TSO request #2678
    - Avoid invalid cache updates after the leader of a Region has changed #2672
    - Add an option to allow `store.GetLimit` to return the tombstone stores #2743
    - Support synchronizing the Region leader change between the PD leader and followers #2795
    - Add commands for querying the GC safepoint service #2797
    - Replace the `region.Clone` call in filters to improve performance #2801
    - Add an option to disable updating Region flow cache to improve the performance of the large cluster #2848

- TiFlash

    - Add more Grafana panels to display metrics of CPU, I/O, RAM usages and metrics of the storage engine
    - Reduce I/O operations by optimizing the processing logic of Raft logs
    - Accelerate Region scheduling for the blocked `add partition` DDL statement
    - Optimize compactions of delta data in DeltaTree to reduce read and write amplification
    - Optimize the performance of applying Region snapshots by preprocessing the snapshots using multiple threads
    - Optimize the number of opening file descriptors when the read load of TiFlash is low to reduce system resource consumption
    - Optimize the number of unnecessary small files created when TiFlash restarts
    - Support encryption at rest for data storage
    - Support TLS for data transfer

- Tools

    - TiCDC
        - Lower the frequency of getting TSO #801
    - Backup & Restore (BR)
        - Optimize some logs #428
    - Dumpling

    - Release FTWRL after connections are created to reduce the lock time for MySQL #121
- TiDB Lightning
    - Optimize some logs #352

16.12.12.4 Bug Fixes

- TiDB
    - Fix the `should ensure all columns have the same length` error that occurs because the `ErrTruncate/Overflow` error is incorrectly handled in the `builtinCastRealAsDecimalSig` function #18967
    - Fix the issue that the `pre_split_regions` table option does not work in the partitioned table #18837
    - Fix the issue that might cause a large transaction to be terminated prematurely #18813
    - Fix the issue that using the `collation` functions get wrong query results #18735
    - Fix the bug that the `getAutoIncrementID()` function does not consider the `tidb_snapshot` session variable, which might cause the dumper tool to fail with the `table not exist` error #18692
    - Fix the `unknown column error` for SQL statement like `select a from t having t.a` #18434
    - Fix the panic issue that writing the 64-bit unsigned type into the hash partitioned table causes overflow and gets an unexpected negative number when the partition key is the integer type #18186
    - Fix the wrong behavior of the `char` function #18122
    - Fix the issue that the `ADMIN REPAIR TABLE` statement cannot parse integer in the expressions on the range partition #17988
    - Fix the wrong behavior of the `SET CHARSET` statement #17289
    - Fix the bug caused by the wrong collation setting which leads to the wrong result of the `collation` function #17231
    - Fix the issue that `STR_TO_DATE`'s handling of the format tokens'%r', '%h' is inconsistent with that of MySQL #18727
    - Fix issues that the TiDB version information is inconsistent with that of PD/TiKV in the `cluster_info` table #18413
    - Fix the existent checks for pessimistic transactions #19004
    - Fix the issue that executing `union select for update` might cause concurrent race #19006
    - Fix the wrong query result when `apply` has a child of the `PointGet` operator #19046
    - Fix the incorrect result that occurs when `IndexLookUp` is in the inner side of the `Apply` operator #19496
    - Fix the incorrect result of `anti-semi-join` queries #19472
    - Fix the incorrect result caused by the mistaken usage of `BatchPointGet` #19456

- Fix the incorrect result that occurs when `UnionScan` is in the inner side of the `Apply` operator #19496
- Fix the panic caused by using the `EXECUTE` statement to print an expensive query log #17419
- Fix the index join error when the join key is `ENUM` or `SET` #19235
- Fix the issue that the query range cannot be built when the `NULL` value exists on the index column #19358
- Fix the data race issue caused by updating the global configuration #17964
- Fix the panic issue occurs when modifying the character set in an uppercase schema #19286
- Fix an unexpected error caused by changing the temporary directory during the disk spill action #18970
- Fix the wrong hash key for the decimal type #19131
- Fix the issue that the `PointGet` and `BatchPointGet` operators do not consider the partition selection syntax and get incorrect results #19141
- Fix the incorrect results when using the `Apply` operator together with the `UnionScan` operator #19104
- Fix the bug that causes the indexed virtual generated column to return wrong value #17989
- Add the lock for runtime statistics to fix a panic caused by concurrent execution #18983

- TiKV

    - Speed up leader election when Hibernate Region is enabled #8292
    - Fix the memory leak issue during scheduling #8357
    - Add the `hibernate-timeout` configuration item to prevent the leader from becoming hibernate too fast #8208

- PD

    - Fix the bug that the TSO request might fail at the time of leader change #2666
    - Fix the issue that sometimes Region replicas cannot be scheduled to the optimal state when placement rules are enabled #2720
    - Fix the issue that `Balance Leader` does not work when placement rules are enabled #2726
    - Fix the issue that unhealthy stores are not filtered from store load statistics #2805

- TiFlash

    - Fix the issue that TiFlash cannot start normally after upgrading from an earlier version if the name of the database or table contains special characters
    - Fix the issue that the TiFlash process cannot exit if any exceptions are thrown during initialization

- Tools

- Backup & Restore (BR)
  - Fix the issue of duplicated calculation of total KV and total bytes in the backup summary log #472
  - Fix the issue that the import mode does not work in the first 5 minutes after switching to this mode #473
- Dumpling
  - Fix the issue that FTWRL lock is not released in time #128
- TiCDC
  - Fix the issue that the failed `changefeed` cannot be removed #782
  - Fix invalid `delete` events by selecting one unique index as the handle index #787
  - Fix the bug that GC safepoint is forwarded beyond the checkpoint of stopped `changefeed` #797
  - Fix the bug that the network I/O waiting blocks tasks to exit #825
  - Fix the bug that some unnecessary data might be mistakenly replicated to the downstream #743
- TiDB Lightning
  - Fix the syntax error on empty binary/hex literals when using TiDB backend #357

16.12.13 TiDB 4.0.4 Release Notes

Release date: July 31, 2020

TiDB version: 4.0.4

16.12.13.1 Bug Fixes

- TiDB
  - Fix the issue of getting stuck when querying `information_schema.columns` #18849
  - Fix the errors that occur when the `PointGet` and `BatchPointGet` operators encounter `in null` #18848
  - Fix the wrong result of `BatchPointGet` #18815
  - Fix the issue of incorrect query result that occurs when the `HashJoin` operator encounters the `set` or `enum` type #18859

16.12.14 TiDB 4.0.3 Release Notes

Release date: July 24, 2020

TiDB version: 4.0.3

16.12.14.1 New Features

- TiDB Dashboard
  - Display detailed TiDB Dashboard version information #679
  - Show browser compatibility notice for unsupported browsers or outdated browsers #654
  - Support searching in the **SQL Statements** page #658
- TiFlash
  - Implement file encryption in TiFlash proxy
- Tools
  - Backup & Restore (BR)
    - Support compressing backup files using zstd, lz4 or snappy #404
  - TiCDC
    - Support configuring `kafka-client-id` in MQ sink-uri #706
    - Support updating `changefeed` configuration offline #699
    - Support setting customized `changefeed` name #727
    - Support TLS and MySQL SSL connection #347
    - Support outputting changes in the Avro format #753
    - Support the Apache Pulsar sink #751
  - Dumpling
    - Support the specialized CSV separator and delimiter #116
    - Support specifying the format of the output file name #122

16.12.14.2 Improvements

- TiDB
  - Add the `tidb_log_desensitization` global variable to control whether to do desensitization when logging SQL queries #18581
  - Enable `tidb_allow_batch_cop` by default #18552
  - Speed up canceling a query #18505
  - Add a header for the `tidb_decode_plan` result #18501
  - Make the configuration checker compatible with earlier versions of the configuration file #18046
  - Enable collecting the execution information by default #18518
  - Add the `tiflash_tables` and `tiflash_segments` system tables #18536
  - Move `AUTO RANDOM` out of experimental features and announce its general availability. The improvements and compatibility changes are as follows:

    - Deprecate `experimental.allow-auto-random` in the configuration file. No matter how this item is configured, you can always define the `AUTO RANDOM` feature on columns. #18613 #18623
    - Add the `tidb_allow_auto_random_explicit_insert` session variable to control the explicit writes on `AUTO RANDOM` columns. The default value is `false`. This is to avoid the unexpected `AUTO_RANDOM_BASE` update caused by explicit writes on columns. #18508
    - Allow defining `AUTO_RANDOM` only on `BIGINT` and `UNSIGNED BIGINT` columns and restrict the maximum number of shard bits to `15`, which avoids the allocatable space being consumed too quickly #18538
    - Do not trigger the `AUTO_RANDOM_BASE` update when defining the `AUTO_RANDOM` ↪ attribute on the `BIGINT` column and inserting the negative value into the primary key #17987
    - Use the highest bit of an integer for ID allocation when defining the `AUTO_RANDOM` attribute on `UNSIGNED BIGINT` columns, which gets more allocable space #18404
    - Support updating the `AUTO_RANDOM` attribute in the result of `SHOW CREATE` ↪ `TABLE` #18316

- TiKV

    - Introduce the new `backup.num-threads` configuration to control the size of the backup thread pool #8199
    - Do not send store heartbeats when receiving snapshots #8136
    - Support dynamically changing the shared block cache's capacity #8232

- PD

    - Support the JSON formatted log #2565

- TiDB Dashboard

    - Improve the Key Visualizer bucket merge for cold logical ranges #674
    - Rename the configuration item of `disable-telemetry` to `enable-telemetry` for consistency #684
    - Show the progress bar when switching pages #661
    - Ensure that the slow log search now follows the same behavior as log search when there are space separators #682

- TiFlash

    - Change the unit of the **DDL Jobs** panel in Grafana to `operations per minute`
    - Add a new dashboard in Grafana to show more metrics about **TiFlash-Proxy**
    - Reduce IOPS in TiFlash proxy

- Tools

- TiCDC
  - Replace table ID with table name in metrics #695
- Backup & Restore (BR)
  - Support outputting JSON logs #336
  - Support enabling pprof during runtime #372
  - Speed up DDL executions by sending DDL concurrently during restore #377
- TiDB Lightning
  - Deprecate `black-white-list` with a newer and easier-to-understand filter format #332

16.12.14.3 Bug Fixes

- TiDB
  - Return an error instead of an empty set for `IndexHashJoin` when an error occurs during execution #18586
  - Fix the recurring panic when gRPC transportReader is broken #18562
  - Fix the issue that Green GC does not scan locks on offline stores which might cause data incompleteness #18550
  - Forbid processing a non-read-only statement using TiFlash engine #18534
  - Return the actual error message when a query connection panics #18500
  - Fix the issue that the `ADMIN REPAIR TABLE` execution fails to reload the table metadata on the TiDB node #18323
  - Fix the data inconsistency issue occurred because the lock of a written and deleted primary key in one transaction is resolved by another transaction #18291
  - Make spilling disk work well #18288
  - Fix the error reported when the `REPLACE INTO` statement works on the table that contains generated columns #17907
  - Return the OOM error when the `IndexHashJoin` and `IndexMergeJoin` workers panic #18527
  - Fix the bug that the execution of `Index Join` might return wrong results in special cases when the index used by `Index Join` contains the integer primary key #18565
  - Fix the issue that when the new collation is enabled on the cluster, the data updated on columns with the new collation in a transaction cannot be read through the unique index #18703
- TiKV
  - Fix the issue that reads might get stale data during merging #8113
  - Fix the issue that collation does not work on the `min`/`max` function when aggregation is pushed down to TiKV #8108

- PD
    - Fix the issue that creating TSO stream might be blocked for a while if the server crashes #2648
    - Fix the issue that `getSchedulers` might cause a data race #2638
    - Fix the issue that deleting the scheduler might cause deadlocks #2637
    - Fix the bug that placement rules are not considered when `balance-leader-scheduler` is enabled #2636
    - Fix the issue that sometimes service `safepoint` cannot be set properly, which might make BR and dumpling fail #2635
    - Fix the issue that the target store in `hot region scheduler` is incorrectly selected #2627
    - Fix the issue that the TSO request might take too long when PD leader is switched #2622
    - Fix the issue of stale scheduler after leader change #2608
    - Fix the issue that sometimes replicas of a Region cannot be adjusted to the best location when placement rules are enabled #2605
    - Fix the issue that the deployment path of the store is not updated according to the change of deployment directory #2600
    - Prevent `store limit` from changing to zero #2588
- TiDB Dashboard
    - Fix the TiDB connection error when TiDB is scaled out #689
    - Fix the issue that TiFlash instances are not displayed in the log searching page #680
    - Fix the issue of metric selection reset after refreshing the overview page #663
    - Fix a connection issue in some TLS scenarios #660
    - Fix the issue that the language dropdown box is not displayed correctly in some cases #677
- TiFlash
    - Fix the issue that TiFlash crashes after renaming the primary key column
    - Fix the issue that concurrent `Learner Read` and `Remove Region` might cause deadlocks
- Tools
    - TiCDC
        - Fix the issue that TiCDC leaks memory in some cases #704
        - Fix the issue that unquoted table name causes the SQL syntax error #676
        - Fix the issue that the processor does not fully exit after `p.stop` is called #693
    - Backup & Restore (BR)
        - Fix the issue that the backup time might be negative #405

- Dumpling
  - Fix the issue that Dumpling omits the `NULL` value when `--r` is specified #119
  - Fix the bug that flushing tables might not work for tables to dump #117
- TiDB Lightning
  - Fix the issue that `--log-file` does not take effect #345
- TiDB Binlog
  - Fix the issue that when TiDB Binlog replicates data to the downstream with TLS enabled, Drainer cannot be started which occurs because TLS is not enabled on the database driver used to update the checkpoint #988

16.12.15 TiDB 4.0.2 Release Notes

Release date: July 1, 2020

TiDB version: 4.0.2

16.12.15.1 Compatibility Changes

- TiDB
  - Remove sensitive information in the slow query log and the statement summary table #18130
  - Forbid negative value in the sequence cache #18103
  - Remove tombstone TiKV and TiFlash stores from the `CLUSTER_INFO` table #17953
  - Change the diagnostic rule from `current-load` to `node-check` #17660
- PD
  - Persist `store-limit` and remove `store-balance-rate` #2557

16.12.15.2 New Change

- By default, TiDB and TiDB Dashboard share usage details with PingCAP to help understand how to improve the product #18180. For details about what is shared and how to disable the sharing, see Telemetry.

16.12.15.3 New Features

- TiDB
    - Support the `MEMORY_QUOTA()` hint in `INSERT` statements #18101
    - Support authentication based on the `SAN` field of TLS certificate #17698
    - Support collation for the `REGEXP()` function #17581
    - Support the `sql_select_limit` session and global variable #17604
    - Support splitting the Region for the newly added partition by default #17665
    - Support pushing the `IF()`/`BITXOR()`/`BITNEG()`/`JSON_LENGTH()` functions to the TiFlash Coprocessor #17651 #17592
    - Support a new aggregate function `APPROX_COUNT_DISTINCT()` to calculate the approximate result of `COUNT(DISTINCT)` #18120
    - Support collation in TiFlash and pushing collation-related functions to TiFlash #17705
    - Add the `STATUS_ADDRESS` column in the `INFORMATION_SCHEMA.INSPECTION_RESULT` table to indicate the status address of servers #17695
    - Add the `SOURCE` column in the `MYSQL.BIND_INFO` table to indicate the how the bindings are created #17587
    - Add the `PLAN_IN_CACHE` and `PLAN_CACHE_HITS` columns in the `PERFORMANCE_SCHEMA.EVENTS_STATEMENTS_SUMMARY_BY_DIGEST` table to indicate the plan cache usage of SQL statements #17493
    - Add the `enable-collect-execution-info` configuration item and the `tidb_enable_collect_execution_info` session variable to control whether to collect execution information of each operator and record the information in the slow query log #18073 #18072
    - Add the `tidb_slow_log_masking` global variable to control whether to desensitize the queries in slow query log #17694
    - Add a diagnostic rule in the `INFORMATION_SCHEMA.INSPECTION_RESULT` table for the `storage.block-cache.capacity` TiKV configuration item #17671
    - Add the `BACKUP` and `RESTORE` SQL statements to back up and restore data #15274
- TiKV
    - Support the `encryption-meta` command in TiKV Control #8103
    - Add a perf context metric for `RocksDB::WriteImpl` #7991
- PD
    - Support the operator to fail immediately when trying to remove a leader peer #2551
    - Set a suitable default store limit for TiFlash stores #2559
- TiFlash
    - Support new aggregation function `APPROX_COUNT_DISTINCT` in Coprocessor

    - Enable the `rough set filter` feature by default
    - Enable TiFlash to run on the ARM architecture
    - Support pushing down the `JSON_LENGTH` function in Coprocessor

- Tools

    - TiCDC

        - Support migrating sub-tasks to new `captures` #665
        - Add a `cli` command to delete the TiCDC GC TTL #652
        - Support canal protocol in MQ sink #649

16.12.15.4 Improvements

- TiDB

    - Reduce the query latency caused by the Golang memory allocation when CM-Sketch consumes too much memory #17545
    - Reduce the QPS recovery duration of a cluster when a TiKV server is in the failure recovery process #17681
    - Support pushing aggregate functions to TiKV/TiFlash Coprocessor on partition tables #17655
    - Improve the accuracy of row count estimation for index equal conditions #17611

- TiKV

    - Improve the PD client panic log #8093
    - Add back the `process_cpu_seconds_total` and `process_start_time_seconds` monitoring metrics #8029

- TiFlash

    - Improve backward compatibility when upgrading from an older version #786
    - Reduce memory consumption of delta index #787
    - Use the more efficient update algorithm for delta index #794

- Tools

    - Backup & Restore (BR)

        - Improve the performance by pipelining the restore process #266

16.12.15.5 Bug Fixes

- TiDB

    - Fix the issue of incorrect execution plan obtained from the plan cache after `tidb_isolation_read_engines` is changed #17570
    - Fix the occasional runtime error that occurs when executing the `EXPLAIN FOR CONNECTION` statement #18124
    - Fix the incorrect result of the `last_plan_from_cache` session variable in some cases #18111
    - Fix the runtime error that occurs when executing the `UNIX_TIMESTAMP()` function from the plan cache #18002 #17673
    - Fix the runtime error when the child of `HashJoin` executor returns the `NULL` column #17937
    - Fix the runtime error caused by concurrently executing the `DROP DATABASE` statement and other DDL statements in the same database #17659
    - Fix the incorrect result of the `COERCIBILITY()` function on user variables #17890
    - Fix the issue that the `IndexMergeJoin` executor occasionally gets stuck #18091
    - Fix the hang issue of the `IndexMergeJoin` executor when out of memory quota and query cancelling is triggered #17654
    - Fix the excessive counting memory usage of the `Insert` and `Replace` executors #18062
    - Fix the issue that the data replication to TiFlash storage is stopped when `DROP DATABASE` and `DROP TABLE` are executed concurrently in the same database #17901
    - Fix the `BACKUP/RESTORE` failure between TiDB and the object storage service #17844
    - Fix the incorrect error message of privilege check failure when access is denied #17724
    - Discard the query feedbacks generated from the `DELETE/UPDATE` statement #17843
    - Forbid altering `AUTO_RANDOM_BASE` for a table without `AUTO_RANDOM` property #17828
    - Fix the issue that the `AUTO_RANDOM` column is allocated wrong results when the table is moved between databases by `ALTER TABLE ... RENAME` #18243
    - Fix the issue that some system tables cannot be accessed when setting the value of `tidb_isolation_read_engines` without `tidb` #17719
    - Fix the inaccurate result of JSON comparison on large integers and float values #17717
    - Fix the incorrect decimal property for the result of the `COUNT()` function #17704
    - Fix the incorrect result of the `HEX()` function when the type of input is the binary string #17620
    - Fix the issue that an empty result is returned when querying the `INFORMATION_SCHEMA.INSPECTION_SUMMARY` table without filter condition #17697

- Fix the issue that the hashed password used by the `ALTER USER` statement to update user information is unexpected #17646
- Support collation for `ENUM` and `SET` values #17701
- Fix the issue that the timeout mechanism for pre-splitting Regions does not work when creating a table #17619
- Fix the issue that the schema is unexpectedly updated when a DDL job is retried, which might break the atomicity of DDL jobs #17608
- Fix the incorrect result of the `FIELD()` function when the argument contains the column #17562
- Fix the issue that the `max_execution_time` hint does not work occasionally #17536
- Fix the issue that the concurrency information is redundantly printed in the result of `EXPLAIN ANALYZE` #17350
- Fix the incompatible behavior of `%h` on the `STR_TO_DATE` function #17498
- Fix the issue that the follower/learner keeps retrying when `tidb_replica_read` is set to `follower` and there is a network partition between the leader and the follower/learner #17443
- Fix the issue that TiDB sends too many pings to PD follower in some cases #17947
- Fix the issue that the range partition table of older versions cannot be loaded in TiDB v4.0 #17983
- Fix the SQL statement timeout issue when multiple Region requests fail at the same time by assigning different `Backoffer` for each Region #17585
- Fix the MySQL incompatible behavior when parsing `DateTime` delimiters #17501
- Fix the issue that TiKV requests are occasionally sent to the TiFlash server #18105
- Fix the data inconsistency issue occurred because the lock of a written and deleted primary key in one transaction is resolved by another transaction #18250

- TiKV

    - Fix a memory safety issue for the status server #8101
    - Fix the issue of lost precision in JSON numeric comparison #8087
    - Fix the wrong query slow log #8050
    - Fix the issue that a peer cannot be removed when its store is isolated during multiple merge processes #8048
    - Fix the issue that `tikv-ctl recover-mvcc` does not remove invalid pessimistic locks #8047
    - Fix the issue that some Titan histogram metrics are missing #7997
    - Fix the issue that TiKV returns `duplicated error` to TiCDC #7887

- PD

    - Check the correctness of the `pd-server.dashboard-address` configuration item #2517

- Fix the panic issue of PD when setting `store-limit-mode` to `auto` #2544
- Fix the issue that hotspots cannot be identified in some cases #2463
- Fix the issue that placement rules prevent the store from changing to `tombstone` in some cases #2546
- Fix the panic issue of PD when upgrading from earlier versions in some cases #2564

- TiFlash

  - Fix the issue that the proxy might panic when the `region not found` error occurs
  - Fix the issue that the I/O exception thrown in `drop table` might lead to synchronization failure of TiFlash schema

16.12.16 TiDB 4.0.1 Release Notes

Release date: June 12, 2020

TiDB version: 4.0.1

16.12.16.1 New Features

- TiKV

  - Add the `--advertise-status-addr` start flag to specify the status address to advertise #8046

- PD

  - Support the internal proxy for the built-in TiDB Dashboard #2511
  - Support setting a custom timeout for PD client #2509

- TiFlash

  - Support the TiDB new collation framework
  - Support pushing down the `If`/`BitAnd`/`BitOr`/`BitXor`/`BitNot`/`Json_length` functions to TiFlash
  - Support the Resolve Lock logic for large transactions in TiFlash

- Tools

  - Backup & Restore (BR)

    - Add a version check when starting BR to avoid the issue that BR and the TiDB cluster are incompatible #311

16.12.16.2 Bug Fixes

- TiKV
    - Fix the issue that the `use-unified-pool` configuration in the startup log is incorrectly printed #7946
    - Fix the issue that the tikv-ctl does not support relative path #7963
    - Fix the bug that the monitoring metric of Point Selects is inaccurate #8033
    - Fix the issue that a peer might not be destroyed after the network isolation disappears #8006
    - Fix the issue that a request for read index might get outdated commit index #8043
    - Improve the reliability of backup and restore with S3 and GCS storages #7917
- PD
    - Prevent misconfiguration of Placement Rules in some situations #2516
    - Fix the issue that deleting the Placement Rule might cause panic #2515
    - Fix a bug that the store information cannot be obtained when the store's used size is zero #2474
- TiFlash
    - Fix the issue that default value of the `bit` type column in TiFlash is incorrectly parsed
    - Fix the miscalculation of `1970-01-01 00:00:00 UTC` in some timezones in TiFlash

16.12.17 TiDB 4.0 GA Release Notes

Release date: May 28, 2020

TiDB version: 4.0.0

16.12.17.1 Compatibility Changes

- TiDB
    - Optimize the error message of the large-sized transactions for easier troubleshooting #17219
- TiCDC
    - Optimize the structure of the `Changefeed` configuration file to improve usability #588
    - Add the `ignore-txn-start-ts` configuration item, and change the condition from `commit_ts` to `start_ts` during transactions filtering #589

16.12.17.2 Important Bug Fixes

- TiKV
  - Fix the `DefaultNotFound` error that occurs when backing up using Backup & Restore (BR) #7937
  - Fix system panics caused by the out-of-order `ReadIndex` packages #7930
  - Fix system panics caused by incorrectly removing snapshot files after TiKV is restarted #7927
- TiFlash
  - Fix the possible data loss issue that occurs when the system panics because of incorrect processing logic of `Raft Admin Command`

16.12.17.3 New Features

- TiDB
  - Add the `committer-concurrency` configuration item to control the number of `goroutines` in the retry-commit phase #16849
  - Support the `show table partition regions` syntax #17294
  - Add the `tmp-storage-quota` configuration item to limit the temporary disk space used by the TiDB server #15700
  - Support checking whether the partitioned table uses a unique prefix index when creating and changing tables #17213
  - Support the `insert/replace into tbl_name partition(partition_name_list` ↪ `)` statement #17313
  - Support checking the value of `collations` when using the `Distinct` function #17240
  - Support the `is null` filter condition during the Hash partition pruning #17310
  - Support `admin check index`, `admin cleanup index`, and `admin recover` ↪ `index` in partitioned tables #17392 #17405 #17317
  - Support range partition pruning for the `in` expressions #17320
- TiFlash
  - Support filtering out the data corresponding to the qualified `TSO` through the `min commit ts` value of the `Lock CF` when the Learner reads the data
  - Add the feature that the system explicitly reports an error to avoid incorrect calculation results when the value of `TIMESTAMP` type is less than `1970-01-01` ↪ `00:00:00`
  - Support using flags in regular expressions when searching logs
- TiKV
  - Support collation rules of `ascii_bin` and `latin1_bin` encoding #7919

- PD
    - Support specifying the reverse proxy resource prefix for built-in TiDB Dashboard #2457
    - Support returning the `pending peer` and `down peer` information in interfaces of the PD client Region #2443
    - Add monitoring items such as `Direction of hotspot move leader`, `Direction of hotspot move peer`, and `Hot cache read entry number` #2448
- Tools
    - Backup & Restore (BR)
        - Support the backup and restore of `Sequence` and `View` #242
    - TiCDC
        - Support checking the validity of `Sink URI` when creating `Changefeed` #561
        - Support checking whether the PD and TiKV versions meet the system requirements during system startup #570
        - Support scheduling multiple tables in the same scheduling task generation cycle #572
        - Add information about node roles in HTTP API #591

16.12.17.4 Bug Fixes

- TiDB
    - Fix the issue of unexpected timeouts when sending and receiving messages by disabling TiDB to send batch commands to TiFlash #17307
    - Fix the issue of incorrectly distinguishing signed and unsigned integers during partition pruning, which improves performance #17230
    - Fix the issue of upgrade failure from v3.1.1 to v4.0 because of the incompatible `mysql.user` table #17300
    - Fix the issue of incorrect selection of the partition in the `update` statement #17305
    - Fix system panics when receiving an unknown error message from TiKV #17380
    - Fix system panics caused by incorrect processing logic when creating the table that is `key` partitioned #17242
    - Fix the issue that the wrong `Index Merge Join` plan is selected because of incorrect optimizer processing logic #17365
    - Fix the issue of inaccurate `duration` monitoring metric of the `SELECT` statement in Grafana #16561
    - Fix the issue that the GC worker is blocked when the system error occurs #16915
    - Fix the issue that the `UNIQUE` constraint on a boolean column results in an incorrect result in a comparison #17306
    - Fix system panics caused by incorrect processing logic when `tidb_opt_agg_push_down` is enabled and the aggregation function pushes down the partitioned table #17328

- Fix the issue of accessing failed TiKV nodes in some cases #17342
- Fix the issue that the `isolation-read` configuration item in `tidb.toml` does not take effect #17322
- Fix the issue of incorrect order of output results due to incorrect processing logic when `hint` is used to enforce the stream aggregation #17347
- Fix the behavior that `insert` processes DIV under different `SQL_MODE` #17314

- TiFlash

  - Fix the issue that the matching behavior of regular expressions in the search log feature is inconsistent with other components
  - Fix the issue of excessive restart time when nodes write large amounts of data by disabling the delay processing optimization of `Raft Compact Log Command` by default
  - Fix the issue that the system fails to start because TiDB incorrectly processes the `DROP DATABASE` statement in some scenarios
  - Fix the issue that the method of collecting CPU information in `Server_info` is different from that in other components
  - Fix the issue that the error `Too Many Pings` is reported when the `Query` statement is executed if `batch coprocessor` is enabled
  - Fix the issue that Dashboard fails to display the correct `deploy path` information because TiFlash does not report the related information

- TiKV

  - Fix the `DefaultNotFound` error that occurs when backing up using BR #7937
  - Fix system panics caused by out-of-order `ReadIndex` packets #7930
  - Fix the issue that an unexpected error is returned because the read request callback function is not called #7921
  - Fix system panics caused by incorrectly removing snapshot files when TiKV is restarted #7927
  - Fix the issue that the `master key` cannot be rotated due to incorrect processing logic in storage encryption #7898
  - Fix the issue that the received `lock cf` file of the snapshot is not encrypted when the storage encryption is enabled #7922

- PD

  - Fix the 404 error when deleting `evict-leader-scheduler` or `grant-leader-scheduler` using pd-ctl #2446
  - Fix the issue that the `presplit` feature might not work properly when the TiFlash replica exists #2447

- Tools

  - Backup & Restore (BR)

- Fix the issue that the data restoration fails due to network issues when BR restores data from cloud storage #298

- TiCDC
  - Fix system panics caused by data race #565 #566
  - Fix resource leaks or system blockages caused by incorrect processing logic #574 #586
  - Fix the issue that the command line gets stuck because CLI cannot connect to PD #579

16.12.18 TiDB 4.0 RC.2 Release Notes

Release date: May 15, 2020

TiDB version: 4.0.0-rc.2

16.12.18.1 Compatibility Changes

- TiDB
  - Remove the size limit for a single transaction (100 MB) when TiDB Binlog is enabled. Now the size limit for a transaction is 10 GB. However, if TiDB Binlog is enabled and the downstream is Kafka, configure the `txn-total-size-limit` parameter according to the message size limit of 1 GB in Kafka #16941
  - Change the behavior from querying the default time range to returning an error and requesting a specified time range if the time range is not specified when querying the `CLUSTER_LOG` table #17003
  - If the unsupported `sub-partition` or `linear hash` option is specified when creating the partitioned table using the `CREATE TABLE` statement, the normal table is created rather than the partitioned table with the options ignored #17197

- TiKV
  - Move the encryption-related configuration to the security-related configuration, which means changing `[encryption]` in the TiKV configuration file to `[security.encryption]` #7810

- Tools
  - TiDB Lightning
    - Change the default SQL mode to `ONLY_FULL_GROUP_BY,NO_AUTO_CREATE_USER` when importing data to improve compatibility #316
    - Disallow accessing the PD or TiKV port in the tidb-backend mode #312
    - Print the log information to the tmp file by default, and print the path of the tmp file when TiDB Lightning is started #313

16.12.18.2 Important Bug Fixes

- TiDB
    - Fix the issue that the wrong partition is chosen when the `WHERE` clause has only one equivalent condition #17054
    - Fix the issue of wrong results caused by building the incorrect Index range when the `WHERE` clause only contains the string column #16660
    - Fix the panic issue that occurs when executing the `PointGet` query in the transaction after the `DELETE` operation #16991
    - Fix the issue that the GC worker might encounter the deadlock when an error occurs #16915
    - Avoid the unnecessary RegionMiss retry when the TiKV response is slow but not down #16956
    - Change the log level in the client in the handshake phase of the MySQL protocol to `DEBUG` to solve the problem that interferes with log output #16881
    - Fix the issue that the Region is not pre-split according to the `PRE_SPLIT_REGIONS` information defined by the table after the `TRUNCATE` operation #16776
    - Fix the issue of soaring goroutine caused by retry when TiKV is unavailable during the second phase of the two-phase commit #16876
    - Fix the panic issue of statement execution when some expressions cannot be pushed down #16869
    - Fix the wrong execution result of the IndexMerge operation on the partitioned table #17124
    - Fix the performance reduction of `wide_table` caused by the mutex contention of Memory Trackers #17234
- TiFlash
    - Fix the issue that the system cannot start normally after the upgrade if the name of the database or table contains special characters

16.12.18.3 New Features

- TiDB
    - Add support for the `BACKUP` and `RESTORE` commands to back up and restore data #16960
    - Support pre-checking the data volume in a single Region before commit and pre-splitting the Region when the data volume exceeds the threshold #16959
    - Add the new `LAST_PLAN_FROM_CACHE` variable with a `Session` scope to indicate whether the last executed statement hits the plan cache #16830
    - Support recording the `Cop_time` information in slow log and the `SLOW_LOG` table #16904

- Add in Grafana more metrics that monitor the memory status of Go Runtime #16928
- Support outputting the `forUpdateTS` and `Read Consistency` isolation level information in General Log #16946
- Support collapsing duplicate requests of resolving locks in TiKV Region #16925
- Support using the `SET CONFIG` statement to modify the configuration of PD/TiKV nodes #16853
- Support the `auto_random` option in the `CREATE TABLE` statement #16813
- Allocate TaskID for the DistSQL request to help TiKV better schedule and process requests #17155
- Support displaying the version information of the TiDB server after logging into the MySQL client #17187
- Support the `ORDER BY` clause in the `GROUP_CONCAT` function #16990
- Support displaying the `Plan_from_cache` information in slow log to indicate whether the statement hits plan cache #17121
- Add the feature that TiDB Dashboard can display the capacity information of TiFlash multi-disk deployment
- Add the feature of querying the TiFlash log using SQL statements in Dashboard

- TiKV

  - Support encryption debugging for tikv-ctl, so that tikv-ctl can be used to operate and manage the cluster when the encryption storage is enabled #7698
  - Support encrypting the lock column family in snapshots #7712
  - Use the heatmap in the Grafana dashboard for Raftstore latency summary to better diagnose the jitter issue #7717
  - Support setting the upper limit for the size of the gRPC message #7824
  - Add in Grafana dashboard the encryption-related monitoring metrics #7827
  - Support Application-Layer Protocol Negotiation (ALPN) #7825
  - Add more statistics about Titan #7818
  - Support using the task ID provided by the client as the identifier in the unified read pool to avoid that the priority of a task is lowered by another task in the same transaction #7814
  - Improve the performance of the `batch insert` request #7718

- PD

  - Eliminate the speed limit of removing peers when making a node offline #2372

- TiFlash

  - Change the name of the Count graph of **Read Index** in Grafana to **Ops**
  - Optimize the data for opening file descriptors when the system load is low to reduce system resource consumption
  - Add the capacity-related configuration parameter to limit the the data storage capacity

- Tools
  - TiDB Lightning
    - Add the `fetch-mode` sub-command in tidb-lightning-ctl to print the TiKV cluster mode #287
  - TiCDC
    - Support managing the replication task by using `cdc cli` (changefeed) #546
  - Backup & Restore (BR)
    - Support automatically adjusting GC time during backup #257
    - Adjust PD parameters when restoring data to speed up the restoration #198

16.12.18.4 Bug Fixes

- TiDB
  - Improve the logic that determines whether to use vectorization for expression execution in multiple operators #16383
  - Fix the issue that the `IndexMerge` hint fails to check the database name correctly #16932
  - Forbid truncating the sequence object #17037
  - Fix the issue that the `INSERT/UPDATE/ANALYZE/DELETE` statements can be performed on a sequence object #16957
  - Fix the issue that the internal SQL statements in the bootstrap phase are not correctly marked as internal queries in the Statement Summary table #17062
  - Fix the error that occurs when a filter condition supported by TiFlash but not by TiKV is pushed down to the `IndexLookupJoin` operator #17036
  - Fix the concurrency issue of the `LIKE` expression that might occur after the collation is enabled #16997
  - Fix the issue that the `LIKE` function cannot correctly build the `Range` query index after the collation is enabled #16783
  - Fix the issue that a wrong value is returned when executing `@@LAST_PLAN_FROM_CACHE` ↪ after the `Plan Cache` statement is triggered #16831
  - Fix the issue that `TableFilter` on the index is missed when calculating candidate paths for `IndexMerge` #16947
  - Fix the issue that a physical query plan cannot be generated when using the `MergeJoin` hint and the `TableDual` operator exists #17016
  - Fix the wrong capitalization of the values in the `Stmt_Type` column of the Statement Summary table #17018
  - Fix the issue that the `Permission Denied` error is reported because the service cannot be started when different users use the same `tmp-storage-path` #16996
  - Fix the issue that the `NotNullFlag` result type is incorrectly set for an expression whose result type is determined by multiple input columns, such as `CASE WHEN` #16995

- Fix the issue that the green GC might leave unresolved locks when dirty stores exist #16949
- Fix the issue that the green GC might leave unresolved locks when encountering a single key with multiple different locks #16948
- Fix the issue of inserting a wrong value in the `INSERT VALUE` statement because a sub-query refers to a parent query column #16952
- Fix the issue of incorrect results when using the `AND` operator on the `Float` value #16666
- Fix the wrong information of the `WAIT_TIME` field in the expensive log #16907
- Fix the issue that the `SELECT FOR UPDATE` statement cannot be recorded in the slow log in the pessimistic transaction mode #16897
- Fix the wrong result that occurs when executing `SELECT DISTINCT` on a column of the `Enum` or `Set` type #16892
- Fix the display error of `auto_random_base` in the `SHOW CREATE TABLE` statement #16864
- Fix the incorrect value of `string_value` in the `WHERE` clause #16559
- Fix the issue that the error message of the `GROUP BY` window function is inconsistent with that of MySQL #16165
- Fix the issue that the `FLASH TABLE` statement fails to execute when the database name contains the uppercase letter #17167
- Fix the inaccurate memory tracing of the Projection executor #17118
- Fix the issue of incorrect time filtering of the `SLOW_QUERY` table in different time zones #17164
- Fix the panic issue that occurs when `IndexMerge` is used with the virtual generated column #17126
- Fix the capitalization issue of the `INSTR` and `LOCATE` function #17068
- Fix the issue that the `tikv server timeout` error is reported frequently after the `tidb_allow_batch_cop` configuration is enabled #17161
- Fix the issue that the result of performing `XOR` operation on the Float type is inconsistent with that of MySQL 8.0 #16978
- Fix the issue that no error is reported when the unsupported `ALTER TABLE ↪ REORGANIZE PARTITION` statement is executed #17178
- Fix the issue that an error is reported when `EXPLAIN FORMAT="dot" FOR ↪ CONNECTION ID` encounters an unsupported plan #17160
- Fix the record issue of the prepared statement in the `EXEC_COUNT` column of the Statement Summary table #17086
- Fix the issue that the value is not validated when setting the Statement Summary system variable #17129
- Fix the issue that an error is reported if an overflow value is used to query the `UNSIGNED BIGINT` primary key when the plan cache is enabled #17120
- Fix the incorrect QPS display by the machine instance and request type on the Grafana **TiDB Summary** dashboard #17105

- TiKV

- Fix the issue that many empty Regions are generated after restoration #7632
- Fix the panic issue of Raftstore when receiving out-of-order read index responses #7370
- Fix the issue that an invalid storage or coprocessor read pool configuration might not be rejected when the unified thread pool is enabled #7513
- Fix the panic issue of the `join` operation when the TiKV server is shut down #7713
- Fix the issue that no result is returned when searching TiKV slow logs via diagnostics API #7776
- Fix the issue that notable memory fragmentation is generated when the TiKV node is running for a long time #7556
- Fix the issue that the SQL statement fails to execute when an invalid date is stored #7268
- Fix the issue that the backup data cannot be restored from GCS #7739
- Fix the issue that KMS key ID is not validated during encryption at rest #7719
- Fix the underlying correctness issue of the Coprocessor in compilers of different architecture #7714 #7730
- Fix the `snapshot ingestion` error when encrytion is enabled #7815
- Fix the `Invalid cross-device link` error when rewriting the configuration file #7817
- Fix the issue of wrong toml format when writing the configuration file to an empty file #7817
- Fix the issue that a destroyed peer in Raftstore can still process requests #7836

- PD

    - Fix the `404` issue that occurs when using the `region key` command in pd-ctl #2399
    - Fix the issue that the monitor metrics of TSO and ID allocation are missing from the Grafana dashboard #2405
    - Fix the issue that pd-recover is not included in the Docker image #2406
    - Parse the path of data directory to an absolute path to fix the issue that TiDB Dashboard might not correctly display PD information #2420
    - Fix the issue that there is no default output when using the `scheduler config` ↪ `shuffle-region-scheduler` command in pd-ctl #2416

- TiFlash

    - Fix the issue that the wrong information of used capacity is report in some scenarios

- Tools

    - TiDB Binlog

        - Fix the issue that data of the `mediumint` type is not processed when the downstream is Kafka #962

- Fix the issue that the reparo fails to parse the DDL statement when the database name in DDL is a keyword #961

- TiCDC
  - Fix the issue of using the wrong time zone when the `TZ` environment variable is not set #512
  - Fix the issue that the owner does not clean up the resources when the server exits because some errors are not handled correctly #528
  - Fix the issue that TiCDC might be stuck when reconnecting to TiKV #531
  - Optimize the memory usage when initializing the table schema #534
  - Use the `watch` mode to monitor the replication status changes and perform quasi-real-time updates to reduce replication delay #481

- Backup & Restore (BR)
  - Fix the issue that inserting data might trigger the `duplicate entry` error after BR restores a table with the `auto_random` attribute #241

16.12.19 TiDB 4.0 RC.1 Release Notes

Release date: April 28, 2020

TiDB version: 4.0.0-rc.1

16.12.19.1 Compatibility Changes

- TiKV
  - Disable the Hibernate Region feature by default #7618

- TiDB Binlog
  - Support the sequence DDL operation in Drainer #950

16.12.19.2 Important Bug Fixes

- TiDB
  - Fix the issue that the `INSERT ... ON DUPLICATE UPDATE` statement might be incorrectly executed on multiple rows in an explicit transaction because `MemBuffer` is not checked #16689
  - Fix the data inconsistency when locking duplicated keys on multiple rows #16769
  - Fix the panic that occurs when recycling the non-superbatch idle connection between TiDB instances #16303

- TiKV

- Fix the deadlock issue caused by the probe request from TiDB #7540
- Fix the issue that the minimum commit timestamp of a transaction might overflow which affects data correctness #7638

- TiFlash

  - Fix the data loss issue caused by the `rename table` operation when multiple data paths are configured
  - Fix the issue that an error occurs when reading data from a merged Region
  - Fix the issue that an error occurs when reading data from a Region that is in the abnormal state
  - Modify the mapping of table names in TiFlash to correctly support `recover table`/`flashback table`
  - Modify the storage path to fix the potential data loss issue that occurs when renaming the table
  - Fix the potential panic of TiDB when Super Batch is enabled
  - Modify the read mode in the online update scenario to improve the read performance

- TiCDC

  - Fix the replication failure that occurs because the schema internally maintained in TiCDC fails to correctly handle the timing issue of read and write operations #438 #450 #478 #496
  - Fix the bug that the TiKV client fails to correctly maintain the internal resources when encountering some TiKV anomalies #499 #492
  - Fix the bug that meta data is not correctly cleaned up and abnormally remains in the TiCDC nodes #488 #504
  - Fix the issue that the TiKV client fails to correctly handle the repeated sending of the prewrite event #446
  - Fix the issue that the TiKV client fails to correctly handle the redundant prewrite events received before the initialization #448

- Backup & Restore (BR)

  - Fix the issue that checksum is still executed when checksum is disabled #223
  - Fix the incremental replication failure when `auto-random` or `alter-pk` is enabled in TiDB #230 #231

16.12.19.3 New Features

- TiDB

  - Support sending Coprocessor requests to TiFlash in batches #16226
  - Enable the Coprocessor cache feature by default #16710

    - Parse only the registered sections of a statement in the special comment of the SQL statement #16157
    - Support using the `SHOW CONFIG` syntax to show the configurations of PD and TiKV instances #16475

- TiKV

    - Support using the user-owned KMS key for the server-side encryption when backing up data to S3 #7630
    - Enable the load-based `split region` operation #7623
    - Support validating common names #7468
    - Add the file lock check to avoid starting multiple TiKV instances that are bound to the same address #7447
    - Support AWS KMS in encryption at rest #7465

- Placement Driver (PD)

    - Remove `config manager` to let other components control their component configurations #2349

- TiFlash

    - Add the metrics report related to the read and write workloads of DeltaTree engine
    - Cache the `handle` and `version` columns to reduce the disk I/O of a single read or write request
    - Support pushing down the `fromUnixTime` and `dateFormat` functions
    - Evaluate the global state according to the first disk and report this evaluation
    - Add the graphics in Grafana related to the read and write workloads of DeltaTree engine
    - Optimize the decimal data encoding in the `Chunk` codec
    - Implement the gRPC API of Diagnostics (SQL diagnosis) to support querying system tables such as `INFORMATION_SCHEMA.CLUSTER_INFO`

- TiCDC

    - Support sending messages in batches in the Kafka sink module #426
    - Support file sorting in the processor #477
    - Support automatic `resolve lock` #459
    - Add the feature that automatically updates the TiCDC service GC safe point to PD #487
    - Add the timezone setting for data replication #498

- Backup and Restore (BR)

    - Support configuring S3/GCS in the storage URL #246

16.12.19.4 Bug Fixes

- TiDB
- Fix the issue that negative numbers cannot be correctly displayed in the system table because the columns are defined as unsigned #16004
- Add a warning when the `use_index_merge` hint contains the invalid index name #15960
- Forbid multiple instances of a TiDB server sharing the same temporary directory #16026
- Fix the panic that occurs during the execution of `explain for connection` when the plan cache is enabled #16285
- Fix the issue that the result of the `tidb_capture_plan_baselines` system variable is incorrectly displayed #16048
- Fix the issue that the `group by` clause in the `prepare` statement is incorrectly parsed #16377
- Fix the panic that might occur during the execution of the `analyze primary key` statement #16081
- Fix the issue that the TiFlash store information in the `cluster_info` system table is wrong #16024
- Fix the panic that might occur during the Index Merge process #16360
- Fix the issue that an incorrect result might occur when the Index Merge reader reads the generated columns #16359
- Fix the incorrect display of the default sequence value in the `show create table` statement #16526
- Fix the issue that the `not-null` error is returned because the sequence is used as the default values of the primary key #16510
- Fix the issue that no error is reported for a blocked SQL execution when TiKV continues to return the `StaleCommand` error #16530
- Fix the issue that an error is reported if you only specify `COLLATE` when creating a database; add the missing `COLLATE` part in the result of `SHOW CREATE DATABASE` #16540
- Fix the partition pruning failure when the plan cache is enabled #16723
- Fix the bug that `PointGet` returns wrong results when handling the overflow #16755

- Fix the issue that a wrong result is returned when querying the `slow_query` system table with equal time values #16806

- TiKV

    - Address the OpenSSL security issue: CVE-2020-1967 #7622
    - Avoid protecting rollback records written by `BatchRollback` to improve performance when many write conflicts exist in optimistic transactions #7604
    - Fix the issue that the needless wake-up of transactions results in useless retry and performance reduction in heavy lock-race workloads #7551
    - Fix the issue that the Region might be stuck in the multi-time merging #7518
    - Fix the issue that the learner is not deleted when deleting the learner #7518
    - Fix the issue that follower read might cause panic in raft-rs #7408
    - Fix the bug that a SQL operation might fail because of the `group by constant` error #7383
    - Fix the issue that an optimistic lock might block reads if the corresponding primary lock is a pessimistic lock #7328

- PD

    - Fix the issue that some APIs might fail in the TLS validation #2363
    - Fix the issue that the configuration API cannot accept a configuration item with a prefix #2354
    - Fix the issue that the `500` error is returned when the scheduler is not found #2328
    - Fix the issue that the `404` error is returned for the `scheduler config balance -hot-region-scheduler list` command #2321

- TiFlash

    - Disable the coarse-grained index optimization for the storage engine
    - Fix the bug that an exception is thrown when resolving locks for Regions and some locks need to be skipped
    - Fix the null pointer exception (NPE) when collecting the Coprocessor statistics
    - Fix the check for Region meta to ensure that the process of Region Split/Region Merge is correct
    - Fix the issue that the message size exceeds the limit for gRPC because the size of Coprocessor response is not estimated
    - Fix the handling of the `AdminCmdType::Split` command in TiFlash

16.12.20 TiDB 4.0 RC Release Notes

Release date: April 8, 2020

TiDB version: 4.0.0-rc

TiUP version: 0.0.3

Warning:

Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 4.0.x version.

16.12.20.1 Compatibility Changes

- TiDB
    - Refuse to get started instead of returning an alert log when the tidb-server status port is occupied #15177
- TiKV
    - Support the `pipelined` feature in pessimistic transactions, which improves the TPC-C performance by 20%. The risk is that the transaction commit might fail because of lock failure during the execution #6984
    - Enable the `unify-read-pool` configuration item in new clusters by default and use the previous setting of this item in old clusters #7059
- Tools
    - TiDB Binlog
        - Add the configuration item for verifying Common Name #934

16.12.20.2 Important Bug Fixes

- TiDB
    - Fix the issue that replication between the upstream and downstream might go wrong when the DDL job is executed using the `PREPARE` statement because of the incorrect job query in the internal records #15435
    - Fix the issue of incorrect subquery result in the `Read Commited` isolation level #15471
    - Fix the issue of incorrect results caused by the Inline Projection optimization #15411
    - Fix the issue that the SQL Hint `INL_MERGE_JOIN` is executed incorrectly in some cases #15515
    - Fix the issue that columns with the `AutoRandom` attribute are rebased when the negative number is explicitly written to these columns #15397

16.12.20.3 New Features

- TiDB
  - Add the case-insensitive collation so that users can enable `utf8mb4_general_ci` and `utf8_general_ci` in a new cluster #33
  - Enhance the `RECOVER TABLE` syntax to support recovering truncated tables #15398
  - Refuse to get started instead of returning an alert log when the the tidb-server status port is occupied #15177
  - Optimize the write performance of using a sequence as the default column values #15216
  - Add the `DDLJobs` system table to query the details of DDL jobs #14837
  - Optimize the `aggFuncSum` performance #14887
  - Optimize the output of `EXPLAIN` #15507
- TiKV
  - Support the `pipelined` feature in pessimistic transactions, which improves the TPC-C performance by 20%. The risk is that the transaction commit might fail because of lock failure during the execution #6984
  - Support TLS in the HTTP port #5393
  - Enable the `unify-read-pool` configuration item in new clusters by default and use the previous setting of this item in old clusters #7059
- PD
  - Support getting the default PD configuration information through the HTTP API #2258
- Tools
  - TiDB Binlog
    - Add the configuration item for verifying Common Name #934
  - TiDB Lightning
    - Optimize the performance of TiDB Lightning #281 #275

16.12.20.4 Bug Fixes

- TiDB
  - Fix the issue that replication between the upstream and downstream might go wrong when the DDL job is executed using the `PREPARE` statement because of the incorrect job query in the internal records #15435

- Fix the issue of incorrect subquery result in the `Read Commited` isolation level #15471
- Fix the issue of possible wrong behavior when using `INSERT ... VALUES` to specify the `BIT(N)` data type #15350
- Fix the issue that the DDL Job internal retry does not fully achieve the expected outcomes because the values of `ErrorCount` fail to be summed correctly #15373
- Fix the issue that Garbage Collection might work abnormally when TiDB connects to TiFlash #15505
- Fix the issue of incorrect result caused by the Inline Projection optimization #15411
- Fix the issue that the SQL Hint `INL_MERGE_JOIN` is executed incorrectly in some cases #15515
- Fix the issue that columns with the `AutoRandom` attribute are rebased when the negative number is explicitly written to these columns #15397

- TiKV

  - Fix the possible panic caused by transferring the leader when the Follower Read feature is enabled #7101

- Tools

  - TiDB Lightning

    - Fix the issue of data error caused by the error of character conversion when the backend is TiDB #283

  - TiCDC

    - Fix the issue that an error is returned if the `test` schema does not exist in the downstream when MySQL sink is executing the DDL statement #353
    - Support the real-time interactive mode in CDC cli #351
    - Support checking whether the table in the upstream can be replicated during data replication #368
    - Support asynchronous write to Kafka #344

16.12.21 TiDB 4.0.0 Beta.2 Release Notes

Release date: March 18, 2020

TiDB version: 4.0.0-beta.2

TiDB Ansible version: 4.0.0-beta.2

16.12.21.1 Compatibility Changes

- Tools

- TiDB Binlog
  - Fix the issue that the system returns an error and exits when `disable-dispatch` and `disable-causality` are configured in Drainer #915

16.12.21.2 New Features

- TiKV
  - Support persisting the dynamically updated configuration into the hardware disk #6684
- PD
  - Support persisting the dynamically updated configuration into the hardware disk #2153
- Tools
  - TiDB Binlog
    - Support the bidirectional data replication between TiDB clusters #879 #903
  - TiDB Lightning
    - Support the TLS configuration #40 #270
  - TiCDC
    - Initial release of the change data capture (CDC), providing the following features:
      - Support capturing changed data from TiKV
      - Support replicating the changed data from TiKV to MySQL compatible databases, and guarantee the eventual data consistency
      - Support replicating the changed data to Kafka, and guarantee either the eventual data consistency or the row-level orderliness
      - Provide process-level high availability
  - Backup & Restore (BR)
    - Enable experimental features such as incremental backup and backing up files to Amazon S3 #175
- TiDB Ansible
  - Support injecting the node information to etcd #1196
  - Support deploying TiDB services on the ARM platform #1204

16.12.21.3 Bug Fixes

- TiKV
  - Fix the panic issue that might occur when meeting empty short values during the backup #6718

    - Fix the issue that Hibernate Regions might not be correctly awakened in some cases #6772 #6648 #6376

- PD

    - Fix the panic issue that the rule checker fails to allocate stores to Regions #2160
    - Fix the issue that after the dynamic configuration is enabled, the configuration might have replication delay when the Leader is being switched #2154

- Tools

    - Backup & Restore (BR)
        - Fix the issue that BR might fail to restore data of a large size because PD cannot process large-sized data #167
        - Fix the BR failure occurred because the BR version is not compatible with the TiDB version #186
        - Fix the BR failure occurred because the BR version is not compatible with TiFlash #194

16.12.22 TiDB 4.0.0 Beta.1 Release Notes

Release date: February 28, 2020

TiDB version: 4.0.0-beta.1

TiDB Ansible version: 4.0.0-beta.1

16.12.22.1 Compatibility Changes

- TiDB

    - Modify the type of the `log.enable-slow-log` configuration item from integer to Boolean #14864
    - Modify the `password` field name to `authentication_string` in the `mysql.user` system table to make it consistent with MySQL 5.7 (**This compatibility change means that you cannot roll back to earlier versions.**) #14598
    - Adjust the default value of the `txn-total-size-limit` configuration item from `1GB` to `100MB` #14522
    - Support dynamically modifying or updating configuration items read from PD #14750 #14303 #14830

- TiKV

    - Add the `readpool.unify-read-pool` configuration item (`True` by default) to control whether point queries use the same threads with Coprocessor #6375 #6401 #6534 #6582 #6585 #6593 #6597 #6677

- PD

    - Optimize the HTTP API to make it compatible with the configuration manager #2080

- TiDB Lightning

    - Use the default configurations specified in the document for certain items not configured in the configuration file #255

- TiDB Ansible

    - Rename `theflash` to `tiflash` #1130
    - Optimize the default values and related configurations in TiFlash's configuration file #1138

16.12.22.2 New Features

- TiDB

    - Support querying slow logs of any time in the `SLOW_QUERY / CLUSTER_SLOW_QUERY` system table #14840 #14878
    - Support SQL performance diagnosis
        - #14843 #14810 #14835 #14801 #14743
        - #14718 #14721 #14670 #14663 #14668
        - #14896
    - Support the `Sequence` function #14731 #14589 #14674 #14442 #14303 #14830
    - Support dynamically modifying or updating configuration items read from PD #14750 #14303 #14830
    - Add a feature of automatically reading data from different roles according to the load balancing policy and add the `leader-and-follower` system variable to enable this feature #14761
    - Add the `Coercibility` function #14739
    - Support setting TiFlash replicas in the partitioned table #14735 #14713 #14644
    - Improve the privilege check for the `SLOW_QUERY` table #14451
    - Support automatically write the intermediate results to the disk file if the memory is insufficient when using a SQL join #14708 #14279
    - Support checking table partitions by querying the `information_schema.PARTITIONS` system table #14347
    - Add the `json_objectagg` aggregate function #11154
    - Support logging rejected connection attempts in the audit log #14594
    - Add the `max-server-connections` configuration item (`4096` by default) to control the number of connections to a single server #14409
    - Support the isolation read specifying multiple storage engines at the server level #14440
    - Optimize the cost model of the `Apply` operator and the `Sort` operator to improve stability #13550 #14708

- TiKV

- Support fetching configuration items from the status port via HTTP API #6480
- Optimize the performance of `Chunk Encoder` in Coprocessor #6341

- PD
  - Support accessing to the distribution of hotspots in the cluster through Dashboard UI #2086
  - Support capturing and displaying `START_TIME` and `UPTIME` of cluster components #2116
  - Add the information of deployment path and component version in the returned message of the `member` API #2130
  - Add the `component` sub-command in pd-ctl to modify and check the configuration of other components (experimental) #2092
- TiDB Binlog
  - Support TLS between the components #904 #894
  - Add the `kafka-client-id` configuration item in Drainer to configure Kafka's client ID #902
  - Support purging the incremental backup data in Drainer #885
- TiDB Ansible
  - Support deploying multiple Grafana/Prometheus/Alertmanagers in one cluster #1142
  - Add the `metric_port` configuration item (`8234` by default) in TiFlash's configuration file #1145
  - Add the `flash_proxy_status_port` configuration item (`20292` by default) in TiFlash's configuration file #1141
  - Add the TiFlash monitoring dashboard #1147 #1151

16.12.22.3 Bug Fixes

- TiDB
  - Fix the issue that an error is reported when creating `view` with a column name that exceeds 64 characters #14850
  - Fix the issue that duplicate data exists in `information_schema.views` because the `create or replace view` statement is incorrectly processed #14832
  - Fix the incorrect results of `BatchPointGet` when `plan cache` is enabled #14855
  - Fix the issue that data is inserted into the wrong partitioned table after the timezone is modified #14370
  - Fix the panic occurred when rebuilding expression using the invalid name of the `IsTrue` function during the outer join simplification #14515
  - Fix the the incorrect privilege check for the`show binding` statement #14443
- TiKV

    - Fix the inconsistent behaviors of the `CAST` function in TiDB and TiKV #6463 #6461 #6459 #6474 #6492 #6569

- TiDB Lightning
    - Fix the bug that the web interface does not work outside the Server mode #259

16.12.23 TiDB 4.0 Beta Release Notes

Release date: January 17, 2020

TiDB version: 4.0.0-beta

TiDB Ansible version: 4.0.0-beta

16.12.23.1 TiDB

- Print the log or cancel the SQL execution when the memory used during the execution of `INSERT/REPLACE/DELETE/UPDATE` exceeds the limit specified by the `MemQuotaQuery` ↪ configuration item. The actual behavior depends on the `OOMAction` configuration. #14179 #14289 #14299
- Increase the accuracy of calculating the cost of `Index Join` by considering the row counts of both driving tables and driven tables #12085
- Add 15 SQL hints to control the behavior of the optimizer and make the optimizer more stable
    - #11253 #11364 #11673 #11740 #11746
    - #11809 #11996 #12043 #12059 #12246
    - #12382
- Improve the performance when the columns involved in a query can be fully covered by indexes #12022
- Improve the performance of table query by supporting the Index Merge feature #10121 #10512 #11245 #12225 #12248 #12305 #12843
- Improve the performance of Range calculation and reduce the CPU overhead by caching index results and eliminating duplicate results #12856
- Decouple the level of slow logs from the level of ordinary logs #12359
- Add the `oom-use-tmp-storage` parameter (`true` by default) to control whether to use temporary files to cache intermediate results when the memory usage for the execution of a single SQL statement exceeds `mem-quota-query` and the SQL contains `Hash Join` #11832 #11937 #12116 #12067
- Support using `create index/alter table` to create expression index and using `drop` ↪ `index` to drop expression index #14117
- Increase the default value of the `query-log-max-len` parameter to `4096` to reduce the number of truncated SQL outputs. This parameter can be adjusted dynamically. #12491

- Support adding the `AutoRandom` keyword in the column attribute to control whether the system automatically assigns a random integer to the primary key, which avoids the hotspot problem caused by the `AUTO_INCREMENT` primary key #13127
- Support Table Locks #11038
- Support using the `LIKE` or `WHERE` clause in `ADMIN SHOW DDL JOBS` for conditional filtering #12484
- Add the `TIDB_ROW_ID_SHARDING_INFO` column in the `information_schema.tables` ↪ table to output the `RowID` scattering information (for example, the value of the `SHARD_ROW_ID_BITS` column in table `A` is `"SHARD_BITS={bit_number}"`) #13418
- Optimize the error code of SQL error messages to avoid the situation that the `ERROR` ↪ `1105 (HY000)` code is used for multiple error messages (the `Unknown Error` type)
    - #14002 #13874 #13733 #13654 #13646
    - #13540 #13366 #13329 #13300 #13233
    - #13033 #12866 #14054
- Convert a narrow data range of the discrete type into `point set` and use CM-Sketch to improve the estimation accuracy when estimating the number of rows #11524
- Extract the `TopN` information from CM-Sketch for normal `Analyze` and separately maintain the frequently occurring values #11409
- Support dynamically adjusting the depth and width of CM-Sketch and the number of `TopN` information #11278
- Support automatically capturing and evolving SQL Binding #13199 #12434
- Optimize the encoding format of communication with TiKV by using `Chunk` to improve communication performance #12023 #12536 #12613 #12621 #12899 #13060 #13349
- Support the new row store format to improve the performance of the wide table #12634
- Optimize the `Recover Binlog` interface to ensure waiting all transactions to be committed before returning to the client #13740
- Support querying the binlog statuses enabled by TiDB servers in the cluster through the HTTP `info/all` interface #13025
- Support the MySQL-compatible `Read Committed` transaction isolation level when using the pessimistic transaction mode #14087
- Support large-sized transactions. The transaction size is limited by the size of the physical memory.
    - #11999 #11986 #11974 #11817 #11807
    - #12133 #12223 #12980 #13123 #13299
    - #13432 #13599
- Improve the stability of `Kill` #10841
- Support hexadecimal and binary expressions as separators in `LOAD DATA` #11029
- Improve the performance of `IndexLookupJoin` and reduce memory consumption during execution by splitting `IndexLookupJoin` into `IndexHashJoin` and `IndexMergeJoin` #8861 #12139 #12349 #13238 #13451 #13714
- Fix several issues relating to RBAC #13896 #13820 #13940 #14090 #13940 #13014
- Fix the issue that `VIEW` cannot be created because the `SELECT` statement contains `union` #12595

- Fix several issues relating to the `CAST` function
    - #12858 #11968 #11640 #11483 #11493
    - #11376 #11355 #11114 #14405 #14323
    - #13837 #13401 #13334 #12652 #12864
    - #12623 #11989
- Output the detailed `backoff` information of TiKV RPC in the slow log to facilitate troubleshooting #13770
- Optimize and unify the format of the memory statistics in the expensive log #12809
- Optimize the explicit format of `EXPLAIN` and support outputting information about the operator's usage of memory and disk #13914 #13692 #13686 #11415 #13927 #13764 #13720
- Optimize the check for duplicate values in `LOAD DATA` based on the transaction size and support setting the transaction size by configuring the `tidb_dml_batch_size` parameter #11132
- Optimize the performance of `LOAD DATA` by separating the data preparing routine and the commit routine and assigning the workload to different Workers #11533 #11284

16.12.23.2 TiKV

- Upgrade the RocksDB version to 6.4.6
- Fix the issue that the system cannot perform the compaction task normally when the disk space is used up by automatically creating a 2GB empty file when TiKV is started #6321
- Support quick backup and restoration
    - #6462 #6395 #6378 #6374 #6349
    - #6339 #6308 #6295 #6286 #6283
    - #6261 #6222 #6209 #6204 #6202
    - #6198 #6186 #6177 #6146 #6071
    - #6042 #5877 #5806 #5803 #5800
    - #5781 #5772 #5689 #5683
- Support reading data from Follower replicas
    - #5051 #5118 #5213 #5316 #5401
    - #5919 #5887 #6340 #6348 #6396
- Improve the performance of TiDB reading data through index #5682
- Fix the issue that the `CAST` function behaves inconsistently in TiKV and in TiDB
    - #6459 #6461 #6458 #6447 #6440
    - #6425 #6424 #6390 #5842 #5528
    - #5334 #5199 #5167 #5146 #5141
    - #4998 #5029 #5099 #5006 #5095
    - #5093 #5090 #4987 #5066 #5038

- #4962 #4890 #4727 #6060 #5761
- #5793 #5468 #5540 #5548 #5455
- #5543 #5433 #5431 #5423 #5179
- #5134 #4685 #4650 #6463

16.12.23.3 PD

- Support optimizing hotspot scheduling according to the load information of storage nodes
  - #1870 #1982 #1998 #1843 #1750
- Add the Placement Rules feature that supports controlling the number of replicas of any data range, the storage location, the storage host type and roles by combining different scheduling rules
  - #2051 #1999 #2042 #1917 #1904
  - #1897 #1894 #1865 #1855 #1834
- Support using plugins (experimental) #1799
- Add the feature that the schedulers support the customized configuration and key ranges (experimental) #1735 #1783 #1791
- Support automatically adjusting the scheduling speed according the cluster load information (experimental, disabled by default) #1875 #1887 #1902

16.12.23.4 Tools

- TiDB Lightning
  - Add the parameter in the command-line tool to set the password of the downstream database #253

16.12.23.5 TiDB Ansible

- Add checksum check in the package in case that the downloaded package is incomplete #1002
- Support checking the systemd version which must be `systemd-219-52` or later #1020 #1074
- Fix the issue that the log directory is incorrectly created when TiDB Lightning is started #1103
- Fix the issue that the customized port of TiDB Lightning is invalid #1107
- Support deploying and maintaining TiFlash #1119

16.13 v3.1

16.13.1 TiDB 3.1.2 Release Notes

Release date: June 4, 2020

TiDB version: 3.1.2

16.13.1.1 Bug Fixes

- TiKV
    - Fix the error handling issue during backup and restoration with S3 and GCS #7965
    - Fix the `DefaultNotFound` error that occurs during restoration #7838
- Tools
    - Backup & Restore (BR)
        - Retry automatically when the network is poor to improve stability with S3 and GCS storages #314 #7965
        - Fix a restoration failure that occurs because the Region leader cannot be found when restoring small tables #303
        - Fix a data loss issue during restoration when a table's row ID exceeds `2^(63)` #323
        - Fix the issue that empty databases and tables cannot be restored #318
        - Support using AWS KMS for server-side encryption (SSE) when targeting the S3 storage #261

16.13.2 TiDB 3.1.1 Release Notes

Release date: April 30, 2020

TiDB version: 3.1.1

TiDB Ansible version: 3.1.1

16.13.2.1 New Features

- TiDB
    - Add the table option for `auto_rand_base` #16812
    - Add the `Feature ID` comment: In the special comments of SQL statements, only the registered statement fragment can be parsed by the parser; otherwise, the statement is ignored #16155

- TiFlash
    - Cache the `handle` and `version` columns to reduce the disk I/O for a single read request
    - Add in Grafana the graphics related to the read and write workloads of DeltaTree engine
    - Optimize the decimal data encoding in the `Chunk` codec
    - Reduce the number of open file descriptors when TiFlash is in low workload

16.13.2.2 Bug Fixes

- TiDB
    - Fix the issue that the isolation read setting at the instance level does not take effect, and that the isolation read setting is incorrectly retained after TiDB is upgraded #16482 #16802
    - Fix the partition selection syntax on the hash partitioned table so that an error is not reported for syntaxes such as `partition (P0)` #16076
    - Fix the issue that when an `UPDATE` SQL statement only queries from a view but does not update the view, the update statement still reports an error #16789
    - Fix the issue of wrong results caused by removing the `not not` from the nested query #16423
- TiFlash
    - Fix the issue that an error occurs when reading data from a Region that is in the abnormal state
    - Modify the mapping of table names in TiFlash to correctly support `recover table`/`flashback table`
    - Modify the storage path to fix the potential data loss issue that occurs when renaming a table
    - Modify the read mode in the online update scenario to improve the read performance
    - Fix the issue that TiFlash fails to start normally after upgrade if the database/table name contains special characters
- Tools
    - Backup & Restore (BR)
        - Fix the issue that after BR restores a table with the `auto_random` attribute, inserting data might trigger the duplicate entry error #241

16.13.3 TiDB 3.1.0 GA Release Notes

Release date: April 16, 2020

TiDB version: 3.1.0 GA

TiDB Ansible version: 3.1.0 GA

16.13.3.1 Compatibility Changes

- TiDB
    - Support directly stopping starting TiDB if the HTTP listening port is unavailable when the `report-status` configuration item is enabled #16291
- Tools
    - Backup & Restore (BR)
        - BR does not support restoring data from the TiKV cluster earlier than 3.1 GA #233

16.13.3.2 New Features

- TiDB
    - Support displaying the information of Coprocessor tasks in `explain format = "dot"` #16125
    - Reduce the redundant stack information of log using the `disable-error-stack` configuration item #16182
- Placement Driver (PD)
    - Optimize the hot Region scheduling #2342
- TiFlash
    - Add the metrics report related to the read and write workloads of DeltaTree engine
    - Support pushing down the `fromUnixTime` and `dateFormat` functions
    - Disable the rough set filter by default
- TiDB Ansible
    - Add TiFlash monitor #1253 #1257
    - Optimize the configuration parameters of TiFlash #1262 #1265 #1271
    - Optimize the TiDB starting script #1268

16.13.3.3 Bug Fixes

- TiDB
    - Fix the panic issue caused by the merge join operation in some scenarios #15920
    - Fix the issue that some expressions are repeatedly counted in selectivity calculation #16052
    - Fix the panic issue occurred when loading the statistics information in extreme cases #15710
    - Fix the issue that an error is returned in some cases when equivalent expressions cannot be recognized in SQL query #16015
    - Fix the issue that an error is returned when querying the `view` of one database from another database #15867
    - Fix the panic issue that occurs when the column is handled using `fast analyze` #16080
    - Fix the incorrect character set of the `current_role` print result #16084
    - Refine the log of MySQL connection handshake error #15799
    - Fix the panic issue caused by port probing after the audit plugin is loaded #16065
    - Fix the panic issue of the `sort` operator on left join because the `TypeNull` class is mistaken as a variable-length type #15739
    - Fix the issue of inaccurate count of monitoring session retry errors #16120
    - Fix the issue of wrong results of `weekday` in the `ALLOW_INVALID_DATES` mode #16171
    - Fix the issue that Garbage Collection (GC) might not work normally when the cluster has TiFlash nodes #15761
    - Fix the issue that TiDB goes out of memory (OOM) when users set a large partition count when creating the hash partitioned table #16219
    - Fix the issue that warnings are mistaken as errors, and make the `UNION` statement have the same behavior as the `SELECT` statement #16138
    - Fix the execution error when `TopN` is pushed down to mocktikv #16200
    - Increase the initial length of `chunk.column.nullBitMap` to avoid unnecessary overhead of `runtime.growslice` #16142
- TiKV
    - Fix the panic issue caused by replica read #7418 #7369
    - Fix the issue that the restoration process creates empty Regions #7419
    - Fix the issue that repeated resolve lock requests might harm the atomicity of pessimistic transactions #7389
- TiFlash
    - Fix the potential issue of the `rename table` operation when replicating the schema from TiDB
    - Fix the issue of data loss caused by the `rename table` operation under multiple data path configurations

- Fix the issue that TiFlash reports incorrect storage space in some scenarios
- Fix the potential issue caused by reading from TiFlash when Region Merge is enabled

- Tools

    - TiDB Binlog

        - Fix the issue that TiFlash-related DDL jobs might interrupt the replication of Drainer #948 #942

    - Backup & Restore (BR)

        - Fix the issue that the `checksum` operation is still executed when it is disabled #223
        - Fix the issue that incremental backup fails when TiDB enables `auto-random` or `alter-pk` #230 #231

16.13.4 TiDB 3.1 RC Release Notes

Release date: April 2, 2020

TiDB version: 3.1.0-rc

TiDB Ansible version: 3.1.0-rc

> **Warning:**
>
> Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 3.1.x version.

16.13.4.1 New Features

- TiDB

    - Use the binary search to re-implement partition pruning for better performance #15678
    - Support using the `RECOVER` syntax to recover the truncated table #15460
    - Add the `AUTO_RANDOM` ID cache for retrying statements and recovering tables #15393
    - Support restoring the state of the `AUTO_RANDOM` ID allocator using the `recover table` statement #15393
    - Support `YEAR`, `MONTH`, and `TO_DAY` functions as the partitioning keys of the Hash partitioned table #15619

- Add the table ID to the schema-change related tables only when keys need to be locked in the `SELECT... FOR UPDATE` statement #15708
- Add the feature of automatically reading data from different roles according to the load balancing policy and add the `leader-and-follower` system variable to enable this feature #15721
- Support dynamically updating the TLS certificate every time TiDB establishes a new connection to update expired client certificate without restarting the RPC client side #15163
- Upgrade PD Client to support loading the latest certificate every time TiDB establishes a new connection #15425
- Forcibly use the HTTPS protocol with the configured TLS certificates between a TiDB server and a PD server, or between two TiDB servers when `cluster-ssl-*` is configured #15430
- Add the MySQL-compatible `--require-secure-transport` startup option to force the client to enable TLS authentication during the configuration #15442
- Add the `cluster-verify-cn` configuration item. After configuration, the status service can only be used when with the corresponding CN certificate #15137

- TiKV

  - Support backing up data with the Raw KV API #7051
  - Support TLS authentication for the status server #7142
  - Support TLS authentication for the KV server #7305
  - Optimize the time to hold locks to improve the performance of backup #7202

- PD

  - Support scheduling learner using `shuffle-region-scheduler` #2235
  - Add commands in pd-ctl to configure Placement Rules #2306

- Tools

  - TiDB Binlog

    - Support TLS authentication between the components #931 #937 #939
    - Add the `kafka-client-id` configuration item in Drainer to configure Kafka's client ID #929

  - TiDB Lightning

    - Optimize the performance of TiDB Lightning #281 #275
    - Support TLS authentication for TiDB Lightning #270

  - Backup & Restore (BR)

    - Optimize the log output #189

- TiDB Ansible

- Optimize the way the TiFlash data directories are created #1242
- Add the `Write Amplification` monitoring item in TiFlash #1234
- Optimize the error message of failed preflight checks when CPU epollexclusive is unavailable #1243

16.13.4.2 Bug Fixes

- TiDB
    - Fix the information schema error caused by frequently updating the TiFlash replica #14884
    - Fix the issue that `last_insert_id` is incorrectly generated when applying `AUTO_RANDOM` #15149
    - Fix the issue that updating the status of TiFlash replica might cause the DDL operation to get stuck #15161
    - Forbid `Aggregation` pushdown and `TopN` pushdown when there are predicates that cannot be pushed down #15141
    - Forbid the nested `view` creation #15440
    - Fix the error occurred when executing `SELECT CURRENT_ROLE()` after `SET ROLE ALL` #15570
    - Fix the failure to identify the `view` name when executing the `select view_name .col_name from view_name` statement #15573
    - Fix the issue that an error might occur when pre-processing DDL statements during the write of binlog information #15444
    - Fix the panic occurred when accessing both `views` and partitioned tables #15560
    - Fix the error occurred when executing the `VALUES` function with the `update duplicate key` statement that contains the `bit(n)` data type #15487
    - Fix the issue that the specified maximum execution time fails to take effect in some scenarios #15616
    - Fix the issue that whether the current `ReadEngine` contains TiKV server is not checked when generating the execution plan using `Index Scan` #15773
- TiKV
    - Fix the issue of conflict check failure or data index inconsistency caused by inserting an existing key into a transaction and then deleting it immediately when disabling the consistency check parameter #7112
    - Fix the calculation error when `TopN` compares unsigned integers #7199
    - Introduce a flow control mechanism in Raftstore to solve the problem that without flow control, it might cause slow log tracking and cause the cluster to be stuck; and the problem that the large transaction size might cause the frequent reconnection among TiKV servers #7087 #7078
    - Fix the issue that pending read requests sent to replicas might be permanently blocked #6543
    - Fix the issue that replica read might be blocked by applying snapshots #7249

- Fix the issue that transferring leader might cause TiKV to panic #7240
- Fix the issue that all SST files are filled with zeroes when backing up data to S3 #6967
- Fix the issue that the size of SST file is not recorded during backup, resulting in many empty Regions after restoration #6983
- Support AWS IAM web identity for backup #7297

- PD

  - Fix the issue of incorrect Region information caused by data race when PD processes Region heartbeats #2234
  - Fix the issue that `random-merge-scheduler` fails to follow location labels and Placement Rules #2212
  - Fix the issue that a placement rule is overwritten by another placement rule with the same `startKey` and `endKey` #2222
  - Fix the issue that the version number of API is inconsistent with that of PD server #2192

- Tools

  - TiDB Lightning

    - Fix the bug that the `&` character is replaced by the `EOF` character in TiDB backend #283

  - Backup & Restore (BR)

    - Fix the issue that BR cannot restore the TiFlash cluster data #194

16.13.5 TiDB 3.1 Beta.2 Release Notes

Release date: March 9, 2020

TiDB version: 3.1.0-beta.2

TiDB Ansible version: 3.1.0-beta.2

> **Warning:**
>
> Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 3.1.x version.

16.13.5.1 Compatibility Changes

- Tools
  - TiDB Lightning
    - Use the default configurations specified in the TiDB Lightning Configuration for certain items not configured in the configuration file #255
    - Add the `--tidb-password` CLI parameter to set the TiDB password #253

16.13.5.2 New Features

- TiDB
  - Support adding the `AutoRandom` keyword in the column attribute to enable TiDB to automatically assign random integers to the primary key, which avoids the write hot spot caused by the `AUTO_INCREMENT` primary key #14555
  - Support creating or deleting column store replicas through DDL statements #14537
  - Add the feature that the optimizer can independently select different storage engines #14537
  - Add the feature that the SQL hint supports different storage engines #14537
  - Support reading data from followers by using the `tidb_replica_read` system variable #13464
- TiKV
  - Raftstore
    - Add the `peer_address` parameter to connect other nodes to the TiKV server #6491
    - Add the `read_index` and `read_index_resp` monitoring metrics to monitor the number of `ReadIndex` requests #6610
- PD Client
  - Support reporting statistics of local threads to PD #6605
- Backup
  - Replace the `RocksIOLimiter` flow control library with Rust's `async-speed-limit` ↪ flow control library to eliminate extra memory copies when backing up a file #6462
- PD
  - Tolerate backslash in the location label name #2084
- TiFlash
  - Initial release

- TiDB Ansible
    - Support deploying multiple Grafana/Prometheus/Alertmanager in one cluster #1143
    - Support deploying the TiFlash component #1148
    - Add monitoring metrics related to the TiFlash component #1152

16.13.5.3 Bug Fixes

- TiKV
    - Raftstore
        - Fix the issue that the read requests cannot be processed because data is not properly read from Hibernate Regions #6450
        - Fix the panic issue caused by the `ReadIndex` requests during the leader transfer process #6613
        - Fix the issue that Hibernate Regions are not correctly awakened in some special conditions #6730 #6737 #6972
    - Backup
        - Fix the inconsistent data index during the restoration caused by the backup of the extra data #6659
        - Fix the panic caused by incorrectly processing the deleted values during the backup #6726
- PD
    - Fix the panic occurred because the rule checker fails to assign stores to Regions #2161
- Tools
    - TiDB Lightning
        - Fix the bug that the web interface does not work outside the Server mode #259
    - BR (Backup and Restore)
        - Fix the issue that BR cannot exit in time due to an unrecoverable error it encounters when restoring data #152
- TiDB Ansible
    - Fix the issue that the rolling update command fails because the PD Leader cannot be obtained in some scenarios #1122

16.13.6 TiDB 3.1 Beta.1 Release Notes

Release date: January 10, 2020

TiDB version: 3.1.0-beta.1

TiDB Ansible version: 3.1.0-beta.1

16.13.6.1 TiKV

- backup
  - Change the name of the backup file from `start_key` to the hash value of `start_key` to reduce the file name's length for easy reading #6198
  - Disable RocksDB's `force_consistency_checks` check to avoid false positives in the consistency check #6249
  - Add the incremental backup feature #6286
- sst_importer
  - Fix the issue that the SST file does not have MVCC properties during restoring #6378
  - Add the monitoring items such as `tikv_import_download_duration`, `tikv_import_download_` ↪ , `tikv_import_ingest_duration`, `tikv_import_ingest_bytes`, and `tikv_import_error_counter` to observe the overheads of downloading and ingesting SST files #6404
- raftstore
  - Fix the issue of Follower Read that the follower reads stale data when the leader changes, thus breaking transaction isolation #6343

16.13.6.2 Tools

- BR (Backup and Restore)
  - Fix the inaccurate backup progress information #127
  - Improve the performance of splitting Regions #122
  - Add the backup and restore feature for partitioned tables #137
  - Add the feature of automatically scheduling PD schedulers #123
  - Fix the issue that data is overwritten after non `PKIsHandle` tables are restored #139

16.13.6.3 TiDB Ansible

- Add the feature of automatically disabling Transparent Huge Pages (THP) in the operating system during the initialization phase #1086
- Add the Grafana monitoring for BR components #1093
- Optimize the deployment of TiDB Lightning by automatically creating related directories #1104

16.13.7 TiDB 3.1 Beta Release Notes

Release date: December 20, 2019

TiDB version: 3.1.0-beta

TiDB Ansible version: 3.1.0-beta

16.13.7.1 TiDB

- SQL Optimizer
    - Enrich SQL hints #12192
- New feature
    - Support the Follower Read feature #12535

16.13.7.2 TiKV

- Support the distributed backup and restore feature #5532
- Support the Follower Read feature #5562

16.13.7.3 PD

- Support the distributed backup and restore feature #1896

16.14 v3.0

16.14.1 TiDB 3.0.20 Release Notes

Release date: December 25, 2020

TiDB version: 3.0.20

16.14.1.1 Compatibility Change

- TiDB
    - Deprecate the `enable-streaming` configuration item #21054

16.14.1.2 Improvements

- TiDB
    - Raise an error when preparing the `LOAD DATA` statement #21222
- TiKV
    - Add the `end_point_slow_log_threshold` configuration item #9145

16.14.1.3 Bug Fixes

- TiDB
    - Fix the incorrect cache of the transaction status for pessimistic transactions #21706
    - Fix the issue of inaccurate statistics that occurs when querying `INFORMATION_SCHEMA.TIDB_HOT_REGIONS` #21319
    - Fix the issue that `DELETE` might not delete data correctly when the database name is not in a pure lower representation #21205
    - Fix the issue of stack overflow that occurs when building the recursive view #21000
    - Fix the issue of goroutine leak in TiKV client #20863
    - Fix the wrong default zero value for the `year` type #20828
    - Fix the issue of goroutine leak in index lookup join #20791
    - Fix the issue that executing `INSERT SELECT FOR UPDATE` returns the malformed packet in the pessimistic transaction #20681
    - Fix the unknown time zone `'posixrules'` #20605
    - Fix the issue that occurs when converting the unsigned integer type to the bit type #20362
    - Fix the corrupted default value of the bit type column #20339
    - Fix the potentially incorrect results when one of the equal condition is the `Enum` or `Set` type #20296
    - Fix a wrong behavior of `!= any()` #20061
    - Fix the issue that type conversion in `BETWEEN...AND...` returns invalid results #21503
    - Fix a compatibility issue with the `ADDDATE` function #21008
    - Set the correct default value for newly added `Enum` column #20999
    - Fix the result of SQL statements like `SELECT DATE_ADD('2007-03-28 22:08:28',INTERVAL "-2.-2" SECOND)` to be compatible with MySQL #20627
    - Fix the incorrect default value when modifying the column type #20532
    - Fix the issue that the `timestamp` function gets wrong result when the input argument is the `float` or `decimal` type #20469
    - Fix a potential deadlock issue in statistics #20424
    - Fix the issue that the overflown float type data is inserted #20251
- TiKV
    - Fix the issue that an error is returned indicating that a key exists when this key is locked and deleted in a committed transaction #8931
- PD
    - Fix the issue that too many logs are printed when starting PD and when there are too many stale Regions #3064

16.14.2 TiDB 3.0.19 Release Notes

Release date: September 25, 2020

TiDB version: 3.0.19

16.14.2.1 Compatibility Changes

- PD
    - Change the import path from `pingcap/pd` to `tikv/pd` #2779
    - Change the copyright information from `PingCAP, Inc` to `TiKV Project Authors` #2777

16.14.2.2 Improvements

- TiDB
    - Mitigate the impact of failure recovery on QPS performance #19764
    - Support adjusting the concurrency of the `union` operator #19885
- TiKV
    - Set `sync-log` to `true` as a nonadjustable value #8636
- PD
    - Add an alert rule for PD restart #2789

16.14.2.3 Bug Fixes

- TiDB
    - Fix the query error that occurs when the `slow-log` file does not exist #20050
    - Add the privilege check for `SHOW STATS_META` and `SHOW STATS_BUCKET` #19759
    - Forbid changing the decimal type to the integer type #19681
    - Fix the issue that the constraint is not checked when altering the `ENUM`/`SET` type column #20045
    - Fix the bug that tidb-server does not release table locks after a panic #20021
    - Fix the bug that the `OR` operator is not handled correctly in the `WHERE` clause #19901
- TiKV

- Fix the bug that TiKV panics when parsing responses with missing reason phrases #8540

- Tools

  - TiDB Lightning

    - Fix the issue that the TiDB Lightning process does not exit in time when encountering illegal UTF characters in CSV in the strict mode #378

16.14.3 TiDB 3.0.18 Release Notes

Release date: August 21, 2020

TiDB version: 3.0.18

16.14.3.1 Improvements

- Tools

  - TiDB Binlog

    - Support the time duration format of Go for the Pump GC configuration #996

16.14.3.2 Bug Fixes

- TiDB

  - Fix the issue that the wrong handling of the `decimal` type by the `Hash` function causes the wrong HashJoin result #19185
  - Fix the issue that the wrong handling of the `set` and `enum` types by the `Hash` function causes the wrong HashJoin result #19175
  - Fix the issue that the check for duplicate keys fails in the pessimistic locking mode #19236
  - Fix the issue that the `Apply` and `Union Scan` operators cause the wrong execution result #19297
  - Fix the issue that some cached execution plans are incorrectly executed in transaction #19274

- TiKV

  - Change the GC failure log from `error` to the `warning` level #8444

- Tools

  - TiDB Lightning

    - Fix the issue that the `--log-file` argument does not take effect #345
    - Fix the syntax error on empty binary/hex literals when using TiDB-backend #357
    - Fix the unexpected `switch-mode` call when using TiDB-backend #368

16.14.4 TiDB 3.0.17 Release Notes

Release date: Aug 3, 2020

TiDB version: 3.0.17

16.14.4.1 Improvements

- TiDB
    - Decrease the default value of the `query-feedback-limit` configuration item from 1024 to 512, and improve the statistics feedback mechanism to ease its impact on the cluster #18770
    - Limit batch split count for one request #18694
    - Accelerate `/tiflash/replica` HTTP API when there are many history DDL jobs in the TiDB cluster #18386
    - Improve row count estimation for index equal condition #17609
    - Speed up the execution of `kill tidb conn_id` #18506
- TiKV
    - Add the `hibernate-timeout` configuration that delays region hibernation to improve rolling update performance #8207
- Tools
    - TiDB Lightning
        - `[black-white-list]` has been deprecated with a newer, easier-to-understand filter format #332

16.14.4.2 Bug Fixes

- TiDB
    - Return the actual error message instead of an empty set when a query which contains `IndexHashJoin` or `IndexMergeJoin` encounters a panic #18498
    - Fix the unknown column error for SQL statements like `SELECT a FROM t HAVING t.a` #18432
    - Forbid adding a primary key for a table when the table has no primary key or when the table already has an integer primary key #18342
    - Return an empty set when executing `EXPLAIN FORMAT="dot" FOR CONNECTION` #17157
    - Fix `STR_TO_DATE`'s handling for format token'%r', '%h' #18725
- TiKV

- Fix a bug that might read stale data during region merging #8111
- Fix the issue of memory leak during the scheduling process #8355

- Tools

    - TiDB Lightning

        - Fix the issue that the `log-file` flag is ignored #345

16.14.5 TiDB 3.0.16 Release Notes

Release date: July 03, 2020

TiDB version: 3.0.16

16.14.5.1 Improvements

- TiDB

    - Support the `is null` filter condition in hash partition pruning #17308
    - Assign different `Backoffer`s to each Region to avoid the SQL timeout issue when multiple Region requests fail at the same time #17583
    - Split separate Regions for the newly added partition #17668
    - Discard feedbacks generated from the `delete` or `update` statement #17841
    - Correct the usage of `json.Unmarshal` in `job.DecodeArgs` to be compatible with future Go versions #17887
    - Remove sensitive information in the slow query log and the statement summary table #18128
    - Match the MySQL behavior with `DateTime` delimiters #17499
    - Handle `%h` in date formats in the range that is consistent with MySQL #17496

- TiKV

    - Avoid sending store heartbeats to PD after snapshots are received #8145
    - Improve the PD client log #8091

16.14.5.2 Bug Fixes

- TiDB

    - Fix the data inconsistency issue occurred because the lock of a written and deleted primary key in one transaction is resolved by another transaction #18248
    - Fix the `Got too many pings` gRPC error log in the PD server-side followers #17944
    - Fix the panic issue that might occur when the child of HashJoin returns the `TypeNull` column #17935

- Fix the error message when access is denied #17722
- Fix JSON comparison issue for the `int` and `float` types #17715
- Update the failpoint which causes data race #17710
- Fix the issue that the timeout pre-split Regions might not work when creating tables #17617
- Fix the panic caused by ambiguous error messages after the sending failure #17378
- Fix the issue that `FLASHBACK TABLE` might fail in some special cases #17165
- Fix the issue of inaccurate range calculation results when statements only have string columns #16658
- Fix the query error occurred when the `only_full_group_by` SQL mode is set #16620
- Fix the issue that the field length of results returned from the `case when` function is inaccurate #16562
- Fix the type inference for the decimal property in the `count` aggregate function #17702

- TiKV

    - Fix the potential wrong result read from ingested files #8039
    - Fix the issue that a peer cannot be removed when its store is isolated during multiple merge processes #8005

- PD

    - Fix the `404` error when querying Region keys in PD Control #2577

16.14.6 TiDB 3.0.15 Release Notes

Release date: June 5, 2020

TiDB version: 3.0.15

16.14.6.1 New Features

- TiDB

    - Forbid the query in partitioned tables to use the plan cache feature #16759
    - Support the `admin recover index` and `admin check index` statements on partitioned tables #17315 #17390
    - Support partition pruning of the `in` condition for Range partitioned tables #17318
    - Optimize the output of `SHOW CREATE TABLE`, and add quotation marks to the partition name #16315
    - Support the `ORDER BY` clause in the `GROUP_CONCAT` function #16988
    - Optimize the memory allocation mechanism of `CMSketch` statistics to reduce the impact of garbage collection (GC) on performance #17543

- PD
    - Add a policy in which PD performs scheduling in terms of the number of Leaders #2479

16.14.6.2 Bug Fixes

- TiDB
    - Use deep copy to copy the `enum` and `set` type data in the `Hash` aggregate function; fix an issue of correctness #16890
    - Fix the issue that `PointGet` returns incorrect results because of the wrong processing logic of integer overflow #16753
    - Fix the issue of incorrect results caused by incorrect processing logic when the `CHAR()` function is used in the query predicate #16557
    - Fix the issue of inconsistent results in the storage layer and calculation layer of the `IsTrue` and `IsFalse` functions #16627
    - Fix the incorrect `NotNull` flags in some expressions, such as `case when` #16993
    - Fix the issue that the optimizer cannot find a physical plan for `TableDual` in some scenarios #17014
    - Fix the issue that the syntax for partition selection does not take effect correctly in the Hash partitioned table #17051
    - Fix the inconsistent results between TiDB and MySQL when XOR operates on a floating-point number #16976
    - Fix the error that occurs when executing DDL statement in the prepared manner #17415
    - Fix the incorrect processing logic of computing the batch size in the ID allocator #17548
    - Fix the issue that the `MAX_EXEC_TIME` SQL hint does not take effect when the time exceeds the expensive threshold #17534
- TiKV
    - Fix the issue that memory defragmentation is not effective after running for a long time #7790
    - Fix the panic issue caused by incorrectly removing snapshot files after TiKV is restarted accidentally #7925
    - Fix the gRPC disconnection caused by too large message packages #7822

16.14.7 TiDB 3.0.14 Release Notes

Release date: May 9, 2020

TiDB version: 3.0.14

16.14.7.1 Compatibility Changes

- TiDB
    - Adjust the user privilege in `performance_schema` and `metrics_schema` from read-write to read-only #15417

16.14.7.2 Important Bug Fixes

- TiDB
    - Fix the issue that the query result of `index join` is incorrect when the `join` ↪ condition has multiple equivalent conditions on the column with the `handle` attribute #15734
    - Fix the panic that occurs when performing the `fast analyze` operation on the column with the `handle` attribute #16079
    - Fix the issue that the `query` field in the DDL job structure is incorrect when the DDL statement is executed in a way of `prepare`. This issue might cause data inconsistency between the upstream and the downstream when Binlog is used for data replication. #15443
- TiKV
    - Fix the issue that repeated requests on the cleanup of lock might destroy the atomicity of the transaction #7388

16.14.7.3 New Features

- TiDB
    - Add the schema name column and the table name column to the query results of the `admin show ddl jobs` statement #16428
    - Enhance the `RECOVER TABLE` syntax to support recovering truncated tables #15458
    - Support the privilege check for the `SHOW GRANTS` statement #16168
    - Support the privilege check for the `LOAD DATA` statement #16736
    - Improve the performance of partition pruning when functions related to time and date are used as partition keys #15618
    - Adjust the log level of `dispatch error` from `WARN` to `ERROR` #16232
    - Support the `require-secure-transport` startup option to force clients to use TLS #15415
    - Support HTTP communication between TiDB components when TLS is configured #15419

- Add the `start_ts` information of the current transaction to the `information_schema` ↪ `.processlist` table #16160
- Support automatically reloading the TLS certificate information used for communication among clusters #15162
- Improve the read performance of the partitioned tables by restructuring the partition pruning #15628
- Support the partition pruning feature when `floor(unix_timestamp(a))` is used as the partition expression of the `range` partition table #16521
- Allow executing the `update` statement that contains a `view` and does not update the `view` #16787
- Prohibit creating nested `views` #15424
- Prohibit truncating `view` #16420
- Prohibit using the `update` statement to explicitly update the values of a column when this column is not in the `public` state #15576
- Prohibit starting TiDB when the `status` port is occupied #15466
- Change the character set of the `current_role` function from `binary` to `utf8mb4` #16083
- Improve `max-execution-time` usability by checking the interrupt signal when the data of a new Region is read #15615
- Add the `ALTER TABLE ... AUTO_ID_CACHE` syntax for explicitly setting the cache step of `auto_id` #16287

- TiKV

  - Improve the performance when many conflicts and the `BatchRollback` condition exist in optimistic transactions #7605
  - Fix the issue of decreased performance that occurs because the pessimistic lock `waiter` is frequently awakened when many conflicts exist in pessimistic transactions #7584

- Tools

  - TiDB Lightning

    - Support printing the TiKV cluster mode using the `fetch-mode` sub-command of tidb-lightning-ctl #287

16.14.7.4 Bug Fixes

- TiDB

  - Fix the issue that `WEEKEND` function is not compatible with MySQL when the SQL mode is `ALLOW_INVALID_DATES` #16170
  - Fix the issue that the `DROP INDEX` statement fails to execute when the index column contains the auto-increment primary key #16008

- Fix the issue of incorrect values of the `TABLE_NAMES` column in the Statement Summary #15231
- Fix the issue that some expressions have incorrect results when the plan cache is enabled #16184
- Fix the issue that the result of the `not`/`istrue`/`isfalse` function is incorrect #15916
- Fix the panic caused by the `MergeJoin` operation on tables with redundant indexes #15919
- Fix the issue caused by incorrectly simplifying the link when the predicate only refers to the outer table #16492
- Fix the issue that the `CURRENT_ROLE` function reports an error caused by the `SET ROLE` statement #15569
- Fix the issue that the result of the `LOAD DATA` statement is incompatible with MySQL when this statement encounters `\` #16633
- Fix the issue that the database visibility is incompatible with MySQL #14939
- Fix the issue of incorrect privilege check for the `SET DEFAULT ROLE ALL` statement #15585
- Fix the issue of partition pruning failure caused by the plan cache #15818
- Fix the issue that `schema change` is reported during the transaction commit when concurrent DDL operations are performed on a table and blocking exists, because the transaction does not lock the related table #15707
- Fix the incorrect behavior of `IF(not_int, *, *)` #15356
- Fix the incorrect behavior of `CASE WHEN (not_int)` #15359
- Fix the issue that the `Unknown column` error message is returned when using a `view` that is not in the current schema #15866
- Fix the issue that the result of parsing time strings is incompatible with MySQL #16242
- Fix the possible panic of the collation operator in `left join` when a `null` column exists in the right child node #16528
- Fix the issue that no error message is returned even though the SQL execution is blocked when TiKV keeps returning the `StaleCommand` error message #16528
- Fix the possible panic caused by the port probing when the audit plugin is enabled #15967
- Fix the panic caused when `fast analyze` works on indices only #15967
- Fix the possible panic of the `SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST` statement execution in some cases #16309
- Fix the issue of TiDB OOM caused by specifying a large number of partitions (for example, `9999999999999`) when the hash partition table is created without checking the number of partitions before allocating memory #16218
- Fix the issue of incorrect information of partitioned tables in `information_schema` ↪ `.tidb_hot_table` #16726
- Fix the issue that the partition selection algorithm does not take effect on the hash partitioned table #16070
- Fix the issue that the HTTP API of the MVCC series does not support partitioned tables #16191

- Keep the error handling of the `UNION` statement consistent with that of the `SELECT` statement #16137
- Fix the issue of incorrect behavior when the parameter type of the `VALUES` function is `bit(n)` #15486
- Fix the issue that the processing logic of TiDB is inconsistent with MySQL when the `view` column name is too long. In this case, the system automatically generates a short column name. #14873
- Fix the issue that `(not not col)` is incorrectly optimized as `col` #16094
- Fix the issue of incorrect `range` of the inner table built by `IndexLookupJoin` plans #15753
- Fix the issue that `only_full_group_by` fails to correctly check expressions with brackets #16012
- Fix the issue that an error is returned when the `select view_name.col_name from view_name` statement is executed #15572

- TiKV

    - Fix the issue that the node cannot be deleted correctly after the isolation recovery in some cases #7703
    - Fix the issue of data loss during network isolation caused by the Region Merge operation #7679
    - Fix the issue that learner cannot be removed correctly in some cases #7598
    - Fix the issue that the scanning result of raw key-value pairs might be out of order #7597
    - Fix the issue of reconnection when the batch of Raft messages is too large #7542
    - Fix the issue of gRPC thread deadlock caused by the empty request #7538
    - Fix the issue that the processing logic of restarting the learner is incorrect during the merge process #7457
    - Fix the issue that repeated requests on the cleanup of lock might destroy the atomicity of the transaction #7388

16.14.8 TiDB 3.0.13 Release Notes

Release date: April 22, 2020

TiDB version: 3.0.13

16.14.8.1 Bug Fixes

- TiDB

    - Fix the issue caused by unchecked `MemBuffer` that the `INSERT ... ON DUPLICATE KEY UPDATE` statement might be executed incorrectly within a transaction when users need to insert multiple rows of duplicate data #16690

- TiKV
    - Fix the issue that the system might get stuck and the service is unavailable if `Region Merge` is executed repeatedly #7612

16.14.9 TiDB 3.0.12 Release Notes

Release date: March 16, 2020

TiDB version: 3.0.12

TiDB Ansible version: 3.0.12

> **Warning:**
>
> Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 3.0.x version.

16.14.9.1 Compatibility Changes

- TiDB
    - Fix the issue of inaccurate timing of prewrite binlog in slow query log. The original timing field was called `Binlog_prewrite_time`. After this fix, the name is changed to `Wait_prewrite_binlog_time`. #15276

16.14.9.2 New Features

- TiDB
    - Support dynamic loading of the replaced certificate file by using the `alter instance` statement #15080 #15292
    - Add the `cluster-verify-cn` configuration item. After configuration, the status service can only be used when with the corresponding CN certificate. #15164
    - Add a flow limiting feature for DDL requests in each TiDB server to reduce the error reporting frequency of DDL request conflicts #15148
    - Support exiting of the TiDB server when binlog write fails #15339
- Tools
    - TiDB Binlog
        - Add the `kafka-client-id` configuration item in Drainer, which supports connecting to Kafka clients to configure the client ID #929

16.14.9.3 Bug Fixes

- TiDB
    - Make `GRANT`, `REVOKE` guarantee atomicity when modifying multiple users #15092
    - Fix the issue that the locking of pessimistic lock on the partition table failed to lock the correct row #15114
    - Make the error message display according to the value of `max-index-length` in the configuration when the index length exceeds the limit #15130
    - Fix the incorrect decimal point issue of the `FROM_UNIXTIME` function #15270
    - Fix the issue of conflict detection failure or data index inconsistency caused by deleting records written by oneself in a transaction #15176
- TiKV
    - Fix the issue of conflict detection failure or data index inconsistency caused by inserting an existing key into a transaction and then deleting it immediately when disabling the consistency check parameter #7054
    - Introduce a flow control mechanism in Raftstore to solve the problem that without flow control, it might lead to too slow tracking and cause the cluster to be stuck, and the transaction size might cause frequent reconnection of TiKV connections #7072 #6993
- PD
    - Fix the issue of incorrect Region information caused by data race when PD processes Region heartbeats #2233
- TiDB Ansible
    - Support deploying multiple Grafana/Prometheus/Alertmanager in a cluster #1198

16.14.10 TiDB 3.0.11 Release Notes

Release date: March 4, 2020

TiDB version: 3.0.11

TiDB Ansible version: 3.0.11

> **Warning:**
>
> Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 3.0.x version.

16.14.10.1 Compatibility Changes

- TiDB
  - Add the `max-index-length` configuration item to control the maximum index length, which is compatible with the behavior of TiDB versions before 3.0.7 or of MySQL #15057

16.14.10.2 New Features

- TiDB
  - Support showing the meta information of partitioned tables in the `information_schema.PARTITIONS` table #14849
- TiDB Binlog
  - Support the bidirectional data replication between TiDB clusters #884 #909
- TiDB Lightning
  - Support the TLS configuration #44 #270
- TiDB Ansible
  - Modify the logic of `create_users.yml` so that users of the control machine do not have to be consistent with `ansible_user` #1184

16.14.10.3 Bug Fixes

- TiDB
  - Fix the issue of Goroutine leaks when retrying an optimistic transaction because queries using `Union` are not marked read-only #15076
  - Fix the issue that `SHOW TABLE STATUS` fails to correctly output the table status at the snapshot time because the value of the `tidb_snapshot` parameter is not correctly used when executing the `SET SESSION tidb_snapshot = 'xxx';` statement #14391
  - Fix the incorrect result caused by a SQL statement that contains `Sort Merge Join` and `ORDER BY DESC` at the same time #14664
  - Fix the panic of TiDB server when creating partition tables using the unsupported expression. The error information `This partition function is not allowed` is returned after fixing this panic. #14769
  - Fix the incorrect result occurred when executing the `select max()from subquery` statement with the subquery containing `Union` #14944
  - Fix the issue that an error message is returned when executing the `SHOW BINDINGS` statement after executing `DROP BINDING` that drops the execution binding #14865

- Fix the issue that the connection is broken because the maximum length of an alias in a query is 256 characters in the MySQL protocol, but TiDB does not cut the alias in the query results according to this protocol #14940
- Fix the incorrect query result that might occur when using the string type in `DIV`. For instance, now you can correctly execute the `select 1 / '2007' div 1` statement #14098

- TiKV
  - Optimize the log output by removing unnecessary logs #6657
  - Fix the panic that might occur when the peer is removed under high loads #6704
  - Fix the issue that Hibernate Regions are not waken up in some cases #6732 #6738
- TiDB Ansible
  - Update outdated document links in `tidb-ansible` #1169
  - Fix the issue that undefined variables might occur in the `wait for region replication complete` task #1173

16.14.11 TiDB 3.0.10 Release Notes

Release date: February 20, 2020

TiDB version: 3.0.10

TiDB Ansible version: 3.0.10

> **Warning:**
>
> Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 3.0.x version.

16.14.11.1 TiDB

- Fix wrong `Join` results when `IndexLookUpJoin` uses `OtherCondition` to construct `InnerRange` #14599
- Delete the `tidb_pprof_sql_cpu` configuration item and add the `tidb_pprof_sql_cpu` variable #14416
- Fix the issue that users can query all databases only when they have global privileges #14386
- Fix the issue that data visibility does not meet expectations due to transaction timeout when executing `PointGet` operations #14480
- Change the timing of pessimistic transaction activation to delayed activation, consistent with the optimistic transaction mode #14474

- Fix the incorrect time zone results when the `unixtimestamp` expression calculates the time zone of the table partitions #14476
- Add the `tidb_session_statement_deadlock_detect_duration_seconds` monitoring item to monitor deadlock detection duration #14484
- Fix the system panic issue caused by some logic errors of GC workers #14439
- Correct the expression name of the `IsTrue` function #14516
- Fix the issue that some memory usage is counted inaccurately #14533
- Fix the system panic issue caused by incorrect processing logic during CM-Sketch statistics initialization #14470
- Fix the issue of inaccurate partition pruning when querying partitioned tables #14546
- Fix the issue that the default database name of the SQL statement in SQL bindings is set incorrectly #14548
- Fix the issue that `json_key` is not compatible with MySQL #14561
- Add the feature of automatically updating the statistics of partitioned tables #14566
- Fix the issue that the plan ID changes when the `PointGet` operation is executed (the plan ID is expected to be `1` always) #14595
- Fix the system panic issue caused by incorrect processing logic when SQL bindings do not match exactly #14263
- Add the `tidb_session_statement_pessimistic_retry_count` monitoring item to monitor the number of retries after the failure to lock pessimistic transactions #14619
- Fix the incorrect privilege check for `show binding` statements #14618
- Fix the issue that the query cannot be killed because the `backoff` logic does not include checking the `killed` tag #14614
- Improve the performance of statement summary by reducing the time to hold internal locks #14627
- Fix the issue that TiDB's result of parsing strings to time is incompatible with MySQL #14570
- Record the user login failures in audit logs #14620
- Add the `tidb_session_ statement_lock_keys_count` monitoring item to monitor the number of lock keys for pessimistic transactions #14634
- Fix the issue that characters in JSON such as `&`, `<`, and `>` are incorrectly escaped #14637
- Fix the system panic issue caused by excessive memory usage when the `HashJoin` operation is building a hash table #14642
- Fix the panic issue caused by incorrect processing logic when an SQL binding processes illegal records #14645
- ix a MySQL incompatibility issue by adding Truncated error detection to decimal division calculation #14673
- Fix the issue of successfully granting users privileges on a table that does not exist #14611

16.14.11.2 TiKV

- Raftstore

- Fix the system panic issue #6460 or data loss issue #598 caused by Region merge failure #6481
- Support `yield` to optimize scheduling fairness, and support pre-transfering the leader to improve leader scheduling stability #6563

16.14.11.3 PD

- Fix the invalid cache issue by supporting automatically updating the Region cache information when the system traffic changes #2103
- Use leader lease time to determine TSO service validity #2117

16.14.11.4 Tools

- TiDB Binlog
  - Support relay log in Drainer #893
- TiDB Lightning
  - Make some configuration items use default values when a config file is missing #255
  - Fix the issue that the web interface cannot be opened in the non-server mode #259

16.14.11.5 TiDB Ansible

- Fix the issue that the command execution fails due to the failure to obtain PD leader in some scenarios #1121
- Add the `Deadlock Detect Duration` monitoring item in the TiDB dashboard #1127
- Add the `Statement Lock Keys Count` monitoring item in the TiDB dashboard #1132
- Add the `Statement Pessimistic Retry Count` monitoring item in the TiDB dashboard #1133

16.14.12 TiDB 3.0.9 Release Notes

Release date: January 14, 2020

TiDB version: 3.0.9

TiDB Ansible version: 3.0.9

> **Warning:**
>
> Some known issues are found in this version, and these issues are fixed in new versions. It is recommended that you use the latest 3.0.x version.

16.14.12.1 TiDB

- Executor
  - Fix the incorrect result when the aggregate function is applied to the `ENUM` column and the collection column #14364
- Server
  - Support the `auto_increment_increment` and `auto_increment_offset` system variables #14396
  - Add the `tidb_tikvclient_ttl_lifetime_reach_total` monitoring metric to monitor the number of pessimistic transactions with a TTL of 10 minutes #14300
  - Output the SQL information in the log when the SQL query causes a panic during its execution #14322
  - Add the `plan` and `plan_digest` fields in the statement summary table to record the `plan` that is being executed and the `plan` signature #14285
  - Adjust the default value of the `stmt-summary.max-stmt-count` configuration item from `100` to `200` #14285
  - Add the `plan_digest` field in the slow query table to record the `plan` signature #14292
- DDL
  - Fix the issue that the results of anonymous indexes created using `alter table ... add index` on the `primary` column is inconsistent with MySQL #14310
  - Fix the issue that `VIEW`s are mistakenly dropped by the `drop table` syntax #14052
- Planner
  - Optimize the performance of statements such as `select max(a), min(a) from t`. If an index exists in the `a` column, the statement is optimized to `select * from (select a from t order by a desc limit 1)as t1, (select a from t order by a limit 1)as t2` to avoid full table scan #14410

16.14.12.2 TiKV

- Raftstore
  - Speed up the configuration change to speed up the Region scattering #6421
- Transaction
  - Add the `tikv_lock_manager_waiter_lifetime_duration`, `tikv_lock_manager_detect_dura` , and `tikv_lock_manager_detect_duration` monitoring metrics to monitor `waiter`s' lifetime, the time cost of detecting deadlocks, and the status of `Wait` table #6392

- Optimize the following configuration items to reduce transaction execution latency caused by changing Region leader or the leader of deadlock detector in extreme situations #6429
  - Change the default value of `wait-for-lock-time` from `3s` to `1s`
  - Change the default value of `wake-up-delay-duration` from `100ms` to `20ms`
- Fix the issue that the leader of the deadlock detector might be incorrect during the Region Merge process #6431

16.14.12.3 PD

- Support using backlash `/` in the location label name #2083
- Fix the incorrect statistics because the tombstone store is mistakenly included by the label counter #2067

16.14.12.4 Tools

- TiDB Binlog
  - Add the unique key information in the binlog protocol output by Drainer #862
  - Support using the encrypted password for database connection for Drainer #868

16.14.12.5 TiDB Ansible

- Support automatically creating directories to optimize the deployment of TiDB Lightning #1105

16.14.13 TiDB 3.0.8 Release Notes

Release date: December 31, 2019

TiDB version: 3.0.8

TiDB Ansible version: 3.0.8

16.14.13.1 TiDB

- SQL Optimizer
  - Fix the wrong SQL binding plan caused by untimely cache updates #13891
  - Fix the issue that the SQL binding might be invalid when an SQL statement contains a symbol list #14004
  - Fix the issue that an SQL binding cannot be created or deleted because an SQL statement ends with `;` #14113

- Fix the issue that a wrong SQL query plan might be selected because the `PhysicalUnionScan` operator sets wrong statistics #14133
- Remove the `minAutoAnalyzeRatio` restriction to make `autoAnalyze` more timely #14015

- SQL Execution Engine
  - Fix issues that the `INSERT/REPLACE/UPDATE ... SET ... = DEFAULT` syntax might report an error and combining the usage of the `DEFAULT` expression with a virtual generated column might report an error #13682
  - Fix the issue that the `INSERT` statement might report an error when converting a string to a float #14011
  - Fix the issue that sometimes the aggregate operation is low effective because the concurrency value of the `HashAgg` executor is incorrectly initialized #13811
  - Fix the issue that an error is reported in the execution of`group by item` when the clause is in the parentheses #13658
  - Fix the issue that the execution of `OUTER JOIN` might report an error because TiDB incorrectly calculates `group by item` #14014
  - Fix the issue that the error message is inaccurate when Range-exceeding data is written into Range partitioned tables #14107
  - Revert PR #10124 and cancel the `PadCharToFullLength` effect to avoid unexpected query results in special cases, considering that MySQL 8 will discard `PadCharToFullLength` soon #14157
  - Fix the goroutine leak issue when executing the `EXPLAIN ANALYZE` statement caused by unguaranteed `close()` calling in `ExplainExec` #14226

- DDL
  - Optimize the error message output of `change column`/`modify column` to make it easier to understand #13796
  - Add the `SPLIT PARTITION TABLE` syntax to support splitting Regions for partitioned tables #13929
  - Fix the issue that the index length exceeds 3072 bytes and no error is reported because the index length is incorrectly checked when an index is created #13779
  - Fix the issue that the `GC life time is shorter than transaction duration` error message might be reported because it takes too much time to add an index in partitioned tables #14132
  - Fix the panic when `SELECT * FROM information_schema.KEY_COLUMN_USAGE` is executed because the foreign key is not checked when `DROP COLUMN`/`MODIFY COLUMN`/`CHANGE COLUMN` is executed #14105

- Server
  - Statement Summary improvements:
    - Add a large number of SQL metric fields to facilitate analyzing SQL statements in more detail #14151, #14168

  - Add the `stmt-summary.refresh-interval` parameter to control whether to move the stale data from the `events_statements_summary_by_digest` table to the `events_statements_summary_by_digest_history` table (the default interval: 30 minutes) #14161
  - Add the `events_statements_summary_by_digest_history` table to save the stale data in `events_statements_summary_by_digest` #14166
- Fix the issue that the binlog is incorrectly output when RBAC-related internal SQL statements are executed #13890
- Add the `server-version` configuration item to control the feature of modifying the TiDB server version #13906
- Add the feature of using the HTTP interface to recover writing the TiDB binlog #13892
- Update the privilege required by `GRANT roles TO user` from `GrantPriv` to `ROLE_ADMIN` or `SUPER`, to keep consistency with the MySQL behavior #13932
- Modify the TiDB behavior from using the current database to reporting the `No database selected` error when the `GRANT` statement does not specify a database name, to keep compatibility with the MySQL behavior #13784
- Modify the execution privilege for the `REVOKE` statement from `SuperPriv` to `REVOKE` being executable only if the user has the privilege for the corresponding schema, to keep consistency with the MySQL behavior #13306
- Fix the issue that `GrantPriv` is mistakenly granted to the target user when the `GRANT ALL` syntax does not contain `WITH GRANT OPTION` #13943
- Fix the issue that the error message does not contain the cause for the `LOAD DATA` statement's wrong behavior when `LoadDataInfo` fails to call `addRecord` #13980
- Fix the issue that wrong slow query information is output because multiple SQL statements in a query share the same `StartTime` #13898
- Fix the issue that the memory might leak when `batchClient` processes a large transaction #14032
- Fix the issue that `system_time_zone` is always displayed as `CST` and now TiDB's `system_time_zone` is obtained from `systemTZ` in the `mysql.tidb` table #14086
- Fix the issue that the `GRANT ALL` syntax does not grant all privileges to the user #14092
- Fix the issue that the `Priv_create_user` privilege is invalid for `CREATE ROLE` and `DROP ROLE` #14088
- Modify the error code of `ErrInvalidFieldSize` from `1105(Unknow Error)` to `3013` #13737
- Add the `SHUTDOWN` command to stop a TiDB server and add the `ShutdownPriv` privilege #14104
- Fix the atomicity issue for the `DROP ROLE` statement to avoid some roles being deleted unexpectedly when TiDB fails to execute a statement #14130
- Fix the issue that the `tidb_enable_window_function` in the `SHOW VARIABLE` result incorrectly outputs `1` when a TiDB version is upgraded to 3.0, and replace the wrong result with `0` #14131
- Fix the issue that the goroutine might leak because `gcworker` continuously retries

when the TiKV node is offline #14106
- Record the binlog `Prewrite` time in the slow query log to improve the usability for issue tracking #14138
- Make the `tidb_enable_table_partition` variable support `GLOBAL SCOPE` #14091
- Fix the issue that the user privilege might be missing or mistakenly added because the newly added privilege is not correctly granted to the corresponding user when a new privilege is added #14178
- Fix the issue that the `CheckStreamTimeoutLoop` goroutine might leak because `rpcClient` does not close when the TiKV server is disconnected #14227
- Support certificate-based authentication (User document) #13955

- Transaction
  - Update the default value of the `tidb_txn_mode` variable from `""` to `"pessimistic ↪ "` when a new cluster is created #14171
  - Fix the issue that the lock waiting time is too long for a pessimistic transaction because the lock waiting time for a single statement is not reset when a transaction is retried #13990
  - Fix the issue that wrong data might be read because unmodified data is unlocked for the pessimistic transaction mode #14050
  - Fix repeated insert value restriction checks because transaction types are not distinguished when prewrite is performed in mocktikv #14175
  - Fix the panic because transactions are not correctly handled when `session. ↪ TxnState` is `Invalid` #13988
  - Fix the issue that the `ErrConfclit` structure in mocktikv does not contain `ConflictCommitTS` #14080
  - Fix the issue that the transaction is blocked because TiDB does not correctly check lock timeout after resolving the lock #14083

- Monitor
  - Add the `pessimistic_lock_keys_duration` monitoring item in `LockKeys` #14194

16.14.13.2 TiKV

- Coprocessor
  - Modify the level of the output log from `error` to `warn` when an error occurs in Coprocessor #6051
  - Modify the update behavior of statistics sampling data from directly updating the row to deleting before inserting, to keep consistency with the update behavior of tidb-server #6069

- Raftstore

    - Fix the panic caused by repeatedly sending the `destroy` message to `peerfsm` and `peerfsm` being destroyed multiple times #6297
    - Update the default value of `split-region-on-table` from `true` to `false` to disable splitting Regions by table by default #6253
- Engine
    - Fix the issue that empty data might be returned because RocksDB iterator errors are not correctly processed in extreme conditions #6326
- Transaction
    - Fix the issue that TiKV fails to write data into keys and GC is blocked because the pessimistic locks are incorrectly cleaned up #6354
    - Optimize the pessimistic lock waiting mechanism to improve the performance in scenarios where the lock conflict is severe #6296
- Update the default value of `tikv_alloc` from `tikv_alloc/default` to `jemalloc` #6206

16.14.13.3 PD

- Client
    - Support using `context` to create a client and setting the timeout duration when creating a new client #1994
    - Support creating the `KeepAlive` connection #2035
- Optimize the performance for the `/api/v1/regions` API #1986
- Fix the issue that deleting stores in a `tombstone` state might cause a panic #2038
- Fix the issue that overlapped Regions are mistakenly deleted when loading the Region information from disks #2011, #2040
- Upgrade etcd from v3.4.0 to v3.4.3 (note that after upgrading you can only degrade etcd using pd-recover) #2058

16.14.13.4 Tools

- TiDB Binlog
    - Fix the issue that the binlog is ignored because Pump does not receive the DDL committed binlog #853

16.14.13.5 TiDB Ansible

- Revert the simplified configuration item #1053
- Optimize the logic for checking the TiDB version when performing a rolling update #1056

- Upgrade TiSpark to v2.1.8 #1061
- Fix the issue that the PD role monitoring item is wrongly displayed on Grafana #1065
- Optimize `Thread Voluntary Context Switches` and `Thread Nonvoluntary Context Switches` monitoring items on the TiKV Detail page on Grafana #1071

16.14.14 TiDB 3.0.7 Release Notes

Release date: December 4, 2019

TiDB version: 3.0.7

TiDB Ansible version: 3.0.7

16.14.14.1 TiDB

- Fix the issue that the lock TTL's value is too large because the TiDB server's local time is behind PD's timestamp #13868
- Fix the issue that the timezone is incorrect after parsing the date from strings using `gotime.Local` #13793
- Fix the issue that the result might be incorrect because the `binSearch` function does not return an error in the implementation of `builtinIntervalRealSig` #13767
- Fix the issue that data is incorrect because the precision is lost when an integer is converted to an unsigned floating point or decimal type #13755
- Fix the issue that the result is incorrect because the `not null` flag is not properly reset when the `USING` clause is used in Natural Outer Join and Outer Join #13739
- Fix the issue that the statistics are not accurate because a data race occurs when statistics are updated #13687

16.14.14.2 TiKV

- Make the deadlock detector only observe valid Regions to make sure the deadlock manager is in a valid Region #6110
- Fix a potential memory leak issue #6128

16.14.15 TiDB 3.0.6 Release Notes

Release date: November 28, 2019

TiDB version: 3.0.6

TiDB Ansible version: 3.0.6

16.14.15.1 TiDB

- SQL Optimizer
  - Fix the issue that the result is incorrect after the window function AST restores SQL text, for example, `over w` being mistakenly restored to `over (w)` #12933
  - Fix the issue of pushing down `STREAM AGG()` to `doubleRead` #12690
  - Fix the issue that quotes are incorrectly handled for SQL binding #13117
  - Optimize the `select max(_tidb_rowid)from t` scenario to avoid full table scans #13095
  - Fix the issue that the query result is incorrect when the query statement contains a variable assignment expression #13231
  - Fix the issue that the result is incorrect when the `UPDATE` statement contains both a sub-query and a generated column; fix the `UPDATE` statement execution error when this statement contains two same-named tables from different source databases #13350
  - Support `_tidb_rowid` for point queries #13416
  - Fix the issue that the generated query execution plan is incorrect, caused by incorrect usage of partitioned table statistics #13628
- SQL Execution Engine
  - Fix the issue that TiDB is incompatible with MySQL when handling invalid values of the year type #12745
  - Reuse `Chunk` in the `INSERT ON DUPLICATE UPDATE` statement to reduce the memory overhead #12998
  - Add the support for the `JSON_VALID` built-in function #13133
  - Support executing `ADMIN CHECK TABLE` on partitioned tables #13140
  - Fix the panic issue when `FAST ANALYZE` is executed on empty tables #13343
  - Fix the panic issue when executing `FAST ANALYZE` on an empty table that contains multi-column indexes #13394
  - Fix the issue that the estimated number of rows is greater than 1 when the `WHERE` clause contains an equal condition on the unique key #13382
  - Fix the issue that the returned data might be duplicated when `Streaming` is enabled in TiDB #13254
  - Extract the top N values from the count-min sketch to improve the estimation accuracy #13429
- Server
  - Make requests sent to TiKV fail quickly when the gRPC dial times out #12926
  - Add the following virtual tables: #13009
    - `performance_schema.tidb_profile_allocs`
    - `performance_schema.tidb_profile_block`
    - `performance_schema.tidb_profile_cpu`
    - `performance_schema.tidb_profile_goroutines`

- Fix the issue that the `kill` command does not work when the query is waiting for pessimistic locking #12989
- Do not do asynchronous rollback when acquiring pessimistic locking fails and the transaction only involves modifying a single key #12707
- Fix the panic issue when the response for the request of splitting Regions is empty #13092
- Avoid unnecessary backoff when `PessimisticLock` returns a locking error #13116
- Modify the TiDB behavior for checking configurations by printing a warning log for unrecognized configuration option #13272
- Support obtaining the binlog status of all TiDB nodes via the `/info/all` interface #13187
- Fix the issue that goroutine might leak when TiDB kills connections #13251
- Make the `innodb_lock_wait_timeout` parameter work in pessimistic transactions to control the lock wait timeout for pessimistic locking #13165
- Stop updating pessimistic transaction TTL when pessimistic transactional queries are killed to prevent other transactions from waiting unnecessarily #13046

- DDL
  - Fix the issue that the execution result of `SHOW CREATE VIEW` in TiDB is inconsistent with that in MySQL #12912
  - Support creating `View` based on `union`, for example, `create view v as select * from t1 union select * from t2` #12955
  - Add more transaction-related fields for the `slow_query` table: #13072
    - `Prewrite_time`
    - `Commit_time`
    - `Get_commit_ts_time`
    - `Commit_backoff_time`
    - `Backoff_types`
    - `Resolve_lock_time`
    - `Local_latch_wait_time`
    - `Write_key`
    - `Write_size`
    - `Prewrite_region`
    - `Txn_retry`
  - Use the table's `COLLATE` instead of the system's default charset in the column when a table is created and the table contains `COLLATE` #13174
  - Limit the length of the index name when creating a table #13310
  - Fix the issue that the table name length is not checked when a table is renamed #13346
  - Add the `alter-primary-key` configuration (disabled by default) to support adding/dropping the primary key in TiDB #13522

16.14.15.2 TiKV

- Fix the issue that the `acquire_pessimistic_lock` interface returns a wrong `txn_size` #5740
- Limit the writes for GC worker per second to reduce the impact on the performance #5735
- Make `lock_manager` accurate #5845
- Support `innodb_lock_wait_timeout` for pessimistic locking #5848
- Add the configuration check for Titan #5720
- Support using tikv-ctl to dynamically modify the GC I/O limit: `tikv-ctl --host=ip:port modify-tikv-config -m server -n gc.max_write_bytes_per_sec -v 10MB` #5957
- Reduce useless `clean up` requests to decrease the pressure on the deadlock detector #5965
- Avoid reducing TTL in pessimistic locking prewrite requests #6056
- Fix the issue that a missing blob file might occur in Titan #5968
- Fix the issue that `RocksDBOptions` might not take effect in Titan #6009

16.14.15.3 PD

- Add an `ActOn` dimension for each filter to indicate that each scheduler and checker is affected by the filter, and delete two unused filters: `disconnectFilter` and `rejectLeaderFilter` #1911
- Print a warning log when it takes more than 5 milliseconds to generate a timestamp in PD #1867
- Lower the client log level when passing unavailable endpoint to the client #1856
- Fix the issue that the gRPC message package might exceed the maximum size in the `region_syncer` replication process #1952

16.14.15.4 Tools

- TiDB Binlog
  - Obtain the initial replication timestamp from PD when `initial-commit-ts` is set to "-1" in Drainer #788
  - Decouple Drainer's `Checkpoint` storage from the downstream and support saving `Checkpoint` in MySQL or local files #790
  - Fix the Drainer panic issue caused by using empty values when configuring replication database/table filtering #801
  - Fix the issue that processes get into the deadlock status instead of exiting after a panic occurs because Drainer fails to apply binlog files to the downstream #807
  - Fix the issue that Pump blocks when it exits because of gRPC's `GracefulStop` #817
  - Fix the issue that Drainer fails when it receives a binlog which misses a column during the execution of a `DROP COLUMN` statement in TiDB (v3.0.6 or later) #827
- TiDB Lightning

- Add the `max-allowed-packet` configuration (64 M by default) for the TiDB backend #248

16.14.16 TiDB 3.0.5 Release Notes

Release date: October 25, 2019

TiDB version: 3.0.5

TiDB Ansible version: 3.0.5

16.14.16.1 TiDB

- SQL Optimizer
  - Support boundary checking on Window Functions #12404
  - Fix the issue that `IndexJoin` on the partition table returns incorrect results #12712
  - Fix the issue that the `ifnull` function on the top of the outer join `Apply` operator returns incorrect results #12694
  - Fix the issue of update failure when a subquery was included in the `where` condition of `UPDATE` #12597
  - Fix the issue that outer join was incorrectly converted to inner join when the `cast` function was included in the query conditions #12790
  - Fix incorrect expression passing in the join condition of `AntiSemiJoin` #12799
  - Fix the statistics error caused by shallow copy when initializing statistics #12817
  - Fix the issue that the `str_to_date` function in TiDB returns a different result from MySQL when the date string and the format string do not match #12725
- SQL Execution Engine
  - Fix the panic issue when the `from_unixtime` function handles null #12551
  - Fix the `invalid list index` error reported when canceling DDL jobs #12671
  - Fix the issue that arrays were out of bounds when Window Functions are used #12660
  - Improve the behavior of the `AutoIncrement` column when it is implicitly allocated, to keep it consistent with the default mode of MySQL auto-increment locking ("consecutive" lock mode): for the implicit allocation of multiple `AutoIncrement` IDs in a single-line `Insert` statement, TiDB guarantees the continuity of the allocated values. This improvement ensures that the JDBC `getGeneratedKeys()` method will get the correct results in any scenario. #12602
  - Fix the issue that the query is hanged when `HashAgg` serves as a child node of `Apply` #12766
  - Fix the issue that the `AND` and `OR` logical expressions return incorrect results when it comes to type conversion #12811
- Server

- Implement the interface function that modifies transaction TTL to help support large transactions later #12397
- Support extending the transaction TTL as needed (up to 10 minutes) to support pessimistic transactions #12579
- Adjust the number of times that TiDB caches schema changes and corresponding changed table information from 100 to 1024, and support modification by using the `tidb_max_delta_schema_count` system variable #12502
- Update the behavior of the `kvrpc.Cleanup` protocol to no longer clean locks of transactions that are not overtime #12417
- Support logging Partition table information to the `information_schema.tables` table #12631
- Support modifying the TTL of Region Cache by configuring `region-cache-ttl` #12683
- Support printing the execution plan compression-encoded information in the slow log. This feature is enabled by default and can be controlled by using the `slow-log-plan` configuration or the `tidb_record_plan_in_slow_log` variable. In addition, the `tidb_decode_plan` function can decode the execution plan column encoded information in the slow log into execution plan information. #12808
- Support displaying memory usage information in the `information_schema.processlist` table #12801
- Fix the issue that an error and an unexpected alarm might occur when the TiKV Client judges an idle connection #12846
- Fix the issue that the `INSERT IGNORE` statement performance is decreased because `tikvSnapshot` does not properly cache the KV results of `BatchGet()` #12872
- Fix the issue that the TiDB response speed was relatively low because of slow connection to some KV services #12814

- DDL
  - Fix the issue that the `Create Table` operation does not correctly set the Int type default value for the Set column #12267
  - Support multiple `uniques` when creating a unique index in the `Create Table` statement #12463
  - Fix the issue that populating the default value of this column for existing rows might cause an error when adding a Bit type column using `Alter Table` #12489
  - Fix the failure of adding a partition when the Range partitioned table uses a Date or Datetime type column as the partitioning key #12815
  - Support checking the consistency of the partition type and the partition key type when creating a table or adding a partition, for the Range partitioned table with the Date or Datetime type column as the partition key #12792
  - Add a check that the Unique Key column set needs to be greater than or equal to the partitioned column set when creating a Range partitioned table #12718

- Monitor
  - Add the monitoring metrics of Commit and Rollback operations to the `Transaction OPS` dashboard #12505

- Add the monitoring metrics of `Add Index` operation progress #12390

16.14.16.2 TiKV

- Storage
  - Add a new feature of pessimistic transactions: the transaction cleanup interface supports only cleaning up locks whose TTL is outdated #5589
  - Fix the issue that Rollback of the transaction Primary key is collapsed #5646, #5671
  - Fix the issue that under pessimistic locks, point queries might return the previous version data #5634
- Raftstore
  - Reduce message flush operations in Raftstore to improve performance and reduce CPU usage #5617
  - Optimize the cost of obtaining the Region size and estimated number of keys, to reduce heartbeat overhead and CPU usage #5620
  - Fix the issue that Raftstore prints an error log and encounters a panic when getting invalid data #5643
- Engine
  - Enable RocksDB `force_consistency_checks` to improve data safety #5662
  - Fix the issue that concurrent flush operations in Titan might cause data loss #5672
  - Update the rust-rocksdb version to avoid the issue of TiKV crash and restart caused by intra-L0 compaction #5710

16.14.16.3 PD

- Improve the precision of storage occupied by Regions #1782
- Improve the output of the `--help` command #1763
- Fix the issue that the HTTP request fails to redirect after TLS is enabled #1777
- Fix the panic issue occurred when pd-ctl uses the `store shows limit` command #1808
- Improve readability of label monitoring metrics and reset the original leader's monitoring data when the leader switches, to avoid false reports #1815

16.14.16.4 Tools

- TiDB Binlog
  - Fix the issue that `ALTER DATABASE` related DDL operations cause Drainer to exit abnormally #769

- Support querying the transaction status information for Commit binlog to improve replication efficiency #757
- Fix the issue that a Pump panic might occur when Drainer's `start_ts` is greater than Pump's largest `commit_ts` #758

- TiDB Lightning
  - Integrate the full logic import feature of Loader and support configuring the backend mode #221

16.14.16.5 TiDB Ansible

- Add the monitoring metrics of adding index speed #986
- Simplify the configuration file content and remove parameters that users do not need to configure #1043c, #998
- Fix the monitoring expression error of performance read and performance write #e90e7
- Update the monitoring display method and the alarm rules of Raftstore CPU usage #992
- Update the TiKV CPU monitoring item in the Overview monitoring dashboard to filter out the excess monitoring content #1001

16.14.17 TiDB 3.0.4 Release Notes

Release date: October 8, 2019

TiDB version: 3.0.4

TiDB Ansible version: 3.0.4

- New features
  - Add the `performance_schema.events_statements_summary_by_digest` system table to troubleshoot performance issues at the SQL level
  - Add the `WHERE` clause in TiDB's `SHOW TABLE REGIONS` syntax
  - Add the `worker-count` and `txn-batch` configuration items in Reparo to control the recovery speed

- Improvements
  - Support batch Region split command and empty split command in TiKV to improve split performance
  - Support double linked list for RocksDB in TiKV to improve performance of reverse scan
  - Add two perf tools `iosnoop` and `funcslower` in TiDB Ansible to better diagnose the cluster state
  - Optimize the output of slow query logs in TiDB by deleting redundant fields

- Changed behaviors

    - Update the default value of `txn-local-latches.enable` to `false` to disable the default behavior of checking conflicts of local transactions in TiDB
    - Add the `tidb_txn_mode` system variable of global scope in TiDB and allow using the pessimistic lock; note that TiDB still adopts the optimistic lock by default
    - Replace the `Index_ids` field in TiDB slow query logs with `Index_names` to improve the usability of slow query logs
    - Add the `split-region-max-num` parameter in the TiDB configuration file to modify the maximum number of Regions allowed in the `SPLIT TABLE` syntax
    - Return the `Out Of Memory Quota` error instead of disconnecting the link when a SQL execution exceeds the memory limit
    - Disallow dropping the `AUTO_INCREMENT` attribute of columns in TiDB to avoid misoperations. To drop this attribute, change the `tidb_allow_remove_auto_inc` system variable

- Fixed issues

    - Fix the issue that the uncommented TiDB-specific syntax `PRE_SPLIT_REGIONS` might cause errors in the downstream database during data replication
    - Fix the issue in TiDB that the slow query logs are incorrect when getting the result of `PREPARE` + `EXECUTE` by using the cursor
    - Fix the issue in PD that adjacent small Regions cannot be merged
    - Fix the issue in TiKV that file descriptor leak in idle clusters might cause TiKV processes to exit abnormally when the processes run for a long time

- Contributors

    Our thanks go to the following contributors from the community for helping this release:

    - sduzh
    - lizhenda

16.14.17.1 TiDB

- SQL Optimizer

    - Fix the issue that invalid query ranges might be resulted when splitted by feedback #12170
    - Display the returned error of the `SHOW STATS_BUCKETS` statement in hexadecimal rather than return errors when the result contains invalid Keys #12094
    - Fix the issue that when a query contains the `SLEEP` function (for example, `select 1 from (select sleep(1))t;`), column pruning causes invalid `sleep(1)` during query #11953
    - Use index scan to lower IO when a query only concerns the number of columns rather than the table data #12112

- Do not use any index when no index is specified in `use index()` to be compatible with MySQL #12100
- Strictly limit the number of `TopN` records in the `CMSketch` statistics to fix the issue that the `ANALYZE` statement fails because the statement count exceeds TiDB's limit on the size of a transaction #11914
- Fix the error occurred when converting the subqueries contained in the `Update` statement #12483
- Optimize execution performance of the `select ... limit ... offset ...` statement by pushing the Limit operator down to the `IndexLookUpReader` execution logic #12378

- SQL Execution Engine
  - Print the SQL statement in the log when the `PREPARED` statement is incorrectly executed #12191
  - Support partition pruning when the `UNIX_TIMESTAMP` function is used to implement partitioning #12169
  - Fix the issue that no error is reported when `AUTO_INCREMENT` incorrectly allocates `MAX int64` and `MAX uint64` #12162
  - Add the `WHERE` clause in the `SHOW TABLE ⋯ REGIONS` and `SHOW TABLE .. INDEX ⋯ REGIONS` syntaxes #12123
  - Return the `Out Of Memory Quota` error instead of disconnecting the link when a SQL execution exceeds the memory limit #12127
  - Fix the issue that incorrect result is returned when `JSON_UNQUOTE` function handles JSON text #11955
  - Fix the issue that `LAST INSERT ID` is incorrect when assigning values to the `AUTO_INCREMENT` column in the first row (for example, `insert into t (pk, c)values (1, 2), (NULL, 3)`) #12002
  - Fix the issue that the `GROUPBY` parsing rule is incorrect in the `PREPARE` statement #12351
  - Fix the issue that the privilege check is incorrect in the point queries #12340
  - Fix the issue that the duration by `sql_type` for the `PREPARE` statement is not shown in the monitoring record #12331
  - Support using aliases for tables in the point queries (for example, `select * from t tmp where a = "aa"`) #12282
  - Fix the error occurred when not handling negative values as unsigned when inserting negative numbers into BIT type columns #12423
  - Fix the incorrectly rounding of time (for example, `2019-09-11 11:17:47.999999666` should be rounded to `2019-09-11 11:17:48`.) #12258
  - Refine the usage of expression blocklist (for example, `<` is equivalent to `lt`.) #11975
  - Add the database prefix to the message of non-existing function error (for example, `[expression:1305]FUNCTION test.std_samp does not exist`) #12111

- Server
  - Add the `Prev_stmt` field in slow query logs to output the previous statement

when the last statement is `COMMIT` #12180
- Optimize the output of slow query logs by deleting redundant fields #12144
- Update the default value of `txn-local-latches.enable` to `false` to disable the default behavior of checking conflicts of local transactions in TiDB #12095
- Replace the `Index_ids` field in TiDB slow query logs with `Index_names` to improve the usability of slow query logs #12061
- Add the `tidb_txn_mode` system variable of global scope in TiDB and allow using pessimistic lock #12049
- Add the `Backoff` field in the slow query logs to record the Backoff information in the commit phase of 2PC #12335
- Fix the issue that the slow query logs are incorrect when getting the result of `PREPARE` + `EXECUTE` by using the cursor (for example, `PREPARE stmt1FROM ↪ SELECT * FROM t WHERE a > ?; EXECUTE stmt1 USING @variable`) #12392
- Support `tidb_enable_stmt_summary`. When this feature is enabled, TiDB counts the SQL statements and the result can be queried by using the system table `performance_schema.events_statements_summary_by_digest` #12308
- Adjust the level of some logs in tikv-client (for example, change the log level of `batchRecvLoop fails` from `ERROR` to `INFO`) #12383

- DDL
  - Add the `tidb_allow_remove_auto_inc` variable. Dropping the `AUTO INCREMENT` attribute of the column is disabled by default #12145
  - Fix the issue that the uncommented TiDB-specific syntax `PRE_SPLIT_REGIONS` might cause errors in the downstream database during data replication #12120
  - Add the `split-region-max-num` variable in the configuration file so that the maximum allowable number of Regions is adjustable #12097
  - Support splitting a Region into multiple Regions and fix the timeout issue during Region scatterings #12343
  - Fix the issue that the `drop index` statement fails when the index that contains an `AUTO_INCREMENT` column referenced by two indexes #12344
- Monitor
  - Add the `connection_transient_failure_count` monitoring metrics to count the number of gRPC connection errors in `tikvclient` #12093

16.14.17.2 TiKV

- Raftstore
  - Fix the issue that Raftstore inaccurately counts the number of keys in empty Regions #5414
  - Support double linked list for RocksDB to improve the performance of reverse scan #5368

    - Support batch Region split command and empty split command to improve split performance #5470

- Server

    - Fix the issue that the output format of the `-V` command is not consistent with the format of 2.X #5501
    - Upgrade Titan to the latest version in the 3.0 branch #5517
    - Upgrade grpcio to v0.4.5 #5523
    - Fix the issue of gRPC coredump and support shared memory to avoid OOM #5524
    - Fix the issue in TiKV that file descriptor leak in idle clusters might cause TiKV processes to exit abnormally when the processes run for a long time #5567

- Storage

    - Support the `txn_heart_beat` API to make the pessimistic lock in TiDB consistent with that in MySQL as much as possible #5507
    - Fix the issue that the performance of point queries is low in some situations #5495 #5463

16.14.17.3 PD

- Fix the issue that adjacent small Regions cannot be merged #1726
- Fix the issue that the TLS enabling parameter in `pd-ctl` is invalid #1738
- Fix the thread-safety issue that the PD operator is accidentally removed #1734
- Support TLS for Region syncer #1739

16.14.17.4 Tools

- TiDB Binlog

    - Add the `worker-count` and `txn-batch` configuration items in Reparo to control the recovery speed #746
    - Optimize the memory usage of Drainer to enhance the efficiency of simultaneous execution #737

- TiDB Lightning

    - Fix the issue that re-importing data from checkpoint might cause TiDB Lightning to panic #237
    - Optimize the algorithm of `AUTO_INCREMENT` to reduce the risk of overflowing `AUTO_INCREMENT` columns #227

16.14.17.5 TiDB Ansible

- Upgrade TiSpark to v2.2.0 #926
- Update the default value of the TiDB configuration item `pessimistic_txn` to `true` #933
- Add more system-level monitoring metrics to `node_exporter` #938
- Add two perf tools `iosnoop` and `funcslower` in TiDB Ansible to better diagnose the cluster state #946
- Replace the raw module to shell module to address the long waiting time in such situations as the password expires #949
- Update the default value of the TiDB configuration item `txn_local_latches` to `false`
- Optimize the monitoring metrics and alert rules of Grafana dashboard #962 #963 #969
- Check the configuration file before the deployment and upgrade #934 #972

16.14.18 TiDB 3.0.3 Release Notes

Release date: August 29, 2019

TiDB version: 3.0.3

TiDB Ansible version: 3.0.3

16.14.18.1 TiDB

- SQL Optimizer
  - Add the `opt_rule_blacklist` table to disable logic optimization rules such as `aggregation_eliminate` and `column_prune` #11658
  - Fix the issue that incorrect results might be returned for `Index Join` when the join key uses a prefix index or an unsigned index column that is equal to a negative value #11759
  - Fix the issue that `"` or `\` in the `SELECT` statements of `create ··· binding ...` might result in parsing errors #11726
- SQL Execution Engine
  - Fix the issue that type errors in the return value might occur when the Quote function handles a null value #11619
  - Fix the issue that incorrect results for `ifnull` might be returned when Max/Min is used for type inferring with `NotNullFlag` retained #11641
  - Fix the potential error that occurs when comparing bit type data in string form #11660
  - Decrease the concurrency for data that requires sequential read to lower the possibility of OOM #11679

- Fix the issue that incorrect type inferring might be caused when multiple parameters are unsigned for some built-in functions (for example, `if` and `coalesce`) #11621
- Fix the incompatibility with MySQL when the `Div` function handles unsigned decimal types #11813
- Fix the issue that panic might occur when executing SQL statements that modify the status of Pump/Drainer #11827
- Fix the issue that panic might occur for `select ... for update` when Autocommit = 1 and there is no `begin` statement #11736
- Fix the permission check error that might occur when the `set default role` statement is executed #11777
- Fix the permission check error that might occur when `create user` or `drop user` is executed #11814
- Fix the issue that the `select ... for update` statement might auto retry when it is constructed into the `PointGetExecutor` function #11718
- Fix the boundary error that might occur when the Window function handles partition #11825
- Fix the issue that the `time` function hits EOF errors when handling an incorrectly formatted argument #11893
- Fix the issue that the Window function does not check the passed-in parameters #11705
- Fix the issue that the plan result viewed via `Explain` is inconsistent with the actually executed plan #11186
- Fix the issue that duplicate memory referenced by the Window function might result in a crash or incorrect results #11823
- Update the incorrect information in the `Succ` field in the slow log #11887

- Server
  - Rename the `tidb_back_off_wexight` variable to `tidb_backoff_weight` #11665
  - Update the minimum TiKV version compatible with the current TiDB to v3.0.0 #11618
  - Support `make testSuite` to ensure the suites in the test are correctly used #11685

- DDL
  - Skip the execution of unsupported partition-related DDL statements, including statements that modify the partition type while deleting multiple partitions #11373
  - Disallow a Generated Column to be placed before its dependent columns #11686
  - Modify the default values of `tidb_ddl_reorg_worker_cnt` and `tidb_ddl_reorg_batch_size` ↪ #11874

- Monitor
  - Add new backoff monitoring types to record duration for each backoff type; add more backoff metrics to cover previously uncounted types such as commit backoff

#11728

16.14.18.2 TiKV

- Fix the issue that ReadIndex might fail to respond to requests because of duplicate context #5256
- Fix potential scheduling jitters caused by premature `PutStore` #5277
- Fix incorrect timestamps reported from Region heartbeats #5296
- Reduce the size of core dump by excluding the shared block cache from it #5322
- Fix potential TiKV panics during region merge #5291
- Speed up leader change check for the dead lock detector #5317
- Support using `grpc env` to create deadlock clients #5346
- Add `config-check` to check whether the configuration is correct #5349
- Fix the issue that ReadIndex does not return anything when there is no leader #5351

16.14.18.3 PD

- Return success message for `pdctl` #1685

16.14.18.4 Tools

- TiDB Binlog
  - Modify the default value of `defaultBinlogItemCount` in Drainer from 65536 to 512 to reduce the chance of OOM on Drainer startup #721
  - Optimize the offline logic for pump server to avoid potential offline congestions #701
- TiDB Lightning:
  - Skip the system databases `mysql`, `information_schema`, `performance_schema`, and `sys` by default when importing #225

16.14.18.5 TiDB Ansible

- Optimize PD operations for rolling update to improve stability #894
- Remove the Grafana Collector components that are not supported by the current Grafana version #892
- Update TiKV alerting rules #898
- Fix the issue that the generated TiKV configuration misses the `pessimistic-txn` parameter #911
- Update Spark to V2.4.3, and update TiSpark to V2.1.4 that is compatible with Spark V2.4.3 #913, #918

16.14.19 TiDB 3.0.2 Release Notes

Release date: August 7, 2019

TiDB version: 3.0.2

TiDB Ansible version: 3.0.2

16.14.19.1 TiDB

- SQL Optimizer
    - Fix the issue that the "Can't find column in schema" message is reported when the same table occurs multiple times in a query and logically the query result is always empty #11247
    - Fix the issue that the query plan does not meet the expectation caused by the `TIDB_INLJ` hint not working correctly in some cases (like `explain select /*+ TIDB_INLJ(t1)*/ t1.b, t2.a from t t1, t t2 where t1.b = t2.a`) #11362
    - Fix the issue that the column name in the query result is wrong in some cases (like `SELECT IF(1,c,c)FROM t`) #11379
    - Fix the issue that some queries like `SELECT 0 LIKE 'a string'` return `TRUE` because the `LIKE` expression is implicitly converted to 0 in some cases #11411
    - Support sub-queries in the `SHOW` statement, like `SHOW COLUMNS FROM tbl WHERE FIELDS IN (SELECT 'a')` #11459
    - Fix the issue that the related column of the aggregate function cannot be found and an error is reported caused by the `outerJoinElimination` optimizing rule not correctly handling the column alias; improve alias parsing in the optimizing process to make optimization cover more query types #11377
    - Fix the issue that no error is reported when the syntax restriction is violated in the Window function (for example, `UNBOUNDED PRECEDING` is not allowed to appear at the end of the Frame definition) #11543
    - Fix the issue that `FUNCTION_NAME` is in uppercase in the `ERROR 3593 (HY000): You cannot use the window function FUNCTION_NAME in this context` error message, which causes incompatibility with MySQL #11535
    - Fix the issue that the unimplemented `IGNORE NULLS` syntax in the Window function is used but no error is reported #11593
    - Fix the issue that the Optimizer does not correctly estimate time equal conditions #11512
    - Support updating the Top-N statistics based on the feedback information #11507
- SQL Execution Engine
    - Fix the issue that the returned value is not `NULL` when the `INSERT` function contains `NULL` in parameters #11248
    - Fix the issue that the computing result might be wrong when the partitioned table is checked by the `ADMIN CHECKSUM` operation #11266

- Fix the issue that the result might be wrong when INDEX JOIN uses the prefix index #11246
- Fix the issue that result might be wrong caused by incorrectly aligning fractions when the `DATE_ADD` function does subtraction on date numbers involving microseconds #11288
- Fix the wrong result caused by the `DATE_ADD` function incorrectly processing the negative numbers in `INTERVAL` #11325
- Fix the issue that the number of fractional digits returned by `Mod(%)`, `Multiple (*)` or `Minus(-)` is different from that in MySQL when `Mod(%)`, `Multiple(*)` or `Minus(-)` returns 0 and the number of fractional digits is large (like `select 0.000 % 0.11234500000000000000`) #11251
- Fix the issue that `NULL` with a warning is incorrectly returned when the length of the result returned by `CONCAT` and `CONCAT_WS` functions exceeds `max_allowed_packet` #11275
- Fix the issue that `NULL` with a warning is incorrectly returned when parameters in the `SUBTIME` and `ADDTIME` functions are invalid #11337
- Fix the issue that `NULL` is incorrectly returned when parameters in the `CONVERT_TZ` function are invalid #11359
- Add the `MEMORY` column to the result returned by `EXPLAIN ANALYZE` to show the memory usage of this query #11418
- Add `CARTESIAN` Join to the result of `EXPLAIN` #11429
- Fix the incorrect data of auto-increment columns of the float and double types #11385
- Fix the panic issue caused by some `nil` information when pseudo statistics are dumped #11460
- Fix the incorrect query result of `SELECT ⋯ CASE WHEN ⋯ ELSE NULL ...` caused by constant folding optimization #11441
- Fix the issue that `floatStrToIntStr` does not correctly parse the input such as `+999.9999e2` #11473
- Fix the issue that `NULL` is not returned in some cases when the result of the `DATE_ADD` and `DATE_SUB` function overflows #11476
- Fix the issue that the conversion result is different from that in MySQL if the string contains an invalid character when a long string is converted to an integer #11469
- Fix the issue that the result of the `REGEXP BINARY` function is incompatible with MySQL caused by case sensitiveness of this function #11504
- Fix the issue that an error is reported when the `GRANT ROLE` statement receives `CURRENT_ROLE`; fix the issue that the `REVOKE ROLE` statement does not correctly revoke the `mysql.default_role` privilege #11356
- Fix the display format issue of the `Incorrect datetime value` warning information when executing statements like `SELECT ADDDATE('2008-01-34', -1)` #11447
- Fix the issue that the error message reports `constant ⋯ overflows float` rather than `constant ⋯ overflows bigint` if the result overflows when a float field of the JSON data is converted to an integer #11534

- Fix the issue that the result might be wrong caused by incorrect type conversion when the `DATE_ADD` function receives `FLOAT`, `DOUBLE` and `DECIMAL` column parameters #11527
- Fix the wrong result caused by incorrectly processing the sign of the INTERVAL fraction in the `DATE_ADD` function #11615
- Fix the incorrect query result when Index Lookup Join contains the prefix index caused by `Ranger` not correctly handling the prefix index #11565
- Fix the issue that the "Incorrect arguments to NAME_CONST" message is reported if the `NAME_CONST` function is executed when the second parameter of `NAME_CONST` is a negative number #11268
- Fix the issue that the result is incompatible with MySQL when an SQL statement involves computing the current time and the value is fetched multiple times; use the same value when fetching the current time for the same SQL statement #11394
- Fix the issue that `Close` is not called for `ChildExecutor` when the `Close` of `baseExecutor` reports an error. This issue might lead to Goroutine leaks when the `KILL` statements do not take effect and `ChildExecutor` is not closed #11576

- Server

  - Fix the issue that the auto-added value is 0 instead of the current timestamp when `LOAD DATA` processes the missing `TIMESTAMP` field in the CSV file #11250
  - Fix issues that the `SHOW CREATE USER` statement does not correctly check related privileges, and `USER` and `HOST` returned by `SHOW CREATE USER CURRENT_USER()` might be wrong #11229
  - Fix the issue that the returned result might be wrong when `executeBatch` is used in JDBC #11290
  - Reduce printing the log information of the streaming client when changing the TiKV server's port #11370
  - Optimize the logic of reconnecting the streaming client to the TiKV server so that the streaming client will not be blocked for a long time #11372
  - Add `REGION_ID` in `INFORMATION_SCHEMA.TIDB_HOT_REGIONS` #11350
  - Cancel the timeout duration of obtaining Region information from the PD API to ensure that obtaining Region information will not end in a failure when TiDB API `http://{TiDBIP}:10080/regions/hot` is called due to PD timeout when the number of Regions is large #11383
  - Fix the issue that Region related requests do not return partitioned table-related Regions in the HTTP API #11466
  - Make the following changes to reduce the probability of locking timeout caused by slow operations when the user manually validates pessimistic locking #11521:
    - Increase the default TTL of pessimistic locking from 30 seconds to 40 seconds
    - Increase the maximum TTL from 60 seconds to 120 seconds
    - Calculate the pessimistic locking duration from the first `LockKeys` request
  - Change the `SendRequest` function logic in the TiKV client: try to immediately connect to another peer instead of keeping waiting when the connect cannot be built #11531

- Optimize the Region cache: label the removed store as invalid when a store is moved while another store goes online with a same address, to update the store information in the cache as soon as possible #11567
- Add the Region ID to the result returned by the `http://{TiDB_ADDRESS:TIDB_IP}/mvcc/key/{db}/{table}/{handle}` API #11557
- Fix the issue that Scatter Table does not work caused by the Scatter Table API not escaping the Range key #11298
- Optimize the Region cache: label the store where the Region exists as invalid when the correspondent store is inaccessible, to avoid reduced query performance caused by accessing this store #11498
- Fix the error that the table schema can still be obtained through the HTTP API after dropping the database with the same name multiple times #11585

- DDL
  - Fix the issue that an error occurs when a non-string column with a zero length is being indexed #11214
  - Disallow modifying the columns with foreign key constraints and full-text indexes (Note: TiDB still supports foreign key constraints and full-text indexes in syntax) #11274
  - Fix the issue that the index offset of the column might be wrong because the position changed by the `ALTER TABLE` statement and the default value of the column are used concurrently #11346
  - Fix two issues that occur when parsing JSON files:
    - `int64` is used as the intermediate parsing result of `uint64` in `ConvertJSONToFloat`, which leads to the precision overflow error #11433
    - `int64` is used as the intermediate parsing result of `uint64` in `ConvertJSONToInt`, which leads to the precision overflow error #11551
  - Disallow dropping indexes on the auto-increment column to avoid that the auto-increment column might get an incorrect result #11399
  - Fix the following issues #11492:
    - The character set and the collation of the column are not consistent when explicitly specifying the collation but not the character set
    - The error is not correctly reported when there is a conflict between the character set and the collation that are specified by `ALTER TABLE ⋯ MODIFY COLUMN`
    - Incompatibility with MySQL when using `ALTER TABLE ⋯ MODIFY COLUMN` to specify character sets and collations multiple times
  - Add the trace details of the subquery to the result of the `TRACE` query #11458
  - Optimize the performance of executing `ADMIN CHECK TABLE` and greatly reduce its execution time #11547
  - Add the result returned by `SPLIT TABLE ⋯ REGIONS/INDEX` and make `TOTAL_SPLIT_REGION` and `SCATTER_FINISH_RATIO` display the number of Regions that have been split successfully before timeout in the result #11484

- Fix the issue that the precision displayed by statements like `SHOW CREATE TABLE` is incomplete when `ON UPDATE CURRENT_TIMESTAMP` is the column attribute and the float precision is specified #11591
- Fix the issue that the index result of the column cannot be correctly calculated when the expression of a virtual generated column contains another virtual generated column #11475
- Fix the issue that the minus sign cannot be added after `VALUE LESS THAN` in the `ALTER TABLE ⋯ ADD PARTITION ⋯` statement #11581

- Monitor
  - Fix the issue that data is not collected and reported because the `TiKVTxnCmdCounter` ↪ monitoring metric is not registered #11316
  - Add the `BindUsageCounter`, `BindTotalGauge` and `BindMemoryUsage` monitoring metrics for the Bind Info #11467

16.14.19.2 TiKV

- Fix the bug that TiKV panics if the Raft log is not written in time #5160
- Fix the bug that the panic information is not written into the log file after TiKV panics #5198
- Fix the bug that the Insert operation might be incorrectly performed in the pessimistic transaction #5203
- Lower the output level of some logs that require no manual intervention to INFO #5193
- Improve the accuracy of monitoring the storage engine size #5200
- Improve the accuracy of the Region size in tikv-ctl #5195
- Improve the performance of the deadlock detector for pessimistic locking #5192
- Improve the performance of GC in the Titan storage engine #5197

16.14.19.3 PD

- Fix the bug that the Scatter Region scheduler cannot work #1642
- Fix the bug that the merge Region operation cannot be performed in pd-ctl #1653
- Fix the bug that the remove-tombstone operation cannot be performed in pd-ctl #1651
- Fix the issue that the Region overlapping with the key scope cannot be found when performing the scan Region operation #1648
- Add the retrying mechanism to make sure that the members are added successfully in PD #1643

16.14.19.4 Tools

TiDB Binlog

- Add the configuration item check feature when starting, which will stop the Binlog service and report an error when an invalid item is found #687

- Add the `node-id` configuration in Drainer to specify a specific logic used by Drainer #684

TiDB Lightning

- Fix the issue that `tikv_gc_life_time` fails to be changed back to its original value when 2 checksums are running at the same time #218
- Add the configuration item check feature when starting, which will stop the Binlog service and report an error when an invalid item is found #217

16.14.19.5 TiDB Ansible

- Fix the unit error that the Disk Performance monitor treats seconds as milliseconds #840
- Add the `log4j` configuration file in Spark #841
- Fix the issue that the Prometheus configuration file is generated in the wrong format when Binlog is enabled and Kafka or ZooKeeper is configured #844
- Fix the issue that the `pessimistic-txn` configuration parameter is left out in the generated TiDB configuration file #850
- Add and optimize metrics on the TiDB Dashboard #853
- Add descriptions for each monitoring item on the TiDB Dashboard #854
- Add the TiDB Summary Dashboard to better view the cluster status and troubleshoot issues #855
- Update the Allocator Stats monitoring item on the TiKV Dashboard #857
- Fix the unit error in the Node Exporter's alerting expression #860
- Upgrade the TiSpark jar package to v2.1.2 #862
- Update the descriptions of the Ansible Task feature #867
- Update the expression of the local reader requests monitoring item on the TiDB Dashboard #874
- Update the expression of the TiKV Memory monitoring item on the Overview Dashboard, and fix the issue of wrongly displayed monitoring #879
- Remove the Binlog support in the Kafka mode #878
- Fix the issue that PD fails to transfer the Leader when executing the `rolling_update` ↪ `.yml` operation #887

16.14.20 TiDB 3.0.1 Release Notes

Release date: July 16, 2019

TiDB version: 3.0.1

TiDB Ansible version: 3.0.1

16.14.20.1 TiDB

- Add support for the `MAX_EXECUTION_TIME` feature #11026
- Add the `tidb_wait_split_region_finish_backoff` session variable to control the backoff time of splitting Regions #11166
- Support automatically adjusting the incremental gap allocated by auto-increment IDs based on the load, and the auto-adjustment scope of the incremental gap is 1000~2000000 #11006
- Add the `ADMIN PLUGINS ENABLE/ADMIN PLUGINS DISABLE` SQL statement to dynamically enable or disable plugins #11157
- Add the session connection information in the Audit plugin #11013
- Change the default behavior during the period of splitting Regions to wait for PD to finish scheduling #11166
- Prohibit Window Functions from being cached in Prepare Plan Cache to avoid incorrect results in some cases #11048
- Prohibit `ALTER` statements from modifying the definition of stored generated columns #11068
- Disallow changing virtual generated columns to stored generated columns #11068
- Disallow changing the generated column expression with indexes #11068
- Support compiling TiDB on the ARM64 architecture #11150
- Support modifying the collation of a database or a table, but the character set of the database/table has to be UTF-8 or utf8mb4 #11086
- Fix the issue that an error is reported when the `SELECT` subquery in the `UPDATE ⋯ SELECT` statement fails to parse the column in the `UPDATE` expression and the column is wrongly pruned #11252
- Fix the panic issue that happens when a column is queried on multiple times and the returned result is NULL during point queries #11226
- Fix the data race issue caused by non-thread safe `rand.Rand` when using the `RAND` function #11169
- Fix the bug that the memory usage of a SQL statement exceeds the threshold but the execution of this statement is not canceled in some cases when `oom-action="cancel"` is configured, and the returned result is incorrect #11004
- Fix the issue that `SHOW PROCESSLIST` shows that the memory usage is not `0` because the memory usage of MemTracker was not correctly cleaned #10970
- Fix the bug that the result of comparing integers and non-integers is not correct in some cases #11194
- Fix the bug that the query result is not correct when the query on table partitions contains a predicate in explicit transactions #11196
- Fix the DDL job panic issue because `infoHandle` might be `NULL` #11022
- Fix the issue that the query result is not correct because the queried column is not referenced in the subquery and is then wrongly pruned when running a nested aggregation query #11020
- Fix the issue that the `Sleep` function does not respond to the `KILL` statement in time #11028

- Fix the issue that the `DB` and `INFO` columns shown by the `SHOW PROCESSLIST` command are incompatible with MySQL #11003
- Fix the system panic issue caused by the `FLUSH PRIVILEGES` statement when `skip-grant-table=true` is configured #11027
- Fix the issue that the primary key statistics collected by `FAST ANALYZE` are not correct when the table primary key is an `UNSIGNED` integer #11099
- Fix the issue that the "invalid key" error is reported by the `FAST ANALYZE` statement in some cases #11098
- Fix the issue that the precision shown by the `SHOW CREATE TABLE` statement is incomplete when `CURRENT_TIMESTAMP` is used as the default value of the column and the float precision is specified #11088
- Fix the issue that the function name is not in lowercase when window functions report an error to make it compatible with MySQL #11118
- Fix the issue that TiDB fails to connect to TiKV and thus cannot provide service after the background thread of TiKV Client Batch gRPC panics #11101
- Fix the issue that the variable is set incorrectly by `SetVar` because of the shallow copy of the string #11044
- Fix the issue that the execution fails and an error is reported when the `INSERT ⋯ ON DUPLICATE` statement is applied on table partitions #11231
- Pessimistic locking (experimental feature)
    - Fix the issue that an incorrect result is returned because of the invalid lock on the row when point queries are run using the pessimistic locking and the returned data is empty #10976
    - Fix the issue that the query result is not correct because `SELECT ⋯ FOR UPDATE` does not use the correct TSO when using the pessimistic locking in the query #11015
    - Change the detection behavior from immediate conflict detection to waiting when an optimistic transaction meets a pessimistic lock to avoid worsening the lock conflict #11051

16.14.20.2 TiKV

- Add the statistics of the size of blob files in statistics information #5060
- Fix the core dump issue caused by the incorrectly cleaned memory resources when the process exits #5053
- Add all monitoring metrics related to the Titan engine #4772, #4836
- Add the number of open file handles for Titan when counting the number of open file handles to avoid the issue that no file handle is available because of inaccurate statistics of file handles #5026
- Set `blob_run_mode` to decide whether to enable the Titan engine on a specific CF #4991
- Fix the issue that the read operations cannot get the commit information of pessimistic transactions #5067

- Add the `blob-run-mode` configuration parameter to control the running mode of the Titan engine, and its value can be `normal`, `read-only` or `fallback` #4865
- Improve the performance of detecting deadlocks #5089

16.14.20.3 PD

- Fix the issue that the scheduling limit is automatically adjusted to 0 when PD schedules hot Regions #1552
- Add the `enable-grpc-gateway` configuration option to enable the gRPC gateway feature of etcd #1596
- Add `store-balance-rate`, `hot-region-schedule-limit` and other statistics related to scheduler configuration #1601
- Optimize the hot Region scheduling strategy and skip the Regions that lack replicas during scheduling to prevent multiple replicas from being scheduled to the same IDC #1609
- Optimize the Region merge processing logic and support giving priority to merging the Regions with smaller sizes to speed up Region merging #1613
- Adjust the default limit of hot Region scheduling in a single time to 64 to prevent too many scheduling tasks from occupying system resources and impacting performance #1616
- Optimize the Region scheduling strategy and support giving high priority to scheduling Regions in the `Pending` status #1617
- Fix the issue that `random-merge` and `admin-merge-region` operators cannot be added #1634
- Adjust the format of the Region key in the log to hexadecimal notation to make it easier to view #1639

16.14.20.4 Tools

TiDB Binlog

- Optimize the Pump GC strategy and remove the restriction that the unconsumed binlog cannot be cleaned to make sure that the resources are not occupied for a long time #646

TiDB Lightning

- Fix the import error that happens when the column names specified by the SQL dump are not in lowercase #210

16.14.20.5 TiDB Ansible

- Add the precheck feature for the ansible command and its `jmespath` and `jinja2` dependency packages #803, #813

- Add the `stop-write-at-available-space` parameter (10 GiB by default) in Pump to stop writing binlog files in Pump when the available disk space is less than the parameter value #806
- Update the I/O monitoring items in the TiKV monitoring information and make them compatible with the monitoring components of the new version #820
- Update the PD monitoring information, and fix the anomaly that Disk Latency is empty in the disk performance dashboard #817
- Add monitoring items for Titan in the TiKV details dashboard #824

16.14.21 TiDB 3.0 GA Release Notes

Release date: June 28, 2019

TiDB version: 3.0.0

TiDB Ansible version: 3.0.0

16.14.21.1 Overview

On June 28, 2019, TiDB 3.0 GA is released. The corresponding TiDB Ansible version is 3.0.0. Compared with TiDB 2.1, this release has greatly improved in the following aspects:

- Stability. TiDB 3.0 has demonstrated long-term stability for large-scale clusters with up to 150+ nodes and 300+ TB of storage.
- Usability. TiDB 3.0 has multi-facet improvements in usability, including standardized slow query logs, well-developed log file specification, and new features such as `EXPLAIN ANALYZE` and SQL Trace to save operation costs for users.
- Performance. The performance of TiDB 3.0 is 4.5 times greater than TiDB 2.1 in TPC-C benchmarks, and over 1.5 times in Sysbench benchmarks. Thanks to the support for Views, TPC-H 50G Q15 can now run normally.
- New features including Window Functions, Views (**Experimental**), partitioned tables, the plugin framework, pessimistic locking (**Experimental**), and `SQL Plan Management`.

16.14.21.2 TiDB

- New Features
  - Support Window Functions; compatible with all window functions in MySQL 8.0, including `NTILE`, `LEAD`, `LAG`, `PERCENT_RANK`, `NTH_VALUE`, `CUME_DIST`, `FIRST_VALUE`, `LAST_VALUE`, `RANK`, `DENSE_RANK`, and `ROW_NUMBER`
  - Support Views (**Experimental**)
  - Improve Table Partition
    - Support Range Partition
    - Support Hash Partition

    - Add the plug-in framework, supporting plugins such as IP Whitelist (**Enterprise**) and Audit Log (**Enterprise**).
    - Support the SQL Plan Management function to create SQL execution plan binding to ensure query stability (**Experimental**)

- SQL Optimizer

    - Optimize the `NOT EXISTS` subquery and convert it to `Anti Semi Join` to improve performance
    - Optimize the constant propagation on the `Outer Join`, and add the optimization rule of `Outer Join` elimination to reduce non-effective computations and improve performance
    - Optimize the `IN` subquery to execute `Inner Join` after aggregation to improve performance
    - Optimize `Index Join` to adapt to more scenarios
    - Improve the Partition Pruning optimization rule of Range Partition
    - Optimize the query logic for `_tidb_rowid`to avoid full table scan and improve performance
    - Match more prefix columns of the indexes when extracting access conditions of composite indexes if there are relevant columns in the filter to improve performance
    - Improve the accuracy of cost estimates by using order correlation between columns
    - Optimize `Join Order` based on the greedy strategy and the dynamic programming algorithm to speed up the join operation of multiple tables
    - Support Skyline Pruning, with some rules to prevent the execution plan from relying too heavily on statistics to improve query stability
    - Improve the accuracy of row count estimation for single-column indexes with NULL values
    - Support `FAST ANALYZE` that randomly samples in each Region to avoid full table scan and improve performance with statistics collection
    - Support the incremental Analyze operation on monotonically increasing index columns to improve performance with statistics collection
    - Support using subqueries in the `DO` statement
    - Support using `Index Join` in transactions
    - Optimize `prepare`/`execute` to support DDL statements with no parameters
    - Modify the system behavior to auto load statistics when the `stats-lease` variable value is 0
    - Support exporting historical statistics
    - Support the `dump`/`load` correlation of histograms

- SQL Execution Engine

    - Optimize log output: `EXECUTE` outputs user variables and `COMMIT` outputs slow query logs to facilitate troubleshooting
    - Support the `EXPLAIN ANALYZE` function to improve SQL tuning usability
    - Support the `admin show next_row_id` command to get the ID of the next row

    - Add six built-in functions: `JSON_QUOTE`, `JSON_ARRAY_APPEND`, `JSON_MERGE_PRESERVE` ↪ , `BENCHMARK` ,`COALESCE`, and `NAME_CONST`
    - Optimize control logics on the chunk size to dynamically adjust based on the query context, to reduce the SQL execution time and resource consumption
    - Support tracking and controlling memory usage in three operators - `TableReader`, `IndexReader` and `IndexLookupReader`
    - Optimize the Merge Join operator to support an empty `ON` condition
    - Optimize write performance for single tables that contains too many columns
    - Improve the performance of `admin show ddl jobs` by supporting scanning data in reverse order
    - Add the `split table region` statement to manually split the table Region to alleviate hotspot issues
    - Add the `split index region` statement to manually split the index Region to alleviate hotspot issues
    - Add a blocklist to prohibit pushing down expressions to Coprocessor
    - Optimize the `Expensive Query` log to print the SQL query in the log when it exceeds the configured limit of execution time or memory
- DDL
    - Support migrating from character set `utf8` to `utf8mb4`
    - Change the default character set from`utf8` to `utf8mb4`
    - Add the `alter schema` statement to modify the character set and the collation of the database
    - Support ALTER algorithm `INPLACE`/`INSTANT`
    - Support `SHOW CREATE VIEW`
    - Support `SHOW CREATE USER`
    - Support fast recovery of mistakenly deleted tables
    - Support adjusting the number of concurrencies of ADD INDEX dynamically
    - Add the `pre_split_regions` option that pre-allocates Regions when creating the table using the `CREATE TABLE` statement, to relieve write hot Regions caused by lots of writes after the table creation
    - Support splitting Regions by the index and range of the table specified using SQL statements to relieve hotspot issues
    - Add the `ddl_error_count_limit` global variable to limit the number of DDL task retries
    - Add a feature to use `SHARD_ROW_ID_BITS` to scatter row IDs when the column contains an AUTO_INCREMENT attribute to relieve hotspot issues
    - Optimize the lifetime of invalid DDL metadata to speed up recovering the normal execution of DDL operations after upgrading the TiDB cluster
- Transactions
    - Support the pessimistic transaction mode (**Experimental**)
    - Optimize transaction processing logics to adapt to more scenarios:
        - Change the default value `tidb_disable_txn_auto_retry` to `on`, which means non-auto committed transactions will not be retried

    - Add the `tidb_batch_commit` system variable to split a transaction into multiple ones to be executed concurrently
    - Add the `tidb_low_resolution_tso` system variable to control the number of TSOs to obtain in batches and reduce the number of times that transactions request for TSOs, to improve performance in scenarios with relatively low requirement of consistency
    - Add the `tidb_skip_isolation_level_check` variable to control whether to report errors when the isolation level is set to SERIALIZABLE
    - Modify the `tidb_disable_txn_auto_retry` system variable to make it work on all retryable errors

- Permission Management
    - Perform permission check on the `ANALYZE`, `USE`, `SET GLOBAL`, and `SHOW PROCESSLIST` statements
    - Support Role Based Access Control (RBAC) (**Experimental**)

- Server
    - Optimize slow query logs:
        - Restructure the log format
        - Optimize the log content
        - Optimize the log query method to support using the `INFORMATION_SCHEMA.SLOW_QUERY` and `ADMIN SHOW SLOW` statements of the memory table to query slow query logs
    - Develop a unified log format specification with restructured log system to facilitate collection and analysis by tools
    - Support using SQL statements to manage TiDB Binlog services, including querying status, enabling TiDB Binlog, maintaining and sending TiDB Binlog strategies.
    - Support using `unix_socket` to connect to the database
    - Support `Trace` for SQL statements
    - Support getting information for a TiDB instance via the `/debug/zip` HTTP interface to facilitate troubleshooting.
    - Optimize monitoring items to facilitate troubleshooting:
        - Add the `high_error_rate_feedback_total` monitoring item to monitor the difference between the actual data volume and the estimated data volume based on statistics
        - Add a QPS monitoring item in the database dimension
    - Optimize the system initialization process to only allow the DDL owner to perform the initialization. This reduces the startup time for initialization or upgrading.
    - Optimize the execution logic of `kill query` to improve performance and ensure resource is release properly
    - Add a startup option `config-check` to check the validity of the configuration file
    - Add the `tidb_back_off_weight` system variable to control the backoff time of internal error retries

    - Add the `wait_timeout`and `interactive_timeout` system variables to control the maximum idle connections allowed
    - Add the connection pool for TiKV to shorten the connection establishing time
- Compatibility
    - Support the `ALLOW_INVALID_DATES` SQL mode
    - Support the MySQL 320 Handshake protocol
    - Support manifesting unsigned BIGINT columns as auto-increment columns
    - Support the `SHOW CREATE DATABASE IF NOT EXISTS` syntax
    - Optimize the fault tolerance of `load data` for CSV files
    - Abandon the predicate pushdown operation when the filtering condition contains a user variable to improve the compatibility with MySQL's behavior of using user variables to simulate Window Functions

16.14.21.3 PD

- Support re-creating a cluster from a single node
- Migrate Region metadata from etcd to the go-leveldb storage engine to solve the storage bottleneck in etcd for large-scale clusters
- API
    - Add the `remove-tombstone` API to clear Tombstone stores
    - Add the `ScanRegions` API to batch query Region information
    - Add the `GetOperator` API to query running operators
    - Optimize the performance of the `GetStores` API
- Configurations
    - Optimize configuration check logic to avoid configuration item errors
    - Add `enable-two-way-merge` to control the direction of Region merge
    - Add `hot-region-schedule-limit` to control the scheduling rate for hot Regions
    - Add `hot-region-cache-hits-threshold` to identify hotspot when hitting multiple thresholds consecutively
    - Add the `store-balance-rate` configuration item to control the maximum numbers of balance Region operators allowed per minute
- Scheduler Optimizations
    - Add the store limit mechanism for separately controlling the speed of operators for each store
    - Support the `waitingOperator` queue to optimize the resource race among different schedulers
    - Support scheduling rate limit to actively send scheduling operations to TiKV. This improves the scheduling rate by limiting the number of concurrent scheduling tasks on a single node.
    - Optimize the `Region Scatter` scheduling to be not restrained by the limit mechanism

    - Add the `shuffle-hot-region` scheduler to facilitate TiKV stability test in scenarios of poor hotspot scheduling

- Simulator

    - Add simulator for data import scenarios
    - Support setting different heartbeats intervals for the Store

- Others

    - Upgrade etcd to solve the issues of inconsistent log output formats, Leader selection failure in prevote, and lease deadlocking
    - Develop a unified log format specification with restructured log system to facilitate collection and analysis by tools
    - Add monitoring metrics including scheduling parameters, cluster label information, and time consumed by PD to process TSO requests, Store ID, and address information.

16.14.21.4 TiKV

- Support distributed GC and concurrent lock resolving for improved GC performance
- Support reversed `raw_scan` and `raw_batch_scan`
- Support Multi-thread Raftstore and Multi-thread Apply to improve scalabilities, concurrency capacity, and resource usage within a single node. Performance improves by 70% under the same level of pressure
- Support batch receiving and sending Raft messages, improving TPS by 7% for write intensive scenarios
- Support checking RocksDB Level 0 files before applying snapshots to avoid write stall
- Introduce Titan, a key-value plugin that improves write performance for scenarios with value sizes greater than 1KiB, and relieves write amplification in certain degrees
- Support the pessimistic transaction mode (**Experimental**)
- Support getting monitoring information via HTTP
- Modify the semantics of `Insert` to allow Prewrite to succeed only when there is no Key
- Develop a unified log format specification with restructured log system to facilitate collection and analysis by tools
- Add performance metrics related to configuration information and key bound crossing
- Support Local Reader in RawKV to improve performance
- Engine

    - Optimize memory management to reduce memory allocation and copying for `Iterator Key Bound Option`
    - Support `block cache` sharing among different column families

- Server

    - Reduce context switch overhead from `batch commands`

    - Remove `txn scheduler`
    - Add monitoring items related to `read index` and `GC worker`
- RaftStore
    - Support Hibernate Regions to optimize CPU consumption from RaftStore (**Experimental**)
    - Remove the local reader thread
- Coprocessor
    - Refactor the computation framework to implement vector operators, computation using vector expressions, and vector aggregations to improve performance
    - Support providing operator execution status for the `EXPLAIN ANALYZE` statement in TiDB
    - Switch to the `work-stealing` thread pool model to reduce context switch cost

16.14.21.5 Tools

- TiDB Lightning
    - Support redirected replication of data tables
    - Support importing CSV files
    - Improve performance for conversion from SQL to KV pairs
    - Support batch import of single tables to improve performance
    - Support separately importing data and indexes for big tables to improve the performance of TiKV-importer
    - Support filling the missing column using the `row_id` or the default column value when column data is missing in the new file
    - Support setting a speed limit in `TIKV-importer` when uploading SST files to TiKV
- TiDB Binlog
    - Add the `advertise-addr` configuration in Drainer to support the bridge mode in the container environment
    - Add the `GetMvccByEncodeKey` function in Pump to speed up querying the transaction status
    - Support compressing communication data among components to reduce network resource consumption
    - Add the Arbiter tool that supports reading binlog from Kafka and replicate the data into MySQL
    - Support filtering out files that don't require replication via Reparo
    - Support replicating generated columns
    - Add the `syncer.sql-mode` configuration item to support using different sql-modes to parse DDL queries
    - Add the `syncer.ignore-table` configuration item to support filtering tables not to be replicated

- sync-diff-inspector
    - Support checkpoint to record verification status and continue the verification from last saved point after restarting
    - Add the `only-use-checksum` configuration item to check data consistency by calculating checksum
    - Support using TiDB statistics and multiple columns to split chunks for comparison to adapt to more scenarios

16.14.21.6 TiDB Ansible

- Upgrade the following monitoring components to a stable version:
    - Prometheus from V2.2.1 to V2.8.1
    - Pushgateway from V0.4.0 to V0.7.0
    - Node_exporter from V0.15.2 to V0.17.0
    - Alertmanager from V0.14.0 to V0.17.0
    - Grafana from V4.6.3 to V6.1.6
    - Ansible from V2.5.14 to V2.7.11
- Add the TiKV summary monitoring dashboard to view cluster status conveniently
- Add the TiKV trouble_shooting monitoring dashboard to remove duplicate items and facilitate troubleshooting
- Add the TiKV details monitoring dashboard to facilitate debugging and troubleshooting
- Add concurrent check for version consistency during rolling updates to improve the update performance
- Support deployment and operations for TiDB Lightning
- Optimize the `table-regions.py` script to support displaying Leader distribution by tables
- Optimize TiDB monitoring and add latency related monitoring items by SQL categories
- Modify the operating system version limit to only support the CentOS 7.0+ and Red Hat 7.0+ operating systems
- Add the monitoring item to predict the maximum QPS of the cluster (hidden by default)

16.14.22 TiDB 3.0.0-rc.3 Release Notes

Release date: June 21, 2019

TiDB version: 3.0.0-rc.3

TiDB Ansible version: 3.0.0-rc.3

16.14.22.1 Overview

On June 21, 2019, TiDB 3.0.0-rc.3 is released. The corresponding TiDB Ansible version is 3.0.0-rc.3. Compared with TiDB 3.0.0-rc.2, this release has greatly improved the stability, usability, features, the SQL optimizer, statistics, and the execution engine.

16.14.22.2 TiDB

- SQL Optimizer
  - Remove the feature of collecting virtual generated column statistics #10629
  - Fix the issue that the primary key constant overflows during point queries #10699
  - Fix the issue that using uninitialized information in `fast analyze` causes panic #10691
  - Fix the issue that executing the `create view` statement using `prepare` causes panic because of wrong column information #10713
  - Fix the issue that the column information is not cloned when handling window functions #10720
  - Fix the wrong estimation for the selectivity rate of the inner table selection in index join #10854
  - Support automatic loading statistics when the `stats-lease` variable value is 0 #10811
- Execution Engine
  - Fix the issue that resources are not correctly released when calling the `Close` function in `StreamAggExec` #10636
  - Fix the issue that the order of `table_option` and `partition_options` is incorrect in the result of executing the `show create table` statement for partitioned tables #10689
  - Improve the performance of `admin show ddl jobs` by supporting scanning data in reverse order #10687
  - Fix the issue that the result of the `show grants` statement in RBAC is incompatible with that of MySQL when this statement has the `current_user` field #10684
  - Fix the issue that UUIDs might generate duplicate values on multiple nodes #10712
  - Fix the issue that the `show view` privilege is not considered in `explain` #10635
  - Add the `split table region` statement to manually split the table Region to alleviate the hotspot issue #10765
  - Add the `split index region` statement to manually split the index Region to alleviate the hotspot issue #10764
  - Fix the incorrect execution issue when you execute multiple statements such as `create user`, `grant`, or `revoke` consecutively #10737
  - Add a blocklist to prohibit pushing down expressions to Coprocessor #10791
  - Add the feature of printing the `expensive query` log when a query exceeds the memory configuration limit #10849

    - Add the `bind-info-lease` configuration item to control the update time of the modified binding execution plan #10727
    - Fix the OOM issue in high concurrent scenarios caused by the failure to quickly release Coprocessor resources, resulted from the `execdetails.ExecDetails` pointer #10832
    - Fix the panic issue caused by the `kill` statement in some cases #10876
- Server
    - Fix the issue that goroutine might leak when repairing GC #10683
    - Support displaying the `host` information in slow queries #10693
    - Support reusing idle links that interact with TiKV #10632
    - Fix the support for enabling the `skip-grant-table` option in RBAC #10738
    - Fix the issue that `pessimistic-txn` configuration goes invalid #10825
    - Fix the issue that the actively canceled ticlient requests are still retried #10850
    - Improve performance in the case where pessimistic transactions conflict with optimistic transactions #10881
- DDL
    - Fix the issue that modifying charset using `alter table` causes the `blob` type change #10698
    - Add a feature to use `SHARD_ROW_ID_BITS` to scatter row IDs when the column contains an `AUTO_INCREMENT` attribute to alleviate the hotspot issue #10794
    - Prohibit adding stored generated columns by using the `alter table` statement #10808
    - Optimize the invalid survival time of DDL metadata to shorten the period during which the DDL operation is slower after cluster upgrade #10795

16.14.22.3 PD

- Add the `enable-two-way-merge` configuration item to allow only one-way merging #1583
- Add scheduling operations for `AddLightLearner` and `AddLightPeer` to make Region Scatter scheduling unrestricted by the limit mechanism #1563
- Fix the issue of insufficient reliability because the data might only have one replica replication when the system is started #1581
- Optimize configuration check logic to avoid configuration item errors #1585
- Adjust the definition of the `store-balance-rate` configuration to the upper limit of the number of balance operators generated per minute #1591
- Fix the issue that the store might have been unable to generate scheduled operations #1590

16.14.22.4 TiKV

- Engine

    - Fix the issue that incomplete snapshots are generated in the system caused by the iterator not checking the status #4936
    - Fix the data loss issue caused by a delay of flushing data to the disk when receiving snapshots after a power failure in abnormal conditions #4850

- Server

    - Add a feature to check the validity of the `block-size` configuration #4928
    - Add `READ_INDEX`-related monitoring metrics #4830
    - Add GC worker-related monitoring metrics #4922

- Raftstore

    - Fix the issue that the cache of the local reader is not cleared correctly #4778
    - Fix the issue that the request delay might be increased when transferring the leader and changing `conf` #4734
    - Fix the issue that a stale command is wrongly reported #4682
    - Fix the issue that the command might be pending for a long time #4810
    - Fix the issue that files are damaged after a power failure, which is caused by a delay of synchronizing the snapshot file to the disk #4807, #4850

- Coprocessor

    - Support Top-N in vector calculation #4827
    - Support `Stream` aggregation in vector calculation #4786
    - Support the `AVG` aggregate function in vector calculation #4777
    - Support the `First` aggregate function in vector calculation #4771
    - Support the `SUM` aggregate function in vector calculation #4797
    - Support the `MAX/MIN` aggregate function in vector calculation #4837
    - Support the `Like` expression in vector calculation #4747
    - Support the `MultiplyDecimal` expression in vector calculation #4849
    - Support the `BitAnd/BitOr/BitXor` expression in vector calculation #4724
    - Support the `UnaryNot` expression in vector calculation #4808

- Transaction

    - Fix the issue that an error occurs caused by non-pessimistic locking conflicts in pessimistic transactions #4801, #4883
    - Reduce unnecessary calculation for optimistic transactions after enabling pessimistic transactions to improve the performance #4813
    - Add a feature of single statement rollback to ensure that the whole transaction does not need a rollback operation in a deadlock situation #4848
    - Add pessimistic transaction-related monitoring items #4852
    - Support using the `ResolveLockLite` command to resolve lightweight locks to improve the performance when severe conflicts exist #4882

- tikv-ctl

    - Add the `bad-regions` command to support checking more abnormal conditions #4862

    - Add a feature of forcibly executing the `tombstone` command #4862

- Misc

    - Add the `dist_release` compiling command #4841

16.14.22.5 Tools

- TiDB Binlog

    - Fix the wrong offset issue caused by Pump not checking the returned value when it fails to write data #640
    - Add the `advertise-addr` configuration in Drainer to support the bridge mode in the container environment #634
    - Add the `GetMvccByEncodeKey` function in Pump to speed up querying the transaction status #632

16.14.22.6 TiDB Ansible

- Add a monitoring item to predict the maximum QPS value of the cluster ("hide" by default) #f5cfa4d

16.14.23 TiDB 3.0.0-rc.2 Release Notes

Release date: May 28, 2019

TiDB version: 3.0.0-rc.2

TiDB Ansible version: 3.0.0-rc.2

16.14.23.1 Overview

On May 28, 2019, TiDB 3.0.0-rc.2 is released. The corresponding TiDB Ansible version is 3.0.0-rc.2. Compared with TiDB 3.0.0-rc.1, this release has greatly improved the stability, usability, features, the SQL optimizer, statistics, and the execution engine.

16.14.23.2 TiDB

- SQL Optimizer

    - Support Index Join in more scenarios #10540
    - Support exporting historical statistics #10291
    - Support the incremental `Analyze` operation on monotonically increasing index columns #10355
    - Neglect the NULL value in the `Order By` clause #10488

- Fix the wrong schema information calculation of the `UnionAll` logical operator when simplifying the column information #10384
- Avoid modifying the original expression when pushing down the `Not` operator #10363
- Support the `dump/load` correlation of histograms #10573

- Execution Engine
  - Handle virtual columns with a unique index properly when fetching duplicate rows in `batchChecker` #10370
  - Fix the scanning range calculation issue for the `CHAR` column #10124
  - Fix the issue of `PointGet` incorrectly processing negative numbers #10113
  - Merge `Window` functions with the same name to improve execution efficiency #9866
  - Allow the `RANGE` frame in a `Window` function to contain no `OrderBy` clause #10496
- Server
  - Fix the issue that TiDB continuously creates a new connection to TiKV when a fault occurs in TiKV #10301
  - Make `tidb_disable_txn_auto_retry` affect all retryable errors instead of only write conflict errors #10339
  - Allow DDL statements without parameters to be executed using `prepare` ↪ `/execute` #10144
  - Add the `tidb_back_off_weight` variable to control the backoff time #10266
  - Prohibit TiDB retrying non-automatically committed transactions in default conditions by setting the default value of `tidb_disable_txn_auto_retry` to `on` #10266
  - Fix the database privilege judgment of `role` in `RBAC` #10261
  - Support the pessimistic transaction mode (experimental) #10297
  - Reduce the wait time for handling lock conflicts in some cases #10006
  - Make the Region cache able to visit follower nodes when a fault occurs in the leader node #10256
  - Add the `tidb_low_resolution_tso` variable to control the number of TSOs obtained in batches and reduce the times of transactions obtaining TSO to adapt for scenarios where data consistency is not so strictly required #10428
- DDL
  - Fix the uppercase issue of the charset name in the storage of the old version of TiDB #10272
  - Support `preSplit` of table partition, which pre-allocates table Regions when creating a table to avoid write hotspots after the table is created #10221
  - Fix the issue that TiDB incorrectly updates the version information in PD in some cases #10324
  - Support modifying the charset and collation using the `ALTER DATABASE` statement #10393

- Support splitting Regions based on the index and range of the specified table to relieve hotspot issues #10203
- Prohibit modifying the precision of the decimal column using the `alter table` statement #10433
- Fix the restriction for expressions and functions in hash partition #10273
- Fix the issue that adding indexes in a table that contains partitions will in some cases cause TiDB panic #10475
- Validate table information before executing the DDL to avoid invalid table schemas #10464
- Enable hash partition by default; and enable range columns partition when there is only one column in the partition definition #9936

16.14.23.3 PD

- Enable the Region storage by default to store the Region metadata #1524
- Fix the issue that hot Region scheduling is preempted by another scheduler #1522
- Fix the issue that the priority for the leader does not take effect #1533
- Add the gRPC interface for `ScanRegions` #1535
- Push operators actively #1536
- Add the store limit mechanism for separately controlling the speed of operators for each store #1474
- Fix the issue of inconsistent `Config` status #1476

16.14.23.4 TiKV

- Engine
  - Support multiple column families sharing a block cache #4563
- Server
  - Remove `TxnScheduler` #4098
  - Support pessimistic lock transactions #4698
- Raftstore
  - Support hibernate Regions to reduce the consumption of the raftstore CPU #4591
  - Fix the issue that the leader does not reply to the `ReadIndex` requests for the learner #4653
  - Fix transferring leader failures in some cases #4684
  - Fix the dirty read issue in some cases #4688
  - Fix the issue that a snapshot may lose applied data in some cases #4716
- Coprocessor
  - Add more RPN functions
    - `LogicalOr` #4691

- `LTReal` #4602
- `LEReal` #4602
- `GTReal` #4602
- `GEReal` #4602
- `NEReal` #4602
- `EQReal` #4602
- `IsNull` #4720
- `IsTrue` #4720
- `IsFalse` #4720
- Support comparison arithmetic for `Int` #4625
- Support comparison arithmetic for `Decimal` #4625
- Support comparison arithmetic for `String` #4625
- Support comparison arithmetic for `Time` #4625
- Support comparison arithmetic for `Duration` #4625
- Support comparison arithmetic for `Json` #4625
- Support plus arithmetic for `Int` #4733
- Support plus arithmetic for `Real` #4733
- Support plus arithmetic for `Decimal` #4733
- Support MOD functions for `Int` #4727
- Support MOD functions for `Real` #4727
- Support MOD functions for `Decimal` #4727
- Support minus arithmetic for `Int` #4746
- Support minus arithmetic for `Real` #4746
- Support minus arithmetic for `Decimal` #4746

16.14.23.5 Tools

- TiDB Binlog
  - Add a metric to track the delay of data replication downstream #594
- TiDB Lightning
  - Support merging sharded databases and tables #95
  - Add the retry mechanism for KV write failure #176
  - Update the default value of `table-concurrency` to 6 #175
  - Reduce required configuration items by automatically discovering `tidb.pd-addr` and `tidb.port` if they are not provided #173

16.14.24 TiDB 3.0.0-rc.1 Release Notes

Release Date: May 10, 2019

TiDB version: 3.0.0-rc.1

TiDB Ansible version: 3.0.0-rc.1

16.14.24.1 Overview

On May 10, 2019, TiDB 3.0.0-rc.1 is released. The corresponding TiDB Ansible version is 3.0.0-rc.1. Compared with TiDB 3.0.0-beta.1, this release has greatly improved the stability, usability, features, the SQL optimizer, statistics, and the execution engine.

16.14.24.2 TiDB

- SQL Optimizer
    - Improve the accuracy of cost estimates by using order correlation between columns; introduce a heuristic parameter `tidb_opt_correlation_exp_factor` to control the preference for index scans for scenarios when correlation cannot be directly used for estimation. #9839
    - Match more prefix columns of the indexes when extracting access conditions of composite indexes if there are relevant columns in the filter #10053
    - Use the dynamic programming algorithm to specify the execution order of join operations when the number of tables participating in the join is less than the value of `tidb_opt_join_reorder_threshold`. #8816
    - Match more prefix columns of the indexes in the inner tables that build the index join when using composite indexes as the access conditions #8471
    - Improve the accuracy of row count estimation for single-column indexes with NULL values #9474
    - Specially handle `GROUP_CONCAT` when eliminating aggregate functions during the logical optimization phase to prevent incorrect executions #9967
    - Properly push the filter down to child nodes of the join operator if the filter is a constant #9848
    - Specially handle some functions such as `RAND()` when pruning columns during the logical optimization phase to prevent incompatibilities with MySQL #10064
    - Support `FAST ANALYZE`, which speeds up statistics collection by sampling the region instead of scanning the entire region. This feature is controlled by the variable `tidb_enable_fast_analyze`. #10258
    - Support SQL Plan Management, which ensures execution stability by performing execution plan binding for SQL statements. This feature is currently in beta and only supports bound execution plans for SELECT statements. It is not recommended to use it in the production environment. #10284
- Execution Engine
    - Support tracking and controlling memory usage in three operators - `TableReader`, `IndexReader` and `IndexLookupReader` #10003
    - Support showing more information about coprocessor tasks in the slow log such as the number of tasks in coprocessor, the average/longest/90% of execution/waiting time and the addresses of the TiKVs which take the longest execution time or waiting time #10165
    - Support the prepared DDL statements with no placeholders #10144

- Server
    - Only allow the DDL owner to execute bootstrap when TiDB is started #10029
    - Add the variable `tidb_skip_isolation_level_check` to prevent TiDB from reporting errors when setting the transaction isolation level to SERIALIZABLE #10065
    - Merge the implicit commit time and the SQL execution time in the slow log #10294
        - Support for SQL Roles (RBAC Privilege Management)
        - Support `SHOW GRANT` #10016
        - Support `SET DEFAULT ROLE` #9949
    - Support `GRANT ROLE` #9721
    - Fix the `ConnectionEvent` error from the `whitelist` plugin that makes TiDB exit #9889
    - Fix the issue of mistakenly adding read-only statements to the transaction history #9723
    - Improve `kill` statements to stop SQL execution and release resources more quickly #9844
    - Add a startup option `config-check` to check the validity of the configuration file #9855
    - Fix the validity check of inserting NULL fields when the strict SQL mode is disabled #10161
- DDL
    - Add the `pre_split_regions` option for `CREATE TABLE` statements; this option supports pre-splitting the Table Region when creating a table to avoid write hot spots caused by lots of writes after the table creation #10138
    - Optimize the execution performance of some DDL statements #10170
    - Add the warning that full-text indexes are not supported for `FULLTEXT KEY` #9821
    - Fix the compatibility issue for the UTF8 and UTF8MB4 charsets in the old versions of TiDB #9820
    - Fix the potential bug in `shard_row_id_bits` of a table #9868
    - Fix the bug that the column charset is not changed after the table charset is changed #9790
    - Fix a potential bug in `SHOW COLUMN` when using `BINARY/BIT` as the column default value #9897
    - Fix the compatibility issue in displaying `CHARSET/COLLATION` descriptions in the `SHOW FULL COLUMNS` statement #10007
    - Fix the issue that the `SHOW COLLATIONS` statement only lists collations supported by TiDB #10186

16.14.24.3 PD

- Upgrade ETCD #1452

    - Unify the log format of etcd and PD server
    - Fix the issue of failing to elect Leader by PreVote
    - Support fast dropping the "propose" and "read" requests that are to fail to avoid blocking the subsequent requests
    - Fix the deadlock issue of Lease
- Fix the issue that a hot store makes incorrect statistics of keys #1487
- Support forcibly rebuilding a PD cluster from a single PD node #1485
- Fix the issue that `regionScatterer` might generate an invalid `OperatorStep` #1482
- Fix the too short timeout issue of the `MergeRegion` operator #1495
- Support giving high priority to hot region scheduling #1492
- Add the metrics for recording the time of handling TSO requests on the PD server side #1502
- Add the corresponding Store ID and Address to the metrics related to the store #1506
- Support the `GetOperator` service #1477
- Fix the issue that the error cannot be sent in the Heartbeat stream because the store cannot be found #1521

16.14.24.4 TiKV

- Engine
    - FIx the issue that may cause incorrect statistics on read traffic #4436
    - Fix the issue that may cause prefix extractor panic when deleting a range #4503
    - Optimize memory management to reduce memory allocation and copying for `Iterator Key Bound Option` #4537
    - Fix the issue that failing to consider learner log gap may in some cases cause panic #4559
    - Support `block cache` sharing among different `column families` #4612
- Server
    - Reduce context switch overhead of `batch commands` #4473
    - Check the validity of seek iterator status #4470
- RaftStore
    - Support configurable `properties index distance` #4517
- Coprocessor
    - Add batch index scan executor #4419
    - Add vectorized evaluation framework #4322
    - Add execution summary framework for batch executors #4433
    - Check the maximum column when constructing the RPN expression to avoid invalid column offset that may cause evaluation panic #4481
    - Add `BatchLimitExecutor` #4469

- Replace the original `futures-cpupool` with `tokio-threadpool` in ReadPool to reduce context switch #4486
- Add batch aggregation framework #4533
- Add `BatchSelectionExecutor` #4562
- Add batch aggression function `AVG` #4570
- Add RPN function `LogicalAnd`#4575

- Misc
  - Support `tcmalloc` as a memory allocator #4370

16.14.24.5 Tools

- TiDB Binlog
  - Fix the replication abortion issue when binlog data for the primary key column of unsigned int type is negative #573
  - Provide no compression option when downstream is `pb`; modify the downstream name from `pb` to `file` #559
  - Add the `storage.sync-log` configuration item in Pump that allows asynchronous flush on local storage #509
  - Support traffic compression for communications between Pump and Drainer #495
  - Add the `syncer.sql-mode` configuration item in Drainer to support parsing DDL queries in different sql-mode #511
  - Add the `syncer.ignore-table` configuration item to support filtering out tables that do not require replication #520
- Lightning
  - Use row IDs or default column values to populate the column data missed in the dump file #170
  - Fix the bug in Importer that import success may still be returned even if part of the SST failed to be imported #4566
  - Support speed limit in Importer when uploading SST to TiKV #4412
  - Support importing tables by size to reduce impacts on the cluster brought by Checksum and Analyze for big tables, and improve the success rate for Checksum and Analyze #156
  - Improve Lightning's SQL encoding performance by 50% by directly parsing data source file as types.Datum of TiDB and saving extra parsing overhead from the KV encoder #145
  - Change log format to Unified Log Format #162
  - Add some command line options for use when the configuration file is missing #157
- sync-diff-inspector
  - Support checkpoint to record verification status and continue the verification from last saved point after restarting #224

- Add the `only-use-checksum` configuration item to check data consistency by calculating checksum #215

16.14.24.6 TiDB Ansible

- Support more TiKV monitoring panels and update versions for Ansible, Grafana, and Prometheus #727
    - Summary dashboard for viewing cluster status
    - trouble_shooting dashboard for troubleshooting issues
    - Details dashboard for developers to analyze issues
- Fix the bug that causes the downloading failure of TiDB Binlog of Kafka version #730
- Modify version limits on supported operating systems as CentOS 7.0+ and later, and Red Hat 7.0 and later #733
- Change version detection mode during the rolling update to multi-concurrent #736
- Update documentation links in README #740
- Remove redundant TiKV monitoring metrics; add new metrics for troubleshooting #735
- Optimize `table-regions.py` script to display leader distribution by table #739
- Update configuration file for Drainer #745
- Optimize TiDB monitoring with new panels that display latencies by SQL categories #747
- Update the Lightning configuration file and add the `tidb_lightning_ctl` script #1e946f8

16.14.25 TiDB 3.0.0 Beta.1 Release Notes

Release Date: March 26, 2019

TiDB version: 3.0.0-beta.1

TiDB Ansible version: 3.0.0-beta.1

16.14.25.1 Overview

On March 26, 2019, TiDB 3.0.0 Beta.1 is released. The corresponding TiDB Ansible version is 3.0.0 Beta.1. Compared with TiDB 3.0.0 Beta, this release has greatly improved the stability, usability, features, the SQL optimizer, statistics, and the execution engine.

16.14.25.2 TiDB

- SQL Optimizer
    - Support calculating the Cartesian product by using `Sort Merge Join` #9032

    - Support Skyline Pruning, with some rules to prevent the execution plan from relying too heavily on statistics #9337
    - Support Window Functions
        - `NTILE` #9682
        - `LEAD` and `LAG` #9672
        - `PERCENT_RANK` #9671
        - `NTH_VALUE` #9596
        - `CUME_DIST` #9619
        - `FIRST_VALUE` and `LAST_VALUE` #9560
        - `RANK` and `DENSE_RANK` #9500
        - `RANGE FRAMED` #9450
        - `ROW FRAMED` #9358
        - `ROW NUMBER` #9098
    - Add a type of statistic that indicates the order correlation between columns and the handle column #9315
- SQL Execution Engine
    - Add built-in functions
        - `JSON_QUOTE` #7832
        - `JSON_ARRAY_APPEND` #9609
        - `JSON_MERGE_PRESERVE` #8931
        - `BENCHMARK` #9252
        - `COALESCE` #9087
        - `NAME_CONST` #9261
    - Optimize the Chunk size based on the query context, to reduce the execution time of SQL statements and resources consumption of the cluster #6489
- Privilege management
    - Support `SET ROLE` and `CURRENT_ROLE` #9581
    - Support `DROP ROLE` #9616
    - Support `CREATE ROLE` #9461
- Server
    - Add the `/debug/zip` HTTP interface to get information of the current TiDB instance #9651
    - Support the `show pump status` and `show drainer status` SQL statements to check the Pump or Drainer status #9456
    - Support modifying the Pump or Drainer status by using SQL statements #9789
    - Support adding HASH fingerprints to SQL text for easy tracking of slow SQL statements #9662
    - Add the `log_bin` system variable ("0" by default) to control the enabling state of binlog; only support checking the state currently #9343
    - Support managing the sending binlog strategy by using the configuration file #9864

    - Support querying the slow log by using the `INFORMATION_SCHEMA.SLOW_QUERY` memory table #9290
    - Change the MySQL version displayed in TiDB from 5.7.10 to 5.7.25 #9553
    - Unify the log format for easy collection and analysis by tools
    - Add the `high_error_rate_feedback_total` monitoring item to record the difference between the actual data volume and the estimated data volume based on statistics #9209
    - Add the QPS monitoring item in the database dimension, which can be enabled by using a configuration item #9151
- DDL
    - Add the `ddl_error_count_limit` global variable ("512" by default) to limit the number of DDL task retries (If this number exceeds the limit, the DDL task is canceled) #9295
    - Support ALTER ALGORITHM `INPLACE/INSTANT` #8811
    - Support the `SHOW CREATE VIEW` statement #9309
    - Support the `SHOW CREATE USER` statement #9240

16.14.25.3 PD

- Unify the log format for easy collection and analysis by tools
- Simulator
    - Support different heartbeat intervals in different stores #1418
    - Add a case about importing data #1263
- Make hotspot scheduling configurable #1412
- Add the store address as the dimension monitoring item to replace the previous Store ID #1429
- Optimize the `GetStores` overhead to speed up the Region inspection cycle #1410
- Add an interface to delete the Tombstone Store #1472

16.14.25.4 TiKV

- Optimize the Coprocessor calculation execution framework and implement the TableScan section, with the Single TableScan performance improved by 5% ~ 30%
    - Implement the definition of the `BatchRows` row and the `BatchColumn` column #3660
    - Implement `VectorLike` to support accessing encoded and decoded data in the same way #4242
    - Define the `BatchExecutor` to interface and implement the way of converting requests to `BatchExecutor` #4243
    - Implement transforming the expression tree into the RPN format #4329

    - Implement the `BatchTableScanExecutor` vectorization operator to accelerate calculation #4351

- Unify the log format for easy collection and analysis by tools
- Support using the Local Reader to read in the Raw Read interface #4222
- Add metrics about configuration information #4206
- Add metrics about key exceeding bound #4255
- Add an option to control panic or return an error when encountering the key exceeding bound error #4254
- Add support for the `INSERT` operation, make prewrite succeed only when keys do not exist, and eliminate `Batch Get` #4085
- Use more fair batch strategy in the Batch System #4200
- Support Raw scan in tikv-ctl #3825

16.14.25.5 Tools

- TiDB Binlog
    - Add the Arbiter tool that supports reading binlog from Kafka and replicate the data into MySQL
    - Support filtering files that do not need to be replicated
    - Support replicating generated columns
- Lightning
    - Support disabling TiKV periodic Level-1 compaction, and when the TiKV cluster version is 2.1.4 or later, Level-1 compaction is automatically executed in the import mode #119, #4199
    - Add the `table_concurrency` configuration item to limit the number of import engines ("16" by default) and avoid overusing the importer disk space #119
    - Support saving the intermediate state SST to the disk, to reduce memory usage #4369
    - Optimize the import performance of TiKV-Importer and support separate import of data and indexes for large tables #132
    - Support importing CSV files #111
- Data replication comparison tool (sync-diff-inspector)
    - Support using TiDB statistics to split chunks to be compared #197
    - Support using multiple columns to split chunks to be compared #197

16.14.26 TiDB 3.0 Beta Release Notes

On January 19, 2019, TiDB 3.0 Beta is released. The corresponding TiDB Ansible 3.0 Beta is also released. TiDB 3.0 Beta builds on TiDB 2.1 with an added focus in stability, the SQL optimizer, statistics, and the execution engine.

16.14.26.1 TiDB

- New Features
  - Support Views
  - Support Window Functions
  - Support Range Partitioning
  - Support Hash Partitioning
- SQL Optimizer
  - Re-support the optimization rule of `AggregationElimination` #7676
  - Optimize the `NOT EXISTS` subquery and convert it to Anti Semi Join #7842
  - Add the `tidb_enable_cascades_planner` variable to support the new Cascades optimizer. Currently, the Cascades optimizer is not yet fully implemented and is turned off by default #7879
  - Support using Index Join in transactions #7877
  - Optimize the constant propagation on the Outer Join, so that the filtering conditions related to the Outer table in the Join result can be pushed down through the Outer Join to the Outer table, reducing the useless calculation of the Outer Join and improving the execution performance #7794
  - Adjust the optimization rule of Projection Elimination to the position after the Aggregation Elimination, to avoid redundant `Project` operators #7909
  - Optimize the `IFNULL` function and eliminate this function when the input parameter has a non-NULL attribute #7924
  - Support Range for `_tidb_rowid` construction queries, to avoid full table scan and reduce cluster stress #8047
  - Optimize the `IN` subquery to do the Inner Join after the aggregation, and add the `tidb_opt_insubq_to_join_and_agg` variable to control whether to enable this optimization rule and open it by default #7531
  - Support using subqueries in the `DO` statement #8343
  - Add the optimization rule of Outer Join elimination to reduce unnecessary table scan and Join operations and improve execution performance #8021
  - Modify the Hint behavior of the `TIDB_INLJ` optimizer, and the optimizer will use the table specified in Hint as the Inner table of Index Join #8243
  - Use `PointGet` in a wide range so that it can be used when the execution plan cache of the `Prepare` statement takes effect #8108
  - Introduce the greedy `Join Reorder` algorithm to optimize the join order selection when joining multiple tables #8394
  - Support View #8757
  - Support Window Function #8630
  - Return warning to the client when `TIDB_INLJ` is not in effect, to enhance usability #9037
  - Support deducing the statistics for filtered data based on filtering conditions and table statistics #7921
  - Improve the Partition Pruning optimization rule of Range Partition #8885

- SQL Executor
    - Optimize the `Merge Join` operator to support the empty `ON` condition #9037
    - Optimize the log and print the user variables used when executing the `EXECUTE` statement #7684
    - Optimize the log to print slow query information for the `COMMIT` statement #7951
    - Support the `EXPLAIN ANALYZE` feature to make the SQL tuning process easier #7827
    - Optimize the write performance of wide tables with many columns #7935
    - Support `admin show next_row_id` #8242
    - Add the `tidb_init_chunk_size` variable to control the size of the initial Chunk used by the execution engine #8480
    - Improve `shard_row_id_bits` and cross-check the auto-increment ID #8936
- `Prepare` Statement
    - Prohibit adding the `Prepare` statement containing subqueries to the query plan cache to guarantee the query plan is correct when different user variables are input #8064
    - Optimize the query plan cache to guarantee the plan can be cached when the statement contains non-deterministic functions #8105
    - Optimize the query plan cache to guarantee the query plan of `DELETE`/`UPDATE` ↪ `/INSERT` can be cached #8107
    - Optimize the query plan cache to remove the corresponding plan when executing the `DEALLOCATE` statement #8332
    - Optimize the query plan cache to avoid the TiDB OOM issue caused by caching too many plans by limiting the memory usage #8339
    - Optimize the `Prepare` statement to support using the `?` placeholder in the `ORDER` ↪ `BY`/`GROUP BY`/`LIMIT` clause #8206
- Privilege Management
    - Add the privilege check for the `ANALYZE` statement #8486
    - Add the privilege check for the `USE` statement #8414
    - Add the privilege check for the `SET GLOBAL` statement #8837
    - Add the privilege check for the `SHOW PROCESSLIST` statement #7858
- Server
    - Support the `Trace` feature #9029
    - Support the plugin framework #8788
    - Support using `unix_socket` and TCP simultaneously to connect to the database #8836
    - Support the `interactive_timeout` system variable #8573
    - Support the `wait_timeout` system variable #8346
    - Support splitting a transaction into multiple transactions based on the number of statements using the `tidb_batch_commit` variable #8293
    - Support using the `ADMIN SHOW SLOW` statement to check slow logs #7785

- Compatibility
    - Support the `ALLOW_INVALID_DATES` SQL mode #9027
    - Improve `LoadData` fault-tolerance for the CSV file #9005
    - Support the MySQL 320 handshake protocol #8812
    - Support using the unsigned `bigint` column as the auto-increment column #8181
    - Support the `SHOW CREATE DATABASE IF NOT EXISTS` syntax #8926
    - Abandon the predicate pushdown operation when the filtering condition contains a user variable to improve the compatibility with MySQL's behavior of using user variables to mock the Window Function behavior #8412
- DDL
    - Support fast recovery of mistakenly deleted tables #7937
    - Support adjusting the number of concurrencies of `ADD INDEX` dynamically #8295
    - Support changing the character set of tables or columns to `utf8`/`utf8mb4` #8037
    - Change the default character set from `utf8` to `utf8mb4` #7965
    - Support Range Partition #8011

16.14.26.2 Tools

- TiDB Lightning
    - Speed up converting SQL statements to KV pairs remarkably #110
    - Support batch import for a single table to improve import performance and stability #113

16.14.26.3 PD

- Add `RegionStorage` to store Region metadata separately #1237
- Add shuffle hot Region scheduler #1361
- Add scheduling parameter related metrics #1406
- Add cluster label related metrics #1402
- Add the importing data simulator #1263
- Fix the `Watch` issue about leader election #1396

16.14.26.4 TiKV

- Support distributed GC #3179
- Check RocksDB Level 0 files before applying snapshots to avoid Write Stall #3606
- Support reverse `raw_scan` and `raw_batch_scan` #3742
- Support using HTTP to obtain monitoring information #3855
- Support DST better #3786
- Support receiving and sending Raft messages in batch #3931
- Introduce a new storage engine Titan #3985

- Upgrade gRPC to v1.17.2 #4023
- Support receiving the client requests and sending replies in batch #4043
- Support multi-thread Apply #4044
- Support multi-thread Raftstore #4066

16.15 v2.1

16.15.1 TiDB 2.1.19 Release Notes

Release date: December 27, 2019

TiDB version: 2.1.19

TiDB Ansible version: 2.1.19

16.15.1.1 TiDB

- SQL Optimizer
  - Optimize the scenario of `select max(_tidb_rowid)from t` to avoid full table scan #13294
  - Fix the incorrect results caused by the incorrect value assigned to the user variable in the query and the push-down of predicates #13230
  - Fix the issue that the statistics are not accurate because a data race occurs when statistics are updated #13690
  - Fix the issue that the result is incorrect when the `UPDATE` statement contains both a sub-query and a stored generated column; fix the `UPDATE` statement execution error when this statement contains two same-named tables from different databases #13357
  - Fix the issue that the query plan might be incorrectly selected because the `PhysicalUnionScan` operator incorrectly sets the statistics #14134
  - Remove the `minAutoAnalyzeRatio` constraint to make the automatic `ANALYZE` more timely #14013
  - Fix the issue that the estimated number of rows is greater than `1` when the `WHERE` clause contains an equal condition on the unique key #13385
- SQL Execution Engine
  - Fix the precision overflow when using `int64` as the intermediate result of `unit64` in `ConvertJSONToInt` #13036
  - Fix the issue that when the `SLEEP` function is in a query (for example, `select 1 from (select sleep(1))t;`), column pruning causes `sleep(1)` in the query to be invalid #13039
  - Reduce memory overhead by reusing `Chunk` in the `INSERT ON DUPLICATE UPDATE` statement #12999
  - Add more transaction-related fields for the `slow_query` table #13129:

    - `Prewrite_time`
    - `Commit_time`
    - `Get_commit_ts_time`
    - `Commit_backoff_time`
    - `Backoff_types`
    - `Resolve_lock_time`
    - `Local_latch_wait_time`
    - `Write_key`
    - `Write_size`
    - `Prewrite_region`
    - `Txn_retry`
  - Fix the issue that a subquery contained in an `UPDATE` statement is incorrectly converted; fix the `UPDATE` execution failure when the `WHERE` clause contains a subquery #13120
  - Support executing `ADMIN CHECK TABLE` on partitioned tables #13143
  - Fix the issue that the precision of statements such as `SHOW CREATE TABLE` is incomplete when `ON UPDATE CURRENT_TIMESTAMP` is used as a column attribute and floating point precision is specified #12462
  - Fix the panic occurred when executing the `SELECT * FROM information_schema` ↪ `.KEY_COLUMN_USAGE` statement because the foreign key is not checked when dropping, modifying or changing the column #14162
  - Fix the issue that the returned data might be duplicated when `Streaming` is enabled in TiDB #13255
  - Fix the `Invalid time format` error caused by daylight saving time #13624
  - Fix the issue that data is incorrect because the precision is lost when an integer is converted to an unsigned floating point or decimal type #13756
  - Fix the issue that an incorrect type of value is returned when the `Quote` function handles the `NULL` value #13681
  - Fix the issue that the timezone is incorrect after parsing the date from strings using `gotime.Local` #13792
  - Fix the issue that the result might be incorrect because the `binSearch` function does not return an error in the implementation of `builtinIntervalRealSig` #13768
  - Fix the issue that an error might occur when converting the string type to the floating point type in the `INSERT` statement execution #14009
  - Fix the incorrect result returned from the `sum(distinct)` function #13041
  - Fix the issue that `data too long` is returned when `CAST` converting the data in `union` of the same location to the merged type because the returned type length of the `jsonUnquoteFunction` function is given an incorrect value #13645
  - Fix the issue that the password cannot be set because the privilege check is too strict #13805
- Server
  - Fix the issue that `KILL CONNECTION` might cause the goroutine leak #13252

- Support getting the binlog status of all TiDB nodes via the `info/all` interface of the HTTP API #13188
- Fix the failure to build the TiDB project on Windows #13650
- Add the `server-version` configuration item to control and modify the version of TiDB server #13904
- Fix the issue that the binary `plugin` compiled with Go1.13 does not run normally #13527

- DDL
  - Use the table's `COLLATE` instead of the system's default charset in the column when a table is created and the table contains `COLLATE` #13190
  - Limit the length of the index name when creating a table #13311
  - Fix the issue that the length of the table name is not checked when renaming a table #13345
  - Check the width range of the `BIT` column #13511
  - Make the error information output from `change/modify column` more understandable #13798
  - Fix the issue that when executing the `drop column` operation that has not yet been handled by the downstream Drainer, the downstream might receive DML operations without the affected column #13974

16.15.1.2 TiKV

- Raftstore
  - Fix the panic occurred when restarting TiKV and `is_merging` is given an incorrect value in the process of merging Regions and applying the Compact log #5884
- Importer
  - Remove the limit on the gRPC message length #5809

16.15.1.3 PD

- Improve the performance of the HTTP API for getting all Regions #1988
- Upgrade etcd to fix the issue that etcd PreVote cannot elect a leader (downgrade not supported) #2052

16.15.1.4 Tools

- TiDB Binlog
  - Optimize the node status information output through binlogctl #777

- Fix the panic occurred because of the `nil` value in the Drainer filter configuration #802
- Optimize the `Graceful` exit of Pump #825
- Add more detailed monitoring metrics when Pump writes binlog data #830
- Optimize Drainer's logic to refresh table information after Drainer has executed a DDL operation #836
- Fix the issue that the commit binlog of a DDL operation is ignored when Pump does not receive this binlog #855

16.15.1.5 TiDB Ansible

- Rename the `Uncommon Error OPM` monitoring item of TiDB service to `Write Binlog Error` and add the corresponding alert message #1038
- Upgrade TiSpark to 2.1.8 #1063

16.15.2 TiDB 2.1.18 Release Notes

Release date: November 4, 2019

TiDB version: 2.1.18

TiDB Ansible version: 2.1.18

16.15.2.1 TiDB

- SQL Optimizer
  - Fix the issue that invalid query ranges might appear when split by feedback #12172
  - Fix the issue that the privilege check is incorrect in point get plan #12341
  - Optimize execution performance of the `select ... limit ... offset ⋯` statement by pushing the Limit operator down to the `IndexLookUpReader` execution logic #12380
  - Support using parameters in `ORDER BY`, `GROUP BY` and `LIMIT OFFSET` #12514
  - Fix the issue that `IndexJoin` on the partition table returns incorrect results #12713
  - Fix the issue that the `str_to_date` function in TiDB returns a different result from MySQL when the date string and the format string do not match #12757
  - Fix the issue that outer join is incorrectly converted to inner join when the `cast` function is included in the query conditions #12791
  - Fix incorrect expression passing in the join condition of `AntiSemiJoin` #12800
- SQL Engine
  - Fix the incorrectly rounding of time (for example, `2019-09-11 11:17:47.999999666` should be rounded to `2019-09-11 11:17:48`) #12259

- Fix the issue that the duration by `sql_type` for the `PREPARE` statement is not shown in the monitoring record #12329
- Fix the panic issue when the `from_unixtime` function handles null #12572
- Fix the compatibility issue that when an invalid value is inserted as the `YEAR` type, the result is `NULL` instead of `0000` #12744
- Improve the behavior of the `AutoIncrement` column when it is implicitly allocated, to keep it consistent with the default mode of MySQL auto-increment locking ("consecutive" lock mode): for the implicit allocation of multiple `AutoIncrement` IDs in a single-line `Insert` statement, TiDB guarantees the continuity of the allocated values. This improvement ensures that the JDBC `getGeneratedKeys()` method will get the correct results in any scenario #12619
- Fix the issue that the query is hanged when `HashAgg` serves as a child node of `Apply` #12769
- Fix the issue that the `AND` and `OR` logical expressions return incorrect results when it comes to type conversion #12813

- Server
  - Fix the issue that the `SLEEP()` function is invalid for the `KILL TIDB QUERY` statements #12159
  - Fix the issue that no error is reported when `AUTO_INCREMENT` incorrectly allocates `MAX int64` and `MAX uint64` #12210
  - Fix the issue that the slow query logs are not recorded when the log level is `ERROR` #12373
  - Adjust the number of times that TiDB caches schema changes and corresponding changed table information from 100 to 1024, and support modification by using the `tidb_max_delta_schema_count` system variable #12515
  - Change the query start time from the point of "starting to execute" to "starting to compile" to make SQL statistics more accurate #12638
  - Add the record of `set session autocommit` in TiDB logs #12568
  - Record SQL query start time in `SessionVars` to prevent it from being reset during plan execution #12676
  - Support `?` placeholder in `ORDER BY`, `GROUP BY` and `LIMIT OFFSET` #12514
  - Add the `Prev_stmt` field in slow query logs to output the previous statement when the last statement is `COMMIT` #12724
  - Record the last statement before `COMMIT` into the log when the `COMMIT` fails in an explicitly committed transaction #12747
  - Optimize the saving method of the previous statement when the TiDB server executes a SQL statement to improve performance #12751
  - Fix the panic issue caused by `FLUSH PRIVILEGES` statements under the `skip-grant-table=true` configuration #12816
  - Increase the default minimum step of applying AutoID from `1000` to `30000` to avoid performance bottleneck when there are many write requests in a short time #12891
  - Fix the issue that the failed `Prepared` statement is not print in the error log when TiDB panics #12954

- Fix the issue that the `COM_STMT_FETCH` time record in slow query logs is inconsistent with that in MySQL #12953
- Add an error code in the error message for write conflicts to quickly locate the cause #12878

- DDL
  - Disallow dropping the `AUTO INCREMENT` attribute of a column by default. Modify the value of the `tidb_allow_remove_auto_inc` variable if you do need to drop this attribute. See System Variables for more details. #12146
  - Support multiple `uniques` when creating a unique index in the `Create Table` statement #12469
  - Fix a compatibility issue that if the foreign key constraint in `CREATE TABLE` statement has no schema, schema of the created table should be used instead of returning a `No Database selected` error #12678
  - Fix the issue that the `invalid list index` error is reported when executing `ADMIN CANCEL DDL JOBS` #12681
- Monitor
  - Add types for backoff monitoring and supplement the backoff time that is not recorded before, such as the backoff time when committing #12326
  - Add a new metric to monitor `Add Index` operation progress #12389

16.15.2.2 PD

- Improve the `--help` command output of pd-ctl #1772

16.15.2.3 Tools

- TiDB Binlog
  - Fix the issue that `ALTER DATABASE` related DDL operations cause Drainer to exit abnormally #770
  - Support querying the transaction status information for Commit binlog to improve replication efficiency #761
  - Fix the issue that a Pump panic might occur when Drainer's `start_ts` is greater than Pump's largest `commit_ts` #759

16.15.2.4 TiDB Ansible

- Add two monitoring items "queue size" and "query histogram" for TiDB Binlog #952
- Update TiDB alerting rules #961
- Check the configuration file before the deployment and upgrade #973
- Add a new metric to monitor index speed in TiDB #987
- Update TiDB Binlog monitoring dashboard to make it compatible with Grafana v4.6.3 #993

16.15.3 TiDB 2.1.17 Release Notes

Release date: September 11, 2019

TiDB version: 2.1.17

TiDB Ansible version: 2.1.17

- New features
  - Add the `WHERE` clause in TiDB's `SHOW TABLE REGIONS` syntax
  - Add the `config-check` feature in TiKV and PD to check the configuration items
  - Add the `remove-tombstone` command in pd-ctl to clear tombstone store records
  - Add the `worker-count` and `txn-batch` configuration items in Reparo to control the recovery speed
- Improvements
  - Optimize PD's scheduling process by supporting actively pushing operators
  - Optimize TiKV's starting process to reduce jitters caused by restarting nodes
- Changed behaviors
  - Change `start ts` in TiDB slow query logs from the last retry time to the first execution time
  - Replace the `Index_ids` field in TiDB slow query logs with the `Index_names` field to improve the usability of slow query logs
  - Add the `split-region-max-num` parameter in TiDB's configuration files to modify the maximum number of Regions allowed by the `SPLIT TABLE` syntax, which is increased from 1,000 to 10,000 in the default configuration

16.15.3.1 TiDB

- SQL Optimizer
  - Fix the issue that the error message is not returned correctly when an error occurs during `EvalSubquery` building `Executor` #11811
  - Fix the issue that the query result might be incorrect when the number of rows in the outer table is greater than that in a single batch in Index Lookup Join; expand the functional scope of Index Lookup Join; `UnionScan` can be used as a subnode of `IndexJoin` #11843
  - Add the display of invalid keys (like `invalid encoded key flag 252`) in the `SHOW STAT_BUCKETS` syntax, for the situation where invalid keys might occur during the statistics feedback process #12098
- SQL Execution Engine
  - Fix some incorrect results (like `select cast(13835058000000000000 as` ↪ `double)`) caused by the number value that is first converted to `UINT` when the `CAST` function is converting the number value type #11712

- Fix the issue that the calculation result might be incorrect when the dividend of the `DIV` calculation is a decimal and this calculation contains a negative number #11812
- Add the `ConvertStrToIntStrict` function to fix the MySQL incompatibility issue caused by some strings being converted to the `INT` type when executing the `SELECT/EXPLAIN` statement #11892
- Fix the issue that the `Explain` result might be incorrect caused by wrong configuration of `stmtCtx` when `EXPLAIN ... FOR CONNECTION` is used #11978
- Fix the issue that the result returned by the `unaryMinus` function is incompatible with MySQL, caused by the non-decimal result when the integer result overflows #11990
- Fix the issue that `last_insert_id()` might be incorrect, caused by the counting order when the `LOAD DATA` statement is being executed #11994
- Fix the issue that `last_insert_id()` might be incorrect when the user writes auto-increment column data in an explicit-implicit mixed way #12001
- Fix an over-quoted bug for the `JSON_UNQUOTE` function: only values enclosed by double quote marks (") should be unquoted. For example, the result of "`SELECT JSON_UNQUOTE("\\\\")`" should be "\\" (not changed) #12096

- Server
  - Change `start ts` recorded in slow query logs from the last retry time to the first execution time when retrying TiDB transactions #11878
  - Add the number of keys of a transaction in `LockResolver` to avoid the scan operation on the whole Region and reduce costs of resolving locking when the number of keys is reduced #11889
  - Fix the issue that the `succ` field value might be incorrect in slow query logs #11886
  - Replace the `Index_ids` filed in slow query logs with the `Index_names` field to improve the usability of slow query logs #12063
  - Fix the connection break issue caused by TiDB parsing `-` into EOF Error when `Duration` contains `-` (like `select time( '--' )`) #11910
  - Remove an invalid Region from `RegionCache` more quickly to reduce the number of requests sent to this Region #11931
  - Fix the connection break issue caused by incorrectly handling the OOM panic issue when `oom-action = "cancel"` and OOM occurs in the `Insert Into ... Select` syntax #12126

- DDL
  - Add the reverse scan interface for `tikvSnapshot` to efficiently query DDL history jobs. After using this interface, the execution time of `ADMIN SHOW DDL JOBS` is remarkably decreased #11789
  - Improve the `CREATE TABLE ... PRE_SPLIT_REGION` syntax: change the number of pre-splitting Regions from 2^(N-1) to 2^N when `PRE_SPLIT_REGION = N` #11797

- Decrease the default parameter value for the background worker thread of the `Add Index` operation to avoid great impacts on online workloads #11875
- Improve the `SPLIT TABLE` syntax behavior: generate N data Region(s) and one index Region when `SPLIT TABLE ... REGIONS N` is used to divide Regions #11929
- Add the `split-region-max-num` parameter (`10000` by default) in the configuration file to make the maximum number of Regions allowed by the `SPLIT TABLE` syntax adjustable #12080
- Fix the issue that the `CREATE TABLE` clause cannot be parsed by the downstream MySQL, caused by uncommented `PRE_SPLIT_REGIONS` in this clause when the system writes the binlog #12121
- Add the `WHERE` sub-clause in `SHOW TABLE ⋯ REGIONS` and `SHOW TABLE .. ↪ INDEX ⋯ REGIONS` #12124

- Monitor
  - Add the `connection_transient_failure_count` monitoring metric to count gRPC connection errors of `tikvclient` #12092

16.15.3.2 TiKV

- Fix the incorrect result of counting keys in a Region in some cases #5415
- Add the `config-check` option in TiKV to check whether the TiKV configuration item is valid #5391
- Optimize the starting process to reduce jitters caused by restarting nodes #5277
- Optimize the resolving locking process in some cases to speed up resolving locking for transactions #5339
- Optimize the `get_txn_commit_info` process to speed up committing transactions #5062
- Simplify Raft-related logs #5425
- Resolve the issue that TiKV exits abnormally in some cases #5441

16.15.3.3 PD

- Add the `config-check` option in PD to check whether the PD configuration item is valid #1725
- Add the `remove-tombstone` command in pd-ctl to support clearing tombstone store records #1705
- Support actively pushing operators to speed up scheduling #1686

16.15.3.4 Tools

- TiDB Binlog
  - Add `worker-count` and `txn-batch` configuration items in Reparo to control the recovery speed #746

- Optimize the memory usage of Drainer to improve the parallel execution efficiency #735
- Fix the bug that Pump cannot quit normally in some cases #739
- Optimize the processing logic of `LevelDB` in Pump to improve the execution efficiency of GC #720

- TiDB Lightning
  - Fix the bug that tidb-lightning might crash caused by re-importing data from the checkpoint #239

16.15.3.5 TiDB Ansible

- Update the Spark version to 2.4.3, and update the TiSpark version to 2.2.0 that is compatible with Spark 2.4.3 #914, #919
- Fix the issue that there is a long waiting time when the remote machine password has expired #937, #948

16.15.4 TiDB 2.1.16 Release Notes

Release date: August 15, 2019

TiDB version: 2.1.16

TiDB Ansible version: 2.1.16

16.15.4.1 TiDB

- SQL Optimizer
  - Fix the issue that row count is estimated inaccurately for the equal condition on the time column #11526
  - Fix the issue that `TIDB_INLJ` Hint does not take effect or take effect on the specified table #11361
  - Change the implementation of `NOT EXISTS` in a query from OUTER JOIN to ANTI JOIN to find a more optimized execution plan #11291
  - Support subqueries within `SHOW` statements, allowing syntaxes such as `SHOW ↪ COLUMNS FROM tbl WHERE FIELDS IN (SELECT 'a')` #11461
  - Fix the issue that the `SELECT ⋯ CASE WHEN ⋯ ELSE NULL ...` query gets an incorrect result caused by the constant folding optimization #11441

- SQL Execution Engine
  - Fix the issue that the `DATE_ADD` function gets a wrong result when `INTERVAL` is negative #11616
  - Fix the issue that the `DATE_ADD` function might get an incorrect result because it performs type conversion wrongly when it accepts an argument of the `FLOAT`, `DOUBLE`, or `DECIMAL` type #11628

- Fix the issue that the error message is inaccurate when CAST(JSON AS SIGNED) overflows #11562
- Fix the issue that other child nodes are not closed when one child node fails to be closed and returns an error during the process of closing Executor #11598
- Support `SPLIT TABLE` statements that return the number of Regions that are successfully split and a finished percentage rather than an error when the scheduling is not finished for Region scatter before the timeout #11487
- Make `REGEXP BINARY` function case sensitive to be compatible with MySQL #11505
- Fix the issue that `NULL` is not returned correctly because the value of `YEAR` in the `DATE_ADD/DATE_SUB` result overflows when it is smaller than 0 or larger than 65535 #11477
- Add in the slow query table a `Succ` field that indicates whether the execution succeeds #11412
- Fix the MySQL incompatibility issue caused by fetching the current timestamp multiple times when a SQL statement involves calculations of the current time (such as `CURRENT_TIMESTAMP` or `NOW`) #11392
- Fix the issue that the AUTO_INCREMENT columns do not handle the FLOAT or DOUBLE type #11389
- Fix the issue that `NULL` is not returned correctly when the `CONVERT_TZ` function accepts an invalid argument #11357
- Fix the issue that an error is reported by the `PARTITION BY LIST` statement. (Currently only the syntax is supported; when TiDB executes the statement, a regular table is created and a prompting message is provided) #11236
- Fix the issue that `Mod(%)`, `Multiple(*)`, and `Minus(-)` operations return an inconsistent `0` result with that in MySQL when there are many decimal digits (such as `select 0.000 % 0.11234500000000000000`) #11353

- Server
    - Fix the issue that the plugin gets a `NULL` domain when `OnInit` is called back #11426
    - Fix the issue that the table information in a schema can still be obtained through the HTTP interface after the schema has been deleted #11586

- DDL
    - Disallow dropping indexes on auto-increment columns to avoid incorrect results of the auto-increment columns caused by this operation #11402
    - Fix the issue that the character set of the column is not correct when creating and modifying the table with different character sets and collations #11423
    - Fix the issue that the column schema might get wrong when `alter table ... ↪ set default...` and another DDL statement that modifies this column are executed in parallel #11374
    - Fix the issue that data fails to be backfilled when Generated Column A depends on Generated Column B and A is used to create an index #11538
    - Speed up `ADMIN CHECK TABLE` operations #11538

16.15.4.2 TiKV

- Support returning an error message when the client accesses a TiKV Region that is being closed #4820
- Support reverse `raw_scan` and `raw_batch_scan` interfaces #5148

16.15.4.3 Tools

- TiDB Binlog
  - Add the `ignore-txn-commit-ts` configuration item in Drainer to skip executing some statements in a transaction #697
  - Add the configuration item check on startup, which stops Pump and Drainer from running and returns an error message when meeting invalid configuration items #708
  - Add the `node-id` configuration in Drainer to specify Drainer's node ID #706
- TiDB Lightning
  - Fix the issue that `tikv_gc_life_time` fails to be changed back to its original value when 2 checksums are running at the same time #224

16.15.4.4 TiDB Ansible

- Add the `log4j` configuration file in Spark #842
- Update the tispark jar package to v2.1.2 #863
- Fix the issue that the Prometheus configuration file is generated in the wrong format when TiDB Binlog uses Kafka or ZooKeeper #845
- Fix the bug that PD fails to switch the Leader when executing the `rolling_update.yml` operation #888
- Optimize the logic of rolling updating PD nodes - upgrade Followers first and then the Leader - to improve stability #895

16.15.5 TiDB 2.1.15 Release Notes

Release date: July 16, 2019

TiDB version: 2.1.15

TiDB Ansible version: 2.1.15

16.15.5.1 TiDB

- Fix the issue that the `DATE_ADD` function returns wrong results due to incorrect alignment when dealing with microseconds #11289
- Fix the issue that an error is reported when the empty value in the string column is compared with `FLOAT` or `INT` #11279
- Fix the issue that the `INSERT` function fails to correctly return the `NULL` value when a parameter is `NULL` #11249
- Fix the issue that an error occurs when indexing the column of the non-string type and `0` length #11215
- Add the `SHOW TABLE REGIONS` statement to query the Region distribution of a table through SQL statements #11238
- Fix the issue that an error is reported when using the `UPDATE ··· SELECT` statement because the projection elimination is used to optimize rules in the `SELECT` subqueries #11254
- Add the `ADMIN PLUGINS ENABLE/ADMIN PLUGINS DISABLE` SQL statement to dynamically enable or disable plugins #11189
- Add the session connection information in the Audit plugin #11189
- Fix the panic issue that happens when a column is queried on multiple times and the returned result is `NULL` during point queries #11227
- Add the `tidb_scatter_region` configuration item to scatter table Regions when creating a table #11213
- Fix the data race issue caused by non-thread safe `rand.Rand` when using the `RAND` function #11170
- Fix the issue that the comparison result of integers and non-integers is incorrect in some cases #11191
- Support modifying the collation of a database or a table, but the character set of the database/table has to be UTF-8 or utf8mb4 #11085
- Fix the issue that the precision shown by the `SHOW CREATE TABLE` statement is incomplete when `CURRENT_TIMESTAMP` is used as the default value of the column and the float precision is specified #11087

16.15.5.2 TiKV

- Unify the log format #5083
- Improve the accuracy of Region's approximate size or keys in extreme cases to improve the accuracy of scheduling #5085

16.15.5.3 PD

- Unify the log format #1625

16.15.5.4 Tools

TiDB Binlog

- Optimize the Pump GC strategy and remove the restriction that the unconsumed binlog cannot be cleaned to make sure that the resources are not occupied for a long time #663

TiDB Lightning

- Fix the import error that happens when the column names specified by the SQL dump are not in lowercase #210

16.15.5.5 TiDB Ansible

- Add the `parse duration` and `compile duration` monitoring items in TiDB Dashboard to monitor the time that it takes to parse SQL statements and execute compilation #815

16.15.6 TiDB 2.1.14 Release Notes

Release date: July 04, 2019

TiDB version: 2.1.14

TiDB Ansible version: 2.1.14

16.15.6.1 TiDB

- Fix wrong query results caused by column pruning in some cases #11019
- Fix the wrongly displayed information in `db` and `info` columns of `show processlist` #11000
- Fix the issue that `MAX_EXECUTION_TIME` as a SQL hint and global variable does not work in some cases #10999
- Support automatically adjust the incremental gap allocated by auto-increment ID based on the load #10997
- Fix the issue that the `Distsql` memory information of `MemTracker` is not correctly cleaned when a query ends #10971
- Add the `MEM` column in the `information_schema.processlist` table to describe the memory usage of a query #10896
- Add the `max_execution_time` global system variable to control the maximum execution time of a query #10940
- Fix the panic caused by using unsupported aggregate functions #10911
- Add an automatic rollback feature for the last transaction when the `load data` statement fails #10862

- Fix the issue that TiDB returns a wrong result in some cases when the `OOMAction` configuration item is set to `Cancel` #11016
- Disable the `TRACE` statement to avoid the TiDB panic issue #11039
- Add the `mysql.expr_pushdown_blacklist` system table that dynamically enables/disables pushing down specific functions to Coprocessor #10998
- Fix the issue that the `ANY_VALUE` function does not work in the `ONLY_FULL_GROUP_BY` mode #10994
- Fix the incorrect evaluation caused by not doing a deep copy when evaluating the user variable of the string type #11043

16.15.6.2 TiKV

- Optimize processing the empty callback when processing the Raftstore message to avoid sending unnecessary message #4682

16.15.6.3 PD

- Adjust the log output level from `Error` to `Warning` when reading an invalid configuration item #1577

16.15.6.4 Tools

TiDB Binlog

- Reparo
  - Add the `safe-mode` configuration item, and support importing duplicated data after this item is enabled #662
- Pump
  - Add the `stop-write-at-available-space` configuration item to limit the available binlog space #659
  - Fix the issue that Garbage Collector does not work sometimes when the number of LevelDB L0 files is 0 #648
  - Optimize the algorithm of deleting log files to speed up releasing the space #648
- Drainer
  - Fix the failure to update `BIT` columns in the downstream #655

16.15.6.5 TiDB Ansible

- Add the precheck feature for the `ansible` command and its `jmespath` and `jinja2` dependency packages #807
- Add the `stop-write-at-available-space` parameter (10 GiB by default) in Pump, and stop writing binlog files in Pump when the available disk space is less than the parameter value #807

16.15.7 TiDB 2.1.13 Release Notes

Release date: June 21, 2019

TiDB version: 2.1.13

TiDB Ansible version: 2.1.13

16.15.7.1 TiDB

- Add a feature to use `SHARD_ROW_ID_BITS` to scatter row IDs when the column contains an `AUTO_INCREMENT` attribute to relieve the hotspot issue #10788
- Optimize the lifetime of invalid DDL metadata to speed up recovering the normal execution of DDL operations after upgrading the TiDB cluster #10789
- Fix the OOM issue in high concurrent scenarios caused by the failure to quickly release Coprocessor resources, resulted from the `execdetails.ExecDetails` pointer #10833
- Add the `update-stats` configuration item to control whether to update statistics #10772
- Add the following TiDB-specific syntax to support Region presplit to solve the hotspot issue:
- Add the `PRE_SPLIT_REGIONS` table option #10863
- Add the `SPLIT TABLE table_name INDEX index_name` syntax #10865
- Add the `SPLIT TABLE [table_name] BETWEEN (min_value...)AND (max_value ...)REGIONS [region_num]` syntax #10882
- Fix the panic issue caused by the `KILL` syntax in some cases #10879
- Improve the compatibility with MySQL for `ADD_DATE` in some cases #10718
- Fix the wrong estimation for the selectivity rate of the inner table selection in index join #10856

16.15.7.2 TiKV

- Fix the issue that incomplete snapshots are generated in the system caused by the iterator not checking the status #4940
- Add a feature to check the validity for the `block-size` configuration #4930

16.15.7.3 Tools

- TiDB Binlog
    - Fix the wrong offset issue caused by Pump not checking the returned value when it fails to write data #640
    - Add the `advertise-addr` configuration in Drainer to support the bridge mode in the container environment #634

16.15.8 TiDB 2.1.12 Release Notes

Release date: June 13, 2019

TiDB version: 2.1.12

TiDB Ansible version: 2.1.12

16.15.8.1 TiDB

- Fix the issue caused by unmatched data types when using the index query feedback #10755
- Fix the issue that the blob column is changed to the text column caused by charset altering in some cases #10745
- Fix the issue that the `GRANT` operation in the transaction mistakenly reports "Duplicate Entry" in some cases #10739
- Improve the compatibility with MySQL of the following features
    - The `DAYNAME` function #10732
    - The `MONTHNAME` function #10733
    - Support the 0 value for the `EXTRACT` function when processing `MONTH` #10702
    - The `DECIMAL` type can be converted to `TIMESTAMP` or `DATETIME` #10734
- Change the column charset while changing the table charset #10714
- Fix the overflow issue when converting a decimal to a float in some cases #10730
- Fix the issue that some extremely large messages report the "grpc: received message larger than max" error caused by inconsistent maximum sizes of messages sent/received by gRPC of TiDB and TiKV #10710
- Fix the panic issue caused by `ORDER BY` not filtering NULL in some cases #10488
- Fix the issue that values returned by the `UUID` function might be duplicate when multiple nodes exist #10711
- Change the value returned by `CAST(-num as datetime)` from `error` to NULL #10703
- Fix the issue that an unsigned histogram meets signed ranges in some cases #10695
- Fix the issue that an error is reported mistakenly for reading data when the statistics feedback meets the bigint unsigned primary key #10307
- Fix the issue that the result of `Show Create Table` for partitioned tables is not correctly displayed in some cases #10690
- Fix the issue that the calculation result of the `GROUP_CONCAT` aggregate function is not correct for some correlated subqueries #10670
- Fix the issue that the result is wrongly displayed when the memory table of slow queries parses the slow query log in some cases #10776

16.15.8.2 PD

- Fix the issue that etcd leader election is blocked in extreme conditions #1576

16.15.8.3 TiKV

- Fix the issue that Regions are not available during the leader transfer process in extreme conditions #4799
- Fix the issue that TiKV loses data when the power of the machine fails abnormally, caused by delayed data flush to the disk when receiving snapshots #4850

16.15.9 TiDB 2.1.11 Release Notes

Release date: June 03, 2019

TiDB version: 2.1.11

TiDB Ansible version: 2.1.11

16.15.9.1 TiDB

- Fix the issue that incorrect schema is used for `delete from join` #10595
- Fix the issue that the built-in `CONVERT()` may return incorrect field type #10263
- Merge non-overlapped feedback when updating bucket count #10569
- Fix calculation errors of `unix_timestamp()-unix_timestamp(now())` #10491
- Fix the incompatibility issue of `period_diff` with MySQL 8.0 #10501
- Skip `Virtual Column` when collecting statistics to avoid exceptions #10628
- Support the `SHOW OPEN TABLES` statement #10374
- Fix the issue that goroutine leak may happen in some cases #10656
- Fix the issue that setting the `tidb_snapshot` variable in some cases may cause incorrect parsing of time format #10637

16.15.9.2 PD

- Fix the issue that hots Region may fail to be scheduled due to `balance-region` #1551
- Set hotspot related scheduling priorities to high #1551
- Add two configuration items #1551
    - `hot-region-schedule-limit` to control the maximum number of concurrent hotspot scheduling tasks
    - `hot-region-cache-hits-threshold` to identify a hot Region

16.15.9.3 TiKV

- Fix the issue that the learner reads an empty index when there is only one leader and one learner #4751
- Process `ScanLock` and `ResolveLock` in the thread pool with a high priority to reduce their impacts on commands with a normal priority #4791
- Sync all files of received snapshots #4811

16.15.9.4 Tools

- TiDB Binlog
  - Limit data deletion speed during GC to avoid QPS degrading caused by `WritePause` #620

16.15.9.5 TiDB Ansible

- Add Drainer parameters #760

16.15.10 TiDB 2.1.10 Release Notes

Release date: May 22, 2019

TiDB version: 2.1.10

TiDB Ansible version: 2.1.10

16.15.10.1 TiDB

- Fix the issue that some abnormalities cause incorrect table schema when using `tidb_snapshot` to read the history data #10359
- Fix the issue that the `NOT` function causes wrong read results in some cases #10363
- Fix the wrong behavior of `Generated Column` in the `Replace` or `Insert on duplicate update` statement #10385
- Fix a bug of the `BETWEEN` function in the `DATE/DATETIME` comparison #10407
- Fix the issue that a single line of a slow log that is too long causes an error report when using the `SLOW_QUERY` table to query a slow log #10412
- Fix the issue that the result of `DATETIME` plus `INTERVAL` is not the same with that of MySQL in some cases #10416, #10418
- Add the check for the invalid time of February in a leap year #10417
- Execute the internal initialization operation limitation only in the DDL owner to avoid a large number of conflict error reports when initializing the cluster #10426
- Fix the issue that `DESC` is incompatible with MySQL when the default value of the output timestamp column is `default current_timestamp on update current_timestamp` #10337
- Fix the issue that an error occurs during the privilege check in the `Update` statement #10439
- Fix the issue that wrong calculation of `RANGE` causes a wrong result in the `CHAR` column in some cases #10455
- Fix the issue that the data might be overwritten after decreasing `SHARD_ROW_ID_BITS` #9868
- Fix the issue that `ORDER BY RAND()` does not return random numbers #10064
- Prohibit the `ALTER` statement modifying the precision of decimals #10458

- Fix the compatibility issue of the `TIME_FORMAT` function with MySQL #10474
- Check the parameter validity of `PERIOD_ADD` #10430
- Fix the issue that the behavior of the invalid `YEAR` string in TiDB is incompatible with that in MySQL #10493
- Support the `ALTER DATABASE` syntax #10503
- Fix the issue that the `SLOW_QUERY` memory engine reports an error when no `;` exists in the slow query statement #10536
- Fix the issue that the `Add index` operation in partitioned tables cannot be canceled in some cases #10533
- Fix the issue that the OOM panic cannot be recovered in some cases #10545
- Improve the security of the DDL operation rewriting the table metadata #10547

16.15.10.2 PD

- Fix the issue that the priority of the leader does not take effect #1533

16.15.10.3 TiKV

- Reject transferring the leader in a Region whose configuration has been changed recently to avoid transfer failure #4684
- Add the priority label for Coprocessor metrics #4643
- Fix the possible dirty read issue during transferring the leader #4724
- Fix the issue that `CommitMerge` causes the restart failure of TiKV in some cases #4615
- Fix unknown logs #4730

16.15.10.4 Tools

- TiDB Lightning
  - Add the retry feature when TiDB Lightning fails to send data to `importer` #176
- TiDB Binlog
  - Optimize the Pump storage log to facilitate troubleshooting #607

16.15.10.5 TiDB Ansible

- Update the configuration file of TiDB Lightning and add the `tidb_lightning_ctl` script #d3a4a368

16.15.11 TiDB 2.1.9 Release Notes

Release date: May 6, 2019

TiDB version: 2.1.9

TiDB Ansible version: 2.1.9

16.15.11.1 TiDB

- Fix compatibility of the `MAKETIME` function when unsigned type overflows #10089
- Fix the stack overflow caused by constant folding in some cases #10189
- Fix the privilege check issue for `Update` when an alias exists in some cases #10157, #10326
- Track and control memory usage in DistSQL #10197
- Support specifying collation as `utf8mb4_0900_ai_ci` #10201
- Fix the wrong result issue of the `MAX` function when the primary key is of the Unsigned type #10209
- Fix the issue that NULL values can be inserted into NOT NULL columns in the non-strict SQL mode #10254
- Fix the wrong result issue of the `COUNT` function when multiple columns exist in `DISTINCT` #10270
- Fix the panic issue occurred when `LOAD DATA` parses irregular CSV files #10269
- Ignore the overflow error when the outer and inner join key types are inconsistent in `Index Lookup Join` #10244
- Fix the issue that a statement is wrongly judged as point-get in some cases #10299
- Fix the wrong result issue when the time type does not convert the time zone in some cases #10345
- Fix the issue that TiDB character set cases are inconsistent in some cases #10354
- Support controlling the number of rows returned by operator #9166
    - Selection & Projection #10110
    - `StreamAgg` & `HashAgg` #10133
    - `TableReader` & `IndexReader` & `IndexLookup` #10169
- Improve the slow query log
    - Add `SQL Digest` to distinguish similar SQL #10093
    - Add version information of statistics used by slow query statements #10220
    - Show memory consumption of a statement in slow query log #10246
    - Adjust the output format of Coprocessor related information so it can be parsed by pt-query-digest #10300
    - Fix the `#` character issue in slow query statements #10275
    - Add some information columns to the memory table of slow query statements #10317
    - Add the transaction commit time to slow query log #10310
    - Fix the issue some time formats cannot be parsed by pt-query-digest #10323

16.15.11.2 PD

- Support the GetOperator service #1514

16.15.11.3 TiKV

- Fix potential quorum changes when transferring leader #4604

16.15.11.4 Tools

- TiDB Binlog
  - Fix the issue that data replication is interrupted because data in the unsigned int type of primary key column are minus numbers #574
  - Remove the compression option when the downstream is `pb` and change the downstream name from `pb` to `file` #597
  - Fix the bug that Reparo introduced in 2.1.7 generates wrong `UPDATE` statements #576
- TiDB Lightning
  - Fix the bug that the bit type of column data is incorrectly parsed by the parser #164
  - Fill the lacking column data in dump files using row id or the default column value #174
  - Fix the Importer bug that some SST files fail to be imported but it still returns successful import result #4566
  - Support setting a speed limit in Importer when uploading SST files to TiKV #4607
  - Change Importer RocksDB SST compression method to `lz4` to reduce CPU consumption #4624
- sync-diff-inspector
  - Support checkpoint #227

16.15.11.5 TiDB Ansible

- Update links in tidb-ansible documentation according to docs refactoring #740, #741
- Remove the `enable_slow_query_log` parameter in the `inventory.ini` file and output the slow query log to a separate log file by default #742

16.15.12 TiDB 2.1.8 Release Notes

Release date: April 12, 2019

TiDB version: 2.1.8

TiDB Ansible version: 2.1.8

16.15.12.1 TiDB

- Fix the issue that the processing logic of `GROUP_CONCAT` function is incompatible with MySQL when there is a NULL-valued parameter #9930
- Fix the equality check issue of decimal values in the `Distinct` mode #9931

- Fix the collation compatibility issue of the date, datetime, and timestamp types for the `SHOW FULL COLUMNS` statement
    - #9938
    - #10114
- Fix the issue that the row count estimation is inaccurate when the filtering condition contains correlated columns #9937
- Fix the compatibility issue between the `DATE_ADD` and `DATE_SUB` functions
    - #9963
    - #9966
- Support the `%H` format for the `STR_TO_DATE` function to improve compatibility #9964
- Fix the issue that the result is wrong when the `GROUP_CONCAT` function groups by a unique index #9969
- Return a warning when the Optimizer Hints contains an unmatched table name #9970
- Unify the log format to facilitate collecting logs using tools for analysis Unified Log Format
- Fix the issue that a lot of NULL values cause inaccurate statistics estimation #9979
- Fix the issue that an error is reported when the default value of the TIMESTAMP type is the boundary value #9987
- Validate the value of `time_zone` #10000
- Support the `2019.01.01` time format #10001
- Fix the issue that the row count estimation is displayed incorrectly in the result returned by the `EXPLAIN` statement in some cases #10044
- Fix the issue that `KILL TIDB [session id]` cannot instantly stop the execution of a statement in some cases #9976
- Fix the predicate pushdown issue of constant filtering conditions in some cases #10049
- Fix the issue that a read-only statement is not processed correctly in some cases #10048

16.15.12.2 PD

- Fix the issue that `regionScatterer` might generate an invalid `OperatorStep` #1482
- Fix the issue that a hot store makes incorrect statistics of keys #1487
- Fix the too short timeout issue of the `MergeRegion` operator #1495
- Add elapsed time metrics of the PD server handling TSO requests #1502

16.15.12.3 TiKV

- Fix the issue of wrong statistics of the read traffic #4441
- Fix the raftstore performance issue when too many Regions exist #4484
- Do not ingest files when the number of level 0 SST files exceeds `level_zero_slowdown_writes_trigg`
  ↪ `/2` #4464

16.15.12.4 Tools

- Optimize the order of importing tables for Lightning to reduce the effects of large tables executing `Checksum` and `Analyze` on the cluster during the importing process and improve the success rate of `Checksum` and `Analyze` #156
- Improve the encoding SQL performance by 50% for Lightning by directly parsing the data source file content to `types.Datum` of TiDB to avoid additional parsing working of the KV encoder #145
- Add the `storage.sync-log` configuration item in TiDB Binlog Pump to support flushing disks of the local storage asynchronously in Pump #529
- Support traffic compression of communication between TiDB Binlog Pump and Drainer #530
- Add the `syncer.sql-mode` configuration item in TiDB Binlog Drainer to support using different `sql-mode`s to parse DDL queries #513
- Add the `syncer.ignore-table` configuration item in TiDB Binlog Drainer to support filtering tables not to be replicated #526

16.15.12.5 TiDB Ansible

- Modify the version limit for the operating system and only support CentOS 7.0 or later and Red Hat 7.0 or later #734
- Add the feature of checking whether `epollexclusive` is supported in every OS #728
- Add the version limit for rolling update to prohibit upgrading a version of 2.0.1 or earlier to a version of 2.1 or later #728

16.15.13 TiDB 2.1.7 Release Notes

Release Date: March 28, 2019

TiDB version: 2.1.7

TiDB Ansible version: 2.1.7

16.15.13.1 TiDB

- Fix the issue of longer startup time when upgrading the program caused by canceling DDL operations #9768
- Fix the issue that the `check-mb4-value-in-utf8` configuration item is in the wrong position in the `config.example.toml` file #9852
- Improve the compatibility of the `str_to_date` built-in function with MySQL #9817
- Fix the compatibility issue of the `last_day` built-in function #9750
- Add the `tidb_table_id` column for `infoschema.tables` to facilitate getting `table_id` ↪ by using SQL statements and add the `tidb_indexes` system table to manage the relationship between Table and Index #9862

- Add a check about the null definition of Table Partition #9663
- Change the privileges required by `Truncate Table` from `Delete` to `Drop` to make it consistent with MySQL #9876
- Support using subqueries in the `DO` statement #9877
- Fix the issue that the `default_week_format` variable does not take effect in the `week` function #9753
- Support the plugin framework #9880, #9888
- Support checking the enabling state of binlog by using the `log_bin` system variable #9634
- Support checking the Pump/Drainer status by using SQL statements #9896
- Fix the compatibility issue about checking mb4 character on utf8 when upgrading TiDB #9887
- Fix the panic issue when the aggregate function calculates JSON data in some cases #9927

16.15.13.2 PD

- Fix the issue that the transferring leader step cannot be created in the balance-region when the number of replicas is one #1462

16.15.13.3 Tools

- Support replicating generated columns by using binlog

16.15.13.4 TiDB Ansible

Change the default retention time of Prometheus monitoring data to 30d

16.15.14 TiDB 2.1.6 Release Notes

On March 15, 2019, TiDB 2.1.6 is released. The corresponding TiDB Ansible 2.1.6 is also released. Compared with TiDB 2.1.5, this release has greatly improved the stability, the SQL optimizer, statistics, and the execution engine.

16.15.14.1 TiDB

- SQL Optimizer/Executor
    - Optimize planner to select the outer table based on cost when both tables are specified in Hint of `TIDB_INLJ` #9615
    - Fix the issue that `IndexScan` cannot be selected correctly in some cases #9587
    - Fix incompatibility with MySQL of check in the `agg` function in subqueries #9551
    - Make `show stats_histograms` only output valid columns to avoid panics #9502

- Server
    - Support the `log_bin` variable to enable/disable Binlog #9634
    - Add a sanity check for transactions to avoid false transaction commit #9559
    - Fix the issue that setting variables may lead to panic #9539
- DDL
    - Fix the issue that the `Create Table Like` statement causes panic in some cases #9652
    - Enable the `AutoSync` feature of etcd clients to avoid connection issues between TiDB and etcd in some cases #9600

16.15.14.2 TiKV

- Fix the issue that a `protobuf` parsing failure would in some cases cause a `StoreNotMatch` error #4303

16.15.14.3 Tools

- Lightning
    - Change the default `region-split-size` of importer to 512 MiB #4369
    - Save the intermediate SST previously cached in memory to the local disk to reduce memory usage #4369
    - Limit the memory usage of RocksDB #4369
    - Fix the issue that Regions are scattered before scheduling is finished #4369
    - Separate importing of data and indexes for large tables to effectively reduce time consumption when importing in batches #132
    - Support CSV #111
    - Fix the error of import failure due to non-alphanumeric characters in schema names #9547

16.15.15 TiDB 2.1.5 Release Notes

On February 28, 2019, TiDB 2.1.5 is released. The corresponding TiDB Ansible 2.1.5 is also released. Compared with TiDB 2.1.4, this release has greatly improved the stability, the SQL optimizer, statistics, and the execution engine.

16.15.15.1 TiDB

- SQL Optimizer/Executor
    - Make `SHOW CREATE TABLE` do not print the column charset information when the charset information of a column is the same with that of a table, to improve the compatibility of `SHOW CREATE TABLE` with MySQL #9306

- Fix the panic or the wrong result of the `Sort` operator in some cases by extracting `ScalarFunc` from `Sort` to a `Projection` operator for computing to simplify the computing logic of `Sort` #9319
- Remove the sorting field with constant values in the `Sort` operator #9335, #9440
- Fix the data overflow issue when inserting data into an unsigned integer column #9339
- Set `cast_as_binary` to `NULL` when the length of the target binary exceeds `max_allowed_packet` #9349
- Optimize the constant folding process of `IF` and `IFNULL` #9351
- Optimize the index selection of TiDB using skyline pruning to improve the stability of simple queries #9356
- Support computing the selectivity of the `DNF` expression #9405
- Fix the wrong SQL query result of `!=ANY()` and `=ALL()` in some cases #9403
- Fix the panic or the wrong result when the Join Key types of two tables on which the `Merge Join` operation is performed are different #9438
- Fix the issue that the result of the `RAND()` function is not compatible with MySQL #9446
- Refactor the logic of `Semi Join` processing `NULL` and the empty result set to get the correct result and improve the compatibility with MySQL #9449

- Server
    - Add the `tidb_constraint_check_in_place` system variable to check the data uniqueness constraint when executing the `INSERT` statement #9401
    - Fix the issue that the value of the `tidb_force_priority` system variable is different from that set in the configuration file #9347
    - Add the `current_db` field in general logs to print the name of the currently used database #9346
    - Add an HTTP API of obtaining the table information with the table ID #9408
    - Fix the issue that `LOAD DATA` loads incorrect data in some cases #9414
    - Fix the issue that it takes a long time to build a connection between the MySQL client and TiDB in some cases #9451

- DDL
    - Fix some issues when canceling the `DROP COLUMN` operation #9352
    - Fix some issues when canceling the `DROP` or `ADD` partitioned table operation #9376
    - Fix the issue that `ADMIN CHECK TABLE` mistakenly reports the data index inconsistency in some cases #9399
    - Fix the time zone issue of the `TIMESTAMP` default value #9108

16.15.15.2 PD

- Provide the `exclude_tombstone_stores` option in the `GetAllStores` interface to remove the Tombstone store from the returned result #1444

16.15.15.3 TiKV

- Fix the issue that Importer fails to import data in some cases #4223
- Fix the `KeyNotInRegion` error in some cases #4125
- Fix the panic issue caused by Region merge in some cases #4235
- Add the detailed `StoreNotMatch` error message #3885

16.15.15.4 Tools

- Lightning
  - Do not report an error or exit when a Tombstone store exists in the cluster #4223
- TiDB Binlog
  - Update the DDL binlog replication plan to guarantee the correctness of DDL event replication #9304

16.15.16 TiDB 2.1.4 Release Notes

On February 15, 2019, TiDB 2.1.4 is released. The corresponding TiDB Ansible 2.1.4 is also released. Compared with TiDB 2.1.3, this release has greatly improved the stability, the SQL optimizer, statistics, and the execution engine.

16.15.16.1 TiDB

- SQL Optimizer/Executor
  - Fix the issue that the `VALUES` function does not handle the FLOAT type correctly #9223
  - Fix the wrong result issue when casting Float to String in some cases #9227
  - Fix the wrong result issue of the `FORMAT` function in some cases #9235
  - Fix the panic issue when handling the Join query in some cases #9264
  - Fix the issue that the `VALUES` function does not handle the ENUM type correctly #9280
  - Fix the wrong result issue of `DATE_ADD`/`DATE_SUB` in some cases #9284
- Server
  - Optimize the "reload privilege success" log and change it to the DEBUG level #9274
- DDL
  - Change `tidb_ddl_reorg_worker_cnt` and `tidb_ddl_reorg_batch_size` to global variables #9134
  - Fix the bug caused by adding an index to a generated column in some abnormal conditions #9289

16.15.16.2 TiKV

- Fix the duplicate write issue when closing TiKV #4146
- Fix the abnormal result issue of the event listener in some cases #4132

16.15.16.3 Tools

- Lightning
  - Optimize the memory usage #107, #108
  - Remove the chunk separation of dump files to avoid an extra parsing of dump files #109
  - Limit the I/O concurrency of reading dump files, to avoid performance degradation caused by too many cache misses #110
  - Support importing data in batches for a single table, to improve import stability #110
  - Enable auto compactions in the import mode in TiKV #4199
  - Support disabling the TiKV periodic Level-1 compaction parameter, because the Level-1 compaction is automatically executed in the import mode when the TiKV cluster version is 2.1.4 or later #119
  - Limit the number of import engines to avoid consuming too much importer disk space #119
- Support splitting chunks using the TiDB statistics in sync-diff-inspector #197

16.15.17 TiDB 2.1.3 Release Notes

On January 28, 2019, TiDB 2.1.3 is released. The corresponding TiDB Ansible 2.1.3 is also released. Compared with TiDB 2.1.2, this release has great improvement in system stability, SQL optimizer, statistics information, and execution engine.

16.15.17.1 TiDB

- SQL Optimizer/Executor
  - Fix the panic issue of Prepared Plan Cache in some cases #8826
  - Fix the issue that Range computing is wrong when the index is a prefix index #8851
  - Make `CAST(str AS TIME(N))` return null if the string is in the illegal `TIME` format when `SQL_MODE` is not strict #8966
  - Fix the panic issue of Generated Column during the process of `UPDATE` in some cases #8980
  - Fix the upper bound overflow issue of the statistics histogram in some cases #8989
  - Support Range for `_tidb_rowid` construction queries, to avoid full table scan and reduce cluster stress #9059

- Return an error when the `CAST(AS TIME)` precision is too big #9058
- Allow using `Sort Merge Join` in the Cartesian product #9037
- Fix the issue that the statistics worker cannot resume after the panic in some cases #9085
- Fix the issue that `Sort Merge Join` returns the wrong result in some cases #9046
- Support returning the JSON type in the `CASE` clause #8355

- Server

  - Return a warning instead of an error when the non-TiDB hint exists in the comment #8766
  - Verify the validity of the configured TIMEZONE value #8879
  - Optimize the `QueryDurationHistogram` metrics item to display more statement types #8875
  - Fix the lower bound overflow issue of bigint in some cases #8544
  - Support the `ALLOW_INVALID_DATES` SQL mode #9110

- DDL

  - Fix a `RENAME TABLE` compatibility issue to keep the behavior consistent with that of MySQL #8808
  - Support making concurrent changes of `ADD INDEX` take effect immediately #8786
  - Fix the `UPDATE` panic issue during the process of `ADD COLUMN` in some cases #8906
  - Fix the issue of concurrently creating Table Partition in some cases #8902
  - Support converting the `utf8` character set to `utf8mb4` #8951 #9152
  - Fix the issue of Shard Bits overflow #8976
  - Support outputting the column character sets in `SHOW CREATE TABLE` #9053
  - Fix the issue of the maximum length limit of the varchar type column in `utf8mb4` #8818
  - Support `ALTER TABLE TRUNCATE TABLE PARTITION` #9093
  - Resolve the charset when the charset is not provided #9147

16.15.17.2 PD

- Fix the Watch issue related to leader election #1396

16.15.17.3 TiKV

- Support obtaining the monitoring information using the HTTP method #3855
- Fix the NULL issue of `data_format` #4075
- Add verifying the range for scan requests #4124

16.15.17.4 Tools

- TiDB Binlog

- Fix the `no available pump` issue while TiDB is started or restarted #157
- Enable outputting the Pump client log #165
- Fix the data inconsistency issue caused by the unique key containing the NULL value when the table only has the unique key and does not have the primary key

16.15.18 TiDB 2.1.2 Release Notes

On December 22, 2018, TiDB 2.1.2 is released. The corresponding TiDB Ansible 2.1.2 is also released. Compared with TiDB 2.1.1, this release has great improvement in system compatibility and stability.

16.15.18.1 TiDB

- Make TiDB compatible with TiDB Binlog of the Kafka version #8747
- Improve the exit mechanism of TiDB in a rolling update #8707
- Fix the panic issue caused by adding the index for the generated column in some cases #8676
- Fix the issue that the optimizer cannot find the optimal query plan when `TIDB_SMJ` ↪ `Hint` exists in the SQL statement in some cases #8729
- Fix the issue that `AntiSemiJoin` returns an incorrect result in some cases #8730
- Improve the valid character check of the `utf8` character set #8754
- Fix the issue that the field of the time type might return an incorrect result when the write operation is performed before the read operation in a transaction #8746

16.15.18.2 PD

- Fix the Region information update issue about Region merge #1377

16.15.18.3 TiKV

- Support the configuration format in the unit of `DAY` (`d`) and fix the configuration compatibility issue #3931
- Fix the possible panic issue caused by `Approximate Size Split` #3942
- Fix two issues about Region merge #3822, #3873

16.15.18.4 Tools

- TiDB Lightning
  - Make TiDB 2.1.0 the minimum cluster version supported by Lightning
  - Fix the content error of the file involving parsed `JSON` data in Lightning #144
  - Fix the issue that `Too many open engines` occurs after the checkpoint is used to restart Lightning

- TiDB Binlog
    - Eliminate some bottlenecks of Drainer writing data to Kafka
    - Support the Kafka version of TiDB Binlog

16.15.19 TiDB 2.1.1 Release Notes

On December 12, 2018, TiDB 2.1.1 is released. Compared with TiDB 2.1.0, this release has great improvement in stability, SQL optimizer, statistics information, and execution engine.

16.15.19.1 TiDB

- SQL Optimizer/Executor
    - Fix the round error of the negative date #8574
    - Fix the issue that the `uncompress` function does not check the data length #8606
    - Reset bind arguments of the `prepare` statement after the `execute` command is executed #8652
    - Support automatically collecting the statistics information of a partition table #8649
    - Fix the wrongly configured integer type when pushing down the `abs` function #8628
    - Fix the data race on the JSON column #8660
- Server
    - Fix the issue that the transaction obtained TSO is incorrect when PD breaks down #8567
    - Fix the bootstrap failure caused by the statement that does not conform to ANSI standards #8576
    - Fix the issue that incorrect parameters are used in transaction retries #8638
- DDL
    - Change the default character set and collation of tables into `utf8mb4` #8590
    - Add the `ddl_reorg_batch_size` variable to control the speed of adding indexes #8614
    - Make the character set and collation options content in DDL case-insensitive #8611
    - Fix the issue of adding indexes for generated columns #8655

16.15.19.2 PD

- Fix the issue that some configuration items cannot be set to `0` in the configuration file #1334

- Check the undefined configuration when starting PD #1362
- Avoid transferring the leader to a newly created peer, to optimize the possible delay #1339
- Fix the issue that `RaftCluster` cannot stop caused by deadlock #1370

16.15.19.3 TiKV

- Avoid transferring the leader to a newly created peer, to optimize the possible delay #3878

16.15.19.4 Tools

- Lightning
  - Optimize the `analyze` mechanism on imported tables to increase the import speed
  - Support storing the checkpoint information to a local file
- TiDB Binlog
  - Fix the output bug of pb files that a table only with the primary key column cannot generate the pb event

16.15.20 TiDB 2.1 GA Release Notes

On November 30, 2018, TiDB 2.1 GA is released. See the following updates in this release. Compared with TiDB 2.0, this release has great improvements in stability, performance, compatibility, and usability.

16.15.20.1 TiDB

- SQL Optimizer
  - Optimize the selection range of `Index Join` to improve the execution performance
  - Optimize the selection of outer table for `Index Join` and use the table with smaller estimated value of Row Count the as the outer table
  - Optimize Join Hint `TIDB_SMJ` so that Merge Join can be used even without proper index available
  - Optimize Join Hint `TIDB_INLJ` to specify the Inner table to Join
  - Optimize correlated subquery, push down Filter, and extend the index selection range, to improve the efficiency of some queries by orders of magnitude
  - Support using Index Hint and Join Hint in the `UPDATE` and `DELETE` statement
  - Support pushing down more functions: `ABS/CEIL/FLOOR/IS TRUE/IS FALSE`

    - Optimize the constant folding algorithm for the `IF` and `IFNULL` built-in functions
    - Optimize the output of the `EXPLAIN` statement and use hierarchy structure to show the relationship between operators

- SQL executor

    - Refactor all the aggregation functions and improve execution efficiency of the `Stream` and `Hash` aggregation operators
    - Implement the parallel `Hash Aggregate` operators and improve the computing performance by 350% in some scenarios
    - Implement the parallel `Project` operators and improve the performance by 74% in some scenarios
    - Read the data of the Inner table and Outer table of `Hash Join` concurrently to improve the execution performance
    - Optimize the execution speed of the `REPLACE INTO` statement and increase the performance nearly by 10 times
    - Optimize the memory usage of the time data type and decrease the memory usage of the time data type by fifty percent
    - Optimize the point select performance and improve the point select efficiency result of Sysbench by 60%
    - Improve the performance of TiDB on inserting or updating wide tables by 20 times
    - Support configuring the memory upper limit of a single statement in the configuration file
    - Optimize the execution of Hash Join, if the Join type is Inner Join or Semi Join and the inner table is empty, return the result without reading data from the outer table
    - Support using the `EXPLAIN ANALYZE` statement to check the runtime statistics including the execution time and the number of returned rows of each operator

- Statistics

    - Support enabling auto ANALYZE statistics only during certain period of the day
    - Support updating the table statistics automatically according to the feedback of the queries
    - Support configuring the number of buckets in the histogram using the `ANALYZE TABLE WITH BUCKETS` statement
    - Optimize the Row Count estimation algorithm using histogram for mixed queries of equality query and range queries

- Expressions

    - Support following built-in function:
        - `json_contains`
        - `json_contains_path`
        - `encode/decode`

- Server

    - Support queuing the locally conflicted transactions within tidb-server instance to optimize the performance of conflicted transactions
    - Support Server Side Cursor
    - Add the HTTP API
        - Scatter the distribution of table Regions in the TiKV cluster
        - Control whether to open the `general log`
        - Support modifying the log level online
        - Check the TiDB cluster information
    - Add the `auto_analyze_ratio` system variables to contorl the ratio of Analyze
    - Add the `tidb_retry_limit` system variable to control the automatic retry times of transactions
    - Add the `tidb_disable_txn_auto_retry` system variable to control whether the transaction retries automatically
    - Support using`admin show slow` statement to obtain the slow queries
    - Add the `tidb_slow_log_threshold` environment variable to set the threshold of slow log automatically
    - Add the `tidb_query_log_max_len` environment variable to set the length of the SQL statement to be truncated in the log dynamically

- DDL

    - Support the parallel execution of the Add index statement and other statements to avoid the time consuming Add index operation blocking other operations
    - Optimize the execution speed of `ADD INDEX` and improve it greatly in some scenarios
    - Support the `select tidb_is_ddl_owner()` statement to facilitate deciding whether TiDB is `DDL Owner`
    - Support the `ALTER TABLE FORCE` syntax
    - Support the `ALTER TABLE RENAME KEY TO` syntax
    - Add the table name and database name in the output information of `admin show ddl jobs`

    - Support using the `ddl/owner/resign` HTTP interface to release the DDL owner and start electing a new DDL owner

- Compatibility

    - Support more MySQL syntaxes
    - Make the `BIT` aggregate function support the `ALL` parameter
    - Support the `SHOW PRIVILEGES` statement
    - Support the `CHARACTER SET` syntax in the `LOAD DATA` statement
    - Support the `IDENTIFIED WITH` syntax in the `CREATE USER` statement
    - Support the `LOAD DATA IGNORE LINES` statement
    - The `Show ProcessList` statement returns more accurate information

16.15.20.2 Placement Driver (PD)

- Optimize availability

    - Introduce the version control mechanism and support rolling update of the cluster compatibly
    - Enable `Raft PreVote` among PD nodes to avoid leader reelection when network recovers after network isolation
    - Enable `raft learner` by default to lower the risk of unavailable data caused by machine failure during scheduling
    - TSO allocation is no longer affected by the system clock going backwards
    - Support the `Region merge` feature to reduce the overhead brought by metadata

- Optimize the scheduler

    - Optimize the processing of Down Store to speed up making up replicas
    - Optimize the hotspot scheduler to improve its adaptability when traffic statistics information jitters
    - Optimize the start of Coordinator to reduce the unnecessary scheduling caused by restarting PD
    - Optimize the issue that Balance Scheduler schedules small Regions frequently
    - Optimize Region merge to consider the number of rows within the Region
    - Add more commands to control the scheduling policy
    - Improve PD simulator to simulate the scheduling scenarios

- API and operation tools

- Add the `GetPrevRegion` interface to support the `TiDB reverse scan` feature
- Add the `BatchSplitRegion` interface to speed up TiKV Region splitting
- Add the `GCSafePoint` interface to support distributed GC in TiDB
- Add the `GetAllStores` interface, to support distributed GC in TiDB
- pd-ctl supports:
  - using statistics for Region split
  - calling `jq` to format the JSON output
  - checking the Region information of the specified store
  - checking topN Region list sorted by versions
  - checking topN Region list sorted by size
  - more precise TSO encoding
- pd-recover doesn't need to provide the `max-replica` parameter

- Metrics
  - Add related metrics for `Filter`
  - Add metrics about etcd Raft state machine

- Performance
  - Optimize the performance of Region heartbeat to reduce the memory overhead brought by heartbeats
  - Optimize the Region tree performance
  - Optimize the performance of computing hotspot statistics

16.15.20.3 TiKV

- Coprocessor
  - Add more built-in functions
  - Add Coprocessor `ReadPool` to improve the concurrency in processing the requests
  - Fix the time function parsing issue and the time zone related issues
  - Optimize the memory usage for pushdown aggregation computing

- Transaction
  - Optimize the read logic and memory usage of MVCC to improve the performance of the scan operation and the performance of full table scan is 1 time better than that in TiDB 2.0
  - Fold the continuous Rollback records to ensure the read performance

    - Add the `UnsafeDestroyRange` API to support to collecting space for the dropping table/index
    - Separate the GC module to reduce the impact on write
    - Add the `upper bound` support in the `kv_scan` command

- Raftstore

    - Improve the snapshot writing process to avoid RocksDB stall
    - Add the `LocalReader` thread to process read requests and reduce the delay for read requests
    - Support `BatchSplit` to avoid large Region brought by large amounts of write
    - Support `Region Split` according to statistics to reduce the I/O overhead
    - Support `Region Split` according to the number of keys to improve the concurrency of index scan
    - Improve the Raft message process to avoid unnecessary delay brought by `Region Split`
    - Enable the `PreVote` feature by default to reduce the impact of network isolation on services

- Storage Engine

    - Fix the `CompactFiles`bug in RocksDB and reduce the impact on importing data using Lightning
    - Upgrade RocksDB to v5.15 to fix the possible issue of snapshot file corruption
    - Improve `IngestExternalFile` to avoid the issue that flush could block write

- tikv-ctl

    - Add the `ldb` command to diagnose RocksDB related issues
    - The `compact` command supports specifying whether to compact data in the bottommost level

16.15.20.4 Tools

- Fast full import of large amounts of data: TiDB Lightning
- Support new TiDB Binlog

16.15.20.5 Upgrade caveat

- TiDB 2.1 does not support downgrading to v2.0.x or earlier due to the adoption of the new storage engine
- Parallel DDL is enabled in TiDB 2.1, so the clusters with TiDB version earlier than 2.0.1 cannot upgrade to 2.1 using rolling update. You can choose either of the following two options:
    - Stop the cluster and upgrade to 2.1 directly
    - Roll update to 2.0.1 or later 2.0.x versions, and then roll update to the 2.1 version
- If you upgrade from TiDB 2.0.6 or earlier to TiDB 2.1, check if there is any ongoing DDL operation, especially the time consuming `Add Index` operation, because the DDL operations slow down the upgrading process. If there is ongoing DDL operation, wait for the DDL operation finishes and then roll update.

16.15.21 TiDB 2.1 RC5 Release Notes

On November 12, 2018, TiDB 2.1 RC5 is released. Compared with TiDB 2.1 RC4, this release has great improvement in stability, SQL optimizer, statistics information, and execution engine.

16.15.21.1 TiDB

- SQL Optimizer
    - Fix the issue that `IndexReader` reads the wrong handle in some cases #8132
    - Fix the issue occurred while the `IndexScan Prepared` statement uses `Plan Cache` #8055
    - Fix the issue that the result of the `Union` statement is unstable #8165
- SQL Execution Engine
    - Improve the performance of TiDB on inserting or updating wide tables #8024
    - Support the unsigned `int` flag in the `Truncate` built-in function #8068
    - Fix the error occurred while converting JSON data to the decimal type #8109
    - Fix the error occurred when you `Update` the float type #8170
- Statistics
    - Fix the incorrect statistics issue during point queries in some cases #8035
    - Fix the selectivity estimation of statistics for primary key in some cases #8149
    - Fix the issue that the statistics of deleted tables are not cleared up for a long period of time #8182
- Server

- Improve the readability of logs and make logs better
  - #8063
  - #8053
  - #8224
- Fix the error occurred when obtaining the table data of `infoschema.profiling` #8096
- Replace the unix socket with the pumps client to write binlogs #8098
- Add the threshold value for the `tidb_slow_log_threshold` environment variable, which dynamically sets the slow log #8094
- Add the original length of a SQL statement truncated while the `tidb_query_log_max_len` environment variable dynamically sets logs #8200
- Add the `tidb_opt_write_row_id` environment variable to control whether to allow writing `_tidb_rowid` #8218
- Add an upper bound to the `Scan` command of ticlient, to avoid overbound scan #8081, #8247

- DDL
  - Fix the issue that executing DDL statements in transactions encounters an error in some cases #8056
  - Fix the issue that executing `truncate table` in partition tables does not take effect #8103
  - Fix the issue that the DDL operation does not roll back correctly after being cancelled in some cases #8057
  - Add the `admin show next_row_id` command to return the next available row ID #8268

16.15.21.2 PD

- Fix the issues related to `pd-ctl` reading the Region key
  - #1298
  - #1299
  - #1308
- Fix the issue that the `regions/check` API returns the wrong result #1311
- Fix the issue that PD cannot restart join after a PD join failure #1279
- Fix the issue that `watch leader` might lose events in some cases #1317

16.15.21.3 TiKV

- Improve the error message of `WriteConflict` #3750
- Add the panic mark file #3746
- Downgrade grpcio to avoid the segment fault issue caused by the new version of gRPC #3650
- Add an upper limit to the `kv_scan` interface #3749

16.15.21.4 Tools

- Support the TiDB-Binlog cluster, which is not compatible with the older version of binlog #8093, documentation

16.15.22 TiDB 2.1 RC4 Release Notes

On October 23, 2018, TiDB 2.1 RC4 is released. Compared with TiDB 2.1 RC3, this release has great improvement in stability, SQL optimizer, statistics information, and execution engine.

16.15.22.1 TiDB

- SQL Optimizer
  - Fix the issue that column pruning of `UnionAll` is incorrect in some cases #7941
  - Fix the issue that the result of the `UnionAll` operator is incorrect in some cases #8007
- SQL Execution Engine
  - Fix the precision issue of the `AVG` function #7874
  - Support using the `EXPLAIN ANALYZE` statement to check the runtime statistics including the execution time and the number of returned rows of each operator during the query execution process #7925
  - Fix the panic issue of the `PointGet` operator when a column of a table appears multiple times in the result set #7943
  - Fix the panic issue caused by too large values in the `Limit` subclause #8002
  - Fix the panic issue during the execution process of the `AddDate`/`SubDate` statement in some cases #8009
- Statistics
  - Fix the issue of judging the prefix of the histogram low-bound of the combined index as out of range #7856
  - Fix the memory leak issue caused by statistics collecting #7873
  - Fix the panic issue when the histogram is empty #7928
  - Fix the issue that the histogram bound is out of range when the statistics is being uploaded #7944
  - Limit the maximum length of values in the statistics sampling process #7982
- Server
  - Refactor Latch to avoid misjudgment of transaction conflicts and improve the execution performance of concurrent transactions #7711
  - Fix the panic issue caused by collecting slow queries in some cases #7874

    - Fix the panic issue when `ESCAPED BY` is an empty string in the `LOAD DATA` statement #8005
    - Complete the "coprocessor error" log information #8006
- Compatibility
    - Set the `Command` field of the `SHOW PROCESSLIST` result to `Sleep` when the query is empty #7839
- Expressions
    - Fix the constant folding issue of the `SYSDATE` function #7895
    - Fix the issue that `SUBSTRING_INDEX` panics in some cases #7897
- DDL
    - Fix the stack overflow issue caused by throwing the `invalid ddl job type` error #7958
    - Fix the issue that the result of `ADMIN CHECK TABLE` is incorrect in some cases #7975

16.15.22.2 PD

- Fix the issue that the tombstone TiKV is not removed from Grafana #1261
- Fix the data race issue when grpc-go configures the status #1265
- Fix the issue that the PD server gets stuck caused by etcd startup failure #1267
- Fix the issue that data race might occur during leader switching #1273
- Fix the issue that extra warning logs might be output when TiKV becomes tombstone #1280

16.15.22.3 TiKV

- Optimize the RocksDB Write stall issue caused by applying snapshots #3606
- Add raftstore `tick` metrics #3657
- Upgrade RocksDB and fix the Write block issue and that the source file might be damaged by the Write operation when performing `IngestExternalFile` #3661
- Upgrade grpcio and fix the issue that "too many pings" is wrongly reported #3650

16.15.23 TiDB 2.1 RC3 Release Notes

On September 29, 2018, TiDB 2.1 RC3 is released. Compared with TiDB 2.1 RC2, this release has great improvement in stability, compatibility, SQL optimizer, and execution engine.

16.15.23.1 TiDB

- SQL Optimizer
  - Fix the incorrect result issue when a statement contains embedded `LEFT OUTER JOIN` #7689
  - Enhance the optimization rule of predicate pushdown on the `JOIN` statement #7645
  - Fix the optimization rule of predicate pushdown for the `UnionScan` operator #7695
  - Fix the issue that the unique key property of the `Union` operator is not correctly set #7680
  - Enhance the optimization rule of constant folding #7696
  - Optimize the data source in which the filter is null after propagation to table dual #7756
- SQL Execution Engine
  - Optimize the performance of read requests in a transaction #7717
  - Optimize the cost of allocating Chunk memory in some executors #7540
  - Fix the "index out of range" panic caused by the columns where point queries get all NULL values #7790
- Server
  - Fix the issue that the memory quota in the configuration file does not take effect #7729
  - Add the `tidb_force_priority` system variable to set the execution priority for each statement #7694
  - Support using the `admin show slow` statement to obtain the slow query log #7785
- Compatibility
  - Fix the issue that the result of `charset/collation` is incorrect in `information_schema.schemata` #7751
  - Fix the issue that the value of the `hostname` system variable is empty #7750
- Expressions
  - Support the `init_vecter` argument in the `AES_ENCRYPT/AES_DECRYPT` built-in function #7425
  - Fix the issue that the result of `Format` is incorrect in some expressions #7770
  - Support the `JSON_LENGTH` built-in function #7739
  - Fix the incorrect result issue when casting the unsigned integer type to the decimal type #7792
- DML

- Fix the issue that the result of the `INSERT ⋯ ON DUPLICATE KEY UPDATE` statement is incorrect while updating the unique key #7675

- DDL
  - Fix the issue that the index value is not converted between time zones when you create a new index on a new column of the timestamp type #7724
  - Support appending new values for the enum type #7767
  - Support creating an etcd session quickly, which improves the cluster availability after network isolation #7774

16.15.23.2 PD

- New feature
  - Add the API to get the Region list by size in reverse order #1254
- Improvement
  - Return more detailed information in the Region API #1252
- Bug fix
  - Fix the issue that `adjacent-region-scheduler` might lead to a crash after PD switches the leader #1250

16.15.23.3 TiKV

- Performance
  - Optimize the concurrency for coprocessor requests #3515
- New features
  - Add the support for Log functions #3603
  - Add the support for the `sha1` function #3612
  - Add the support for the `truncate_int` function #3532
  - Add the support for the `year` function #3622
  - Add the support for the `truncate_real` function #3633
- Bug fixes
  - Fix the reporting error behavior related to time functions #3487, #3615
  - Fix the issue that the time parsed from string is inconsistent with that in TiDB #3589

16.15.24 TiDB 2.1 RC2 Release Notes

On September 14, 2018, TiDB 2.1 RC2 is released. Compared with TiDB 2.1 RC1, this release has great improvement in stability, SQL optimizer, statistics information, and execution engine.

16.15.24.1 TiDB

- SQL Optimizer
  - Put forward a proposal of the next generation Planner #7543
  - Improve the optimization rules of constant propagation #7276
  - Enhance the computing logic of `Range` to enable it to handle multiple `IN` or `EQUAL` conditions simultaneously #7577
  - Fix the issue that the estimation result of `TableScan` is incorrect when `Range` is empty #7583
  - Support the `PointGet` operator for the `UPDATE` statement #7586
  - Fix the panic issue during the process of executing the `FirstRow` aggregate function in some conditions #7624
- SQL Execution Engine
  - Fix the potential `DataRace` issue when the `HashJoin` operator encounters an error #7554
  - Make the `HashJoin` operator read the inner table and build the hash table simultaneously #7544
  - Optimize the performance of Hash aggregate operators #7541
  - Optimize the performance of Join operators #7493, #7433
  - Fix the issue that the result of `UPDATE JOIN` is incorrect when the Join order is changed #7571
  - Improve the performance of Chunk's iterator #7585
- Statistics
  - Fix the issue that the auto Analyze work repeatedly analyzes the statistics #7550
  - Fix the statistics update error that occurs when there is no statistics change #7530
  - Use the RC isolation level and low priority when building `Analyze` requests #7496
  - Support enabling statistics auto-analyze on certain period of a day #7570
  - Fix the panic issue when logging the statistics information #7588
  - Support configuring the number of buckets in the histogram using the `ANALYZE TABLE WITH BUCKETS` statement #7619
  - Fix the panic issue when updating an empty histogram #7640
  - Update `information_schema.tables.data_length` using the statistics information #7657
- Server
  - Add Trace related dependencies #7532
  - Enable the `mutex profile` feature of Golang #7512
  - The `Admin` statement requires the `Super_priv` privilege #7486
  - Forbid users to `Drop` crucial system tables #7471
  - Switch from `juju/errors` to `pkg/errors` #7151
  - Complete the functional prototype of SQL Tracing #7016

- Remove the goroutine pool #7564
- Support viewing the goroutine information using the `USER1` signal #7587
- Set the internal SQL to high priority while TiDB is started #7616
- Use different labels to filter internal SQL and user SQL in monitoring metrics #7631
- Store the top 30 slow queries in the last week to the TiDB server #7646
- Put forward a proposal of setting the global system time zone for the TiDB cluster #7656
- Enrich the error message of "GC life time is shorter than transaction duration" #7658
- Set the global system time zone when starting the TiDB cluster #7638

- Compatibility
  - Add the unsigned flag for the `Year` type #7542
  - Fix the issue of configuring the result length of the `Year` type in the `Prepare` ↪ `/Execute` mode #7525
  - Fix the issue of inserting zero timestamp in the `Prepare/Execute` mode #7506
  - Fix the error handling issue of the integer division #7492
  - Fix the compatibility issue when processing `ComStmtSendLongData` #7485
  - Fix the error handling issue during the process of converting string to integer #7483
  - Optimize the accuracy of values in the `information_schema.columns_in_table` table #7463
  - Fix the compatibility issue when writing or updating the string type of data using the MariaDB client #7573
  - Fix the compatibility issue of aliases of the returned value #7600
  - Fix the issue that the `NUMERIC_SCALE` value of the float type is incorrect in the `information_schema.COLUMNS` table #7602
  - Fix the issue that Parser reports an error when the single line comment is empty #7612

- Expressions
  - Check the value of `max_allowed_packet` in the `insert` function #7528
  - Support the built-in function `json_contains` #7443
  - Support the built-in function `json_contains_path` #7596
  - Support the built-in function `encode/decode` #7622
  - Fix the issue that some time related functions are not compatible with the MySQL behaviors in some cases #7636
  - Fix the compatibility issue of parsing the time type of data in string #7654
  - Fix the issue that the time zone is not considered when computing the default value of the `DateTime` data #7655

- DML
  - Set correct `last_insert_id` in the `InsertOnDuplicateUpdate` statement #7534
  - Reduce the cases of updating the `auto_increment_id` counter #7515

- Optimize the error message of `Duplicate Key` #7495
- Fix the `insert...select...on duplicate key update` issue #7406
- Support the `LOAD DATA IGNORE LINES` statement #7576

- DDL
  - Add the DDL job type and the current schema version information in the monitor #7472
  - Complete the design of the `Admin Restore Table` feature #7383
  - Fix the issue that the default value of the `Bit` type exceeds 128 #7249
  - Fix the issue that the default value of the `Bit` type cannot be `NULL` #7604
  - Reduce the interval of checking `CREATE TABLE/DATABASE` in the DDL queue #7608
  - Use the `ddl/owner/resign` HTTP interface ro release the DDL owner and start electing a new owner #7649

- TiKV Go Client
  - Support the issue that the `Seek` operation only obtains `Key` #7419

- Table Partition (Experimental)
  - Fix the issue that the `Bigint` type cannot be used as the partition key #7520
  - Support the rollback operation when an issue occurs during adding an index in the partitioned table #7437

16.15.24.2 PD

- Features
  - Support the `GetAllStores` interface #1228
  - Add the statistics of scheduling estimation in Simulator #1218

- Improvements
  - Optimize the handling process of down stores to make up replicas as soon as possible #1222
  - Optimize the start of Coordinator to reduce the unnecessary scheduling caused by restarting PD #1225
  - Optimize the memory usage to reduce the overhead caused by heartbeats #1195
  - Optimize error handling and improve the log information #1227
  - Support querying the Region information of a specific store in pd-ctl #1231
  - Support querying the topN Region information based on version comparison in pd-ctl #1233
  - Support more accurate TSO decoding in pd-ctl #1242

- Bug fix
  - Fix the issue that pd-ctl uses the `hot store` command to exit wrongly #1244

16.15.24.3 TiKV

- Performance
  - Support splitting Regions based on statistics estimation to reduce the I/O cost #3511
  - Reduce clone in the transaction scheduler #3530
- Improvements
  - Add the pushdown support for a large number of built-in functions
  - Add the `leader-transfer-max-log-lag` configuration to fix the failure issue of leader scheduling in specific scenarios #3507
  - Add the `max-open-engines` configuration to limit the number of engines opened by `tikv-importer` simultaneously #3496
  - Limit the cleanup speed of garbage data to reduce the impact on `snapshot apply` #3547
  - Broadcast the commit message for crucial Raft messages to avoid unnecessary delay #3592
- Bug fixes
  - Fix the leader election issue caused by discarding the `PreVote` message of the newly split Region #3557
  - Fix follower related statistics after merging Regions #3573
  - Fix the issue that the local reader uses obsolete Region information #3565

16.15.25 TiDB 2.1 RC1 Release Notes

On August 24, 2018, TiDB 2.1 RC1 is released! Compared with TiDB 2.1 Beta, this release has great improvement in stability, SQL optimizer, statistics information, and execution engine.

16.15.25.1 TiDB

- SQL Optimizer
  - Fix the issue that a wrong result is returned after the correlated subquery is decorrelated in some cases #6972
  - Optimize the output result of `Explain` #7011#7041
  - Optimize the choosing strategy of the outer table for `IndexJoin` #7019
  - Remove the Plan Cache of the non-`PREPARE` statement #7040
  - Fix the issue that the `INSERT` statement is not parsed and executed correctly in some cases #7068
  - Fix the issue that the `IndexJoin` result is not correct in some cases #7150
  - Fix the issue that the `NULL` value cannot be found using the unique index in some cases #7163

- Fix the range computing issue of the prefix index in UTF-8 #7194
- Fix the issue that result is not correct caused by eliminating the `Project` operator in some cases #7257
- Fix the issue that `USE INDEX(PRIMARY)` cannot be used when the primary key is an integer #7316
- Fix the issue that the index range cannot be computed using the correlated column in some cases #7357

- SQL Execution Engine
  - Fix the issue that the daylight saving time is not computed correctly in some cases #6823
  - Refactor the aggregation function framework to improve the execution efficiency of the `Stream` and `Hash` aggregation operators #6852
  - Fix the issue that the `Hash` aggregation operator cannot exit normally in some cases #6982
  - Fix the issue that `BIT_AND/BIT_OR/BIT_XOR` does not handle the non-integer data correctly #6994
  - Optimize the execution speed of the `REPLACE INTO` statement and increase the performance nearly 10 times #7027
  - Optimize the memory usage of time type data and decrease the memory usage of the time type data by fifty percent #7043
  - Fix the issue that the returned result is mixed with signed and unsigned integers in the `UNION` statement is not compatible with MySQL #7112
  - Fix the panic issue caused by the too much memory applied by `LPAD/RPAD` ↪ `/TO_BASE64/FROM_BASE64/REPEAT` #7171 #7266 #7409 #7431
  - Fix the incorrect result when `MergeJoin/IndexJoin` handles the `NULL` value #7255
  - Fix the incorrect result of `Outer Join` in some cases #7288
  - Improve the error message of `Data Truncated` to facilitate locating the wrong data and the corresponding field in the table #7401
  - Fix the incorrect result for `decimal` in some cases #7001 #7113 #7202 #7208
  - Optimize the point select performance #6937
  - Prohibit the isolation level of `Read Committed` to avoid the underlying problem #7211
  - Fix the incorrect result of `LTRIM/RTRIM/TRIM` in some cases #7291
  - Fix the issue that the `MaxOneRow` operator cannot guarantee that the returned result does not exceed one row #7375
  - Divide the Coprocessor requests with too many ranges #7454

- Statistics
  - Optimize the mechanism of statistics dynamic collection #6796
  - Fix the issue that `Auto Analyze` does not work when data is updated frequently #7022
  - Decrease the Write conflicts during the statistics dynamic update process #7124
  - Optimize the cost estimation when the statistics is incorrect #7175

    - Optimize the `AccessPath` cost estimation strategy #7233

- Server
    - Fix the bug in loading privilege information #6976
    - Fix the issue that the `Kill` command is too strict with privilege check #6954
    - Fix the issue of removing some binary numeric types #6922
    - Shorten the output log #7029
    - Handle the `mismatchClusterID` issue #7053
    - Add the `advertise-address` configuration item #7078
    - Add the `GrpcKeepAlive` option #7100
    - Add the connection or `Token` time monitor #7110
    - Optimize the data decoding performance #7149
    - Add the `PROCESSLIST` table in `INFORMMATION_SCHEMA` #7236
    - Fix the order issue when multiple rules are hit in verifying the privilege #7211
    - Change some default values of encoding related system variables to UTF-8 #7198
    - Make the slow query log show more detailed information #7302
    - Support registering tidb-server related information in PD and obtaining this information by HTTP API #7082

- Compatibility
    - Support Session variables `warning_count` and `error_count` #6945
    - Add `Scope` check when reading the system variables #6958
    - Support the `MAX_EXECUTION_TIME` syntax #7012
    - Support more statements of the `SET` syntax #7020
    - Add validity check when setting system variables #7117
    - Add the verification of the number of `PlaceHolders` in the `Prepare` statement #7162
    - Support `set character_set_results = null` #7353
    - Support the `flush status` syntax #7369
    - Fix the column size of `SET` and `ENUM` types in `information_schema` #7347
    - Support the `NATIONAL CHARACTER` syntax of statements for creating a table #7378
    - Support the `CHARACTER SET` syntax in the `LOAD DATA` statement #7391
    - Fix the column information of the `SET` and `ENUM` types #7417
    - Support the `IDENTIFIED WITH` syntax in the `CREATE USER` statement #7402
    - Fix the precision losing issue during `TIMESTAMP` computing process #7418
    - Support the validity verification of more `SYSTEM` variables #7196
    - Fix the incorrect result when the `CHAR_LENGTH` function computes the binary string #7410
    - Fix the incorrect `CONCAT` result in a statement involving `GROUP BY` #7448
    - Fix the imprecise type length issue when casting the `DECIMAL` type to the `STRING` type #7451

- DML
    - Fix the stability issue of the `Load Data` statement #6927
    - Fix the memory usage issue when performing some `Batch` operations #7086

    - Improve the performance of the `Replace Into` statement #7027
    - Fix the inconsistent precision issue when writing `CURRENT_TIMESTAMP` #7355
- DDL
    - Improve the method of DDL judging whether `Schema` is replicated to avoid misjudgement in some cases #7319
    - Fix the `SHOW CREATE TABLE` result in adding index process #6993
    - Allow the default value of `text/blob/json` to be NULL in non-restrict `sql-mode` #7230
    - Fix the `ADD INDEX` issue in some cases #7142
    - Increase the speed of adding `UNIQUE-KEY` index operation largely #7132
    - Fix the truncating issue of the prefix index in UTF-8 character set #7109
    - Add the environment variable `tidb_ddl_reorg_priority` to control the priority of the `add-index` operation #7116
    - Fix the display issue of `AUTO-INCREMENT` in `information_schema.tables` #7037
    - Support the `admin show ddl jobs <number>` command and support output specified number of DDL jobs #7028
    - Support parallel DDL job execution #6955
- Table Partition (Experimental)
    - Support top level partition
    - Support `Range Partition`

16.15.25.2 PD

- Features
    - Introduce the version control mechanism and support rolling update of the cluster with compatibility
    - Enable the `region merge` feature
    - Support the `GetPrevRegion` interface
    - Support splitting Regions in batch
    - Support storing the GC safepoint
- Improvements
    - Optimize the issue that TSO allocation is affected by the system clock going backwards
    - Optimize the performance of handling Region heartbeats
    - Optimize the Region tree performance
    - Optimize the performance of computing hotspot statistics
    - Optimize returning the error code of API interface
    - Add options of controlling scheduling strategies
    - Prohibit using special characters in `label`
    - Improve the scheduling simulator

    - Support splitting Regions using statistics in pd-ctl
    - Support formatting JSON output by calling `jq` in pd-ctl
    - Add metrics about etcd Raft state machine

- Bug fixes

    - Fix the issue that the namespace is not reloaded after switching Leader
    - Fix the issue that namespace scheduling exceeds the schedule limit
    - Fix the issue that hotspot scheduling exceeds the schedule limit
    - Fix the issue that wrong logs are output when the PD client closes
    - Fix the wrong statistics of Region heartbeat latency

16.15.25.3 TiKV

- Features

    - Support `batch split` to avoid too large Regions caused by the Write operation on hot Regions
    - Support splitting Regions based on the number of rows to improve the index scan efficiency

- Performance

    - Use `LocalReader` to separate the Read operation from the raftstore thread to lower the Read latency
    - Refactor the MVCC framework, optimize the memory usage and improve the scan Read performance
    - Support splitting Regions based on statistics estimation to reduce the I/O usage
    - Optimize the issue that the Read performance is affected by continuous Write operations on the rollback record
    - Reduce the memory usage of pushdown aggregation computing

- Improvements

    - Add the pushdown support for a large number of built-in functions and better charset support
    - Optimize the GC workflow, improve the GC speed and decrease the impact of GC on the system
    - Enable `prevote` to speed up service recovery when the network is abnormal
    - Add the related configuration items of RocksDB log files
    - Adjust the default configuration of `scheduler_latch`
    - Support setting whether to compact the data in the bottom layer of RocksDB when using tikv-ctl to compact data manually
    - Add the check for environment variables when starting TiKV
    - Support dynamically configuring the `dynamic_level_bytes` parameter based on the existing data
    - Support customizing the log format

    - Integrate tikv-fail in tikv-ctl
    - Add I/O metrics of threads
- Bug fixes
    - Fix decimal related issues
    - Fix the issue that `gRPC max_send_message_len` is set mistakenly
    - Fix the issue caused by misconfiguration of `region_size`

16.15.26 TiDB 2.1 Beta Release Notes

On June 29, 2018, TiDB 2.1 Beta is released! Compared with TiDB 2.0, this release has great improvement in stability, SQL optimizer, statistics information, and execution engine.

16.15.26.1 TiDB

- SQL Optimizer
    - Optimize the selection range of `Index Join` to improve the execution performance
    - Optimize correlated subquery, push down `Filter`, and extend the index range, to improve the efficiency of some queries by orders of magnitude
    - Support `Index Hint` and `Join Hint` in the `UPDATE` and `DELETE` statements
    - Validate Hint `TIDM_SMJ` when no available index exists
    - Support pushdown of the `ABS`, `CEIL`, `FLOOR`, `IS TRUE`, and `IS FALSE` functions
    - Handle the `IF` and `IFNULL` functions especially in the constant folding process
- SQL Execution Engine
    - Implement parallel `Hash Aggregate` operators and improve the computing performance of `Hash Aggregate` by 350% in some scenarios
    - Implement parallel `Project` operators and improve the performance by 74% in some scenarios
    - Read the data of the `Inner` table and `Outer` table of `Hash Join` concurrently to improve the execution performance
    - Fix incorrect results of `INSERT ⋯ ON DUPLICATE KEY UPDATE ⋯` in some scenarios
    - Fix incorrect results of the `CONCAT_WS`, `FLOOR`, `CEIL`, and `DIV` built-in functions
- Server
    - Add the HTTP API to scatter the distribution of table Regions in the TiKV cluster
    - Add the `auto_analyze_ratio` system variable to control the threshold value of automatic `Analyze`
    - Add the HTTP API to control whether to open the general log
    - Add the HTTP API to modify the log level online
    - Add the user information in the general log and the slow query log

    - Support the server side cursor
- Compatibility
    - Support more MySQL syntax
    - Make the `bit` aggregate function support the `ALL` parameter
    - Support the `SHOW PRIVILEGES` statement
- DML
    - Decrease the memory usage of the `INSERT INTO SELECT` statement
    - Fix the performance issue of `PlanCache`
    - Add the `tidb_retry_limit` system variable to control the automatic retry times of transactions
    - Add the `tidb_disable_txn_auto_retry` system variable to control whether the transaction tries automatically
    - Fix the accuracy issue of the written data of the `time` type
    - Support the queue of locally conflicted transactions to optimize the conflicted transaction performance
    - Fix `Affected Rows` of the `UPDATE` statement
    - Optimize the statement performance of `insert ignore on duplicate key update`
- DDL
    - Optimize the execution speed of the `CreateTable` statement
    - Optimize the execution speed of `ADD INDEX` and improve it greatly in some scenarios
    - Fix the issue that the number of added columns by `Alter table add column` exceeds the limit of the number of table columns
    - Fix the issue that DDL job retries lead to an increasing pressure on TiKV in abnormal conditions
    - Fix the issue that TiDB continuously reloads the schema information in abnormal conditions
    - Do not output the `FOREIGN KEY` related information in the result of `SHOW CREATE TABLE`
    - Support the `select tidb_is_ddl_owner()` statement to facilitate judging whether TiDB is `DDL Owner`
    - Fix the issue that the index is deleted in the `Year` type in some scenarios
    - Fix the renaming table issue in the concurrent execution scenario
    - Support the `AlterTableForce` syntax
    - Support the `AlterTableRenameIndex` syntax with `FromKey` and `ToKey`
    - Add the table name and database name in the output information of `admin show ddl jobs`

16.15.26.2 PD

- Enable Raft PreVote between PD nodes to avoid leader reelection when network recovers after network isolation
- Optimize the issue that Balance Scheduler schedules small Regions frequently
- Optimize the hotspot scheduler to improve its adaptability in traffic statistics information jitters
- Skip the Regions with a large number of rows when scheduling `region merge`
- Enable `raft learner` by default to lower the risk of unavailable data caused by machine failure during scheduling
- Remove `max-replica` from `pd-recover`
- Add `Filter` metrics
- Fix the issue that Region information is not updated after tikv-ctl unsafe recovery
- Fix the issue that TiKV disk space is used up caused by replica migration in some scenarios
- Compatibility notes
    - Do not support rolling back to v2.0.x or earlier due to update of the new version storage engine
    - Enable `raft learner` by default in the new version of PD. If the cluster is upgraded from 1.x to 2.1, the machine should be stopped before upgrade or a rolling update should be first applied to TiKV and then PD

16.15.26.3 TiKV

- Upgrade Rust to the `nightly-2018-06-14` version
- Enable `Raft PreVote` to avoid leader reelection generated when network recovers after network isolation
- Add a metric to display the number of files and `ingest` related information in each layer of RocksDB
- Print `key` with too many versions when GC works
- Use `static metric` to optimize multi-label metric performance (YCSB `raw get` is improved by 3%)
- Remove `box` in multiple modules and use patterns to improve the operating performance (YCSB `raw get` is improved by 3%)
- Use `asynchronous log` to improve the performance of writing logs
- Add a metric to collect the thread status
- Decease memory copy times by decreasing `box` used in the application to improve the performance

16.16 v2.0

16.16.1 TiDB 2.0.11 Release Notes

On January 03, 2019, TiDB 2.0.11 is released. The corresponding TiDB Ansible 2.0.11 is also released. Compared with TiDB 2.0.10, this release has great improvement in system compatibility and stability.

16.16.1.1 TiDB

- Fix the issue that the error is not handled properly when PD is in an abnormal condition #8764
- Fix the issue that the `Rename` operation on a table in TiDB is not compatible with that in MySQL #8809
- Fix the issue that the error message is wrongly reported when the `ADMIN CHECK TABLE` operation is performed in the process of executing the `ADD INDEX` statement #8750
- Fix the issue that the prefix index range is incorrect in some cases #8877
- Fix the panic issue of the `UPDATE` statement when columns are added in some cases #8904

16.16.1.2 TiKV

- Fix two issues about Region merge #4003, #4004

16.16.2 TiDB 2.0.10 Release Notes

On December 18, 2018, TiDB 2.0.10 is released. The corresponding TiDB Ansible 2.0.10 is also released. Compared with TiDB 2.0.9, this release has great improvement in system compatibility and stability.

16.16.2.1 TiDB

- Fix the possible issue caused by canceling a DDL job #8513
- Fix the issue that the `ORDER BY` and `UNION` clauses cannot quote the column including a table name #8514
- Fix the issue that the `UNCOMPRESS` function does not judge the incorrect input length #8607
- Fix the issue encountered by `ANSI_QUOTES SQL_MODE` when upgrading TiDB #8575
- Fix the issue that `select` returns the wrong result in some cases #8570
- Fix the possible issue that TiDB cannot exit when it receives the exit signal #8501
- Fix the issue that `IndexLookUpJoin` returns the wrong result in some cases #8508
- Avoid pushing down the filter containing `GetVar` or `SetVar` #8454
- Fix the issue that the result length of the `UNION` clauses is incorrect in some cases #8491
- Fix the issue of `PREPARE FROM @var_name` #8488
- Fix the panic issue when dumping statistics information in some cases #8464
- Fix the statistics estimation issue of point queries in some cases #8493
- Fix the panic issue when the returned default `enum` value is a string #8476
- Fix the issue that too much memory is consumed in the scenario of wide tables #8467
- Fix the issue encountered when Parser incorrectly formats the mod opcode #8431

- Fix the panic issue caused by adding foreign key constraints in some cases #8421, #8410
- Fix the issue that the `YEAR` column type incorrectly converts the zero value #8396
- Fix the panic issue occurred when the argument of the `VALUES` function is not a column #8404
- Disable Plan Cache for statements containing subqueries #8395

16.16.2.2 PD

- Fix the possible issue that RaftCluster cannot stop caused by deadlock #1370

16.16.2.3 TiKV

- Avoid transferring the leader to a newly created peer, to optimize the possible delay #3929
- Fix redundant Region heartbeats #3930

16.16.3 TiDB 2.0.9 Release Notes

On November 19, 2018, TiDB 2.0.9 is released. Compared with TiDB 2.0.8, this release has great improvement in system compatibility and stability.

16.16.3.1 TiDB

- Fix the issue caused by the empty statistics histogram #7927
- Fix the panic issue of the `UNION ALL` statement in some cases #7942
- Fix the stack overflow issue caused by wrong DDL Jobs #7959
- Add the slow log for the `Commit` operation #7983
- Fix the panic issue caused by the too large `Limit` value #8004
- Support specifying the `utf8mb4` character set in the `USING` clause #8048
- Make the `TRUNCATE` built-in function support parameters of unsigned integer type #8069
- Fix the selectivity estimation issue of the primary key for the statistics module in some cases #8150
- Add the `Session` variable to control whether `_tidb_rowid` is allowed to be written in #8126
- Fix the panic issue of `PhysicalProjection` in some cases #8154
- Fix the unstable results of the `Union` statement in some cases #8168
- Fix the issue that `NULL` is not returned by `values` in the non-`Insert` statement #8179
- Fix the issue that the statistics module cannot clear the outdated data in some cases #8184
- Make the maximum allowed running time for a transaction a configurable option #8209
- Fix the wrong comparison algorithm of `expression rewriter` in some cases #8288

- Eliminate the extra columns generated by the `UNION ORDER BY` statement #8307
- Support the `admin show next_row_id` statement #8274
- Fix the escape issue of special characters in the `Show Create Table` statement #8321
- Fix the unexpected errors in the `UNION` statement in some cases #8318
- Fix the issue that canceling a DDL job causes no rollback of a schema in some cases #8312
- Change `tidb_max_chunk_size` to a global variable #8333
- Add an upper bound to the `Scan` command of ticlient, to avoid overbound scan #8309 #8310

16.16.3.2 PD

- Fix the issue that the PD server gets stuck caused by etcd startup failure #1267
- Fix the issues related to `pd-ctl` reading the Region key #1298 #1299 #1308
- Fix the issue that the `regions/check` API returns the wrong result #1311
- Fix the issue that PD cannot restart join after a PD join failure #1279

16.16.3.3 TiKV

- Add the `end-key` limit to the `kv_scan` interface #3749
- Abandon the `max-tasks-xxx` configuration and add `max-tasks-per-worker-xxx` #3093
- Fix the `CompactFiles` issue in RocksDB #3789

16.16.4 TiDB 2.0.8 Release Notes

On October 16, 2018, TiDB 2.0.8 is released. Compared with TiDB 2.0.7, this release has great improvement in system compatibility and stability.

16.16.4.1 TiDB

- Improvement
  - Slow down the AUTO-ID increasing speed when the `Update` statement does not modify the corresponding AUTO-INCREMENT column #7846
- Bug fixes
  - Quickly create a new etcd session to recover the service when the PD leader goes down #7810
  - Fix the issue that the time zone is not considered when the default value of the `DateTime` type is calculated #7672
  - Fix the issue that `duplicate key update` inserts values incorrectly in some conditions #7685

- Fix the issue that the predicate conditions of `UnionScan` are not pushed down #7726
- Fix the issue that the time zone is not correctly handled when you add the `TIMESTAMP` index #7812
- Fix the memory leak issue caused by the statistics module in some conditions #7864
- Fix the issue that the results of `ANALYZE` cannot be obtained in some abnormal conditions #7871
- Do not fold the function `SYSDATE`, to ensure the returned results are correct #7894
- Fix the `substring_index` panic issue in some conditions #7896
- Fix the issue that `OUTER JOIN` is mistakenly converted to `INNER JOIN` in some conditions #7899

16.16.4.2 TiKV

- Bug fix
  - Fix the issue that the memory consumed by Raftstore `EntryCache` keeps increasing when a node goes down #3529

16.16.5 TiDB 2.0.7 Release Notes

On September 7, 2018, TiDB 2.0.7 is released. Compared with TiDB 2.0.6, this release has great improvement in system compatibility and stability.

16.16.5.1 TiDB

- New Feature
  - Add the `PROCESSLIST` table in `information_schema` #7286
- Improvement
  - Collect more details about SQL statement execution and output the information in the `SLOW QUERY` log #7364
  - Drop the partition information in `SHOW CREATE TABLE` #7388
  - Improve the execution efficiency of the `ANALYZE` statement by setting it to the RC isolation level and low priority #7500
  - Speed up adding a unique index #7562
  - Add an option of controlling the DDL concurrency #7563
- Bug Fixes
  - Fix the issue that `USE INDEX(PRIMARY)` cannot be used in a table whose primary key is an integer #7298

- Fix the issue that `Merge Join` and `Index Join` output incorrect results when the inner row is `NULL` #7301
- Fix the issue that `Join` outputs an incorrect result when the chunk size is set too small #7315
- Fix the panic issue caused by a statement of creating a table involving `range column` #7379
- Fix the issue that `admin check table` mistakenly reports an error of a time-type column #7457
- Fix the issue that the data with a default value `current_timestamp` cannot be queried using the `=` condition #7467
- Fix the issue that the zero-length parameter inserted by using the `ComStmtSendLongData` command is mistakenly parsed to NULL #7508
- Fix the issue that `auto analyze` is repeatedly executed in specific scenarios #7556
- Fix the issue that the parser cannot parse a single line comment ended with a newline character #7635

16.16.5.2 TiKV

- Improvement
  - Open the `dynamic-level-bytes` parameter in an empty cluster by default, to reduce space amplification
- Bug Fix
  - Update `approximate size` and `approximate keys count` of a Region after Region merging

16.16.6 TiDB 2.0.6 Release Notes

On August 6, 2018, TiDB 2.0.6 is released. Compared with TiDB 2.0.5, this release has great improvement in system compatibility and stability.

16.16.6.1 TiDB

- Improvements
  - Make "set system variable" log shorter to save disk space #7031
  - Record slow operations during the execution of `ADD INDEX` in the log, to make troubleshooting easier #7083
  - Reduce transaction conflicts when updating statistics #7138
  - Improve the accuracy of row count estimation when the values pending to be estimated exceeds the statistics range #7185
  - Choose the table with a smaller estimated row count as the outer table for `Index Join` to improve its execution efficiency #7277

- Add the recover mechanism for panics occurred during the execution of `ANALYZE TABLE`, to avoid that the tidb-server is unavailable caused by abnormal behavior in the process of collecting statistics #7228
- Return `NULL` and the corresponding warning when the results of `RPAD`/`LPAD` exceed the value of the `max_allowed_packet` system variable, compatible with MySQL #7244
- Set the upper limit of placeholders count in the `PREPARE` statement to 65535, compatible with MySQL #7250

- Bug Fixes
  - Fix the issue that the `DROP USER` statement is incompatible with MySQL behavior in some cases #7014
  - Fix the issue that statements like `INSERT`/`LOAD DATA` meet OOM after opening `tidb_batch_insert` #7092
  - Fix the issue that the statistics fail to automatically update when the data of a table keeps updating #7093
  - Fix the issue that the firewall breaks inactive gPRC connections #7099
  - Fix the issue that prefix index returns a wrong result in some scenarios #7126
  - Fix the panic issue caused by outdated statistics in some scenarios #7155
  - Fix the issue that one piece of index data is missed after the `ADD INDEX` operation in some scenarios #7156
  - Fix the wrong result issue when querying `NULL` values using the unique index in some scenarios #7172
  - Fix the messy code issue of the `DECIMAL` multiplication result in some scenarios #7212
  - Fix the wrong result issue of `DECIMAL` modulo operation in some scenarios #7245
  - Fix the issue that the `UPDATE`/`DELETE` statement in a transaction returns a wrong result under some special sequence of statements #7219
  - Fix the panic issue of the `UNION ALL`/`UPDATE` statement during the process of building the execution plan in some scenarios #7225
  - Fix the issue that the range of prefix index is calculated incorrectly in some scenarios #7231
  - Fix the issue that the `LOAD DATA` statement fails to write the binlog in some scenarios #7242
  - Fix the wrong result issue of `SHOW CREATE TABLE` during the execution process of `ADD INDEX` in some scenarios #7243
  - Fix the issue that panic occurs when `Index Join` does not initialize timestamps in some scenarios #7246
  - Fix the false alarm issue when `ADMIN CHECK TABLE` mistakenly uses the timezone in the session #7258
  - Fix the issue that `ADMIN CLEANUP INDEX` does not clean up the index in some scenarios #7265
  - Disable the Read Committed isolation level #7282

16.16.6.2 TiKV

- Improvements
    - Enlarge scheduler's default slots to reduce false conflicts
    - Reduce continuous records of rollback transactions, to improve the Read performance when conflicts are extremely severe
    - Limit the size and number of RocksDB log files, to reduce unnecessary disk usage in long-running condition
- Bug Fixes
    - Fix the crash issue when converting the data type from string to decimal

16.16.7 TiDB 2.0.5 Release Notes

On July 6, 2018, TiDB 2.0.5 is released. Compared with TiDB 2.0.4, this release has great improvement in system compatibility and stability.

16.16.7.1 TiDB

- New Features
    - Add the `tidb_disable_txn_auto_retry` system variable which is used to disable the automatic retry of transactions #6877
- Improvements
    - Optimize the cost calculation of `Selection` to make the result more accurate #6989
    - Select the query condition that completely matches the unique index or the primary key as the query path directly #6966
    - Execute necessary cleanup when failing to start the service #6964
    - Handle `\N` as NULL in the `Load Data` statement #6962
    - Optimize the code structure of CBO #6953
    - Report the monitoring metrics earlier when starting the service #6931
    - Optimize the format of slow queries by removing the line breaks in SQL statements and adding user information #6920
    - Support multiple asterisks in comments #6858
- Bug Fixes
    - Fix the issue that `KILL QUERY` always requires SUPER privilege #7003
    - Fix the issue that users might fail to login when the number of users exceeds 1024 #6986
    - Fix an issue about inserting unsigned `float/double` data #6940
    - Fix the compatibility of the `COM_FIELD_LIST` command to resolve the panic issue in some MariaDB clients #6929
    - Fix the `CREATE TABLE IF NOT EXISTS LIKE` behavior #6928
    - Fix an issue in the process of TopN pushdown #6923
    - Fix the ID record issue of the currently processing row when an error occurs in executing `Add Index` #6903

16.16.7.2 PD

- Fix the issue that replicas migration uses up TiKV disks space in some scenarios
- Fix the crash issue caused by `AdjacentRegionScheduler`

16.16.7.3 TiKV

- Fix the potential overflow issue in decimal operations
- Fix the dirty read issue that might occur in the process of merging

16.16.8 TiDB 2.0.4 Release Notes

On June 15, 2018, TiDB 2.0.4 is released. Compared with TiDB 2.0.3, this release has great improvement in system compatibility and stability.

16.16.8.1 TiDB

- Support the `ALTER TABLE t DROP COLUMN a CASCADE` syntax
- Support configuring the value of `tidb_snapshot` to TSO
- Refine the display of statement types in monitoring items
- Optimize the accuracy of query cost estimation
- Configure the `backoff max delay` parameter of gRPC
- Support configuring the memory threshold of a single statement in the configuration file
- Refactor the error of Optimizer
- Fix the side effects of the `Cast Decimal` data
- Fix the wrong result issue of the `Merge Join` operator in specific scenarios
- Fix the issue of converting the Null object to String
- Fix the issue of casting the JSON type of data to the JSON type
- Fix the issue that the result order is not consistent with MySQL in the condition of `Union` + `OrderBy`
- Fix the compliance rules issue when the `Union` statement checks the `Limit/OrderBy` clause
- Fix the compatibility issue of the `Union All` result
- Fix a bug in predicate pushdown
- Fix the compatibility issue of the `Union` statement with the `For Update` clause
- Fix the issue that the `concat_ws` function mistakenly truncates the result

16.16.8.2 PD

- Improve the behavior of the unset scheduling argument `max-pending-peer-count` by changing it to no limit for the maximum number of `PendingPeer`s

16.16.8.3 TiKV

- Add the RocksDB `PerfContext` interface for debugging
- Remove the `import-mode` parameter
- Add the `region-properties` command for `tikv-ctl`
- Fix the issue that `reverse-seek` is slow when many RocksDB tombstones exist
- Fix the crash issue caused by `do_sub`
- Make GC record the log when GC encounters many versions of data

16.16.9 TiDB 2.0.3 Release Notes

On June 1, 2018, TiDB 2.0.3 is released. Compared with TiDB 2.0.2, this release has great improvement in system compatibility and stability.

16.16.9.1 TiDB

- Support modifying the log level online
- Support the `COM_CHANGE_USER` command
- Support using the `TIME` type parameters under the binary protocol
- Optimize the cost estimation of query conditions with the `BETWEEN` expression
- Do not display the `FOREIGN KEY` information in the result of `SHOW CREATE TABLE`
- Optimize the cost estimation for queries with the `LIMIT` clause
- Fix the issue about the `YEAR` type as the unique index
- Fix the issue about `ON DUPLICATE KEY UPDATE` in conditions without the unique index
- Fix the compatibility issue of the `CEIL` function
- Fix the accuracy issue of the `DIV` calculation in the `DECIMAL` type
- Fix the false alarm of `ADMIN CHECK TABLE`
- Fix the panic issue of `MAX/MIN` under specific expression parameters
- Fix the issue that the result of `JOIN` is null in special conditions
- Fix the `IN` expression issue when building and querying Range
- Fix a Range calculation issue when using `Prepare` to query and `Plan Cache` is enabled
- Fix the issue that the Schema information is frequently loaded in abnormal conditions

16.16.9.2 PD

- Fix the panic issue when collecting hot-cache metrics in specific conditions
- Fix the issue about scheduling of the obsolete Regions

16.16.9.3 TiKV

- Fix the bug that the learner flag mistakenly reports to PD
- Report an error instead of getting a result if `divisor/dividend` is 0 in `do_div_mod`

16.16.10 TiDB 2.0.2 Release Notes

On May 21, 2018, TiDB 2.0.2 is released. Compared with TiDB 2.0.1, this release has great improvement in system stability.

16.16.10.1 TiDB

- Fix the issue of pushing down the Decimal division expression
- Support using the `USE INDEX` syntax in the `Delete` statement
- Forbid using the `shard_row_id_bits` feature in columns with `Auto-Increment`
- Add the timeout mechanism for writing Binlog

16.16.10.2 PD

- Make the balance leader scheduler filter the disconnected nodes
- Modify the timeout of the transfer leader operator to 10s
- Fix the issue that the label scheduler does not schedule when the cluster Regions are in an unhealthy state
- Fix the improper scheduling issue of `evict leader scheduler`

16.16.10.3 TiKV

- Fix the issue that the Raft log is not printed
- Support configuring more gRPC related parameters
- Support configuring the timeout range of leader election
- Fix the issue that the obsolete learner is not deleted
- Fix the issue that the snapshot intermediate file is mistakenly deleted

16.16.11 TiDB 2.0.1 Release Notes

On May 16, 2018, TiDB 2.0.1 is released. Compared with TiDB 2.0.0 (GA), this release has great improvement in MySQL compatibility and system stability.

16.16.11.1 TiDB

- Update the progress of `Add Index` to the DDL job information in real time
- Add the `tidb_auto_analyze_ratio` session variable to control the threshold value of automatic statistics update
- Fix an issue that not all residual states are cleaned up when the transaction commit fails
- Fix a bug about adding indexes in some conditions

- Fix the correctness related issue when DDL modifies surface operations in some concurrent scenarios
- Fix a bug that the result of `LIMIT` is incorrect in some conditions
- Fix a capitalization issue of the `ADMIN CHECK INDEX` statement to make its index name case insensitive
- Fix a compatibility issue of the `UNION` statement
- Fix a compatibility issue when inserting data of `TIME` type
- Fix a goroutine leak issue caused by `copIteratorTaskSender` in some conditions
- Add an option for TiDB to control the behaviour of Binlog failure
- Refactor the `Coprocessor` slow log to distinguish between the scenario of tasks with long processing time and long waiting time
- Log nothing when meeting MySQL protocol handshake error, to avoid too many logs caused by the load balancer Keep Alive mechanism
- Refine the "Out of range value for column" error message
- Fix a bug when there is a subquery in an `Update` statement
- Change the behaviour of handling `SIGTERM`, and do not wait for all queries to terminate anymore

16.16.11.2 PD

- Add the `Scatter Range` scheduler to balance Regions with the specified key range
- Optimize the scheduling of Merge Region to prevent the newly split Region from being merged
- Add Learner related metrics
- Fix the issue that the scheduler is mistakenly deleted after restart
- Fix the error that occurs when parsing the configuration file
- Fix the issue that the etcd leader and the PD leader are not replicated
- Fix the issue that Learner still appears after it is closed
- Fix the issue that Regions fail to load because the packet size is too large

16.16.11.3 TiKV

- Fix the issue that `SELECT FOR UPDATE` prevents others from reading
- Optimize the slow query log
- Reduce the number of `thread_yield` calls
- Fix the bug that raftstore is accidentally blocked when generating the snapshot
- Fix the issue that Learner cannot be successfully elected in special conditions
- Fix the issue that split might cause dirty read in extreme conditions
- Correct the default value of the read thread pool configuration
- Speed up Delete Range

16.16.12 TiDB 2.0 Release Notes

On April 27, 2018, TiDB 2.0 GA is released! Compared with TiDB 1.0, this release has great improvement in MySQL compatibility, SQL optimizer, executor, and stability.

16.16.12.1 TiDB

- SQL Optimizer
    - Use more compact data structure to reduce the memory usage of statistics information
    - Speed up loading statistics information when starting a tidb-server process
    - Support updating statistics information dynamically experimental
    - Optimize the cost model to provide more accurate query cost evaluation
    - Use `Count-Min Sketch` to estimate the cost of point queries more accurately
    - Support analyzing more complex conditions to make full use of indexes
    - Support manually specifying the `Join` order using the `STRAIGHT_JOIN` syntax
    - Use the Stream Aggregation operator when the `GROUP BY` clause is empty to improve the performance
    - Support using indexes for the `MAX/MIN` function
    - Optimize the processing algorithms for correlated subqueries to support decorrelating more types of correlated subqueries and transform them to `Left Outer Join`
    - Extend `IndexLookupJoin` to be used in matching the index prefix
- SQL Execution Engine
    - Refactor all operators using the Chunk architecture, improve the execution performance of analytical queries, and reduce memory usage. There is a significant improvement in the TPC-H benchmark result.
    - Support the Streaming Aggregation operators pushdown
    - Optimize the `Insert Into Ignore` statement to improve the performance by over 10 times
    - Optimize the `Insert On Duplicate Key Update` statement to improve the performance by over 10 times
    - Optimize `Load Data` to improve the performance by over 10 times
    - Push down more data types and functions to TiKV
    - Support computing the memory usage of physical operators, and specifying the processing behavior in the configuration file and system variables when the memory usage exceeds the threshold
    - Support limiting the memory usage by a single SQL statement to reduce the risk of OOM
    - Support using implicit RowID in CRUD operations
    - Improve the performance of point queries
- Server
    - Support the Proxy Protocol
    - Add more monitoring metrics and refine the log
    - Support validating the configuration files
    - Support obtaining the information of TiDB parameters through HTTP API
    - Resolve Lock in the Batch mode to speed up garbage collection

    - Support multi-threaded garbage collection
    - Support TLS
- Compatibility
    - Support more MySQL syntaxes
    - Support modifying the `lower_case_table_names` system variable in the configuration file to support the OGG data replication tool
    - Improve compatibility with the Navicat management tool
    - Support displaying the table creating time in `Information_Schema`
    - Fix the issue that the return types of some functions/expressions differ from MySQL
    - Improve compatibility with JDBC
    - Support more SQL Modes
- DDL
    - Optimize the `Add Index` operation to greatly improve the execution speed in some scenarios
    - Attach a lower priority to the `Add Index` operation to reduce the impact on online business
    - Output more detailed status information of the DDL jobs in `Admin Show DDL Jobs`
    - Support querying the original statements of currently running DDL jobs using `Admin Show DDL Job Queries JobID`
    - Support recovering the index data using `Admin Recover Index` for disaster recovery
    - Support modifying Table Options using the `Alter` statement

16.16.12.2 PD

- Support `Region Merge`, to merge empty Regions after deleting data experimental
- Support `Raft Learner` experimental
- Optimize the scheduler
    - Make the scheduler to adapt to different Region sizes
    - Improve the priority and speed of restoring data during TiKV outage
    - Speed up data transferring when removing a TiKV node
    - Optimize the scheduling policies to prevent the disks from becoming full when the space of TiKV nodes is insufficient
    - Improve the scheduling efficiency of the balance-leader scheduler
    - Reduce the scheduling overhead of the balance-region scheduler
    - Optimize the execution efficiency of the the hot-region scheduler
- Operations interface and configuration
    - Support TLS

    - Support prioritizing the PD leaders
    - Support configuring the scheduling policies based on labels
    - Support configuring stores with a specific label not to schedule the Raft leader
    - Support splitting Region manually to handle the hotspot in a single Region
    - Support scattering a specified Region to manually adjust Region distribution in some cases
    - Add check rules for configuration parameters and improve validity check of the configuration items
- Debugging interface
    - Add the `Drop Region` debugging interface
    - Add the interfaces to enumerate the health status of each PD
- Statistics
    - Add statistics about abnormal Regions
    - Add statistics about Region isolation level
    - Add scheduling related metrics
- Performance
    - Keep the PD leader and the etcd leader together in the same node to improve write performance
    - Optimize the performance of Region heartbeat

16.16.12.3 TiKV

- Features
    - Protect critical configuration from incorrect modification
    - Support `Region Merge` experimental
    - Add the `Raw DeleteRange` API
    - Add the `GetMetric` API
    - Add `Raw Batch Put`, `Raw Batch Get`, `Raw Batch Delete` and `Raw Batch Scan`
    - Add Column Family options for the RawKV API and support executing operation on a specific Column Family
    - Support Streaming and Streaming Aggregation in Coprocessor
    - Support configuring the request timeout of Coprocessor
    - Carry timestamps with Region heartbeats
    - Support modifying some RocksDB parameters online, such as `block-cache-size`
    - Support configuring the behavior of Coprocessor when it encounters some warnings or errors
    - Support starting in the importing data mode to reduce write amplification during the data importing process
    - Support manually splitting Region in halves
    - Improve the data recovery tool `tikv-ctl`

    - Return more statistics in Coprocessor to guide the behavior of TiDB
    - Support the `ImportSST` API to import SST files experimental
    - Add the TiKV Importer binary to integrate with TiDB Lightning to import data quickly experimental

- Performance

    - Optimize read performance using `ReadPool` and increase the `raw_get/get/batch_get` by 30%
    - Improve metrics performance
    - Inform PD immediately once the Raft snapshot process is completed to speed up balancing
    - Solve performance jitter caused by RocksDB flushing
    - Optimize the space reclaiming mechanism after deleting data
    - Speed up garbage cleaning while starting the server
    - Reduce the I/O overhead during replica migration using `DeleteFilesInRanges`

- Stability

    - Fix the issue that gRPC call does not get returned when the PD leader switches
    - Fix the issue that it is slow to offline nodes caused by snapshots
    - Limit the temporary space usage consumed by migrating replicas
    - Report the Regions that cannot elect a leader for a long time
    - Update the Region size information in time according to compaction events
    - Limit the size of scan lock to avoid request timeout
    - Limit the memory usage when receiving snapshots to avoid OOM
    - Increase the speed of CI test
    - Fix the OOM issue caused by too many snapshots
    - Configure `keepalive` of gRPC
    - Fix the OOM issue caused by an increase of the Region number

16.16.12.4 TiSpark

TiSpark uses a separate version number. The current TiSpark version is 1.0 GA. The components of TiSpark 1.0 provide distributed computing of TiDB data using Apache Spark.

- Provide a gRPC communication framework to read data from TiKV
- Provide encoding and decoding of TiKV component data and communication protocol
- Provide calculation pushdown, which includes:

    - Aggregate pushdown
    - Predicate pushdown
    - TopN pushdown
    - Limit pushdown

- Provide index related support

    - Transform predicate into Region key range or secondary index

    - Optimize `Index Only` queries - Adaptively downgrade index scan to table scan per Region
- Provide cost-based optimization
    - Support statistics
    - Select index
    - Estimate broadcast table cost
- Provide support for multiple Spark interfaces
    - Support Spark Shell
    - Support ThriftServer/JDBC
    - Support Spark-SQL interaction
    - Support PySpark Shell
    - Support SparkR

16.16.13 TiDB 2.0 RC5 Release Notes

On April 17, 2018, TiDB 2.0 RC5 is released. This release has great improvement in MySQL compatibility, SQL optimization and stability.

16.16.13.1 TiDB

- Fix the issue about applying the `Top-N` pushdown rule
- Fix the estimation of the number of rows for the columns that contain NULL values
- Fix the zero value of the Binary type
- Fix the `BatchGet` issue within a transaction
- Clean up the written data while rolling back the `Add Index` operation, to reduce consumed space
- Optimize the `insert on duplicate key update` statement to improve the performance by 10 times
- Fix the issue about the type of the results returned by the `UNIX_TIMESTAMP` function
- Fix the issue that the NULL value is inserted while adding NOT NULL columns
- Support showing memory usage of the executing statements in the `Show Process List` statement
- Fix the issue that `Alter Table Modify Column` reports an error in extreme conditions
- Support setting the table comment using the `Alter` statement

16.16.13.2 PD

- Add support for Raft Learner
- Optimize the Balance Region Scheduler to reduce scheduling overhead
- Adjust the default value of `schedule-limit` configuration
- Fix the issue of allocating ID frequently
- Fix the compatibility issue when adding a new scheduler

16.16.13.3 TiKV

- Support the Region specified by `compact` in `tikv-ctl`
- Support Batch Put, Batch Get, Batch Delete and Batch Scan in the RawKVClient
- Fix the OOM issue caused by too many snapshots
- Return more detailed error information in Coprocessor
- Support dynamically modifying the `block-cache-size` in TiKV through `tikv-ctl`
- Further improve `importer`
- Simplify the `ImportSST::Upload` interface
- Configure the `keepalive` property of gRPC
- Split `tikv-importer` from TiKV as an independent binary
- Provide statistics about the number of rows scanned by each `scan range` in Coprocessor
- Fix the compilation issue on the macOS system
- Fix the issue of misusing a RocksDB metric
- Support the `overflow as warning` option in Coprocessor

16.16.14 TiDB 2.0 RC4 Release Notes

On March 30, 2018, TiDB 2.0 RC4 is released. This release has great improvement in MySQL compatibility, SQL optimization and stability.

16.16.14.1 TiDB

- Support `SHOW GRANTS FOR CURRENT_USER();`
- Fix the issue that the `Expression` in `UnionScan` is not cloned
- Support the `SET TRANSACTION` syntax
- Fix the potential goroutine leak issue in `copIterator`
- Fix the issue that `admin check table` misjudges the unique index including null
- Support displaying floating point numbers using scientific notation
- Fix the type inference issue during binary literal computing
- Fix the issue in parsing the `CREATE VIEW` statement
- Fix the panic issue when one statement contains both `ORDER BY` and `LIMIT 0`
- Improve the execution performance of `DecodeBytes`
- Optimize `LIMIT 0` to `TableDual`, to avoid building useless execution plans

16.16.14.2 PD

- Support splitting Region manually to handle the hot spot in a single Region
- Fix the issue that the label property is not displayed when `pdctl` runs `config show all`
- Optimize metrics and code structure

16.16.14.3 TiKV

- Limit the memory usage during receiving snapshots, to avoid OOM in extreme conditions
- Support configuring the behavior of Coprocessor when it encounters warnings
- Support importing the data pattern in TiKV
- Support splitting Region in the middle
- Increase the speed of CI test
- Use `crossbeam channel`
- Fix the issue that too many logs are output caused by leader missing when TiKV is isolated

16.16.15 TiDB 2.0 RC3 Release Notes

On March 23, 2018, TiDB 2.0 RC3 is released. This release has great improvement in MySQL compatibility, SQL optimization and stability.

16.16.15.1 TiDB

- Fix the wrong result issue of `MAX/MIN` in some scenarios
- Fix the issue that the result of `Sort Merge Join` does not show in order of Join Key in some scenarios
- Fix the error of comparison between `uint` and `int` in boundary conditions
- Optimize checks on length and precision of the floating point type, to improve compatibility with MySQL
- Improve the parsing error log of time type and add more error information
- Improve memory control and add statistics about `IndexLookupExecutor` memory
- Optimize the execution speed of `ADD INDEX` to greatly increase the speed in some scenarios
- Use the Stream Aggregation operator when the `GROUP BY` substatement is empty, to increase the speed
- Support closing the `Join Reorder` optimization in the optimizer using `STRAIGHT_JOIN`
- Output more detailed status information of DDL jobs in `ADMIN SHOW DDL JOBS`
- Support querying the original statements of currently running DDL jobs using `ADMIN SHOW DDL JOB QUERIES`
- Support recovering the index data using `ADMIN RECOVER INDEX` for disaster recovery
- Attach a lower priority to the `ADD INDEX` operation to reduce the impact on online business
- Support aggregation functions with JSON type parameters, such as `SUM/AVG`
- Support modifying the `lower_case_table_names` system variable in the configuration file, to support the OGG data replication tool
- Improve compatibility with the Navicat management tool
- Support using implicit RowID in CRUD operations

16.16.15.2 PD

- Support Region Merge, to merge empty Regions or small Regions after deleting data
- Ignore the nodes that have a lot of pending peers during adding replicas, to improve the speed of restoring replicas or making nodes offline
- Fix the frequent scheduling issue caused by a large number of empty Regions
- Optimize the scheduling speed of leader balance in scenarios of unbalanced resources within different labels
- Add more statistics about abnormal Regions

16.16.15.3 TiKV

- Support Region Merge
- Inform PD immediately once the Raft snapshot process is completed, to speed up balancing
- Add the Raw DeleteRange API
- Add the GetMetric API
- Reduce the I/O fluctuation caused by RocksDB sync files
- Optimize the space reclaiming mechanism after deleting data
- Improve the data recovery tool `tikv-ctl`
- Fix the issue that it is slow to make nodes down caused by snapshot
- Support streaming in Coprocessor
- Support Readpool and increase the `raw_get/get/batch_get` by 30%
- Support configuring the request timeout of Coprocessor
- Support streaming aggregation in Coprocessor
- Carry time information in Region heartbeats
- Limit the space usage of snapshot files to avoid consuming too much disk space
- Record and report the Regions that cannot elect a leader for a long time
- Speed up garbage cleaning when starting the server
- Update the size information about the corresponding Region according to compaction events
- Limit the size of `scan lock` to avoid request timeout
- Use `DeleteRange` to speed up Region deletion
- Support modifying RocksDB parameters online

16.16.16 TiDB 2.0 RC1 Release Notes

On March 9, 2018, TiDB 2.0 RC1 is released. This release has great improvement in MySQL compatibility, SQL optimization and stability.

16.16.16.1 TiDB

- Support limiting the memory usage by a single SQL statement, to reduce the risk of OOM

- Support pushing the Stream Aggregate operator down to TiKV
- Support validating the configuration file
- Support obtaining the information of TiDB configuration through HTTP API
- Compatible with more MySQL syntax in Parser
- Improve the compatibility with Navicat
- Improve the optimizer and extract common expressions with multiple OR conditions, to choose better query plan
- Improve the optimizer and convert subqueries to Join operators in more scenarios, to choose better query plan
- Resolve Lock in the Batch mode to increase the garbage collection speed
- Fix the length of Boolean field to improve compatibility
- Optimize the Add Index operation and give lower priority to all write and read operations, to reduce the impact on online business

16.16.16.2 PD

- Optimize the logic of code used to check the Region status to improve performance
- Optimize the output of log information in abnormal conditions to facilitate debugging
- Fix the monitor statistics that the disk space of TiKV nodes is not enough
- Fix the wrong reporting issue of the health interface when TLS is enabled
- Fix the issue that concurrent addition of replicas might exceed the threshold value of configuration, to improve stability

16.16.16.3 TiKV

- Fix the issue that gRPC call is not cancelled when PD leaders switch
- Protect important configuration which cannot be changed after initial configuration
- Add gRPC APIs used to obtain metrics
- Check whether SSD is used when you start the cluster
- Optimize the read performance using ReadPool, and improve the performance by 30% in the `raw get` test
- Improve metrics and optimize the usage of metrics

16.16.17 TiDB 1.1 Beta Release Notes

On February 24, 2018, TiDB 1.1 Beta is released. This release has great improvement in MySQL compatibility, SQL optimization, stability, and performance.

16.16.17.1 TiDB

- Add more monitoring metrics and refine the log
- Compatible with more MySQL syntax
- Support displaying the table creating time in `information_schema`

- Optimize queries containing the `MaxOneRow` operator
- Configure the size of intermediate result sets generated by Join, to further reduce the memory used by Join
- Add the `tidb_config` session variable to output the current TiDB configuration
- Fix the panic issue in the `Union` and `Index Join` operators
- Fix the wrong result issue of the `Sort Merge Join` operator in some scenarios
- Fix the issue that the `Show Index` statement shows indexes that are in the process of adding
- Fix the failure of the `Drop Stats` statement
- Optimize the query performance of the SQL engine to improve the test result of the Sysbench Select/OLTP by 10%
- Improve the computing speed of subqueries in the optimizer using the new execution engine; compared with TiDB 1.0, TiDB 1.1 Beta has great improvement in tests like TPC-H and TPC-DS

16.16.17.2 PD

- Add the Drop Region debug interface
- Support setting priority of the PD leader
- Support configuring stores with a specific label not to schedule Raft leaders
- Add the interfaces to enumerate the health status of each PD
- Add more metrics
- Keep the PD leader and the etcd leader together as much as possible in the same node
- Improve the priority and speed of restoring data when TiKV goes down
- Enhance the validity check of the `data-dir` configuration item
- Optimize the performance of Region heartbeat
- Fix the issue that hot spot scheduling violates label constraint
- Fix other stability issues

16.16.17.3 TiKV

- Traverse locks using offset + limit to avoid potential GC problems
- Support resolving locks in batches to improve GC speed
- Support GC concurrency to improve GC speed
- Update the Region size using the RocksDB compaction listener for more accurate PD scheduling
- Delete the outdated data in batches using `DeleteFilesInRanges`, to make TiKV start faster
- Configure the Raft snapshot max size to avoid the retained files taking up too much space
- Support more recovery operations in `tikv-ctl`
- Optimize the ordered flow aggregation operation
- Improve metrics and fix bugs

16.16.18 TiDB 1.1 Alpha Release Notes

On January 19, 2018, TiDB 1.1 Alpha is released. This release has great improvement in MySQL compatibility, SQL optimization, stability, and performance.

16.16.18.1 TiDB

- SQL parser
    - Support more syntax
- SQL query optimizer
    - Use more compact structure to reduce statistics info memory usage
    - Speed up loading statistics info when starting tidb-server
    - Provide more accurate query cost evaluation
    - Use `Count-Min Sketch` to estimate the cost of queries using unique index more accurately
    - Support more complex conditions to make full use of index
- SQL executor
    - Refactor all executor operators using Chunk architecture, improve the execution performance of analytical statements and reduce memory usage
    - Optimize performance of the `INSERT IGNORE` statement
    - Push down more types and functions to TiKV
    - Support more `SQL_MODE`
    - Optimize the `Load Data` performance to increase the speed by 10 times
    - Optimize the `Use Database` performance
    - Support statistics on the memory usage of physical operators
- Server
    - Support the PROXY protocol

16.16.18.2 PD

- Add more APIs
- Support TLS
- Add more cases for scheduling Simulator
- Schedule to adapt to different Region sizes
- Fix some bugs about scheduling

16.16.18.3 TiKV

- Support Raft learner
- Optimize Raft Snapshot and reduce the I/O overhead
- Support TLS
- Optimize the RocksDB configuration to improve performance
- Optimize `count (*)` and query performance of unique index in Coprocessor
- Add more failpoints and stability test cases
- Solve the reconnection issue between PD and TiKV
- Enhance the features of the data recovery tool `tikv-ctl`
- Support splitting according to table in Region
- Support the `Delete Range` feature
- Support setting the I/O limit caused by snapshot
- Improve the flow control mechanism

16.17 v1.0

16.17.1 TiDB 1.0.8 Release Notes

On February 11, 2018, TiDB 1.0.8 is released with the following updates:

16.17.1.1 TiDB

- Fix issues in the `Outer Join` result in some scenarios
- Optimize the performance of the `InsertIntoIgnore` statement
- Fix the issue in the `ShardRowID` option
- Add limitation (Configurable, the default value is 5000) to the DML statements number within a transaction
- Fix an issue in the Table/Column aliases returned by the `Prepare` statement
- Fix an issue in updating statistics delta
- Fix a panic error in the `Drop Column` statement
- Fix an DML issue when running the `Add Column After` statement
- Improve the stability of the GC process by ignoring the regions with GC errors
- Run GC concurrently to accelerate the GC process
- Provide syntax support for the `CREATE INDEX` statement

16.17.1.2 PD

- Reduce the lock overheat of the region heartbeats
- Fix the issue that a hot region scheduler selects the wrong Leader

16.17.1.3 TiKV

- Use `DeleteFilesInRanges` to clear stale data and improve the TiKV starting speed
- Using `Decimal` in Coprocessor sum
- Sync the metadata of the received Snapshot compulsorily to ensure its safety

To upgrade from 1.0.7 to 1.0.8, follow the rolling upgrade order of PD -> TiKV -> TiDB.

16.17.2 TiDB 1.0.7 Release Notes

On January 22, 2018, TiDB 1.0.7 is released with the following updates:

16.17.2.1 TiDB

- Optimize the `FIELD_LIST` command
- Fix data race of the information schema
- Avoid adding read-only statements to history
- Add the `session` variable to control the log query
- Fix the resource leak issue in statistics
- Fix the goroutine leak issue
- Add schema info API for the http status server
- Fix an issue about `IndexJoin`
- Update the behavior when `RunWorker` is false in DDL
- Improve the stability of test results in statistics
- Support `PACK_KEYS` syntax for the `CREATE TABLE` statement
- Add `row_id` column for the null pushdown schema to optimize performance

16.17.2.2 PD

- Fix possible scheduling loss issue in abnormal conditions
- Fix the compatibility issue with proto3
- Add the log

16.17.2.3 TiKV

- Support `Table Scan`
- Support the remote mode in tikv-ctl
- Fix the format compatibility issue of tikv-ctl proto
- Fix the loss of scheduling command from PD
- Add timeout in Push metric

To upgrade from 1.0.6 to 1.0.7, follow the rolling upgrade order of PD -> TiKV -> TiDB.

16.17.3 TiDB 1.0.6 Release Notes

On January 08, 2018, TiDB 1.0.6 is released with the following updates:

16.17.3.1 TiDB

- Support the `Alter Table Auto_Increment` syntax
- Fix the bug in Cost Based computation and the `Null Json` issue in statistics
- Support the extension syntax to shard the implicit row ID to avoid write hot spot for a single table
- Fix a potential DDL issue
- Consider the timezone setting in the `curtime`, `sysdate` and `curdate` functions
- Support the `SEPARATOR` syntax in the `GROUP_CONCAT` function
- Fix the wrong return type issue of the `GROUP_CONCAT` function.

16.17.3.2 PD

- Fix store selection problem of hot-region scheduler

16.17.3.3 TiKV

None.

To upgrade from 1.0.5 to 1.0.6, follow the rolling upgrade order of PD -> TiKV -> TiDB.

16.17.4 TiDB 1.0.5 Release Notes

On December 26, 2017, TiDB 1.0.5 is released with the following updates:

16.17.4.1 TiDB

- Add the max value for the current Auto_Increment ID in the `Show Create Table` statement.
- Fix a potential goroutine leak.
- Support outputting slow queries into a separate file.
- Load the `TimeZone` variable from TiKV when creating a new session.
- Support the schema state check so that the `Show Create Table`and `Analyze` statements process the public table/index only.
- The `set transaction read only` should affect the `tx_read_only` variable.
- Clean up incremental statistic data when rolling back.
- Fix the issue of missing index length in the `Show Create Table` statement.

16.17.4.2 PD

- Fix the issue that the leaders stop balancing under some circumstances.
    - 869
    - 874
- Fix potential panic during bootstrapping.

16.17.4.3 TiKV

- Fix the issue that it is slow to get the CPU ID using the `get_cpuid` function.
- Support the `dynamic-level-bytes` parameter to improve the space collection situation.

To upgrade from 1.0.4 to 1.0.5, follow the rolling upgrade order of PD -> TiKV -> TiDB.

16.17.5 TiDB 1.0.4 Release Notes

On December 11, 2017, TiDB 1.0.4 is released with the following updates:

16.17.5.1 TiDB

- Speed up the loading of the statistics when starting the `tidb-server`
- Improve the performance of the `show variables` statement
- Fix a potential issue when using the `Add Index` statement to handle the combined indexes
- Fix a potential issue when using the `Rename Table` statement to move a table to another database
- Accelerate the effectiveness for the `Alter/Drop User` statement

16.17.5.2 TiKV

- Fix a possible performance issue when a snapshot is applied
- Fix the performance issue for reverse scan after removing a lot of data
- Fix the wrong encoded result for the Decimal type under special circumstances

To upgrade from 1.0.3 to 1.0.4, follow the rolling upgrade order of PD -> TiKV -> TiDB.

16.17.6 TiDB 1.0.3 Release Notes

On November 28, 2017, TiDB 1.0.3 is released with the following updates:

16.17.6.1 TiDB

- Optimize the performance in transaction conflicts scenario
- Add the `TokenLimit` option in the config file
- Output the default database in slow query logs
- Remove the DDL statement from query duration metrics
- Optimize the query cost estimation
- Fix the index prefix issue when creating tables
- Support pushing down the expressions for the Float type to TiKV
- Fix the issue that it is slow to add index for tables with discrete integer primary index
- Reduce the unnecessary statistics updates
- Fix a potential issue during the transaction retry

16.17.6.2 PD

- Support adding more types of schedulers using API

16.17.6.3 TiKV

- Fix the deadlock issue with the PD client
- Fix the issue that the wrong leader value is prompted for `NotLeader`
- Fix the issue that the chunk size is too large in the coprocessor

To upgrade from 1.0.2 to 1.0.3, follow the rolling upgrade order of PD -> TiKV -> TiDB.

16.17.7 TiDB 1.0.2 Release Notes

On November 13, 2017, TiDB 1.0.2 is released with the following updates:

16.17.7.1 TiDB

- Optimize the cost estimation of index point query
- Support the `Alter Table Add Column (ColumnDef ColumnPosition)` syntax
- Optimize the queries whose `where` conditions are contradictory
- Optimize the `Add Index` operation to rectify the progress and reduce repetitive operations
- Optimize the `Index Look Join` operator to accelerate the query speed for small data size
- Fix the issue with prefix index judgment

16.17.7.2 Placement Driver (PD)

- Improve the stability of scheduling under exceptional situations

16.17.7.3 TiKV

- Support splitting table to ensure one region does not contain data from multiple tables
- Limit the length of a key to be no more than 4 KB
- More accurate read traffic statistics
- Implement deep protection on the coprocessor stack
- Fix the `LIKE` behavior and the `do_div_mod` bug

16.17.8 TiDB 1.0.1 Release Notes

On November 1, 2017, TiDB 1.0.1 is released with the following updates:

16.17.8.1 TiDB

- Support canceling DDL Job.
- Optimize the `IN` expression.
- Correct the result type of the `Show` statement.
- Support log slow query into a separate log file.
- Fix bugs.

16.17.8.2 TiKV

- Support flow control with write bytes.
- Reduce Raft allocation.
- Increase coprocessor stack size to 10MB.
- Remove the useless log from the coprocessor.

16.17.9 TiDB 1.0 Release Notes

On October 16, 2017, TiDB 1.0 is now released! This release is focused on MySQL compatibility, SQL optimization, stability, and performance.

16.17.9.1 TiDB

- The SQL query optimizer:
    - Adjust the cost model
    - Analyze pushdown
    - Function signature pushdown
- Optimize the internal data format to reduce the interim data size
- Enhance the MySQL compatibility
- Support the `NO_SQL_CACHE` syntax and limit the cache usage in the storage engine
- Refactor the Hash Aggregator operator to reduce the memory usage
- Support the Stream Aggregator operator

16.17.9.2 PD

- Support read flow based balancing
- Support setting the Store weight and weight based balancing

16.17.9.3 TiKV

- Coprocessor now supports more pushdown functions
- Support pushing down the sampling operation
- Support manually triggering data compact to collect space quickly
- Improve the performance and stability
- Add a Debug API for debugging
- TiSpark Beta Release:
- Support configuration framework
- Support ThriftSever/JDBC and Spark SQL

16.17.9.4 Acknowledgement

16.17.9.4.1 Special thanks to the following enterprises and teams

- Archon
- Mobike
- Samsung Electronics
- SpeedyCloud
- Tencent Cloud
- UCloud

16.17.9.4.2 Thanks to the open source software and services from the following organizations and individuals

- Asta Xie
- CNCF
- CoreOS
- Databricks
- Docker
- Github
- Grafana
- gRPC
- Jepsen
- Kubernetes
- Namazu
- Prometheus
- RedHat
- RocksDB Team
- Rust Team

16.17.9.4.3 Thanks to the individual contributors

- 8cbx
- Akihiro Suda
- aliyx
- alston111111
- andelf
- Andy Librian
- Arthur Yang
- astaxie
- Bai, Yang
- bailaohe
- Bin Liu
- Blame cosmos
- Breezewish
- Carlos Ferreira
- Ce Gao
- Changjian Zhang
- Cheng Lian
- Cholerae Hu
- Chu Chao
- coldwater
- Cole R Lawrence
- cuiqiu
- cuiyuan
- Cwen
- Dagang
- David Chen
- David Ding
- dawxy
- dcadevil
- Deshi Xiao
- Di Tang
- disksing
- dongxu
- dreamquster
- Drogon
- Du Chuan
- Dylan Wen
- eBoyy
- Eric Romano
- Ewan Chou
- Fiisio
- follitude
- Fred Wang

- fud
- fudali
- gaoyangxiaozhu
- Gogs
- goroutine
- Gregory Ian
- Guanqun Lu
- Guilherme Hübner Franco
- Haibin Xie
- Han Fei
- hawkingrei
- Hiroaki Nakamura
- hiwjd
- Hongyuan Wang
- Hu Ming
- Hu Ziming
- Huachao Huang
- HuaiyuXu
- Huxley Hu
- iamxy
- Ian
- insion
- iroi44
- Ivan.Yang
- Jack Yu
- jacky liu
- Jan Mercl
- Jason W
- Jay
- Jay Lee
- Jianfei Wang
- Jiaxing Liang
- Jie Zhou
- jinhelin
- Jonathan Boulle
- Karl Ostendorf
- knarfeh
- Kuiba
- leixuechun
- li
- Li Shihai
- Liao Qiang
- Light
- lijian
- Lilian Lee

- Liqueur Librazy
- Liu Cong
- Liu Shaohui
- liubo0127
- liyanan
- lkk2003rty
- Louis
- louishust
- luckcolors
- Lynn
- Mae Huang
- maiyang
- maxwell
- mengshangqi
- Michael Belenchenko
- mo2zie
- morefreeze
- MQ
- mxlxm
- Neil Shen
- netroby
- ngaut
- Nicole Nie
- nolouch
- onlymellb
- overvenus
- PaladinTyrion
- paulg
- Priya Seth
- qgxiaozhan
- qhsong
- Qiannan
- qiukeren
- qiuyesuifeng
- queenypingcap
- qupeng
- Rain Li
- ranxiaolong
- Ray
- Rick Yu
- shady
- ShawnLi
- Shen Li
- Sheng Tang
- Shirly

- Shuai Li
- ShuNing
- ShuYu Wang
- siddontang
- silenceper
- Simon J Mudd
- Simon Xia
- skimmilk6877
- sllt
- soup
- Sphinx
- Steffen
- sumBug
- sunhao2017
- Tao Meng
- Tao Zhou
- tennix
- tiancaiamao
- TianGuangyu
- Tristan Su
- ueizhou
- UncP
- Unknwon
- v01dstar
- Van
- WangXiangUSTC
- wangyanjun
- wangyisong1996
- weekface
- wegel
- Wei Fu
- Wenbin Xiao
- Wenting Li
- Wenxuan Shi
- winkyao
- woodpenker
- wuxuelian
- Xiang Li
- xiaojian cai
- Xuanjia Yang
- Xuanwo
- XuHuaiyu
- Yang Zhexuan
- Yann Autissier
- Yanzhe Chen

- Yiding Cui
- Yim
- youyouhu
- Yu Jun
- Yuwen Shen
- Zejun Li
- Zhang Yuning
- zhangjinpeng1987
- ZHAO Yijun
- Zhe-xuan Yang
- ZhengQian
- ZhengQianFang
- zhengwanbo
- ZhiFeng Hu
- Zhiyuan Zheng
- Zhou Tao
- Zhoubirdblue
- zhouningnan
- Ziyi Yan
- zs634134578
- zxylvlp
- zyguan
- zz-jason

16.17.10 Pre-GA Release Notes

On August 30, 2017, TiDB Pre-GA is released! This release is focused on MySQL compatibility, SQL optimization, stability, and performance.

16.17.10.1 TiDB

- The SQL query optimizer:
  - Adjust the cost model
  - Use index scan to handle the `where` clause with the `compare` expression which has different types on each side
  - Support the Greedy algorithm based Join Reorder
- Many enhancements have been introduced to be more compatible with MySQL
- Support `Natural Join`
- Support the JSON type (Experimental), including the query, update and index of the JSON fields
- Prune the useless data to reduce the consumption of the executor memory
- Support configuring prioritization in the SQL statements and automatically set the prioritization for some of the statements according to the query type
- Completed the expression refactor and the speed is increased by about 30%

16.17.10.2 Placement Driver (PD)

- Support manually changing the leader of the PD cluster

16.17.10.3 TiKV

- Use dedicated Rocksdb instance to store Raft log
- Use `DeleteRange` to speed up the deleting of replicas
- Coprocessor now supports more pushdown operators
- Improve the performance and stability

16.17.10.4 TiDB Connector for Spark Beta Release

- Implement the predicates pushdown
- Implement the aggregation pushdown
- Implement range pruning
- Capable of running full set of TPC+H except for one query that needs view support

16.17.11 TiDB RC4 Release Notes

On August 4, 2017, TiDB RC4 is released! This release is focused on MySQL compatibility, SQL optimization, stability, and performance.

16.17.11.1 Highlight

- For performance, the write performance is improved significantly, and the computing task scheduling supports prioritizing to avoid the impact of OLAP on OLTP.
- The optimizer is revised for a more accurate query cost estimating and for an automatic choice of the `Join` physical operator based on the cost.
- Many enhancements have been introduced to be more compatible with MySQL.
- TiSpark is now released to better support the OLAP business scenarios. You can now use Spark to access the data in TiKV.

16.17.11.2 Detailed updates

16.17.11.2.1 TiDB

- The SQL query optimizer refactoring:
  - Better support for TopN queries
  - Support the automatic choice of the of the `Join` physical operator based on the cost

    - Improved Projection Elimination

- The version check of schema is based on Table to avoid the impact of DDL on the ongoing transactions
- Support `BatchIndexJoin`
- Improve the `Explain` statement
- Improve the `Index Scan` performance
- Many enhancements have been introduced to be more compatible with MySQL
- Support the JSON type and operations
- Support the configuration of query prioritizing and isolation level

16.17.11.2.2 Placement Driver (PD)

- Support using PD to set the TiKV location labels
- Optimize the scheduler
    - PD is now supported to initialize the scheduling commands to TiKV.
    - Accelerate the response speed of the region heartbeat.
    - Optimize the `balance` algorithm
- Optimize data loading to speed up failover

16.17.11.2.3 TiKV

- Support the configuration of query prioritizing
- Support the RC isolation level
- Improve Jepsen test results and the stability
- Support Document Store
- Coprocessor now supports more pushdown functions
- Improve the performance and stability

16.17.11.2.4 TiSpark Beta Release

- Implement the prediction pushdown
- Implement the aggregation pushdown
- Implement range pruning
- Capable of running full set of TPC-H except one query that needs view support

16.17.12 TiDB RC3 Release Notes

On June 16, 2017, TiDB RC3 is released! This release is focused on MySQL compatibility, SQL optimization, stability, and performance.

16.17.12.1 Highlight

- The privilege management is refined to enable users to manage the data access privileges using the same way as in MySQL.
- DDL is accelerated.
- The load balancing policy and process are optimized for performance.
- TiDB Ansible is open sourced. By using TiDB-Ansible, you can deploy, upgrade, start and shutdown a TiDB cluster with one click.

16.17.12.2 Detailed updates

16.17.12.3 TiDB

- The following features are added or improved in the SQL query optimizer:
    - Support incremental statistics
    - Support the `Merge Sort Join` operator
    - Support the `Index Lookup Join` operator
    - Support the `Optimizer Hint` Syntax
    - Optimize the memory consumption of the `Scan`, `Join`, `Aggregation` operators
    - Optimize the Cost Based Optimizer (CBO) framework
    - Refactor `Expression`
- Support more complete privilege management
- DDL acceleration
- Support using HTTP API to get the data distribution information of tables
- Support using system variables to control the query concurrency
- Add more MySQL built-in functions
- Support using system variables to automatically split a big transaction into smaller ones to commit

16.17.12.4 Placement Driver (PD)

- Support gRPC
- Provide the Disaster Recovery Toolkit
- Use Garbage Collection to clear stale data automatically
- Support more efficient data balance
- Support hot Region scheduling to enable load balancing and speed up the data importing
- Performance
    - Accelerate getting Client TSO
    - Improve the efficiency of Region Heartbeat processing
- Improve the `pd-ctl` function

- Update the Replica configuration dynamically
- Get the Timestamp Oracle (TSO)
- Use ID to get the Region information

16.17.12.5 TiKV

- Support gRPC
- Support the Sorted String Table (SST) format snapshot to improve the load balancing speed of a cluster
- Support using the Heap Profile to uncover memory leaks
- Support Streaming SIMD Extensions (SSE) and speed up the CRC32 calculation
- Accelerate transferring leader for faster load balancing
- Use Batch Apply to reduce CPU usage and improve the write performance
- Support parallel Prewrite to improve the transaction write speed
- Optimize the scheduling of the coprocessor thread pool to reduce the impact of big queries on point get
- The new Loader supports data importing at the table level, as well as splitting a big table into smaller logical blocks to import concurrently to improve the data importing speed.

16.17.13 TiDB RC2 Release Notes

On March 1, 2017, TiDB RC2 is released! This release is focused on the compatibility with MySQL, SQL query optimizer, system stability and performance in this version. What's more, a new permission management mechanism is added and users can control data access in the same way as the MySQL privilege management system.

16.17.13.1 TiDB

- Query optimizer
  - Collect column/index statistics and use them in the query optimizer
  - Optimize the correlated subquery
  - Optimize the Cost Based Optimizer (CBO) framework
  - Eliminate aggregation using unique key information
  - Refactor expression evaluation framework
  - Convert Distinct to GroupBy
  - Support the topn operation push-down
- Support basic privilege management
- Add lots of MySQL built-in functions
- Improve the Alter Table statement and support the modification of table name, default value and comment
- Support the Create Table Like statement

- Support the Show Warnings statement
- Support the Rename Table statement
- Restrict the size of a single transaction to avoid the cluster blocking of large transactions
- Automatically split data in the process of Load Data
- Optimize the performance of the AddIndex and Delete statement
- Support "ANSI_QUOTES" sql_mode
- Improve the monitoring system
- Fix Bugs
- Solve the problem of memory leak

16.17.13.2 PD

- Support location aware replica scheduling
- Conduct fast scheduling based on the number of region
- pd-ctl support more features
    - Add or delete PD
    - Obtain Region information with Key
    - Add or delete scheduler and operator
    - Obtain cluster label information

16.17.13.3 TiKV

- Support Async Apply to improve the entire write performance
- Use prefix seek to improve the read performance of Write CF
- Use memory hint prefix to improve the insert performance of Raft CF
- Optimize the single read transaction performance
- Support more push-down expressions
- Improve the monitoring system
- Fix Bugs

16.17.14 TiDB RC1 Release Notes

On December 23, 2016, TiDB RC1 is released. See the following updates in this release:

16.17.14.1 TiKV

- The write speed has been improved.
- The disk space usage is reduced.
- Hundreds of TBs of data can be supported.
- The stability is improved and TiKV can support a cluster with 200 nodes.
- Supports the Raw KV API and the Golang client.

16.17.14.2 Placement Driver (PD)

- The scheduling strategy framework is optimized and now the strategy is more flexible and reasonable.
- The support for `label` is added to support Cross Data Center scheduling.
- PD Controller is provided to operate the PD cluster more easily.

16.17.14.3 TiDB

- The following features are added or improved in the SQL query optimizer:
    - Eager aggregation
    - More detailed `EXPLAIN` information
    - Parallelization of the `UNION` operator
    - Optimization of the subquery performance
    - Optimization of the conditional push-down
    - Optimization of the Cost Based Optimizer (CBO) framework
- The implementation of the time related data types are refactored to improve the compatibility with MySQL.
- More built-in functions in MySQL are supported.
- The speed of the `add index` statement is enhanced.
- The following statements are supported:
    - Use the `CHANGE COLUMN` statement to change the name of a column.
    - Use `MODIFY COLUMN` and `CHANGE COLUMN` of the `ALTER TABLE` statement for some of the column type transfer.

16.17.14.4 New tools

- `Loader` is added to be compatible with the `mydumper` data format in Percona and provides the following functions:
    - Multi-thread import
    - Retry if error occurs
    - Breakpoint resume
    - Targeted optimization for TiDB
- The tool for one-click deployment is added.

17 Glossary

17.1 A

17.1.1 ACID

ACID refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability. Each of these properties is described below.

- **Atomicity** means that either all the changes of an operation are performed, or none of them are. TiDB ensures the atomicity of the Region that stores the Primary Key to achieve the atomicity of transactions.
- **Consistency** means that transactions always bring the database from one consistent state to another. In TiDB, data consistency is ensured before writing data to the memory.
- **Isolation** means that a transaction in process is invisible to other transactions until it completes. This allows concurrent transactions to read and write data without sacrificing consistency. TiDB currently supports the isolation level of `REPEATABLE READ`.
- **Durability** means that once a transaction is committed, it remains committed even in the event of a system failure. TiKV uses persistent storage to ensure durability.

17.2 B

17.2.1 Batch Create Table

Batch Create Table is a feature introduced in TiDB v6.0.0. This feature is enabled default. When restoring data with a large number of tables (nearly 50000) using BR (Backup & Restore), the feature can greatly speed up the restore process by creating tables in batches. For details, see Batch Create Table.

17.2.2 Baseline Capturing

Baseline Capturing captures queries that meet capturing conditions and create bindings for them. It is used for preventing regression of execution plans during an upgrade.

17.2.3 Bucket

A Region is logically divided into several small ranges called bucket. TiKV collects query statistics by buckets and reports the bucket status to PD. For details, see the Bucket design doc.

17.3 C

17.3.1 Cached Table

With the cached table feature, TiDB loads the data of an entire table into the memory of the TiDB server, and TiDB directly gets the table data from the memory without accessing TiKV, which improves the read performance.

17.3.2 Continuous Profiling

Introduced in TiDB 5.3.0, Continuous Profiling is a way to observe resource overhead at the system call level. With the support of Continuous Profiling, TiDB provides performance insight as clear as directly looking into the database source code, and helps R&D and operation and maintenance personnel to locate the root cause of performance problems using a flame graph. For details, see TiDB Dashboard Instance Profiling - Continuous Profiling.

17.4 D

17.4.1 Dynamic Pruning

Dynamic proning mode is one of the modes that TiDB accesses partitioned tables. In dynamic pruning mode, each operator supports direct access to multiple partitions. Therefore, TiDB no longer uses Union. Omitting the Union operation can improve the execution efficiency and avoid the problem of Union concurrent execution.

17.5 I

17.5.1 Index Merge

Index Merge is a method introduced in TiDB v4.0 to access tables. Using this method, the TiDB optimizer can use multiple indexes per table and merge the results returned by each index. In some scenarios, this method makes the query more efficient by avoiding full table scans. Since v5.4, Index Merge has become a GA feature.

17.5.2 In-Memory Pessimistic Lock

The in-memory pessimistic lock is a new feature introduced in TiDB v6.0.0. When this feature is enabled, pessimistic locks are usually stored in the memory of the Region leader only, and are not persisted to disk or replicated through Raft to other replicas. This feature can greatly reduce the overhead of acquiring pessimistic locks and improve the throughput of pessimistic transactions.

17.6 L

17.6.1 leader/follower/learner

Leader/Follower/Learner each corresponds to a role in a Raft group of peers. The leader services all client requests and replicates data to the followers. If the group leader fails, one of the followers will be elected as the new leader. Learners are non-voting followers that only serves in the process of replica addition.

17.7 M

17.7.1 Multi-version concurrency control (MVCC)

MVCC is a concurrency control mechanism in TiDB and other databases. It processes the memory read by transactions to achieve concurrent access to TiDB, thereby avoiding blocking caused by conflicts between concurrent reads and writes.

17.8 O

17.8.1 Old value

The "original value" in the incremental change log output by TiCDC. You can specify whether the incremental change log output by TiCDC contains the "original value".

17.8.2 Operator

An operator is a collection of actions that applies to a Region for scheduling purposes. Operators perform scheduling tasks such as "migrate the leader of Region 2 to Store 5" and "migrate replicas of Region 2 to Store 1, 4, 5".

An operator can be computed and generated by a scheduler, or created by an external API.

17.8.3 Operator step

An operator step is a step in the execution of an operator. An operator normally contains multiple Operator steps.

Currently, available steps generated by PD include:

- `TransferLeader`: Transfers leadership to a specified member
- `AddPeer`: Adds peers to a specified store
- `RemovePeer`: Removes a peer of a Region
- `AddLearner`: Adds learners to a specified store
- `PromoteLearner`: Promotes a specified learner to a voting member
- `SplitRegion`: Splits a specified Region into two

17.9 P

17.9.1 pending/down

"Pending" and "down" are two special states of a peer. Pending indicates that the Raft log of followers or learners is vastly different from that of leader. Followers in pending cannot be elected as leader. "Down" refers to a state that a peer ceases to respond to leader for a long time, which usually means the corresponding node is down or isolated from the network.

17.9.2 Predicate columns

In most cases, when executing SQL statements, the optimizer only uses statistics of some columns (such as columns in the `WHERE`, `JOIN`, `ORDER BY`, and `GROUP BY` statements). These used columns are called predicate columns. For details, see Collect statistics on some columns.

17.10 Q

17.10.1 Quota Limiter

Quota Limiter is an experimental feature introduced in TiDB v6.0.0. If the machine on which TiKV is deployed has limited resources, for example, with only 4v CPU and 16 G memory, and the foreground of TiKV processes too many read and write requests, the CPU resources used by the background are occupied to help process such requests, which affects the performance stability of TiKV. To avoid this situation, the quota-related configuration items can be set to limit the CPU resources to be used by the foreground.

17.11 R

17.11.1 Raft Engine

Raft Engine is an embedded persistent storage engine with a log-structured design. It is built for TiKV to store multi-Raft logs. Since v5.4, TiDB supports using Raft Engine as the log storage engine. For details, see Raft Engine.

17.11.2 Region/peer/Raft group

Region is the minimal piece of data storage in TiKV, each representing a range of data (96 MiB by default). Each Region has three replicas by default. A replica of a Region is called a peer. Multiple peers of the same Region replicate data via the Raft consensus algorithm, so peers are also members of a Raft instance. TiKV uses Multi-Raft to manage data. That is, for each Region, there is a corresponding, isolated Raft group.

17.11.3 Region split

Regions are generated as data writes increase. The process of splitting is called Region split.

The mechanism of Region split is to use one initial Region to cover the entire key space, and generate new Regions through splitting existing ones every time the size of the Region or the number of keys has reached a threshold.

17.11.4 restore

Restore is the reverse of the backup operation. It is the process of bringing back the system to an earlier state by retrieving data from a prepared backup.

17.12 S

17.12.1 scheduler

Schedulers are components in PD that generate scheduling tasks. Each scheduler in PD runs independently and serves different purposes. The commonly used schedulers are:

- `balance-leader-scheduler`: Balances the distribution of leaders
- `balance-region-scheduler`: Balances the distribution of peers
- `hot-region-scheduler`: Balances the distribution of hot Regions
- `evict-leader-{store-id}`: Evicts all leaders of a node (often used for rolling upgrades)

17.12.2 Store

A store refers to the storage node in the TiKV cluster (an instance of `tikv-server`). Each store has a corresponding TiKV instance.

17.13 T

17.13.1 Top SQL

Top SQL helps locate SQL queries that contribute to a high load of a TiDB or TiKV node in a specified time range. For details, see Top SQL user document.

17.13.2 TSO

Because TiKV is a distributed storage system, it requires a global timing service, Timestamp Oracle (TSO), to assign a monotonically increasing timestamp. In TiKV, such a feature

is provided by PD, and in Google Spanner, this feature is provided by multiple atomic clocks and GPS.

© 2025 PingCAP. All Rights Reserved.