TDSQL-C for MySQL

Kernel Features

Product Documentation

Contents

- Kernel Features
  - Kernel Overview
  - Kernel Version Release Notes
    - TXSQL Engine Kernel Version Release Notes
    - Database Proxy Version Release Notes
  - Optimized Kernel Version
    - Compilation Optimization High-Performance Version
  - Functionality Features
    - Automatic Killing of Idle Transactions
    - Instant DDL Overview
    - Dynamic Thread Pool
    - NOWAIT
    - RETURNING
    - Flashback Query
  - Performance Features
    - Optimization of Plan Caching Point Query
    - Auto-Increment Column Persistence
    - Invisible Index
    - Computation Pushdown
    - Parallel Initialization of InnoDB Buffer Pool
  - Stability Feature
    - Statement Outline
    - Hotspot Update Protection

Kernel Features
Kernel Overview

Last updated：2023-11-01 16:22:57

The engine kernel of TDSQL-C for MySQL is fully compatible with native MySQL. You can migrate MySQL data to TDSQL-C for MySQL without modifying any application code or configuration.
The engine kernel of TDSQL-C for MySQL provides various features similar to those in MySQL Enterprise Edition, including enterprise-level transparent data encryption, audit, thread pool, encryption function, and backup and restoration.
The engine kernel of TDSQL-C for MySQL not only deeply optimizes the InnoDB storage engine and query performance, but also improves the ease of use and maintainability of databases. While providing all the benefits of MySQL, it offers more enterprise-grade advanced features such as disaster recovery, restoration, monitoring, performance optimization, read/write separation, transparent data encryption (TDE), and database audit.

More information on the engine kernel of TDSQL-C for MySQL

For updates of the TDSQL-C for MySQL engine kernel version, see Kernel Version Release Notes.
The kernel minor version of TDSQL-C for MySQL can be upgraded automatically or manually. For more information, see Upgrading Kernel Minor Version.

More information on the proxy kernel of TDSQL-C for MySQL:

The proxy minor version of TDSQL-C for MySQL can be upgraded. For more information, see Upgrading Kernel Minor Version of Database Proxy.

Release dates of different TDSQL-C for MySQL kernel versions:

TDSQL-C for MySQL 8.0

Kernel Version	Release Date
3.1.10	June 2023
3.1.9	November 2022
3.1.8	October 2022
3.1.7	September 2022
3.1.6	August 2022
3.1.5	July 2022
3.1.3	June 2022
3.1.2	February 2022

3.1.1	November 2021
3.0.1	August 2021

TDSQL-C for MySQL 5.7

Kernel Version	Release Date
2.1.10	July 2023
2.0.23/2.1.9	May 2023
2.0.22/2.1.8	November 2022
2.0.21/2.1.7	September 2022
2.0.20/2.1.6	August 2022
2.0.19	July 2022
2.0.17	June 2022
2.0.16	January 2022
2.0.15	October 2021
2.0.14	July 2021
2.0.13	March 2021
2.0.12	November 2020
2.0.11	June 2020

Release dates of different TDSQL-C for MySQL proxy kernel versions:

Database Proxy Kernel Version	Release Date
1.3.7	May 2023
1.3.5	November 2022
1.3.4	September 2022
1.3.3	August 2022
1.2.1	July 2022

Kernel Version Release Notes
TXSQL Engine Kernel Version Release Notes

Last updated：2024-07-31 14:57:39

This document describes the version updates of the TDSQL-C for MySQL kernel.

Note:

If you need to upgrade the kernel version, see Upgrading Kernel Minor Version. If you want to learn about the release dates of each kernel version, see Kernel Overview.

TDSQL-C for MySQL 8.0 Kernel Release Notes

TDSQL-C for MySQL 5.7 Kernel Release Notes

Minor Version	Description
3.1.14	Feature updates Supported Columnar Storage Index(CSI) (in regional grayscale release), which uses the columnar data format for storing, searching, and managing data, achieving better query performance and higher data compression rates. Supported enhanced parallel query features: parallel subqueries, rollup, Nested-Loop Join (NLJ) inner table parallelism, and in-mem hash join parallelism. Bug Fixes Fixed the issue of RO crash caused by DDL operations on partition tables. Fixed the issue of garbled characters when a rotate event checksum is received during CDC subscription. Fixed the issue of precision loss when {} is used to directly insert a JSON string.
3.1.13	Performance optimizations Optimized the issue of slow data access due to low cyclic efficiency in full table scan. Optimized the issue of decreased large query scan performance in concurrent scenarios. Bug Fixes Fixed the issue of CPU occupancy after the database auditing feature is enabled. Fixed the issue of duplicate doc IDs in full-text indexes. Fixed the issue of read-only instance deadlock caused by full-text indexes in certain scenarios. Fixed the issue of read-only instance crash caused by DDL operations on partition tables. Fixed some official bugs as follows: Fixed bugs related to json prepare: Bug #101284. Fixed bugs related to sort buffer size: Bug #32738705, Bug #33501541. Fixed bugs related to temporary table index scan: Bug #33700735. Fixed bugs related to subquery: Bug #31216115, Bug #31946448, Bug #32813547, Bug #32813554. Fixed some segmentation faults: Bug #32813554, Bug #32813547.

3.1.12	Feature Updates Enhanced visibility in slow logs, now including metrics on storage layer network IO traffic, execution time, frequency, and transaction commit latency. Database auditing supported the addition of transaction ID fields and table names. Supports the BLACKHOLE storage engine (`ENGINE = BLACKHOLE;`). Supports the default algorithms for DDL execution (inplace/instant), which is configurable via the `innodb_alter_table_default_algorithm` parameter. Performance Optimization Large transaction commit binlog optimization: Large transaction logs are first written to a temporary file, which is then renamed the next binlog to reduce the time used by large transactions blocking other transactions. Binlog write optimization: Supports the conversion of row-format binlogs to statement-format binlogs, with regular expression filtering for effective databases and tables. Enhanced the performance of purging binlogs, reducing performance fluctuations. Removed CDC, LSN, and CLUSTER as reserved keywords. Optimized parallel DDL functionality: Fixed an issue where created index data is incorrect when the number of scan threads is greater than 1 but `innodb_parallel_ddl_threads = 1` . Addressed a crash caused by `m_page` becoming invalid after a subtree is created. Resolved an issue where error codes are not reset before indexes are created. Fixed a crash that occurs when the `innodb_disable_sort_file_cache` switch is enabled during parallel DDL index creation. Corrected a crash that happens when a unique key is added to a column with duplicate data. Bug Fixes Rectified an issue where database auditing incorrectly records the type of prepare statements. Fixed the problem where database auditing incorrectly classifies sql_type in execute statements as “other”. Addressed an error in retrieving custom variable strings in session track: now supports constructing the return value of session track by specifying the string length when fetching custom variables. Resolved an issue in parallel queries where referencing worker table fields in related subqueries leads to incorrect query results.
3.1.10	Feature updates Supported binlog subscription for read-only instances. Supported blackhole engine. Updated parallel query: Support parallel queries for full-table scans, full index scans, and index range scans; support variance and standard deviation functions; support setting parallel policies with the LIMIT syntax; and support Prepared Statement (PS) query mode. Supported automatic killing of idle transactions. Supported [dynamic thread pool]. Supported NOWAIT syntax. Supported flashback query. Supported the optimization of plan caching point query. Supported invisible index. Supported statement outline.

	Bug fixes Fixed the issue of excessive columns caused by INSTANT ADD on partitioned tables. The default algorithm for DDL has been changed from INSTANT to INPLACE. To use instant DDL, you need to specify `algorithm = instant` . Fixed the issue of prohibiting tables with a column name "fts_doc_id" from executing instant DDL. Optimized buffer pool resizing. Optimized the splitting logic of operator splitting for items in parallel queries. Fixed the performance degradation caused by the failure of binary search in the IN operator in PS mode. Optimized the query efficiency for full table count(*) in parallel queries. Fixed the issue where the stage variable error in Parallel DDL caused the stage null pointer to crash when creating FTS indexes. Fixed the possible crash when adding full-text indexes. Fixed the issue of GTID loss caused by the HA switch in certain scenarios. Fixed the issue of possible crash triggered by executing `show create table` after killing the mysqld process during DDL. Fixed the issue of premature release of the FTS cache lock in full-text indexing, which caused inconsistency between reads and writes. Fixed the issue of data inconsistency between source and replica caused by transaction rollback during the DDL commit process. Fixed the issue of deadlock occurred when killing a transaction and dropping a database during the process of creating a temp table and inserting data. Fixed the issue of failure when users perform operations on a database with the same name as the MySQL system database. Fixed the issue where Canal was unable to obtain the GTID starting point through `show master status` when extracting binlogs via read-only instances.
3.1.9	Feature updates Supported CDC, which can directly and repeatedly backtrack/extract the binlogs in the custom log retention period. This solves the problem where the compute node loses local binlogs in scenarios such as HA. For more information on how to set the binlog retention period, see Setting Backup Retention Period. Optimized the pages purge rate to improve the database performance. Bug fixes Fixed the issue where the worker couldn't pass the table-level NULL ROW FLAG to the coordinator and thus caused incorrect results. Fixed the core issue of parallel query caused by the failure of the sort operator to get the order list during operator splitting because sort order was pushed down to the table.
3.1.8	Feature updates Supported parallel query, which can automatically identify complicated queries. The parallel query capability leverages multiple compute cores to greatly shorten the response time of large queries. For more information on how to use this feature, see Enabling/Disabling Parallel Query. Bug fixes Fixed several database issues in debug mode.

	Fixed the issue where abnormal information appeared in database proxy-related fields when `show detailed processlist` was used to display connection information.
3.1.7	Feature updates Added the password dictionary parameter as described in Configuring Custom Password Strength, and optimized HA's dependence on dictionary files. Supported completing purged binlogs in the kernel. Supported the built-in database proxy for the Serverless architecture to implement connection persistence and momentary disconnection prevention, and fixed the connection error reported during the first wakeup.
3.1.6	Bug fixes Fixed the crash caused by full-text index (non-tree index) encountered during index check. Fixed the issue where the liveness probe of the database proxy caused the kernel to output a large number of error logs, and blocked the printing of redundant logs. Fixed the issue where the uninitialized GCR LSN in the session LSN tracker might return a random value of the database proxy and thus cause the statement execution to time out.
3.1.5	Feature updates Supported traffic throttling for `bulk insert` . Supported setting the change buffer and merge modes. Supported database proxy. For more information on how to use this feature, see Database Proxy Overview. Supported logical backup for read-only instances. Supported parallel replication of binlogs at the table level. Supported SQL throttling. Supported hotspot update protection. Supported interesting order judgment in sort merge join. Supported TABLESAMPLE. Supported the HISTOGRAM() function. Supported histogram versioning and compressed histogram. Supported `show detail processlist` . Performance optimizations Optimized the physical replication of transactions to greatly improve the write-only performance. Optimized the parallel initialization of the transaction system to accelerate the system startup. Optimized the logic of page locking during log replay in read-only instances to accelerate the replay thread. Bug fixes Fixed the issue where the MySQL client exited abnormally when receiving an incomplete package. Fixed the crash of the FSP management fraction caused by the incorrect commit order of the nested MTRs generated by blob. Fixed the transaction consistency issue that might be caused by the purge of the host when RO accessed the secondary index. Fixed the issue where `backup lock` couldn't be locked due to the `lock table` statement. Fixed the issue where several keywords introduced by TDSQL-C couldn't be used as identifiers, such as CDB_GET_TABLE_VERSION, CLUSTER, and THREADPOOL.

	Fixed the issue where the startup time was prolonged due to the failure to add `dict op lock` to the main thread caused by large transactions or long-line transaction rollbacks during instance startup. Fixed the crash caused by TRX reuse after the failure to allocate the undo page. Fixed the crash caused by executing `alter table` on a partitioned table to migrate from the extended tablespace to the system tablespace. Fixed the crash caused by the startup before the truncate log was completely written. Fixed the crash caused by inserting data after `drop table partition force` . Fixed the possible crash caused by rollbacks after instant DDL. Fixed the issue where creating tables in the extended tablespace with `create temporary table like` failed. Fixed the OOM caused by the continuous increase of the cache during data writes to the full-text index table. Fixed the error of unstable performance after hotspot update was enabled after optimization. Fixed the issue where `select count(*)` parallel scans caused full-table scans in extreme cases. Fixed the issue where the statistics were read as zero in various cases, and fixed the official Bug#31889883. Fixed the bug where queries were in the `query end` status for a long time. Fixed the case sensitivity issue of column names in the `json_table` function (official Bug#32591074). Fixed the bug where an error was reported when the Temptable engine was used and the number of aggregate functions in the selected column exceeded 255. Fixed the bug that caused correctness issues in window functions because expressions returned early during `return true` . Fixed the correctness issue caused by the pushdown by `derived condition pushdown` when it contained user variables. Fixed the issue where SQL filters were prone to crash when no namespaces were added in a rule. Fixed the QPS jitters when the thread pool was enabled under high concurrency and high conflict. Fixed the crash when information was not cleared during execution of the `update` statement or stored procedures. Fixed the issue where the histogram couldn't be stopped by CTRL+C on the existing version.
3.1.3	Feature updates Supported adding the binlog with the specified filename to an index file. Added a backup lock in the syntax of LOCK TABLES FOR BACKUP, UNLOCK TABLES. Added a binlog lock in the syntax of LOCK BINLOG FOR BACKUP, UNLOCK BINLOG. Bug fixes Fixed the bug where dynamic metadata persistence caused instance table corruptions or visibility errors. Merged the official bugfix #32897503 to solve the issue where the execution path of some query statements was incorrect under the `prepare` statement. Merged the official bugfix to solve the crash when `set resource group` failed. Fixed the bug where the previous `gtid` was empty after HA switch.

	Fixed the bug where an auto-increment column could be set to a value smaller than the inserted maximum value. Fixed the issue where explicit transactions in read-only instances would block the replay thread from replaying DDL logs. Fixed the issue where tables with "-" in the name might crash when replicated in a read-only instance after DDL. Fixed the crash caused by the `undo` request to the DDL log system table when a read-only instance experienced DDL recovery upon startup.
3.1.2	Feature updates Supported MySQL 8.0 for read-only nodes and source-replica physical replication. Supported table space expansion and up to above 1 PB of capacity per instance. Supported limiting the number of preloaded rows, which achieved a 1%-5% performance increase during point query testing. Supports extended ANALYZE syntax (UPDATE HISTOGRAM c USING DATA 'json') and direct writes to histograms. Performance optimizations Replaced index dive with histogram to reduce evaluation errors and I/O overheads (this capability is not enabled by default). Bug fixes Fixed the issue where updates related to large object pages were not written to the log when a full-text index containing large object columns was created. Fixed the issue with inconsistent formats of `undo page` and different definitions of `TRX_UNDO_HISTORY_NODE` in the computing and storage layers. Fixed the issue where statistics information might be zero during online-DDL. Fixed the issue where columns generated from replica instances were not updated. Fixed the issue where the instance hung when binlog was compressed. Fixed the issue of missing GTID in the previous_gtids event of the newly generated binlog file. Fixed possible deadlocks when system variables were modified. Fixed the issue where the information of the SQL thread of the replica instance in SHOW PROCESSLIST was incorrectly displayed. Implemented the bug fix related to hash join provided in MySQL 8.0.23. Implemented the bug fix related to writeset provided in MySQL. Implemented the bug fix related to the query optimizer provided in MySQL 8.0.24. Fixed the concurrency bugs of optimizing flush list and releasing pages in FAST DDL. Optimizes the memory usage during data dictionary update in instances with a large number of tables. Fixed the crash caused by new primary key creation after INSTANT ADD COLUMN. Fixed the OOM caused by memory growth in full-text index query. Fixed the issue where -1 was included in the TIME field in the result set returned by SHOW PROCESSLIST. Fixed the issue where tables might fail to be opened due to histogram compatibility. Fixed the floating point accumulation error when Singleton histograms were constructed. Fixed the replication interruption caused by using many Chinese characters in the table name of a row format log.

3.1.1	Feature updates Supported the official updates of MySQL 8.0.19, 8.0.20, 8.0.21, and 8.0.22. Supported dynamic setting of thread pooling mode or connection pooling mode by using the `thread_handling` parameter. Supported source-replica buffer pool sync: after a high-availability (HA) source-replica switch occurs, it usually takes a long time to warm up the replica, that is, to load hotspot data into its buffer pool. To accelerate the replica's warmup, TXSQL now supports the buffer pool sync between the source and the replica. Supported sort merge join. Supported async commit: With the thread pool enabled and binlog disabled, async commit can be enabled by setting the parameter `innodb_log_sync_method` to `async` . Supported fast DDL operations. Supported querying the value of the `character_set_client_handshake` parameter. Supported database audit. Performance optimizations Optimized the mechanism of scanning and flushing the dirty pages tracked in the flush list, so as to solve the performance fluctuation issue while creating indexes and thus improve the system stability. Optimized the `BINLOG LOCK_done` conflict to improve write performance. Optimized the `trx_sys mutex` conflict by using lock free hash and improve performance. Optimized redo log flushing. Optimized the buffer pool initialization time. Optimized the clearing of adaptive hash indexes (AHI) during the `drop table` operations on big tables. Bug fixes Fixed the deadlocks caused by the modification of the `offline_mode` and `cdb_working_mode` parameters. Fixed the concurrency issue while persistently storing `max_trx_id` of global object `trx_sys` . Fixed performance fluctuation when cleaning InnoDB temporary tables. Fixed the read-only performance decrease when the instance has many cores. Fixed the error (error code: 1032) caused by hash scans. Fixed concurrency security issues caused by hotspot update.
3.0.1	Feature updates Supported three methods of querying `cynos_version` : `select CYNOS_VERSION()` , `select @@cynos_version` , and `show variables like 'cynos_version'` . Added a space limit parameter. If the total space usage exceeds the limit, an error will be reported to prompt you to release the space or upgrade the specification. Added the `innodb_ncdb_log_priority` read-only parameter, which indicates the priority of the source instance's backend log thread. Added the `innodb_ncdb_apply_priority` read-only parameter, which indicates the priority of the read-only instance's log replay thread. Added the `innodb_ncdb_fast_shutdown` dynamic parameter, which controls whether to quickly shut down processes. After it is enabled, when a process exits, no destruction operations on the global structure will be performed, which reduces the shutdown time. It is disabled by default.

	Added the `innodb_max_temp_data_file_size` read-only parameter. Its default value is 128 MB. If its value is greater than 0, it indicates the maximum size of the temp tablespace in the local storage.

Minor Version	Description
2.1.12	Feature Updates Read-only instances now support binlog subscription via log positions. Performance Enhancements Optimized the performance of purging binlogs, reducing performance fluctuations. Bug Fixes Resolved compatibility issues in certain scenarios where Flink subscribes to the binlog of TDSQL-C read-only instances. Addressed the official Bug #27422376. Fixed an issue where traversing a B-Tree in reverse order would cause retries when encountering expired data pages. Corrected a problem where reducing the size of the buffer pool might lead to the repeated recycling of pages.
2.1.11	Feature Updates Database auditing now supports the addition of transaction ID fields and table names. Supports multi-threaded logical backup. Database proxy now supports Transaction Split Feature. Performance Optimization Optimized memory usage mechanisms, reducing the risk of Out Of Memory (OOM). Removed CDC, LSN, and CLUSTER as reserved keywords. Enhanced the logic for applying logs, improving the storage layer's ability to process redo logs. Bug Fixes Rectified an issue where database auditing incorrectly records the type of prepare statements. Fixed the problem where database auditing incorrectly classifies sql_type in execute statements as “other”. Fixed official Bug #111686 and official Bug #26225783.
2.1.10	Feature updates Supported binlog subscription for read-only instances. Supported blackhole engine. Supported buffer pool resizing. Bug fixes Fixed the issue of frequent disconnections when extracting binlogs via read-only instances.
2.0.23/2.1.9	Feature updates Supported automatic killing of idle transactions. Supported dynamic thread pool. Supported NOWAIT syntax.

	Supported returning. Supported auto-increment column persistence. Supported invisible index. Supported computation pushdown. Supported buffer pool initialization. Supported hotspot update protection. Bug fixes Fixed the issue of GTID loss caused by the HA switch in certain scenarios. Fixed the issue of possible crash triggered by executing `show create table` after killing the mysqld process during DDL. Fixed the issue of premature release of the FTS cache lock in full-text indexing, which caused inconsistency between reads and writes.
2.0.22/2.1.8	Feature updates Supported CDC, which can directly and repeatedly backtrack/extract the binlogs in the custom log retention period. This solves the problem where the compute node loses local binlogs in scenarios such as HA. For more information on how to set the binlog retention period, see Setting Backup Retention Period. Optimized the pages purge rate to improve the database performance. Bug fixes Fixed the repeated crashes of the compute instance when the upper limit of the space was reached.
2.0.21/2.1.7	Feature updates Added the password dictionary parameter as described in Configuring Custom Password Strength, and optimized HA's dependence on dictionary files. Supported completing purged binlogs in the kernel. Supported the built-in database proxy for the serverless architecture to implement connection persistence and momentary disconnection prevention, and fixed the connection error reported during the first wakeup.
2.0.20/2.1.6	Bug fixes Fixed the issue where the liveness probe of the database proxy caused the kernel to output a large number of error logs, and blocked the printing of redundant logs. Fixed the issue where the uninitialized GCR LSN in the session LSN tracker might return a random value of the database proxy and thus cause the statement execution to time out. Fixed the issue where creating a temp table in a read-only instance might cause a deadlock.
2.0.19	Feature updates Supported logical backup for read-only instances. Supported database proxy. For more information on how to use this feature, see Database Proxy Overview. Supported parallel replication of binlogs at the table level. Supported setting the change buffer and merge modes. Supported `show detail processlist` . Bug fixes

	Fixed the official Bug#22991924 related to the JSON character set. Merged the official bugfix for Bug#25865525 to solve the issue where `LOAD DATA INFILE` failed to read escape characters plus UTF8 characters. Merged the official bugfix for Bug#31529221 to fix the issue where the error `Incorrect key file` was reported upon ALTER TABLE failure. Merged several official bugfixes related to column generation and cascading deletion, including Bug#33053297, Bug#32124113, and Bug#29127203. Fixed the official Bug#31599938 where resetting the source caused a crash when binary logging was disabled in the replica. Fixed the issue where the startup time was prolonged due to the failure to add `dict op lock` to the main thread caused by large transactions or long-line transaction rollbacks during instance startup. Fixed the crash caused by TRX reuse after the failure to allocate the undo page. Fixed the crash caused by executing `alter table` on a partitioned table to migrate from the extended tablespace to the system tablespace. Fixed the crash caused by the startup before the truncate log was completely written. Fixed the crash caused by inserting data after `drop table partition force` . Fixed the possible crash caused by rollbacks after instant DDL. Fixed the issue where creating tables in the extended tablespace with `create temporary table like` failed. Fixed the OOM caused by the continuous increase of the cache during data writes to the full-text index table.
2.0.17	Feature updates Supported adding the binlog with the specified filename to an index file. Bug fixes Merged the official bugfix for Bug#25865525 to solve the issue where `LOAD DATA INFILE` failed to read escape characters plus UTF8 characters.
2.0.16	Performance optimizations Optimized `undo space truncate` to improve the speed of `undo truncate` on large-spec instances. Optimized the performance of large-scale queries on read-only instances. Bug fixes Fixed the issue where `backup lock` couldn't be locked due to the `lock table` statement. Fixed the issue where `table share` went wrong for the read-only instances after thousands of columns were added through `INSTANT ADD` . Fixed the replay error when the content of binlog contained escaped keywords. Fixed the issue where externally prepared XA transactions were not explicitly rolled back and thus blocked normal shutdown. Fixed the issue where the warning "tablespace -1 not found" was reported during read-only instance startup.
2.0.15	Feature updates

	Supported the extended table space: When a single table space exceeds the `innodb_ncdb_extend_space_threshold` configuration, a new table will be created in the extended table space. Added new JSON functions: JSON_MERGE_PRESERVE, JSON_MERGE_PATCH, JSON_PRETTY, JSON_STORAGE_SIZE, JSON_ARRAYAGG, JSON_OBJECTAGG. Optimized the table lock recovery process at system startup to shorten the startup time. Bug fixes Fixed the bugs for the `group by` performance issue in text columns and multiple issues related to virtual columns. Fixed the possible crash when large transaction rollback and shutdown operations were performed concurrently after instance startup. Fixed the issue where repeatedly failed IO retries of related pages caused instance exit after `undo space truncate` failed. Fixed the crash when statistics update accessed the snapshot cache for disabled read-only instances. Fixed the issue where the truncation log might be behind the `truncate` operation in `undo space truncate` . Fixed the bug where an error in ICP check for partitioned table scan resulted in slow query. Fixed the crash when the previous scan in a read-only instance encountered the partial replay of a split index log.
2.0.14	Feature updates Supported INSTANT MODIFY COLUMN. For more information, see Instant DDL Overview. Bug fixes Fixed the issue where the used space was not reclaimed when a temp table in a read-only instance was dropped. Fixed the issue where the process exited when a read-only instance read the old page version due to changes in storage routes. Fixed the issue of possible crash when `information_schema.metadata_locks` was queried. Fixed the concurrency error of forward scan and B-tree SMO in read-only instance. Fixed the issue where when multiple tables had complex foreign key dependencies and the foreign key attribute was `ON DELETE CASCADE` , the corresponding record in a child table might be deleted twice when a record was deleted in its parent table with DELETE. Fixed the issue where the process exited due to operations such as CREATE USER in a read-only instance. Fixed the issue where SHOW VOLUME STATUS in a read-only instance might trigger an assertion failure. Fixed the issue where a read-only instance had abnormal query results and crashed when DDL operations were performed in partitioned tables frequently. Fixed the issue where the process crashed during historical version construction when an read-only instance scanned a purged undo log.
2.0.13	Feature updates Supported INSTANT ADD COLUMN. For more information, see Instant DDL Overview.

	Optimized the audit performance under high load and added the `lock_usleep_time` dynamic parameter. Bug fixes Fixed the issue where `dict_operation_lock` might cause deadlock during foreign key check. Fixed the issue where the process might exit when the ACL change log was replayed during read-only instance startup. Fixed the issue where data in source and replica instances was inconsistent after INSTANT ADD COLUMN and TRUNCATE TABLE. Fixed the log replay error occurring when data was inserted again after INSTANT ADD COLUMN and TRUNCATE TABLE. Fixed the issue where the process exited when a primary key containing a column added by INSTANT ADD COLUMN was created. Fixed the issue where the process exited during table structure query in a read-only instance when INSTANT COLUMN was used to create or rebuild a partition.
2.0.12	Feature updates Supported database audit. For more information on how to use it, see Enabling TDSQL-C for MySQL Audit. Supported purging page read-ahead to accelerate space reclaim. Supported real-time update of the size information of tables and indexes in the system view. Added a thread to accelerate recycle LSN and storage GC. Bug fixes Fixed official bugs in full-text index, including Bug#24938374, Bug#21625016, Bug#27082268, Bug#27155294, Bug#27326796, Bug#27304661, Bug#25289359, Bug#29717909, and Bug#30787535. Fixed official bugs where concurrent update might cause system crashes, including Bug#30950714, Bug#31205266, and Bug#25669686. Fixed the official Bug#28104394 where uncommitted INSERT operations would affect the range scan created by an index and made it return an incorrect number of rows. Fixed the official Bug#30488700 where an incorrect query execution plan of a derived table could result in a poor performance. Fixed the issue where the process exited when a read-only instance replayed statistics logs after the source (read-write) instance executed a DDL statement. Fixed the issue where RENAME TABLE was performed on a database that did not exist. Fixed the issue where a read-only instance might exit when OPTIMIZE TABLE was used for the source instance. Fixed the issue where transactions were inconsistent after a table with a full-text index was updated in a read-only instance. Fixed the deadlock occurring during SET OFFLINE MODE and SHOW VARIABLES operations.
2.0.11	Feature updates Optimized the binlog file index to accelerate binlog file scan. Optimized the shutdown process to make it faster. Optimized the instance memory to reduce the memory usage by structures such as buffer, ZIP, and hash.

	Optimized the DDL lock recovery process during read-only instance startup to accelerate replay. Optimized system table loading in read-only instance to accelerate startup. Optimized worker thread assignment during PURGE COORDINATOR to accelerate purge. Bug fixes Fixed the issue where the process crashed during OPEN TABLE due to incorrect index structure mapping. Fixed the issue where read-only instance replication was abnormal during XA transaction rollback. Fixed the issue where startup crashed when binlog replication was started in a read-only instance. Fixed the memory leak caused by FLUSH LOGS. Fixed the shutdown failure of `mysqladmin shutdown` . Fixed the issue where a read-only instance failed to replay logs when DROP COLUMN was performed on the source instance. Fixed the issue where data could still be input after space restriction was triggered when a BLOB was inserted. Fixed the issue of read-only instance replication availability that might be caused by DDL statements in big tables or slow log storage in the source instance. Fixed the issue where storage wasted small tables after `innodb_max_temp_data_file_size` was set for a temp table.

Database Proxy Version Release Notes

Last updated：2024-07-31 14:58:46

This document describes the updates in each release of the TDSQL-C for MySQL database proxy kernel version.

Note:

If the kernel version requirements of TDSQL-C for MySQL are not met, you can upgrade the database kernel version first as instructed in Upgrading Kernel Minor Version. To learn about the release dates of various database proxy kernel versions, see Kernel Overview.

Version	TDSQL-C for MySQL Kernel Version Requirement	Description
1.3.10	TDSQL-C for MySQL version 5.7 is equal to or greater than 2.0.20/2.1.6. TDSQL-C for MySQL version 8.0 is equal to or greater than 3.1.6.	Bug Fixes Resolved an issue where handling response packets could result in exceptions when encountering specific messages.
1.3.8	TDSQL-C for MySQL version 5.7 is compatible with versions 2.0.20/2.1.6 and later. TDSQL-C for MySQL version 8.0 is compatible with version 3.1.6 and later.	Feature Updates Enhanced support for updated functions in MySQL versions 5.7 and 8.0. Optimized parser caching to reduce the likelihood of Out of Memory (OOM) errors under complex SQL queries. Introduced traffic monitoring metrics for database proxies. Implemented dynamic load balancing capabilities. Introduced anti-flapping features.
1.3.7	TDSQL-C for MySQL 5.7 ≥ 2.0.20/2.1.6 TDSQL-C for MySQL 8.0 ≥ 3.1.6	Bug fixes Fixed the routing error of the select for update statement in some cases. Modified the select @@read_only statement and made it possible to be routed to the source database. This prevents some frameworks that use read_only flags from misjudging the database proxy as unwritable. Fixed the database proxy node exceptions caused by a database instance HA in some scenarios.
1.3.5	TDSQL-C for MySQL 5.7 ≥ 2.0.20/2.1.6 TDSQL-C for MySQL 8.0 ≥ 3.1.6	Bug fixes Resolved the issue of decreased and fluctuating read performance in read-only instances under high concurrency scenarios.
1.3.4	TDSQL-C for MySQL 5.7 ≥ 2.0.20/2.1.6TDSQL-C for MySQL	Bug fixes

	8.0 ≥ 3.1.6	Resolved the issue of incomplete data returned by the show processlist command.
1.3.3	TDSQL-C for MySQL 5.7 ≥ 2.0.20/2.1.6 TDSQL-C for MySQL 8.0 ≥ 3.1.6	Bug fixes Fixed the issue where an error was reported when the session connection pool reused connections to send change_user to the backend, and the issue where the PREPARE statement was not correctly handled by the database proxy after a new connection was established. Fixed the issue where the execute statement lacked parameter types.
1.2.1	-	Feature updates Supported MySQL 5.7/8.0. Supported cluster deployment, enabling the deployment of multiple instances in a single database proxy. Supported read/write separation along with corresponding weight configuration. Supported the failover feature, rerouting read requests to the read-write instance in case of read-only instance exceptions. Supported the load balancing feature to address uneven connection counts among proxy nodes. Supported using the hint syntax to specify routing nodes. Supported session-level connection pooling, handling scenarios where frequent connections need to be established with the database in short lived connection-based businesses. Database proxy supported saving connections and reusing them for subsequent connection attempts. Supported hot loading, allowing online modifications of configurations without the need of restarting the dedicated database proxy. Supported the reconnection feature for read-only instances. In scenarios involving long lived connections, the database proxy will automatically re-establish a connection and restore the routing nodes if a read-only instance is restarted or a read-only instance is added.

Optimized Kernel Version Compilation Optimization High-Performance Version

Last updated：2024-04-25 09:42:49

The TXSQL kernel of TencentDB for MySQL supports a compilation optimization high-performance version, which can maintain the original compatibility without changing the internal implementation logic of the kernel. By leveraging dynamic compiler optimization techniques to identify possible user input behavior, the database kernel demonstrates stronger performance in common business scenarios, while also reducing power consumption. This article introduces the compilation optimization high-performance version of TencentDB for MySQL.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.1.11) or later.
TDSQL-C for MySQL 8.0 (kernel version 3.1.12) or later.
Note:
The compilation optimization high-performance version is currently in grayscale release. If you want to experience it in advance, please submit a ticket to apply for use under the precondition of meeting the above database kernel version requirements.

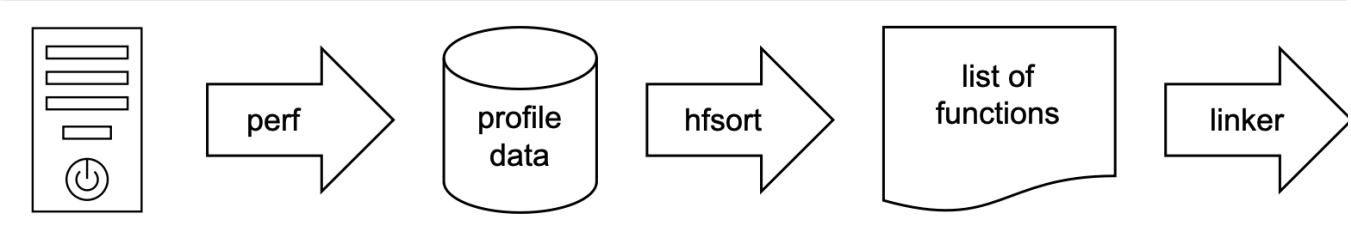
Overview

As the internal implementation of modern CPUs becomes increasingly complex, the default compilation, configuration, and execution methods of cloud databases can hardly fully exploit the performance potential of CPUs, leading to a significant number of CPU cycles idling. This phenomenon, in the case of the compounded scale effect, not only results in the wastage of hardware resources but also consumes a lot of power. Therefore, it is necessary to optimize cloud databases to maximize the performance potential of CPUs, reduce the idling of CPU cycles, improve the utilization rate of hardware resources, and decrease power consumption waste.
TDSQL-C for MySQL, without changing the database kernel's business logic code precondition, employs dynamic compiler optimization techniques to achieve kernel performance improvement and power consumption reduction with minimal cost. By collecting behavior and performance consumption data of cloud databases in typical/real business scenarios, it improves the default compilation method being unaware of business behavior, analyzes the database runtime behavior characteristics along with the CPU microarchitecture features, and utilizes compilation optimization technologies to make the optimized version more friendly to the CPU microarchitecture, fully unleashing the CPU's

performance potential; and ensures the optimization effect does not degrade under various conditions through extensive scenario testing.

The following optimizations were achieved through the above technical measures:

1. Based on database operation behavior data, feedback optimization improves the optimization capabilities of function inlining/function reordering/basic library reordering, thereby significantly reducing database CPU ICache/ITLB miss rates and enhancing performance.
2. By utilizing link-time optimization techniques, the compilation optimization perspective is expanded from a single file/single function to across files/entire binary files, significantly enhancing the optimization space for inlining and reducing the directive count.
3. In practice, a set of efficient verification and analysis methods has been developed to ensure the effects are close to theoretical thresholds and guarantee no degradation under various scenarios.



Definition of Compilation Optimization

Compilation optimization refers to the process of enhancing a program's execution efficiency and performance by optimizing the code and adjusting compilation parameters during code compilation.

Optimization Principles

The high-performance version of TDSQL-C for MySQL uses Profile-Guided Optimization (PGO) technology for compilation optimization. PGO technology addresses the issue that traditional compilers, during optimization, rely solely on static code information without considering potential user inputs, thus failing to effectively optimize the code. PGO technology is divided into the following three stages:

1. Instrument: During the instrument stage, an initial compilation is performed on the application. In this compilation, the compiler inserts directives into the code so that data can be collected in the next stage. These directives are of three types, used to track how many times each function is executed, how many times each branch is executed (for example, in if-else scenarios), and certain variable values (primarily for switch-case scenarios).
2. Train: In the train stage, users need to run the application compiled in the previous stage using the most common inputs. Since the previous stage has prepared for data collection, the data corresponding to the most common usage scenarios of that application will be collected after the train stage.

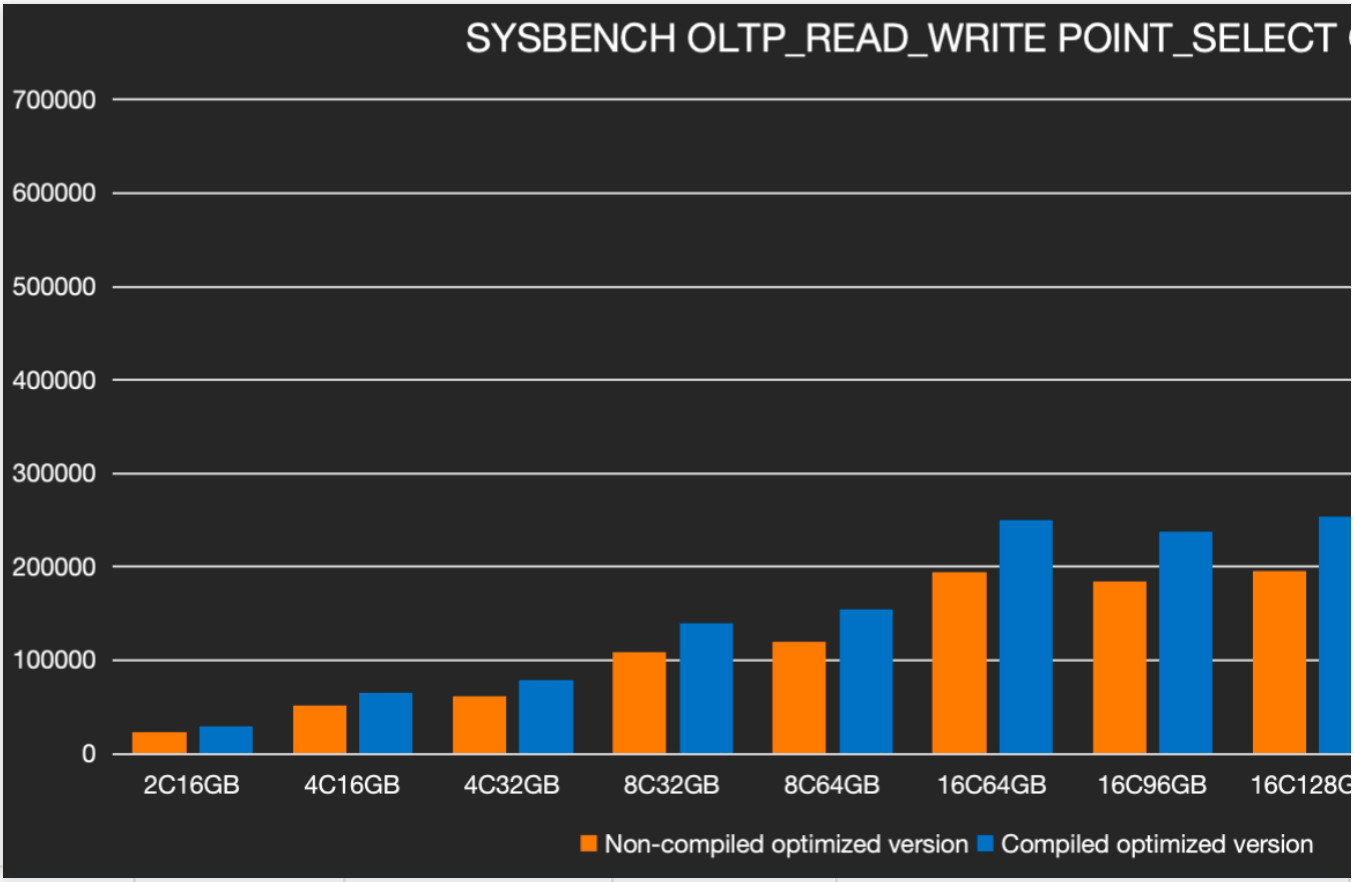
3. Optimization: In the optimization stage, the compiler recompiles the application using the data collected in the previous stage. Since the data from the previous stage comes from the most common user input scenarios, the final optimized result will perform better in these scenarios.
Through the optimization of these three stages, the high-performance version of TDSQL-C for MySQL can better meet the needs of users, improving the performance and efficiency of the application.

Performance Testing

Test Scenario

Mixed read-write (POINT SELECT) test scenario mainly tests the performance of the database while conducting read and write operations concurrently. It can help evaluate the database's performance in real application scenarios, including the handling capability of concurrent read-write operations, response time, throughput, and other metrics.

Test Result



Specification	Concurrency	Single table data volume (table_size)	Total number of tables (tables)	QPS	
				Using the compilation optimization version	Boost percentage

2-core 16 GB MEM	64	800,000	150	29207	27%
4-core 16GB	256	800,000	300	65562	27%
4-core 32 GB MEM	256	800,000	300	78973	27%
8-core 32 GB	256	800,000	300	139,845	28%
8-core 64 GB MEM	256	800,000	450	154,894	28%
16-core 64 GB	256	800,000	450	249,954	29%
16-core 96 GB	256	800,000	600	238,061	29%
16-core 128 GB	512	5,000,000	300	253,848	29%
32-core 128 GB MEM	512	5,000,000	300	399647	30%
32-core 256 GB MEM	512	5,000,000	400	402,105	30%
64-core 256 GB MEM	512	6,000,000	450	596,706	31%

Functionality Features
Automatic Killing of Idle Transactions

Last updated：2023-11-01 16:50:53

Overview

This feature kills transactions that have been idle for the specified time period to release resources in time.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.
TDSQL-C for MySQL on kernel version 3.1.10 or later.

Use Cases

If a connection starts a transaction (explicitly using `begin` / `start transaction` or implicitly) but no new statement has been executed for the specified threshold period, the connection will be killed.

Use Limit

Use the `cdb_kill_idle_trans_timeout` parameter to enable or disable the feature. If it is `0`, the feature is disabled; otherwise, a connection idle for `cdb_kill_idle_trans_timeout` or `wait_timeout` seconds, whichever is smaller, will be killed. `wait_timeout` is a session parameter.

Parameter	Effective Immediately	Type	Default Value	Value Range	Description
cdb_kill_idle_trans_timeout	YES	ulong	0	[0,31536000]	If it is `0`, the feature is dis otherwise, a transaction idl `cdb_kill_idle_trans` seconds will be killed.

Instant DDL Overview

Last updated：2024-04-25 10:54:27

Use Cases

This feature can use DDL operations to alter ultra big tables in online businesses within seconds.

Overview

The instant DDL feature can quickly modify columns in big tables while avoiding data replication. This feature does not replicate the data or consume disk capacity or I/O, and can implement changes within seconds during peak hours.

Supported Versions

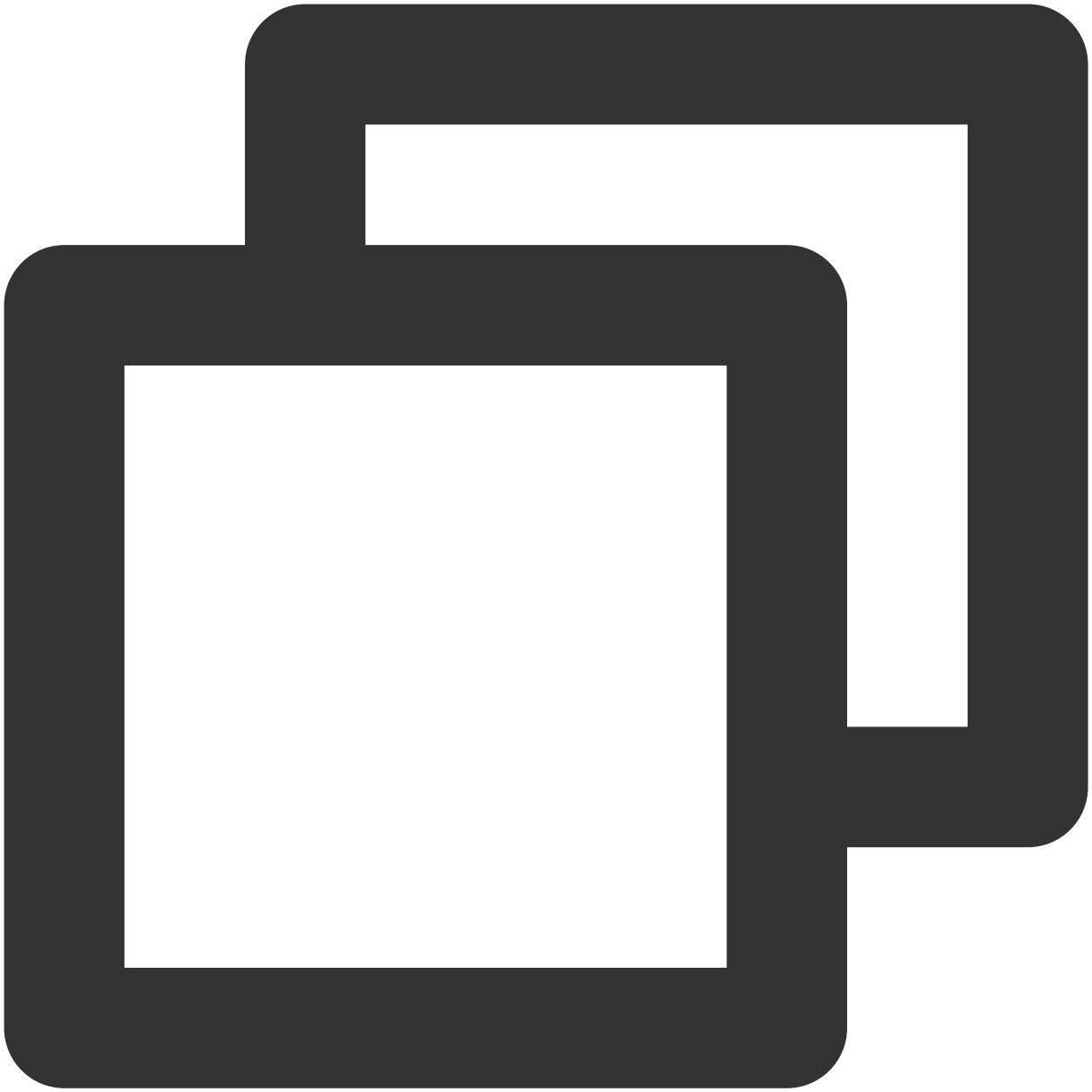
MySQL 5.7 with kernel minor version 2.1.3 or later.
MySQL 8.0 with kernel minor version 3.1.1 or later.

Instructions

Instant DDL supports the ADD COLUMN operation.

Description of INSTANT ADD COLUMN

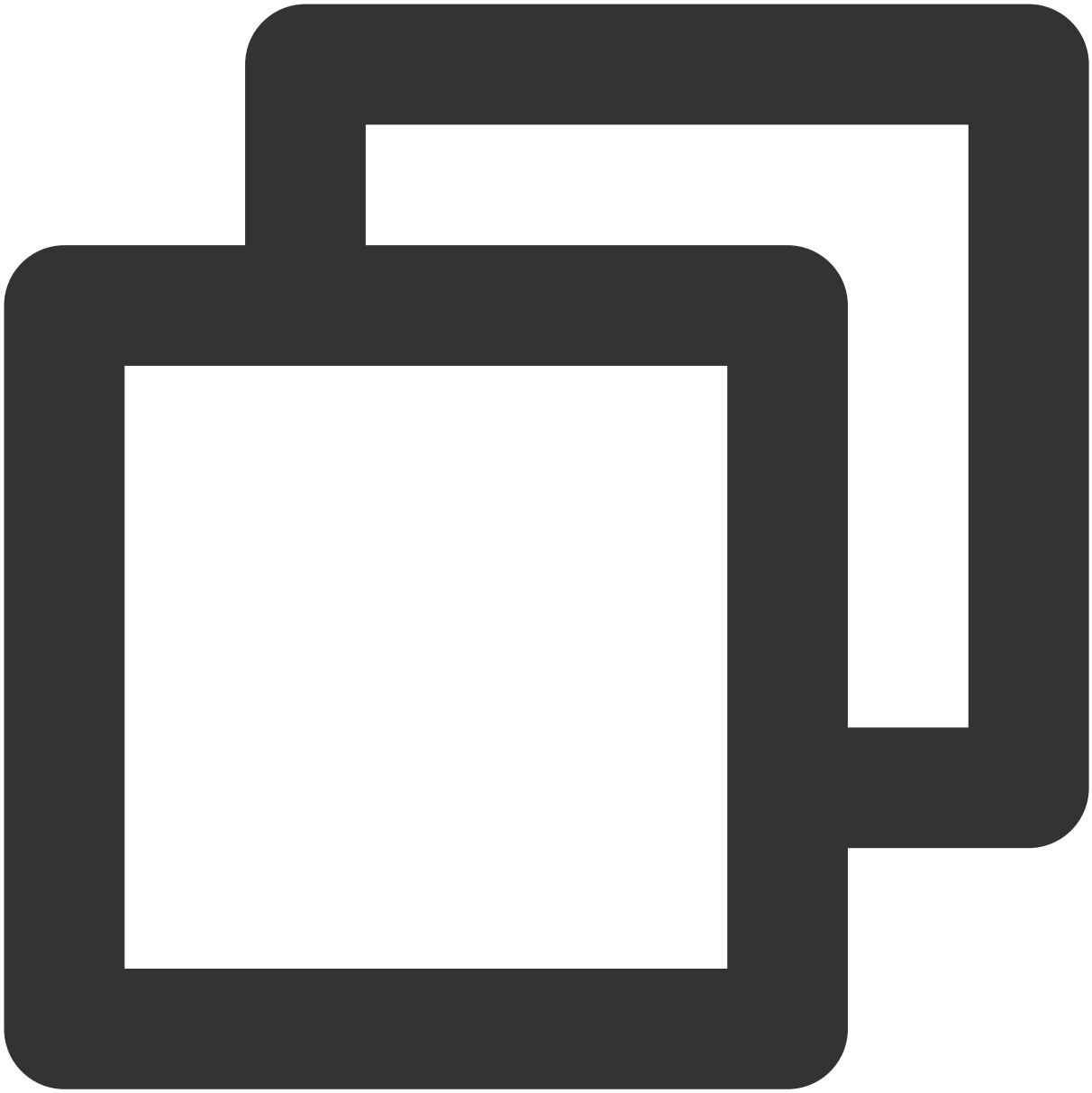
MySQL 8.0 kernel minor version from 3.1.1 to 3.1.10 operation instructions
MySQL 8.0 kernel minor version 3.1.12 and later operation instructions
Syntax of INSTANT ADD COLUMN.
To add a column, you can use the following statements by including the new 'algorithm = instant' clause:



```
ALTER TABLE t1 ADD COLUMN c INT, ADD COLUMN d INT DEFAULT 1000, ALGORITHM=INSTANT;
```

Added parameter innodb_alter_table_default_algorithm, which can be set to inplace or instant, with the default value being instant.

Column addition can be done with the following statement:



```
ALTER TABLE t1 ADD COLUMN c INT, ADD COLUMN d INT DEFAULT 1000;
```

Restrictions on INSTANT ADD COLUMN

A statement can contain only column addition operations.

A new column will be added to the end, and column order cannot be changed.

INSTANT ADD COLUMN is not supported in tables with the row format being COMPRESSED.

INSTANT ADD COLUMN is not supported in tables with a full-text index.

INSTANT ADD COLUMN is not supported for temp tables.

Dynamic Thread Pool

Last updated：2024-04-25 10:59:00

Overview

A thread pool (Thread_pool) is a collection of worker threads that are used to handle connection requests. It is often ideal for OLTP workloads. However, those worker threads may get stuck on high-latency operations when dealing with many slow queries, resulting in delayed responses to new requests. This limitation makes the thread pool less efficient than the traditional one-thread-per-connection mode (Per_thread) in terms of system throughput.
The Thread_pool and Per_thread modes each have their own advantages and disadvantages, and they can be flexibly switched by the system based on business types. Unfortunately, the switch between these two modes requires a server restart. This usually happens during peak hours, and forcing a server restart during this time can have a severe effect on the business.
To facilitate the switch between Per_thread and Thread_pool, TDSQL-C for MySQL has make an optimization called dynamic thread pool switch. This means that the thread pool can be dynamically enabled or disabled without restarting the database service.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.
TDSQL-C for MySQL 8.0 (kernel version 3.1.10) or later.

Use Cases

This feature is suitable for the business which is sensitive to performance and needs to flexibly change the database working mode based on the business type.

Impact on Performance

For switching from the thread pool mode to the one-thread-per-connection mode, there will be no backlog of queries or performance degradation when the QPS has dropped significantly.
For switching from the one-thread-per-connection mode to the thread pool mode, there may be a backlog of queries when the QPS stays extremely high. This is because the thread pool is disabled and remains dormant before the switch and many queries continue to arrive during the switch. The solutions are as follows:

Option 1. You can increase the value of the `thread_pool_oversubscribe` parameter and decrease the value of the `thread_pool_stall_limit` parameter to quickly enable the thread pool. After the backlog of SQL queries are processed, you can restore the parameters to their original values as needed.
Option 2. If the backlog of SQL queries occurs, you can suspend or reduce service traffic for a few seconds, wait for the thread pool to become active, and then resume the service traffic to handle high load.

Use Limits

You can use the `thread_handling_switch_mode` parameter to control whether to dynamically change the thread working mode. Parameter values are described as follows:

Valid Value	Description
disabled	The mode cannot be changed dynamically.
stable	The mode can only be changed for new connections.
fast	(Default value) The mode can be changed for new connections and new requests.
sharp	Active connections will be killed in order to force a reconnection so that the mode can be changed quickly.

The `show threadpool status` command displays the following new status:
connections_moved_from_per_thread: The number of connections switched from Per_thread to Thread_pool.
connections_moved_to_per_thread: The number of connections switched from Thread_pool to Per_thread.
events_consumed: The total number of events consumed by the worker thread group in each thread pool. After the thread working mode is switched from Thread_pool to Per_thread, the total number of events won't increase any more.
average_wait_usecs_in_queue: The average time each event waits in the queue.
The `show full processlist` command displays the following new status:
Moved_to_per_thread: The number of times that the connection is switched to Per_thread.
Moved_to_thread_pool: The number of times that the connection is switched to Thread_pool.

Parameter Description

Thread pool parameters are described as follows:

Parameter	Effective Immediately	Type	Default Value	Valid Values/Value Range

thread_pool_idle_timeout	Yes	uint	60	[1, UINT_MAX]
thread_pool_oversubscribe	Yes	uint	3	[1,1000]
thread_pool_size	Yes	uint	The number of CPUs on the current machine.	[1,1000]
thread_pool_stall_limit	Yes	uint	500	[10, UINT_MAX]
thread_pool_max_threads	Yes	uint	100000	[1,100000]
thread_pool_high_prio_mode	Yes, session	enum	transactions	transactions\\statement\\none
thread_pool_high_prio_tickets	Yes,	uint	UINT_MAX	[0, UINT_MAX]

	session			
threadpool_workaround_epoll_bug	Yes	bool	false	true/false

The `show threadpool status` command displays the following status:

Status	Description
groupid	Thread group ID
connection_count	The number of user connections in the thread group
thread_count	The number of worker threads in the thread group
havelistener	Whether the thread group has a listener
active_thread_count	The number of active worker threads in the thread group
waiting_thread_count	The number of worker threads calling `wait_begin` in the thread group
waiting_threads_size	The number of sleeping worker threads waiting to be woken up in the thread group when there is no network event to handle (such worker threads will wait for `thread_pool_idle_timeout` seconds before being automatically killed).
queue_size	The length of the ordinary queue of the thread group
high_prio_queue_size	The length of the high priority queue of the thread group
get_high_prio_queue_num	The total number of times that events in the thread group are removed from the high priority queue
get_normal_queue_num	The total number of times that events in the thread group are removed from the ordinary queue
create_thread_num	The total number of worker threads created in the thread group
wake_thread_num	The total number of worker threads in the thread group awakened from the waiting_threads queue
oversubscribed_num	The number of times that worker threads are ready to go to sleep because the thread group is oversubscribed
mysql_cond_timedwait_num	The total number of times that worker threads in the thread group enter the waiting_threads queue

check_stall_nolistener	The total number of times that no listener is detected in the thread group in the stall check performed by the timer thread
check_stall_stall	The total number of times that the thread group is considered stalled in the stall check performed by the timer thread
max_req_latency_us	The maximum time in milliseconds for a user connection to wait in the queue in the thread group
conns_timeout_killed	The total number of times that user connections in the thread group are killed because there has been no new message on the client for the threshold period
connections_moved_in	The total number of connections migrated from other thread groups to this thread group
connections_moved_out	The total number of connections migrated from this thread group to other thread groups
connections_moved_from_per_thread	The total number of connections switched from the one-thread-per-connection mode to this thread group
connections_moved_to_per_thread	The total number of connections switched from this thread group to the one-thread-per-connection mode
events_consumed	The total number of events processed by the thread group
average_wait_usecs_in_queue	The average waiting time of all events in the queue in the thread group

NOWAIT

Last updated：2024-04-25 11:00:51

Overview

DDL statements support `NO_WAIT` and `WAIT` options. If a DDL statement with `WAIT` enabled fails to obtain an MDL lock, it will wait for `WAIT` seconds before it directly returns the query result. If a DDL statement with `NO_WAIT` enabled, it will directly return the query result without waiting for the MDL lock.
SELECT FOR UPDATE statement supports `NOWAIT` and `SKIP LOCKED` options. If target rows are locked by another transaction, a SELECT FOR UPDATE statement is supposed to wait for the transaction to release the lock. But in some use cases like flash sales, you do not want to wait for a lock. You can use `SKIP LOCKED` to skip locked rows (as a result, the locked rows won't be returned in the query result set) or `NOWAIT` to return an error without waiting for the lock.
Note that `NO_WAIT` and `NOWAIT` are different keywords.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.
TDSQL-C for MySQL 8.0 (kernel version 3.1.10) or later.

Use Cases

Currently, DevAPI/XPlugin does not support using `SKIP LOCKED` or `NOWAIT` in SELECT FOR UPDATE/SHARE statements. Note that `NO_WAIT` in DDL statements and `NOWAIT` in SELECT FOR UPDATE statements are different keywords for historical reasons.

RETURNING

Last updated：2024-04-25 11:03:22

Overview

In some scenarios, you need to retrieve the rows manipulated by DML statements. There are generally two ways to do so:
Add a SELECT statement after the DML statement if the transaction is enabled.
Use a trigger or other complex operations.
However, running a SELECT statement increases query costs, and creating a trigger makes SQL implementation more complex and inflexible.
Therefore, TXSQL supports the RETURNING keyword to optimize such scenarios. The above requirements can be flexibly and efficiently met by appending RETURNING to a DML statement.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.
TDSQL-C for MySQL 8.0 (kernel version 3.1.10) or later.

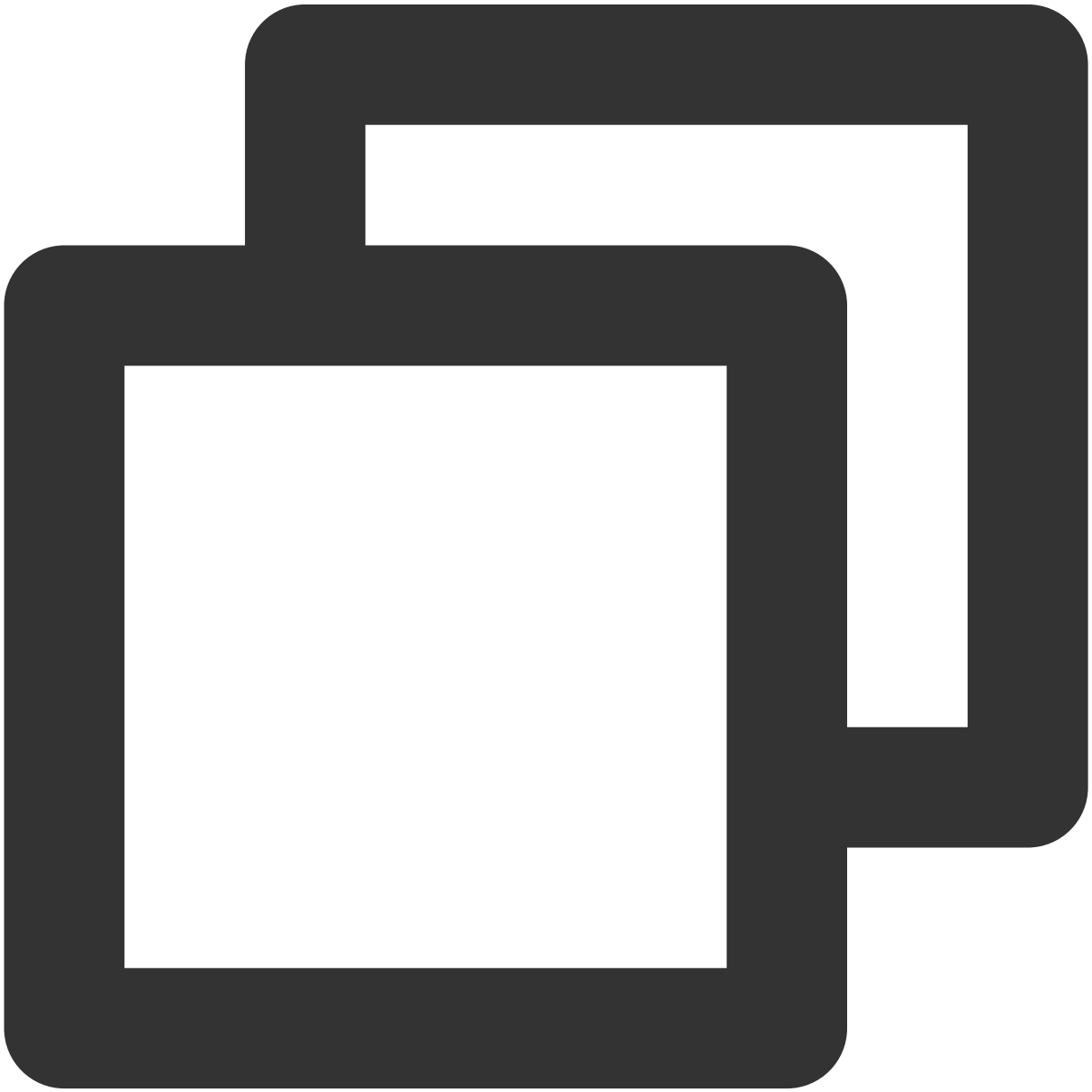
Use Cases

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later support `INSERT ... RETURNING`, `REPLACE ... RETURNING`, and `DELETE ... RETURNING`. The RETURNING keyword returns all rows that have been manipulated by an INSERT/REPLACE/DELETE statement. RETURNING can also be used in prepared statements and stored procedures.
TDSQL-C for MySQL 3.1.10 or later supports `DELETE ... RETURNING`, `INSERT ... RETURNING`, `REPLACE ... RETURNING`, and `UPDATE ... RETURNING`. The RETURNING keyword returns all rows that have been manipulated by this statement.
Notes:
1. For `DELETE ... RETURNING`, the returned data rows are pre-images, while for `INSERT/REPLACE ... RETURNING`, they are post-images.
2. For `INSERT/REPLACE ... RETURNING`, columns in the outer table are currently invisible to the subquery in the RETURNING clause.

3. `INSERT/REPLACE ... RETURNING` only returns the value of `last_insert_id()` before the statement is executed successfully. To obtain the true value of `last_insert_id()`, you should use RETURNING to return the auto-increment column ID of the table.

Use Limits

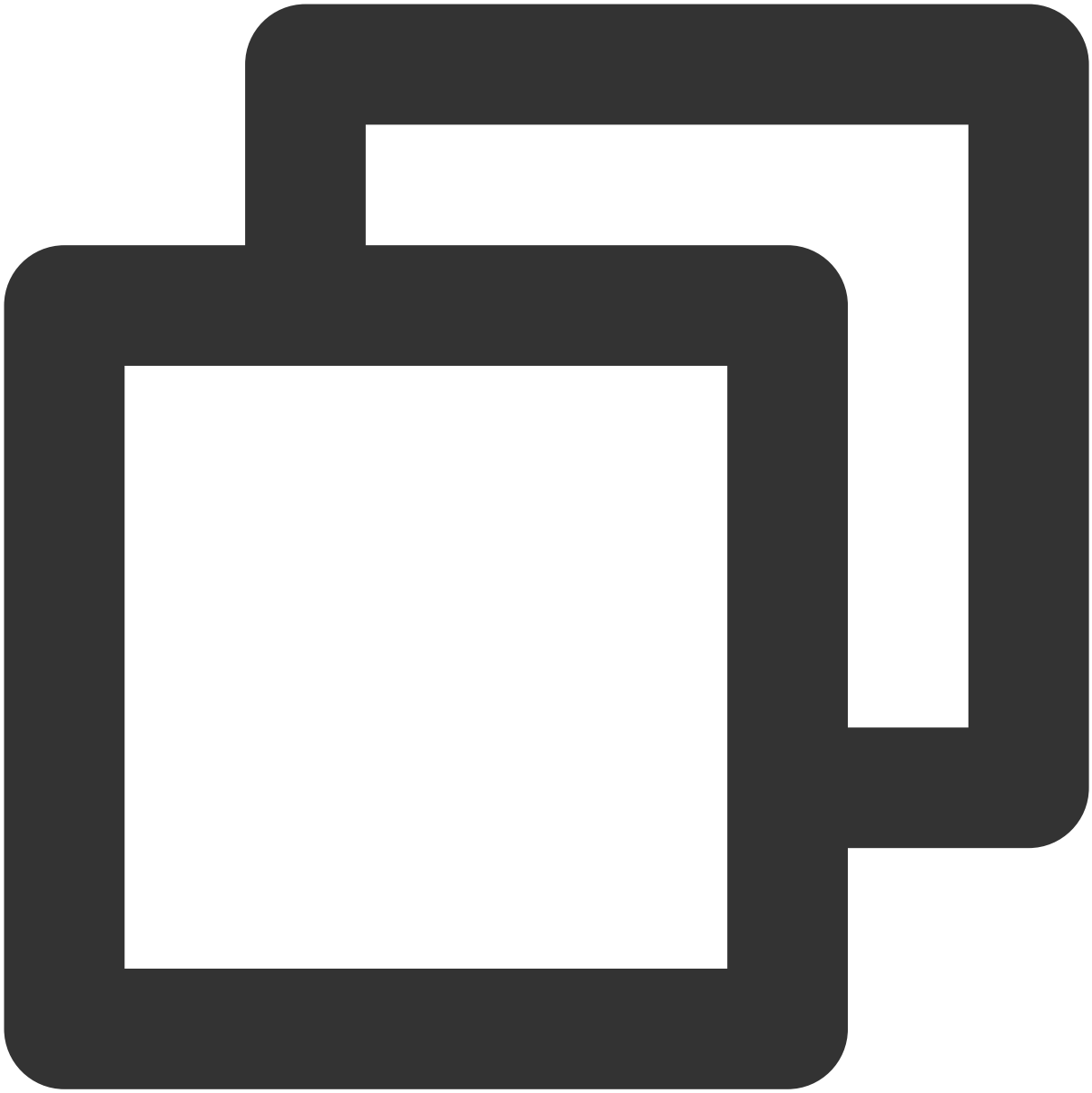
INSERT ... RETURNING



```
MySQL [test]> CREATE TABLE `t1` (id1 INT);
```

```
Query OK, 0 rows affected (0.04 sec)

MySQL [test]> CREATE TABLE `t2` (id2 INT);
Query OK, 0 rows affected (0.03 sec)

MySQL [test]> INSERT INTO  t2 (id2) values (1);
Query OK, 1 row affected (0.00 sec)

MySQL [test]> INSERT INTO t1 (id1) values (1) returning *, id1 * 2, id1 + 1, id1 *
+------+---------+---------+-------+--------------------+
| id1  | id1 * 2 | id1 + 1 | alias | (select * from t2) |
+------+---------+---------+-------+--------------------+
|    1 |       2 |       2 |     1 |                  1 |
+------+---------+---------+-------+--------------------+
1 row in set (0.01 sec)

MySQL [test]> INSERT INTO t1 (id1) SELECT id2 from t2 returning id1;
+------+
| id1  |
+------+
|    1 |
+------+
1 row in set (0.01 sec)
```

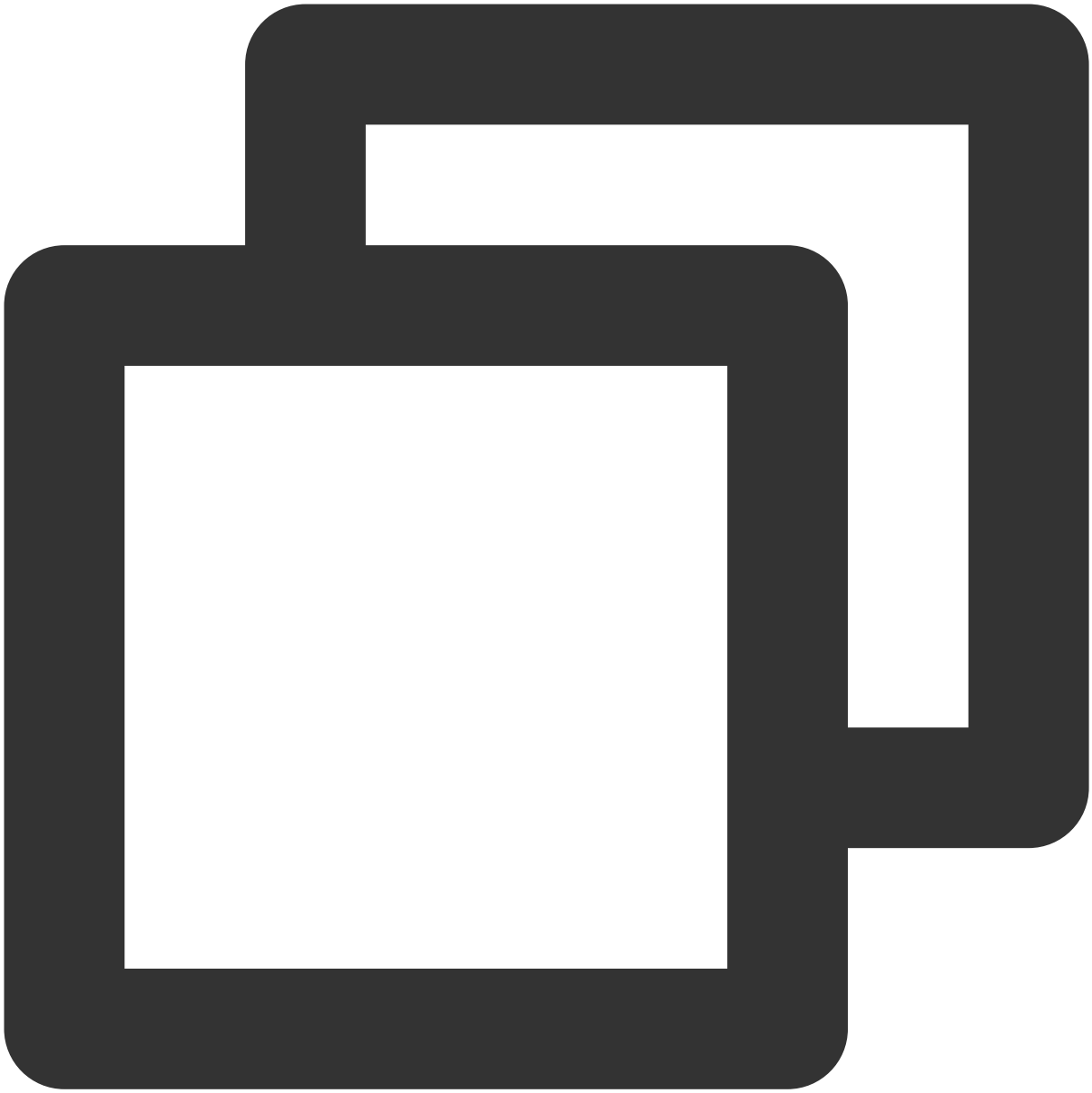
REPLACE ... RETURNING



```
MySQL [test]> CREATE TABLE t1(id1 INT PRIMARY KEY, val1 VARCHAR(1));
Query OK, 0 rows affected (0.04 sec)

MySQL [test]> CREATE TABLE t2(id2 INT PRIMARY KEY, val2 VARCHAR(1));
Query OK, 0 rows affected (0.03 sec)

MySQL [test]> INSERT INTO t2 VALUES (1,'a'),(2,'b'),(3,'c');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

MySQL [test]> REPLACE INTO t1 (id1, val1) VALUES (1, 'a');
```

```
Query OK, 1 row affected (0.00 sec)

MySQL [test]> REPLACE INTO t1 (id1, val1) VALUES (1, 'b') RETURNING *;
+-----+------+
| id1 | val1 |
+-----+------+
|   1 | b    |
+-----+------+
1 row in set (0.01 sec)
```

DELETE ... RETURNING



```
MySQL [test]> CREATE TABLE t1 (a int, b varchar(32));
Query OK, 0 rows affected (0.04 sec)

MySQL [test]> INSERT INTO t1 VALUES
(7,'ggggggg'),
(1,'a'),
(3,'ccc'),
(4,'dddd'),
(1,'A'),
(2,'BB'),
(4,'DDDD'),
```

```
(5,'EEEEE'),
(7,'GGGGGGG'),
(2,'bb');
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0

MySQL [test]> DELETE FROM t1 WHERE a=2 RETURNING *;
+------+------+
| a    | b    |
+------+------+
|    2 | BB   |
|    2 | bb   |
+------+------+
2 rows in set (0.01 sec)

MySQL [test]> DELETE FROM t1 RETURNING *;
+------+---------+
| a    | b       |
+------+---------+
|    7 | ggggggg |
|    1 | a       |
|    3 | ccc     |
|    4 | dddd    |
|    1 | A       |
|    4 | DDDD    |
|    5 | EEEEE   |
|    7 | GGGGGGG |
+------+---------+
8 rows in set (0.01 sec)
```

Flashback Query

Last updated：2024-04-25 11:05:59

Overview

Maloperations may occur in the process of database Ops and severely affect the business. Rollback and cloning are common recovery methods for maloperations, but they are error-prone and time-consuming in case of minor data changes and urgent troubleshooting, and are uncontrollable in recovery time when dealing with major data changes. The TXSQL team has developed and implemented the flashback query feature for the InnoDB engine. It allows you to query the historical data before a maloperation with a simple SQL statement and query the data at a specified time point through specific SQL syntax. This greatly saves the data query and recovery time and enables fast data recovery for better business continuity.

Supported Versions

TDSQL-C for MySQL 8.0 (kernel version 3.1.10) or later.

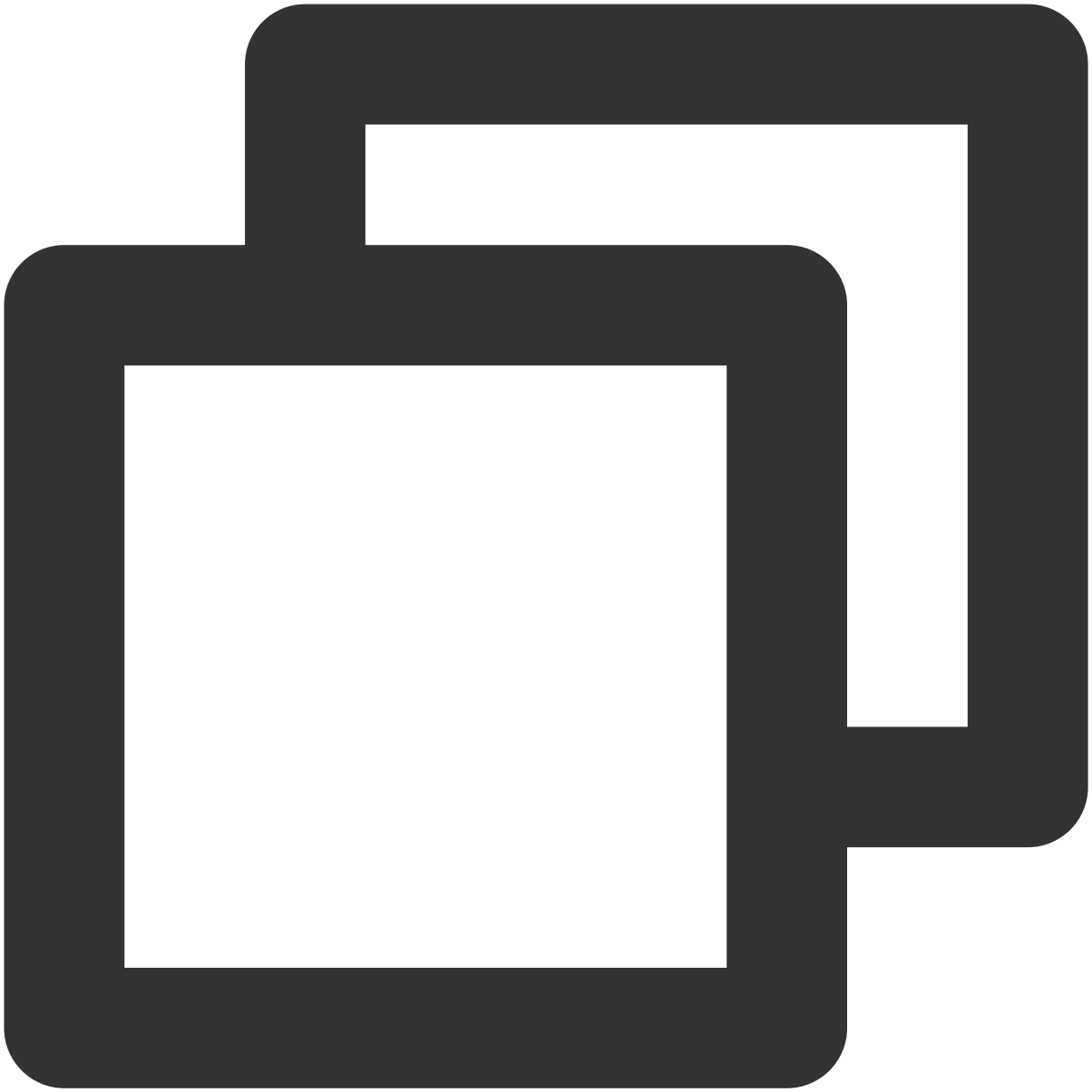
Use Cases

The flashback query feature is used to quickly query the historical data after a maloperation during database Ops.
Notes:
Flashback query is supported only for InnoDB physical tables but not views, other engines, or functions without actual columns such as `last_insert_id()` .
Only second-level flashback query is supported, and the accuracy cannot be fully guaranteed. If there are multiple changes within one second, any of them may be returned.
Flashback query is supported only for primary keys (or GEN_CLUST_INDEX).
Flashback query cannot be used in prepared statements or stored procedures.
Flashback query does not support DDL. If you perform DDL on a table (such as TRUNCATE TABLE, which should be recovered through the recycle bin), the results obtained by flashback query may not be as expected.
In the same statement, if multiple flashback query times are specified for the same table, the earliest time will be selected.
Due to the time difference between the read-write and read-only instances, if you specify the same time for flashback query, the results obtained for the instances may be different.

Enabling the flashback query feature will delay undo log cleanup and increase the memory usage. We recommend that you not set `Innodb_backquery_window` to a large value (preferably between 900 and 1,800), especially for instances with frequent business access requests.
If the database instance restarts or crashes, the historical information before the restart or crash cannot be queried.
The specified time should be within the supported range (which can be viewed through the status variables `Innodb_backquery_up_time` and `Innodb_backquery_low_time` by running `show status like '%backquery%')` .

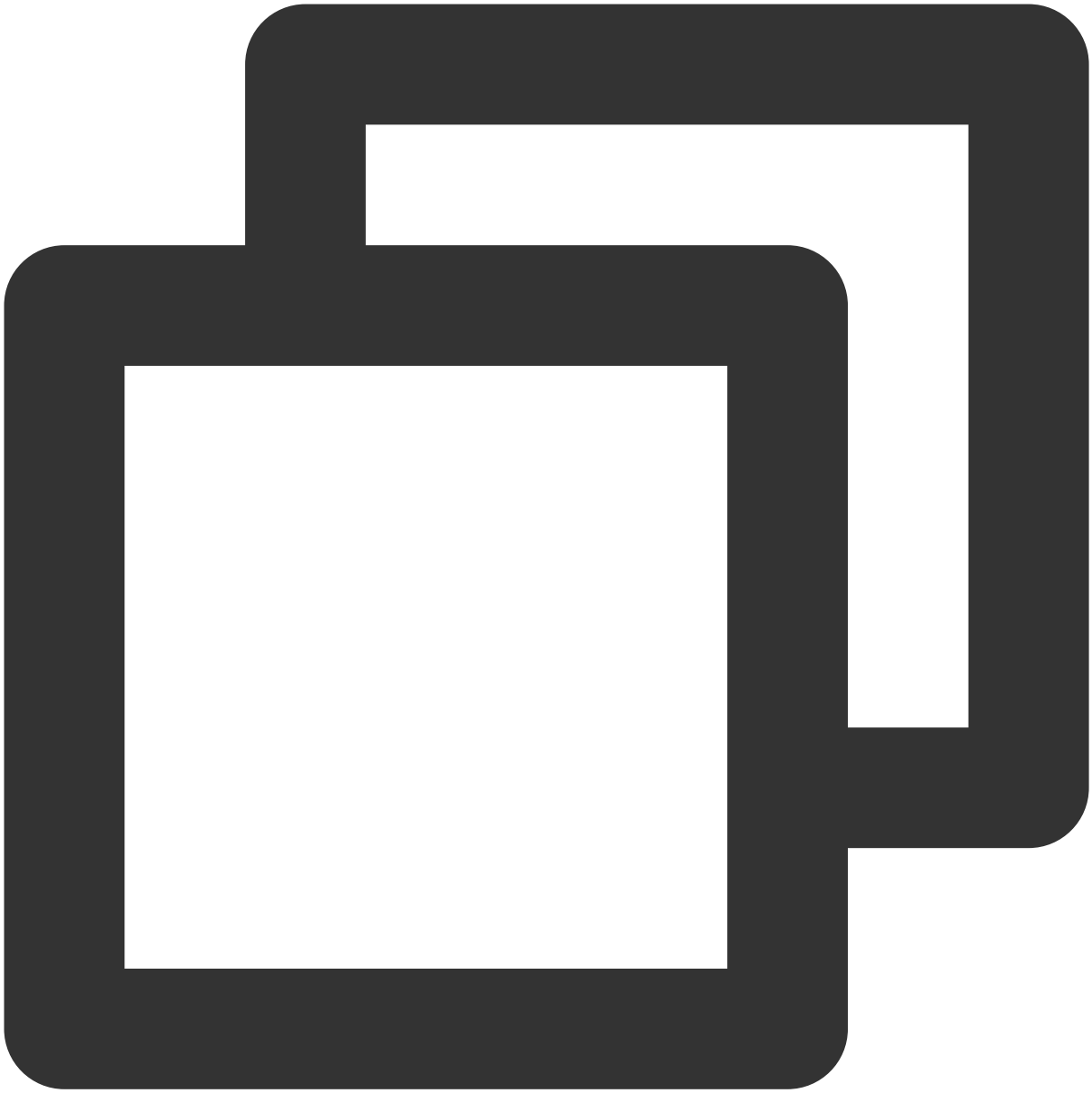
Use Limits

Flashback query provides a new AS OF syntax. You can set the `Innodb_backquery_enable` parameter to `ON` to enable the flashback query feature and then query data at the specified time. The syntax involved is as follows:



```
SELECT ... FROM <table name>
AS OF TIMESTAMP <time>;
```

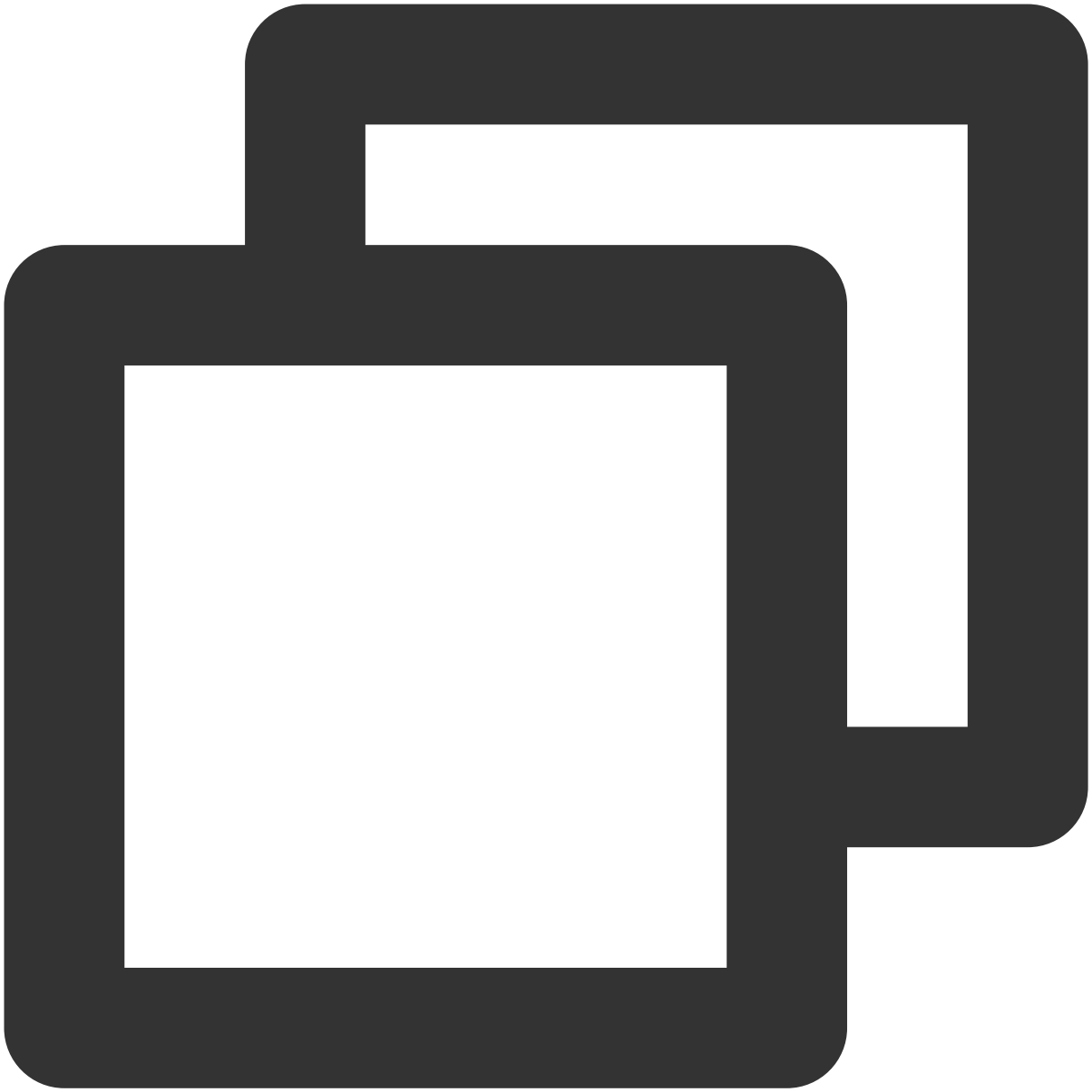
Sample of querying data at the specified time



```
MySQL [test]> create table t1(id int,c1 int) engine=innodb;
Query OK, 0 rows affected (0.06 sec)

MySQL [test]> insert into t1 values(1,1),(2,2),(3,3),(4,4);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

MySQL [test]> select now();
+---------------------+
| now()               |
+---------------------+
```

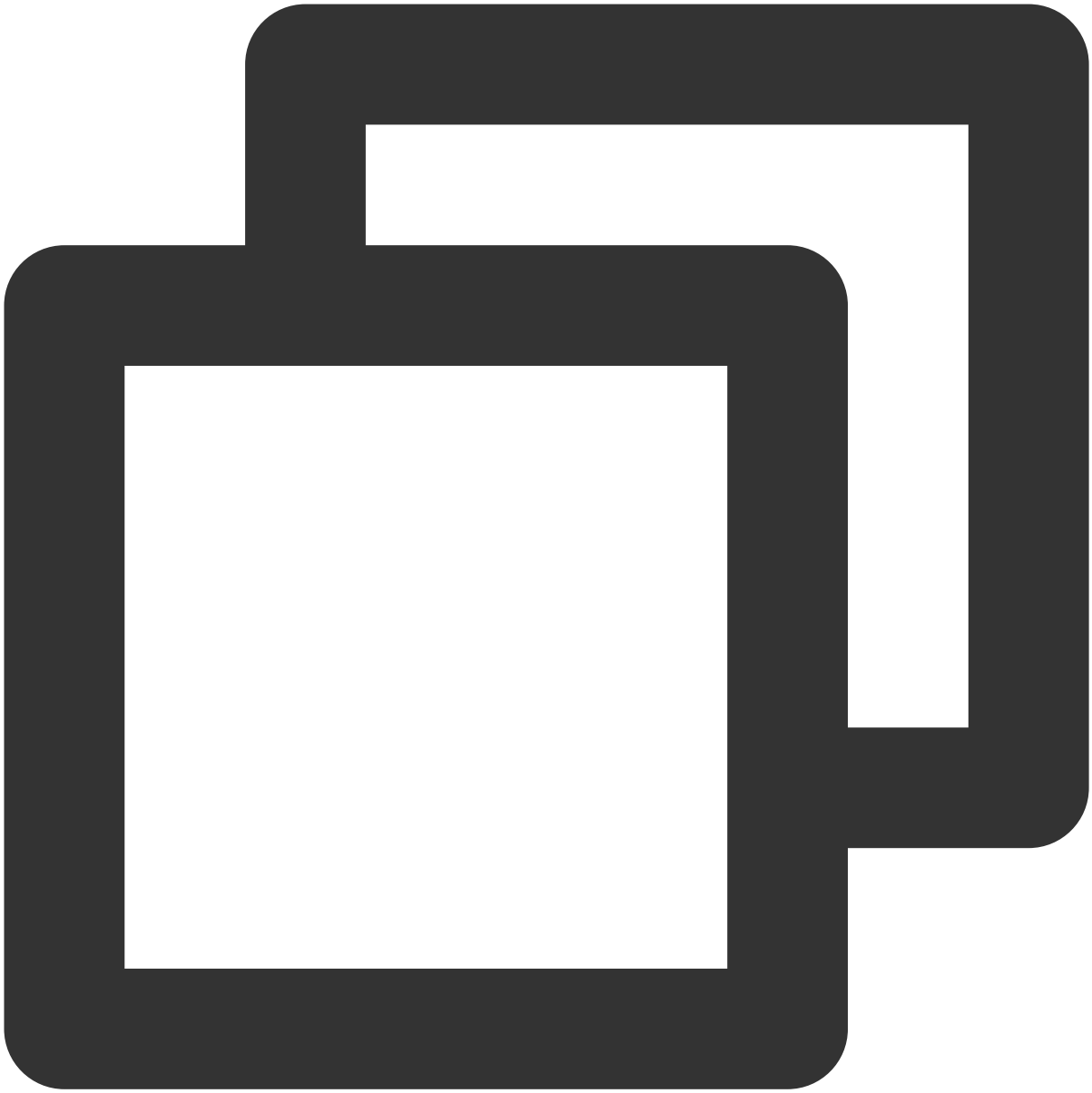
```
| 2023-08-17 15:50:01 |
+---------------------+
1 row in set (0.00 sec)

MySQL [test]> delete from t1 where id=4;
Query OK, 1 row affected (0.00 sec)

MySQL [test]> select * from t1;
+------+------+
| id   | c1   |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    3 |    3 |
+------+------+
3 rows in set (0.00 sec)

MySQL [test]> select * from t1 as of timestamp '2023-08-17 15:50:01';
+------+------+
| id   | c1   |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    3 |    3 |
|    4 |    4 |
+------+------+
4 rows in set (0.00 sec)
```

Sample of creating a table from historical data



```
create table t3 select * from t1 as of timestamp '2023-08-17 15:50:01';
```

Sample of inserting historical data into a table



```
insert into t4 select * from t1 as of timestamp '2023-08-17 15:50:01';
```

Parameter Description

The following table lists the configurable parameters of the flashback query feature.

Parameter	Scope	Type	Default Value	Value Range/Valid Values	Restart Required

innodb_backquery_enable	Global	Boolean	OFF	ON/OFF	No
innodb_backquery_window	Global	Integer	900	1–86400	No
innodb_backquery_history_limit	Global	Integer	8000000	1–9223372036854476000	No

Performance Features
Optimization of Plan Caching Point Query

Last updated：2024-04-25 11:10:02

Overview

In TDSQL-C for MySQL, SQL statement execution is divided into four stages: parsing, preparation, optimization, and execution. The execution plan cache feature is only available for prepared statements. After the feature is enabled, the first three stages will be skipped when executing a prepared statement, greatly boosting query performance.

Supported Versions

TDSQL-C for MySQL 8.0 (kernel version 3.1.10) or later.

Use Cases

This feature is mainly used to improve the query performance when executing many online short point queries with prepared statements. The specific extent of performance improvement depends on the online business.

Use Limits

You can use the `cdb_plan_cache` parameter to enable or disable the execution plan cache and the `cdb_plan_cache_stats` parameter to query information about cache hits.

Parameter	Status	Type	Default Value	Valid Values	Description
cdb_plan_cache	yes	bool	false	true/false	Whether to enable the feature. Only accounts with the feature permission can use the parameter.

Note:
Currently, you cannot directly modify the values of the above parameter. If needed, submit a ticket for assistance.

Upon enabling the `cdb_plan_cache_stats` parameter, one can then view related data through the `show cdb_plan_cache_stat` command.

You can run the `show cdb_plan_cache_stat` command to query information about execution plan cache hits.

The command will return the following fields:

Field	Description
sql	A SQL statement with the question mark (?) which represents that the execution plan of this statement has been cached.
mode	SQL cache mode. Currently, only the prepare mode is supported.
hit	Number of hits for this session

Note：

When the `cdb_plan_cache_stats` switch is enabled, it essentially functions as an information record, which will have an impact on performance.

Auto-Increment Column Persistence

Last updated：2023-11-01 16:44:24

Overview

The auto-increment column persistence feature can persist an auto-increment column into a page to avoid duplicate auto-increment values.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.

Use Cases

This feature is suitable for scenarios where you don't want duplicate auto-increment values, such as legacy data archive.

Use Limits

This feature is enabled in the kernel by default.

Invisible Index

Last updated：2024-04-25 11:14:26

Overview

You may require the capability to make an index invisible to determine whether it can be deleted By making an index as invisible, you can test the impact of its deletion on query performance before deleting it. If the index is being used by any program or database user, an error will occur or be reported. This feature is now available to MySQL 5.7 and later versions, not just limited to MySQL 8.0.

Supported Versions

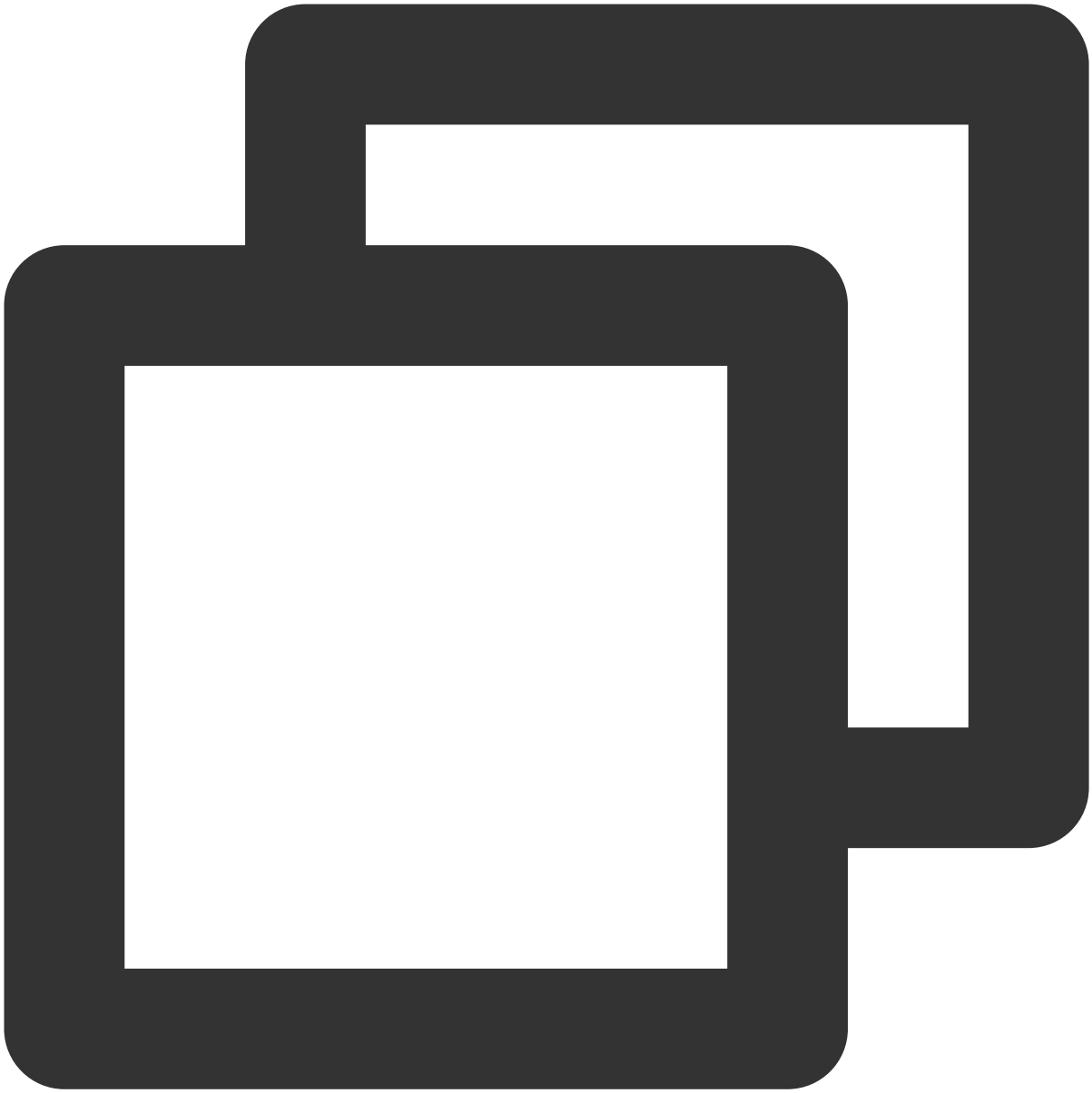
TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.
TDSQL-C for MySQL 8.0 (kernel version 3.1.10) or later.

Use Cases

Before deleting an index, you can make it invisible to see if it is still in use. If not, it can be securely deleted.

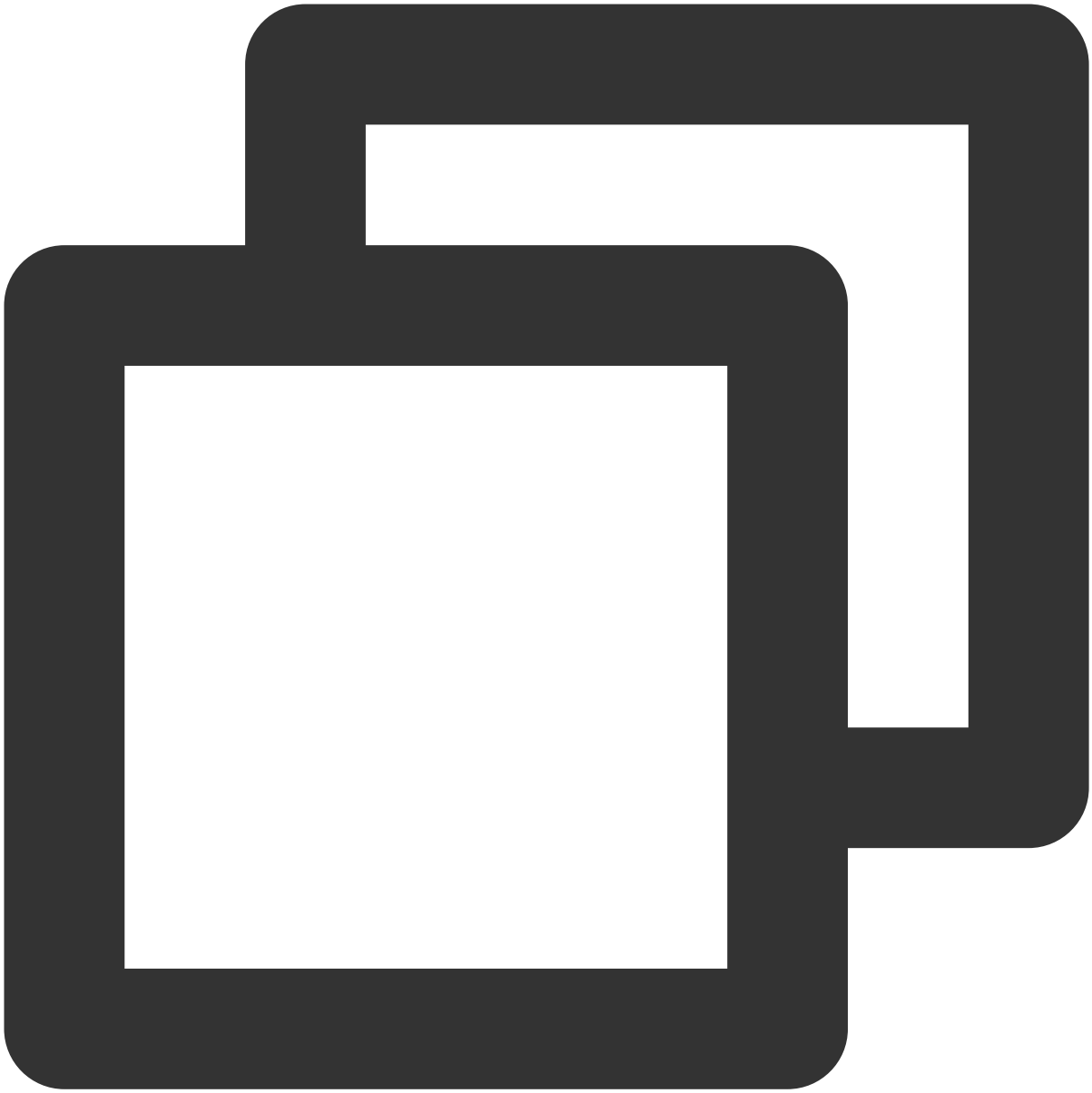
Use Limits

Run the following statements to create an invisible index or make an index invisible:



```
CREATE TABLE t1 (
  i INT,
  j INT,
  k INT,
  INDEX i_idx (i) INVISIBLE
) ENGINE = InnoDB;
CREATE INDEX j_idx ON t1 (j) INVISIBLE;
ALTER TABLE t1 ADD INDEX k_idx (k) INVISIBLE;
```

Run the following statements to make an index visible:



```
ALTER TABLE t1 ALTER INDEX i_idx INVISIBLE;
ALTER TABLE t1 ALTER INDEX i_idx VISIBLE
```

Computation Pushdown

Last updated：2023-11-01 16:47:39

Overview

This feature pushes LIMIT/OFFSET and SUM operations down to the storage engine InnoDB when querying single tables, effectively reducing query latency.
When LIMIT/OFFSET is executed using secondary indexes, this feature can avoid using the clustered index values as pointers to find the full table rows, effectively cutting the cost of scanning table data.
This feature pushes SUM operations down to InnoDB. In other words, instead of sending rows to the MySQL server, InnoDB calculates data itself and returns the final result to the MySQL server.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.

Use Cases

This feature is mainly used to optimize single-table queries with LIMIT/OFFSET or SUM clauses, such as "Select *from tbl Limit 10", "Select* from tbl Limit 10,2", and "Select sum(c1) from tbl".
This feature cannot optimize the following queries:
Queries with DISTINCT, GROUP BY, or HAVING clauses
Nested subqueries
Queries with FULLTEXT indexes
Queries with ORDER BY clauses, where the optimizer fails to use indexes to implement ORDER BY
Queries with multi-range read (MRR).
Queries with SQL_CALC_FOUND_ROWS.

Parameter Description

During the execution of a SQL statement, the optimizer automatically modifies the query execution plan to implement computation pushdown according to the following parameters.
Parameters are as follows:

Parameter	Effective	Type	Default	Valid	Description

	Immediately		Value	Values	
cdb_enable_offset_pushdown	Yes	bool	ON	{ON,OFF}	Enable or disable LIMIT/OFFSET pushdown. It is enabled by default.
cdb_enable_sumagg_pushdown	Yes	bool	OFF	{ON,OFF}	Enable or disable SUM pushdown. It is disabled by default.

Note:

Currently, you cannot directly modify the values of the above parameters. If needed, submit a ticketfor assistance.

Parallel Initialization of InnoDB Buffer Pool

Last updated：2024-06-17 16:31:45

Overview

This feature speeds up the initialization of the buffer pool, reducing the startup time of the database instance.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.

Use Cases

This feature is used to speed up the startup of the database instance.

Performance Test Data

Performance test data collected from eight instances:

buffer_pool_size	Buffer Pool Initialization Time (Before Optimization)	Buffer Pool Initialization Time (After Optimization)	Speed Enhancement
50GB	2.55s	0.13s	1962%
200GB	10.28s	0.52s	1977%
500GB	25.72s	1.32s	1948%

Stability Feature
Statement Outline

Last updated：2024-04-25 11:19:15

Overview

SQL tuning is a crucial step in improving database performance. To avoid the impact when the optimizer fails to select an appropriate execution plan, TXSQL provides the outline feature for you to bind execution plans. TDSQL-C for MySQL allows you to use hints to manually bind execution plans. The hint information contains the optimization rule for SQL statements, algorithm to be used, and index for data scan. An outline relies on hints to specify execution plans. TDSQL-C for MySQL provides the `mysql.outline` system table for you to add plan binding rules and the `cdb_opt_outline_enabled` switch for you to enable/disable the outline feature.

Supported Versions

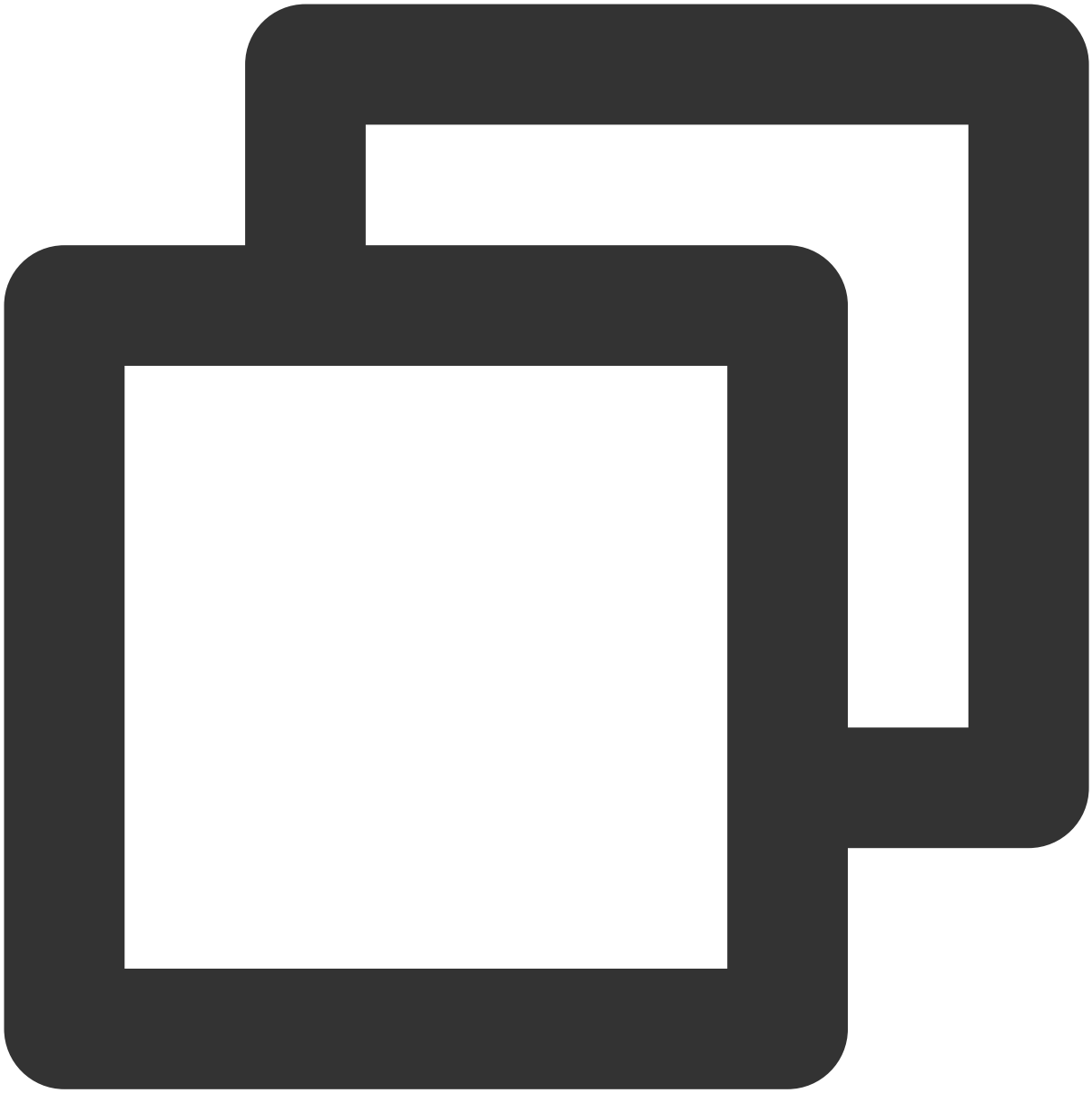
TDSQL-C for MySQL 8.0 (kernel version 3.1.10) or later.

Use Cases

This feature is suitable for scenarios where an execution plan in the production environment has poor performance (for example, the index in the execution plan is incorrect), but you don't want to modify SQL statements and release a new version to fix this problem.

Use Limits

The outline syntax uses a new syntax form:
Configure outline information: `outline "sql" set outline_info "outline";`
Clear outline information: `outline reset ""; outline reset all;`
Refresh outline information: `outline flush;`
Below are the outline use methods with the following schemas as examples:



```
create table t1(a int, b int, c int, primary key(a));
create table t2(a int, b int, c int, unique key idx2(a));
create table t3(a int, b int, c int, unique key idx3(a));
```

Parameter	Effective Immediately	Type	Default Value	Valid Values	Description
cdb_opt_outline_enabled	yes	bool	false	true/false	Whether to enable the outline feature.

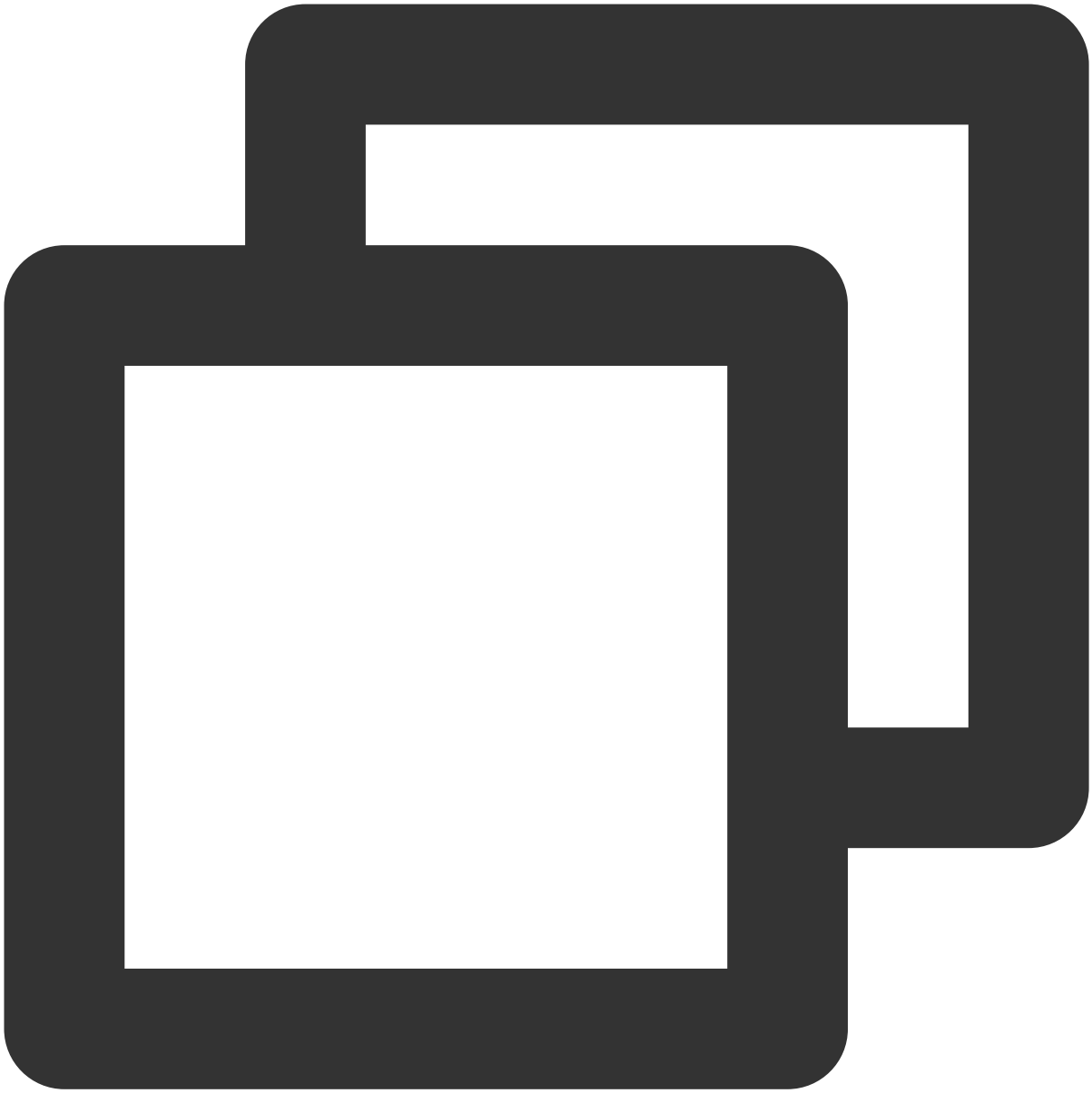
Note:

Currently, you cannot directly modify the values of the above parameter. If needed, submit a ticket for assistance.

Binding an outline

To bind an outline directly, you can replace one SQL statement with another, without changing the syntax of the SQL. This simply adds some HINT information to instruct the optimizer on how to execute the statement.

The syntax is in the format of `outline "sql" set outline_info "outline";` . Note that the string after `outline_info` must start with `"OUTLINE:"` , which is followed by the SQL statement with the hint information added. For example, you can add the index in column `a` to table `t2` in the SQL statement `select *from t1, t2 where t1.a = t2.a` as follows:



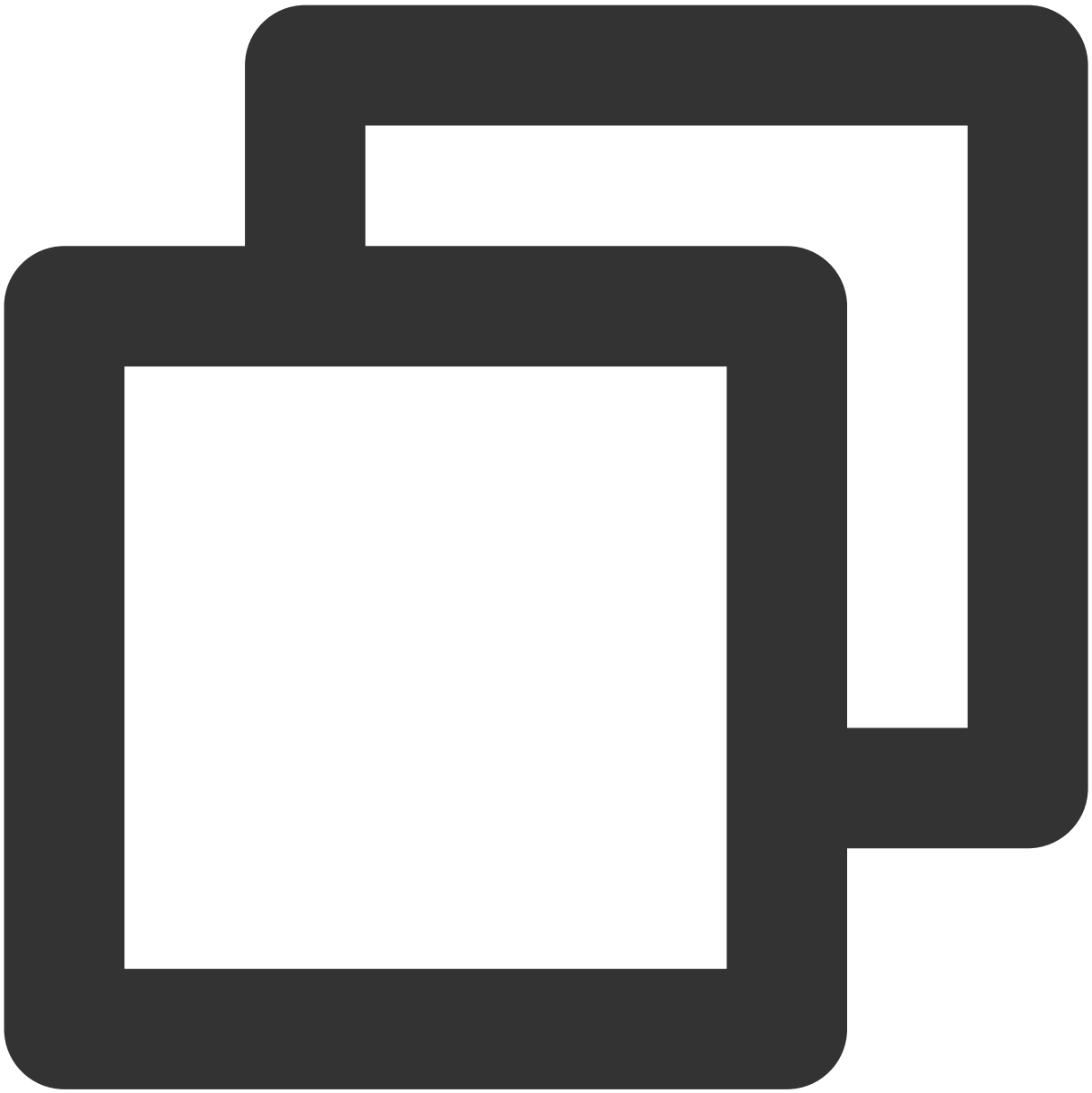
```
outline "select* from t1, t2 where t1.a = t2.a" set outline_info "OUTLINE:select *
```

Binding optimizer hint

To make the feature more flexible, TXSQL allows you to add optimizer hints incrementally to SQL statements. You can also implement the same feature by directly binding an outline.

The syntax is in the format of `outline "sql" set outline_info "outline";` . Note that the string after `outline_info` must start with `"OPT:"` , which is followed by the optimizer hint information to be added. For

example, you can specify `SEMIJOIN` of `MATERIALIZATION/DUPSWEEDOUT` for the SQL statement `select *from t1 where t1.a in (select b from t2)` as follows:



```
outline "select* from t1 where t1.a in (select b from t2)" set outline_info "OPT:2#
outline "select * from t1 where t1.a in (select b from t2)" set outline_info "OPT:1
```

You can add only one optimizer hint to the original SQL statement at a time and must comply with the following rules:

The `OPT` keyword must follow ".

':' must be placed before the new statement to be bound.

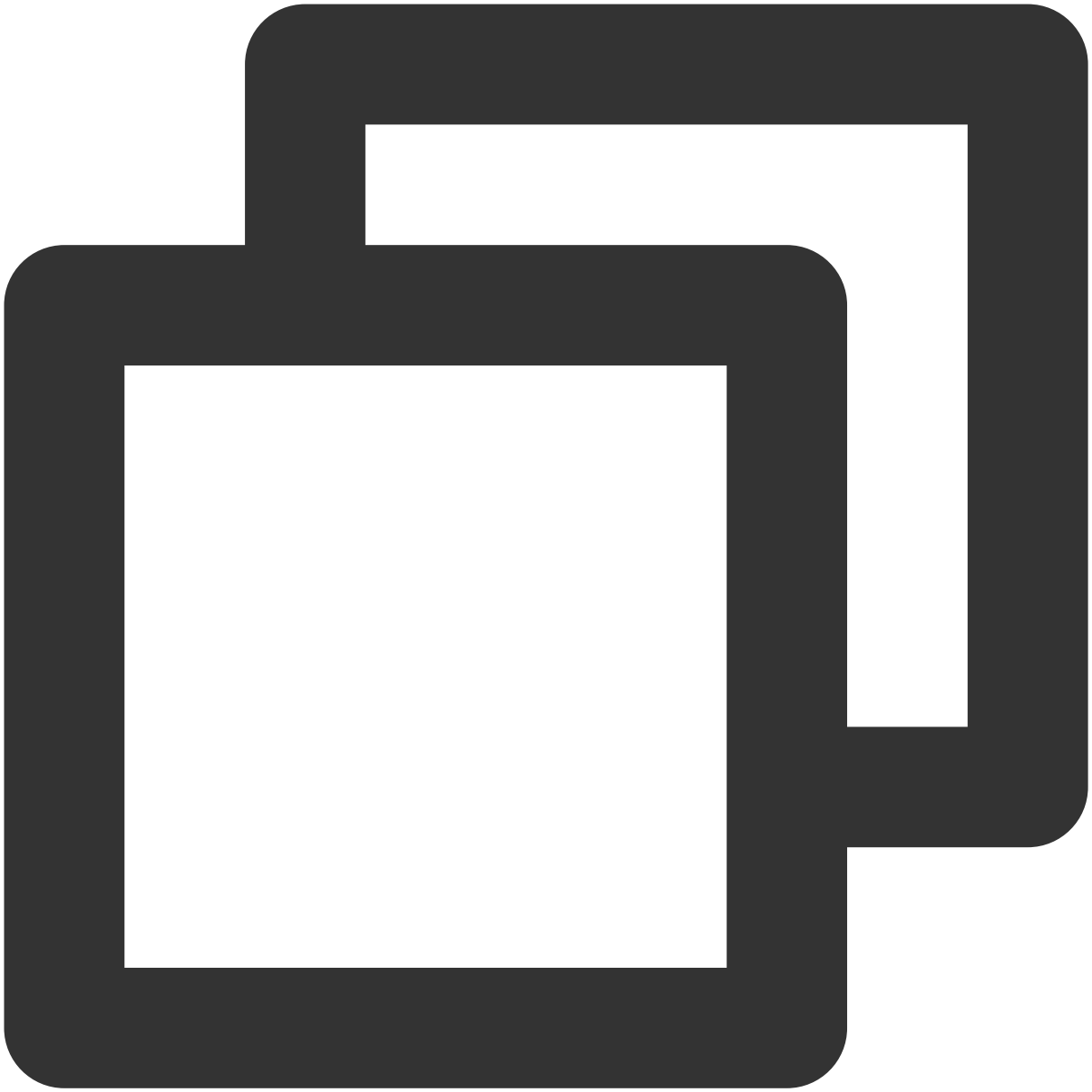
You must add two fields (query block number#optimizer hint string), which must be separated with "#" (e.g., `"OPT:1#max_execution_time(1000)"`).

Binding index hint

To make the feature more flexible, TXSQL allows you to add index hints incrementally to SQL statements. You can also implement the same feature by directly binding an outline.

The syntax is in the format of `outline "sql" set outline_info "outline";` . Note that the string after `outline_info` must start with `"INDEX:"` , which is followed by the index hint information to be added.

For example, you can add the index `idx1` of `USE INDEX` in `FOR JOIN` type to the table `t1` in the database `test` in query block 3 for the SQL statement `select *from t1 where t1.a in (select t1.a from t1 where t1.b in (select t1.a from t1 left join t2 on t1.a = t2.a))` as follows:



```
outline "select* from t1 where t1.a in (select t1.a from t1 where t1.b in (select t
```

You can add only one index hint to the original SQL statement at a time and must comply with the following rules:

The `INDEX` keyword must follow ".

':' must be placed before the new statement to be bound.

You must add five fields (query block number#db_name#table_name#index_name#index_type#clause).

Here, `index_type` has three valid values (0: INDEX_HINT_IGNORE; 1: INDEX_HINT_USE; 2: INDEX_HINT_FORCE), and `clause` also has three valid values (1: FOR JOIN; 2: FOR ORDER BY; 3: FOR

GROUP BY), which must be separated by "#" (e.g., `"INDEX:2#test#t2#idx2#1#1"`, indicating to bind the index `idx1` in `USE INDEX FOR JOIN` type to the table `test.t2` in the second query block).

Deleting the outline information of a SQL statement

TXSQL allows you to delete the outline binding information from a SQL statement.
The syntax is in the format of `outline reset "sql";`. For example, to delete the outline information from `select *from t1, t2 where t1.a = t2.a`, run the following statement: `outline reset "select* from t1, t2 where t1.a = t2.a";`.

Clearing all outline information

TXSQL allows you to clear all outline binding information in the kernel. The syntax is `outline reset all`, and the execution statement is `outline reset all;`.
There may be some specific problems in the production environment where you must bind an index. In this case, you can directly configure an outline for binding.
You should analyze the possible performance compromise after configuring an outline and bind an outline only if the compromised performance is acceptable. You can consult kernel engineers if necessary.

Hotspot Update Protection

Last updated：2024-04-25 11:21:31

Overview

For businesses with frequent updates or flash sales, the hotspot update feature greatly optimizes the performance of the UPDATE operation on frequently updated rows. If automatic hotspot update detection is enabled, the system will automatically detect whether there is a single row of hotspot update. If such an update is detected, it will queue the large number of concurrent UPDATE operations and execute them sequentially. This helps reduce the risk of concurrency performance being compromised by numerous row locks.

Supported Versions

TDSQL-C for MySQL 5.7 (kernel version 2.0.23/2.1.9) or later.
TDSQL-C for MySQL 8.0 (kernel version 3.1.5) or later.

Use Cases

This feature is suitable for scenarios where the pressure of updating a single row or multiple rows with the primary key specified is very high, such as flash sales.

Parameter Description

Parameter	Effective Immediately	Type	Default Value	Valid Values	Description
cdb_sql_filter_enable	yes	bool	off	on/off	Whether to enable hotspot update

Use Limits

Hotspot Update Protection