TDSQL-C MySQL 版

自研内核

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

- 自研内核
  - 内核概述
  - 内核版本更新动态
    - TXSQL 引擎内核版本更新动态
    - 数据库代理版本更新动态
  - 内核优化版本
    - 编译优化高性能版本
  - 功能类特性
    - 自动 kill 空闲事务
    - Instant DDL 功能介绍
    - 动态线程池
    - 支持 NOWAIT 语法
    - 支持 returning
    - 闪回查询
  - 性能类特性
    - 计划缓存点查优化
    - 支持自增列持久化
    - Invisible Index
    - 计算下推
    - InnoDB buffer pool 并行初始化
  - 稳定性特性
    - statement outline
    - 热点更新保护

版权所有：腾讯云计算（北京）有限责任公司

自研内核
内核概述

最近更新时间：2023-11-01 16:22:05

TDSQL-C MySQL 版引擎内核100%兼容原生 MySQL，您可以在不修改应用程序任何代码和配置的情况下，将 MySQL 数据库迁移至 TDSQL-C MySQL 版。
TDSQL-C MySQL 版引擎内核提供了多种 MySQL 企业版功能，如企业级透明数据加密、审计、线程池、加密函数、备份恢复等功能。
TDSQL-C MySQL 版引擎内核不仅对 InnoDB 存储引擎、查询优化等方面进行了大量优化，同时提升了数据库的易用性和可维护性，为用户提供 MySQL 全部功能的同时，还提供了企业级的容灾、恢复、监控、性能优化、读写分离、透明数据加密、数据库审计等高级特性。
有关 TDSQL-C MySQL 版引擎内核的更多信息：
TDSQL-C MySQL 版引擎内核版本更新动态，请参见 内核版本更新动态。
TDSQL-C MySQL 版支持自动或手动升级内核小版本，请参见 升级内核小版本。
有关 TDSQL-C MySQL 版数据库代理内核的更多信息：
TDSQL-C MySQL 版支持升级数据库代理内核小版本，请参见 升级数据库代理内核小版本。
有关 TDSQL-C MySQL 版各个内核版本发布时间：
TDSQL-C MySQL 版 8.0

内核版本	发布时间
3.1.10	2023年06月
3.1.9	2022年11月
3.1.8	2022年10月
3.1.7	2022年09月
3.1.6	2022年08月
3.1.5	2022年07月
3.1.3	2022年06月
3.1.2	2022年02月
3.1.1	2021年11月
3.0.1	2021年08月

版权所有：腾讯云计算（北京）有限责任公司

TDSQL-C MySQL 版 5.7

内核版本	发布时间
2.1.10	2023年07月
2.0.23/2.1.9	2023年05月
2.0.22/2.1.8	2022年11月
2.0.21/2.1.7	2022年09月
2.0.20/2.1.6	2022年08月
2.0.19	2022年07月
2.0.17	2022年06月
2.0.16	2022年01月
2.0.15	2021年10月
2.0.14	2021年07月
2.0.13	2021年03月
2.0.12	2020年11月
2.0.11	2020年06月

有关 TDSQL-C MySQL 版数据库代理各个内核版本发布时间：

数据库代理内核版本	发布时间
1.3.7	2023年05月
1.3.5	2022年11月
1.3.4	2022年09月
1.3.3	2022年08月
1.2.1	2022年07月

版权所有：腾讯云计算（北京）有限责任公司

内核版本更新动态

TXSQL 引擎内核版本更新动态

最近更新时间：2024-07-31 11:42:46

本文为您介绍 TDSQL-C MySQL 版内核版本更新动态。

说明:

如需升级，请参见 升级内核小版本，如需了解各个内核版本的发布时间，请参见 内核版本发布时间。

TDSQL-C MySQL 版 8.0 内核更新说明

TDSQL-C MySQL 版 5.7 内核更新说明

小版本	说明
3.1.14	功能更新 支持 列存索引功能（地域级灰度中），该功能使用列式数据格式存储、检索和管理数据的技术，可实现更好的查询性能和更高的数据压缩率。 支持并行查询增强功能：子查询并行、rollup、NLJ 内表并行、in-mem hash join 并行。 问题修复 修复分区表做 DDL 时可能导致 RO crash 的问题。 修复 cdc 订阅时收到 rotate event checksum 混乱导致出现乱码的问题。 修复使用 '{}' 直接插入 json 字符串，精度会丢失的问题。
3.1.13	性能优化 优化了全表扫描循环效率低导致的访问数据较慢的问题。 优化了并发场景下大查询扫描性能下降的问题。 问题修复 修复了开启数据库审计功能后 CPU 占用的问题。 修复了全文索引 doc id 重复的问题。 修复了全文索引在部分场景下会导致只读实例死锁的问题。 修复了分区表 DDL 会导致只读实例 crash 的问题。 修复了部分官方 Bug，具体如下： 修复 json prepare 相关 Bug：Bug #101284。 修复 sort buffer size 相关 Bug：Bug #32738705、Bug #33501541。 修复临时表 index scan 相关 Bug：Bug #33700735。 修复子查询相关 Bug：Bug #31216115、Bug #31946448、Bug #31216115、Bug #31216115、Bug #32813547、Bug #32813554。 修复部分段错误：Bug #32813554、Bug #32813547。
3.1.12	功能更新 支持在慢日志中查看存储层网络 IO 流量、所用时间和次数以及事务提交的时延。 数据库审计支持添加事务 ID 字段与表名。 支持黑洞引擎（ENGINE = BLACKHOLE;）。

	支持 DDL 执行所用的默认算法（inplace/instant），可以通过 innodb_alter_table_default_algorithm 参数来指定。 性能优化 大事务提交 binlog 优化：支持大事务的日志先写入临时文件，然后将临时文件重命名为下一个 binlog，以减少大事务提交阻塞其他事务的时间。 binlog 写入优化：支持 row 模式的 binlog 转化为 statement 格式的 binlog，并支持使用正则表达式设置生效的库表。 优化了 purge binlog 的性能，减少了性能抖动。 移除了 CDC、LSN、CLUSTER 作为保留关键字。 优化了并行 DDL 功能： 修复扫描线程大于1，但是 innodb_parallel_ddl_threads = 1时，创建的索引数据错误的问题。 修复 m_page 在创建子树后变得非法而导致 crash 的问题。 修复在创建索引前错误码未重置的问题。 修复在开启 innodb_disable_sort_file_cache 开关时，parallel ddl 创建索引 crash 的问题。 修复在有重复数据的列上添加 unique key 时会 crash 的问题。 问题修复 修复了数据库审计将 prepare 语句归属类型记录错误的问题。 修复了数据库审计将 execute 语句中的 sql_type 统一为“other”类型的问题。 修复了在 session track 中获取用户自定义变量字符串错误的问题：支持在获取自定义变量时，通过指定的字符串长度构造 session track 的返回值。 修复了并行查询中相关子查询引用 worker 表字段导致查询结果错误的问题。
3.1.10	功能更新 只读实例支持订阅 binlog。 支持黑洞引擎。 并行查询功能更新：支持全表扫描/全索引扫描/索引范围扫描并行查询，支持方差和标准差函数，支持在 LIMIT 语法下设置并行策略，支持 Prepared Statement（PS）查询模式。 支持 自动 kill 空闲事务。 支持 动态线程池。 支持 NOWAIT 语法。 支持 闪回查询。 支持 计划缓存点查优化。 支持 invisible index。 支持 statement outline。 问题修复 修复了分区表 instant add 导致 column 过多的问题，将 ddl 的默认算法由 instant 改成 inplace，使用 instant ddl 需要指明 algorithm = instant。 禁止有列名称为 fts_doc_id 的表执行 instant ddl。 优化了 resize buffer pool 功能。 优化并行查询中算子拆分 item 的拆分逻辑。 修复 PS 模式 IN 二分查找失效导致的性能衰退。 优化了并行查询中对于整表 count(*) 的查询效率。 修复 Parallel DDL 中 stage 变量错误，导致创建 FTS 索引场景出现 stage 空指针 crash 的问题。 修复新增全文索引过程中可能引发 crash 的问题。 修复了部分情况下 HA 导致 gtid 丢失的问题。 修复 ddl 时 kill mysqld 进程，重新拉起后 show create table 概率性触发 crash 的问题。

	修复全文索引过早释放 fts cache lock 导致读写不一致的问题。 修复了 ddl 提交过程中事务回滚导致主从数据不一致的问题。 修复了在创建临时表并插入数据的过程中，kill 事务并删库时出现死锁的问题。 修复了用户操作与 mysql 系统库的同名库时失败的问题。 修复 canal 通过 RO 抽取 binlog 无法用 show master status 获取 gtid 起点的问题。
3.1.9	功能更新 新增 cdc 能力，可直接重复回溯/抽取用户自定义日志保留时间段的 binlog，解决了计算节点在 HA 等场景下丢失本地 binlog 的问题。设置 binlog 保留时间请参见 设置备份保留时间。 优化 pages purge 速率，提高数据库性能。 问题修复 并行查询修复 worker 无法向 coordinator 传递表级别的 NULL ROW FLAG 标志导致结果错误的问题。 并行查询修复 sort order 被下推到 table 导致算子拆分过程中 sort 算子获取不到 order list 而产生 core 的问题。
3.1.8	功能更新 增加并行查询特性，自动识别复杂查询，利用并行查询能力，调动多核计算资源，大幅缩短大查询响应时间，使用请参见 开启或关闭并行查询 功能介绍。 问题修复 修复若干在 debug mode 下出现的数据库问题。 修复使用 show detailed processlist 显示连接信息的时候，数据库代理相关字段出现异常信息的问题。
3.1.7	功能更新 新增密码字典参数，使用请参见 自定义密码强度 功能介绍，优化了 HA 对字典文件依赖的问题。 内核支持补全 purged 的 binlog。 Serverless 架构支持内置数据库代理，实现连接保持、防闪断功能，解决首次唤醒连接报错的问题。
3.1.6	问题修复 修复 check index 中遇到全文索引（非树索引）导致 crash 的问题。 修复数据库代理探活导致内核输出大量 errlog 的问题，屏蔽冗余的日志打印。 修复 session lsn tracker 中 gcr lsn 未初始化可能返回数据库代理随机值导致语句执行超时的问题。
3.1.5	功能更新 支持 bulk insert 的限流。 支持 change buffer 以及 merge 方式设置。 支持数据库代理，使用请参见 数据库代理 功能介绍。 支持只读实例执行逻辑备份。 支持 binlog 在 table 级别的并行复制。 支持 SQL 限流功能。 支持 热点更新保护。 支持 sort merge join 下 interesting order 判断。 支持 TABLESAMPLE 功能。 支持 HISTOGRAM() 函数。 支持直方图历史版本功能、compressed 直方图。

	支持 show detail processlist。 性能优化 优化事务的物理复制，大幅提升只写性能。 优化事务系统的并行初始化，缩短系统的启动时间。 优化只读实例回放日志时对页面加锁的逻辑，加速回放线程的速度。 问题修复 修复 mysql client 出现接受不完整包异常退出的问题。 修复 blob 产生嵌套 mtr 提交顺序错误，导致 fsp 管理段 crash 的问题。 修复主机的 purge 可能导致 RO 访问二级索引时出现的事务一致性问题。 修复 backup lock 受 lock table 影响无法加锁的问题。 修复 cynosdb 引入的几个关键字不能作为标识符的问题，如 CDB_GET_TABLE_VERSION、CLUSTER、THREADPOOL。 修复实例启动时大事务或者长行事务回滚，使得主线程无法加上 dict op lock，导致启动时间长的问题。 修复 undo 页分配失败后续复用 trx 引起 crash 的问题。 修复对分区表执行 alter table 从扩展表空间往系统表空间迁移引起 crash 的问题。 修复 truncate log 未完整写入后启动 crash 的问题。 修复 drop table partition force 后插入数据导致 crash 的问题。 修复 instant DDL 更新数据然后 rollback 可能导致 crash 的问题。 修复 create temporary table like 扩展表空间的表创建失败的问题。 修复全文索引表灌数时 cache 持续上涨导致 OOM 的问题。 修复热点更新优化开启后性能不稳定的问题。 修复 `select count(*)` 并行扫描在极端情况下会全表扫描的问题。 修复多种情况下统计信息读零的问题，以及修复官方 Bug#31889883。 修复 query 长时间处于 query end 状态的问题。 修复 json_table 函数列名称大小写敏感的问题，官方 Bug#32591074。 修复使用 Temptable 引擎时，选择列中的聚合函数超过255个时报错的问题。 修复窗口函数因为表达式在 return true 时提前返回导致正确性问题的 bug。 修复 derived condition pushdown 在含有 user variables 的时候依然下压导致的正确性问题。 修复 SQL filter 在 Rule 规则没加 namespace 下容易导致 crash 的问题。 修复高并发高冲突情况下开启线程池的 QPS 抖动的问题。 修复移植执行 update 语句或存储过程未清理信息导致的 crash 问题。 修复现有版本中无法通过 CTRL+C 停止 histogram 的问题。
3.1.3	功能更新 支持将指定 filename 的 binlog 加入到 index 文件中。 新增 backup 锁，语法 LOCK TABLES FOR BACKUP，UNLOCK TABLES。 新增 binlog 锁，语法 LOCK BINLOG FOR BACKUP，UNLOCK BINLOG。 问题修复 修复动态元信息持久化导致实例表损坏或可见性错误的 bug。 合入官方 bugfix Bug #32897503，解决 prepare 语句下，部分查询语句执行路径错误的问题。 合入官方 bugfix：set resource group 失败会 crash 的问题。 修复 HA 切换后 previous gtid 为空的 bug。 修复自增列可设置为小于已插入最大值的 bug。 修复只读实例上的显式事务会阻塞回放线程回放 DDL 日志。 修复表名中存在"-"的表 DDL 后在只读实例上复制可能会 crash 的问题。

	修复只读实例启动遇到 DDL recover 导致操作 DDL log 系统表申请 undo 时 crash 的问题。
3.1.2	功能更新 支持 MySQL 8.0，增加只读节点，主备物理复制。 支持扩展表空间，实例容量最大支持1PB+容量。 支持预加载行数限制功能 ，在点查询相关测试中，得到了1~5%的性能提升。 支持扩展 ANALYZE 语法（UPDATE HISTOGRAM c USING DATA 'json'），支持直接写入直方图功能。 性能优化 使用直方图替代索引下探，降低评估误差以及 I/O 开销，该能力默认未打开。 问题修复 修复创建包含大对象列的全文索引时，大对象页相关更新没有写日志的问题。 修复计算和存储层 undo page 格式不一致的问题，TRX_UNDO_HISTORY_NODE 定义不一样。 修复 online-DDL 期间统计信息可能为零的情况。 修复从机 generated column 不更新的情况。 修复 binlog 压缩时实例 hang 住的问题。 修复新产生的 binlog 文件的 previous_gtids event 中的 gtid 缺失问题。 修复修改系统变量时可能死锁的问题。 修复 show processlist 中从机 sql 线程的 info 显示不正确的问题。 移植官方8.0.23中 hash join 相关的 bugfix。 移植官方 writeset 相关 bugfix。 移植官方8.0.24中查询优化器相关的 bugfix。 修复 FAST DDL 中优化 flush list 释放页面并发 bug。 优化海量个数表的实例升级数据字典时占用大量内存。 修复 instant add column 后在创建新主键场景下的 crash 问题。 修复全文索引查询中内存增长导致 OOM 问题。 修复 show processlist 返回结果集中 TIME 字段出现-1的问题。 修复直方图兼容性可能导致表无法打开的问题。 修复构建 Singleton 直方图的浮点累加误差。 修复 row 格式日志时表名为较长的中文字符导致复制中断问题。
3.1.1	功能更新 合并官方 8.0.19、8.0.20、8.0.21、8.0.22 变更。 支持动态设置 thread_handling 线程模式或连接池模式。 支持主从 bp 同步功能：当发生 HA 并进行主备切换后，备库通常需要一段比较长的时间来 warmup，把热点数据加载到buffer pool。为加速备机的预热，TXSQL 支持了主从 bp 同步功能。 支持 Sort Merge Join 功能。 支持异步提交，当打开线程池和关闭 binlog 时设置参数 innodb_log_sync_method =async 即可以开启异步提交。 支持 FAST DDL 功能。 支持用户侧查询 character_set_client_handshake 参数显示当前值功能。 支持数据库审计。 性能优化 优化扫描 flush list 刷脏：通过优化刷脏机制，解决了创建索引过程中的性能抖动问题，提升了系统稳定性。 优化 BINLOG LOCK_done 锁冲突，提升写入性能。

	使用 Lock Free Hash 优化 trx_sys mutex 冲突，提升性能。 redo log 刷盘优化。 buffer pool 初始化时间优化。 大表 drop table 清理 AHI 优化。 问题修复 修复修改 offline_mode、cdb_working_mode 参数的死锁问题。 修复 trx_sys的max_trx_id 持久化并发问题。 修复清理 innodb 临时表时造成的性能抖动问题。 修复核数较多的实例 read only 性能下降的问题。 修复 hash scan 导致1032问题。 修复热点更新功能的并发安全问题。
3.0.1	功能更新 支持 cynos_version，通过三种方式查询 select CYNOS_VERSION()、select @@cynos_version、show variables like 'cynos_version'。 增加空间限制参数，总空间超过限制时扩展的更新会报错，警告提示用户释放空间或者升级规格。 增加只读参数 innodb_ncdb_log_priority，表示主实例后台日志线程优先级。 增加只读参数 innodb_ncdb_apply_priority，表示只读实例日志回放线程优先级。 增加动态参数 innodb_ncdb_fast_shutdown，控制进程是否快速 shutdown，开启之后进程退出将不再做一些全局结构的析构操作，缩短 shutdown 时间，默认关闭。 增加参数 innodb_max_temp_data_file_size，只读参数，默认值128M，当此参数大于0时，代表本地存储中最大可以容纳的临时表空间。

小版本	说明
2.1.12	功能更新 只读实例支持通过位点订阅 binlog。 性能优化 优化了 purge binlog 的性能，减少了性能抖动。 问题修复 修复了 Flink 订阅 TDSQL-C 只读实例 binlog 的部分场景不兼容的问题。 修复了官方 Bug #27422376。 修复了 B-Tree 逆序遍历时，遇到过期的数据页会读重试的问题。 修复了 buffer pool resize 缩小时可能重复回收 page 的问题。
2.1.11	功能更新 数据库审计支持添加事务 ID 字段与表名。 支持多线程逻辑备份。 数据库代理支持 事务拆分。 性能优化 优化了内存使用机制，降低了 OOM 风险。 移除了 CDC、LSN、CLUSTER 作为保留关键字。 优化存储 apply 日志逻辑，提升存储层处理 redo 日志的能力。 问题修复 修复了数据库审计将 prepare 语句归属类型记录错误的问题

	修复了数据库审计将 execute 语句中的 sql_type 统一为“other”类型的问题。 修复了官方 Bug #111686和官方 Bug #26225783。
2.1.10	功能更新 只读实例支持订阅 binlog。 支持黑洞引擎。 支持 resize buffer pool。 问题修复 修复 RO 抽取 binlog 可能频繁断连的问题。
2.0.23/2.1.9	功能更新 支持 自动 kill 空闲事务。 支持 动态线程池。 支持 NOWAIT 语法。 支持 returning。 支持 自增列持久化。 支持 invisible index。 支持 计算下推。 支持 bufferpool 初始化。 支持 热点更新保护。 问题修复 修复了部分情况下 HA 导致 gtid 丢失的问题。 修复 ddl 时 kill mysqld 进程，重新拉起后 show create table 概率性触发 crash 的问题。 修复全文索引过早释放 fts cache lock 导致读写不一致的问题。
2.0.22/2.1.8	功能更新 新增 cdc 能力，可直接重复回溯/抽取用户自定义日志保留时间段的 binlog，解决了计算节点在 HA 等场景下丢失本地 binlog 的问题。设置 binlog 保留时间请参见 设置备份保留时间 。 优化 pages purge 速率，提高数据库性能。 问题修复 修复计算实例在触发空间上限场景下出现反复 crash 的问题。
2.0.21/2.1.7	功能更新 新增密码字典参数，使用请参见 自定义密码强度 功能介绍，优化了 HA 对字典文件依赖的问题。 内核支持补全 purged 的 binlog。 Serverless 架构支持内置数据库代理，实现连接保持、防闪断功能，解决首次唤醒连接报错的问题
2.0.20/2.1.6	问题修复 修复数据库代理探活导致内核输出大量 errlog 的问题，屏蔽冗余的日志打印。 修复 session lsn tracker 中 gcr lsn 未初始化可能返回数据库代理随机值导致语句执行超时的问题。 修复在只读实例创建临时表可能造成死锁的问题。
2.0.19	功能更新

	支持只读实例执行逻辑备份。 支持数据库代理，使用请参见 数据库代理 功能介绍。 支持 binlog 在 table 级别的并行复制。 支持 change buffer 以及 merge 方式设置。 支持 show detail processlist。 问题修复 修复官方 json 字符集相关的 Bug#22991924。 合入官方 bugfix Bug#25865525，解决 LOAD DATA INFILE 读入转义字符加 utf8 字符失败的问题。 合入官方 bugfix Bug#31529221，解决 ALTER TABLE 失败报 Incorrect key file 错误的问题。 合入官方生成列和级联删除相关的几个 bugfix，包括 Bug#33053297、Bug#32124113、Bug#29127203。 修复官方 Bug#31599938，slave 中关闭 log_bin 追 binlog 时 reset master 导致卡死的问题。 修复实例启动时大事务或者长行事务回滚，使得主线程无法加上 dict op lock，导致启动时间长的问题。 修复 undo 页分配失败后续复用 trx 引起 crash 的问题。 修复对分区表执行 alter table 从扩展表空间往系统表空间迁移引起 crash 的问题。 修复 truncate log 未完整写入后启动 crash 的问题。 修复 drop table partition force 后插入数据导致 crash 的问题。 修复 instant DDL 更新数据然后 rollback 可能导致 crash 的问题。 修复 create temporary table like 扩展表空间的表创建失败的问题。 修复全文索引表灌数时 cache 持续上涨导致 OOM 的问题。
2.0.17	功能更新 支持将指定 filename 的 binlog 加入到 index 文件中。 问题修复 合入官方 bugfix BUG#25865525，解决 LOAD DATA INFILE 读入转义字符加 utf8 字符失败的问题。
2.0.16	性能优化 优化 undo space truncate，提升大规格实例 undo truncate 速度。 优化只读实例上大范围查询的性能。 问题修复 修复 backup lock 受 lock table 语句影响无法加锁备份的问题。 修复 instant add 上千列后，RO 出现 table share 错乱的问题。 修复 binlog 的内容包含转义关键字回放报错的问题。 修复外部 prepared XA 事务不显式回滚阻塞正常 shutdown 的问题。 修复只读实例启动过程中报 warning“tablespace -1 not found” 警告的问题。
2.0.15	功能更新 支持扩展表空间，当单个表空间超过 innodb_ncdb_extend_space_threshold 配置新表会创建到扩展表空间中。 JSON 新增函数 JSON_MERGE_PRESERVE，JSON_MERGE_PATCH，JSON_PRETTY，JSON_STORAGE_SIZE，JSON_ARRAYAGG，JSON_OBJECTAGG。 优化系统启动时表锁恢复流程，缩短启动时间。 问题修复

	修复官方 bugfix，包含 text 列 group by 性能问题和多个虚拟列相关的问题。 修复实例启动后大事务回滚与 shutdown 操作并发时，可能导致实例 crash 的问题。 修复 undo space truncate 失败重试时相关页面 io 重试多次失败，导致实例退出的问题。 修复只读实例关闭时统计信息更新访问快照缓存，导致 crash 的问题。 修复 undo space truncate 可能存在 truncate log 落后于 truncate 操作的问题。 修复官方 bugfix，分区表扫描 ICP 检查错误，导致查询慢的问题。 修复只读实例中前项扫描遇到索引分裂日志部分回放，导致 crash 的问题。
2.0.14	功能更新 支持 instant modify column，使用请参见 Instant DDL 功能介绍。 问题修复 修复只读实例中临时表删除时，占用空间不回收的问题。 修复只读实例因为存储路由变更读到老的页面版本，导致退出的问题。 修复查询 information_schema.metadata_locks 时，可能发生的 crash 问题。 修复只读实例前向扫描与 btree SMO 的并发问题。 修复当多表出现复杂外键依赖，且外键属性为 ON DELETE CASCADE，在 DELETE 父表记录时可能导致子表对应的记录被删除两次的问题。 修复只读实例 create user 等相关操作，导致退出的问题。 修复只读实例中 show volume status，可能触发断言失败的问题。 修复分区表频繁 DDL 后，只读实例查询结果异常和 crash 的问题。 修复只读实例查询扫描到已经被 purge 的 undo log，导致构造历史版本 crash 的问题。
2.0.13	功能更新 支持 instant add column，使用请参见 Instant DDL 功能介绍。 优化高负载下的审计性能，增加动态参数 lock_usleep_time。 问题修复 修复官方针对 dict_operation_lock 在外键检查中可能引起的死锁问题。 修复只读实例启动阶段回放 acl change 日志，可能导致退出的问题。 修复 instant add column 和 truncate table 后，主从数据不一致的问题。 修复 instant add column 和 truncate table 后，再次插入数据导致日志回放错误的问题。 修复创建包含 instant add column 列的主键，导致退出的问题。 修复使用 instant column 新建 partition，rebuild partition，只读实例查询表结构退出的问题。
2.0.12	功能更新 支持数据库审计，使用请参见 开通 TDSQL-C MySQL 版审计。 支持 purge 页面预读，加快空间回收速度。 实时更新系统视图中表和索引大小信息。 增加额外的线程用以加快推进 recycle lsn，加快存储的 GC。 问题修复 修复全文索引官方 BUG，包括：BUG#24938374，BUG#21625016，BUG#27082268，BUG#27155294，BUG#27326796，BUG#27304661，BUG#25289359，BUG#29717909，BUG#30787535。 修复官方 BUG，涉及并发更新可能导致系统 crash，包括 BUG#30950714、BUG#31205266、BUG#25669686。 修复官方 BUG#28104394，未提交的插入操作会影响索引建的 range scan，使其返回错误的行数。

	修复官方 BUG#30488700，derived table 查询计划有误导致性能差的问题。 修复主机执行 DDL 后，只读实例回放统计信息日志可能导致退出的问题。 修复 rename table 到不存在 DB 的问题。 修复主机 optimize table，导致只读实例可能退出的问题。 修复全文索引表的更新在只读实例存在事务一致性的问题。 修复 set offline mode 和 show variables 操作发生死锁的问题。
2.0.11	功能更新 优化 binlog 文件索引，提升扫描 binlog 文件的速度。 优化 shutdown 流程，提升 shutdown 速度。 优化实例内存，压缩 buffer zip hash 等结构的内存占用。 优化只读实例启动时恢复 DDL lock 的流程，加快回放速度。 优化只读实例 load 系统表的流程，加快启动速度。 优化 purge coordinator 工作线程分配，提升 purge 速度。 问题修复 修复 index 结构映射错误，导致的 open table 时挂掉的问题。 修复 xa 事务回滚时，在只读实例复制出现异常的问题。 修复在只读实例中启动 binlog 复制，导致启动 crash 的问题。 修复 flush logs 导致的内存泄漏问题。 修复 mysqladmin shutdown 无法退出的问题。 修复主机 drop column时，只读实例连接无法回放日志的问题。 修复插入 blob 大对象触发空间限制后，依然可以灌数的问题。 修复主机大表 DDL 和日志写盘慢，可能导致的只读实例复制可用性问题。 修复临时表设置 innodb_max_temp_data_file_size 后，存储浪费小表的问题。

数据库代理版本更新动态

最近更新时间：2024-07-31 11:44:07

本文介绍 TDSQL-C MySQL 版数据库代理的内核版本更新说明。

说明：

如不满足 TDSQL-C MySQL 版内核版本要求，可先升级数据库内核版本，详细操作请参见 升级内核小版本，如需了解数据库代理各个内核版本发布时间，请参见 数据库代理内核版本发布时间。

版本	TDSQL-C MySQL 版内核版本要求	说明
1.3.10	TDSQL-C MySQL 版 5.7 ≥ 2.0.20/2.1.6 TDSQL-C MySQL 版 8.0 ≥ 3.1.6	问题修复 修复了处理回包时遇到特定报文可能出现异常的问题。
1.3.8	TDSQL-C MySQL 版 5.7 ≥ 2.0.20/2.1.6 TDSQL-C MySQL 版 8.0 ≥ 3.1.6	功能更新 支持了更多 MySQL 5.7和8.0中新版本更新的函数。 优化了解析器缓存，减少大量复杂 SQL 下的 OOM 概率。 增加了数据库代理的流量监控指标。 支持了动态负载均衡功能。 支持防闪断功能。
1.3.7	TDSQL-C MySQL 版 5.7 ≥ 2.0.20/2.1.6TDSQL-C MySQL 版 8.0 ≥ 3.1.6	问题修复 修复了某些情况 select for update 语句路由错误的问题。 更改了 select @@read_only 语句路由，现在 select @@read_only 会被路由到主库，避免某些框架使用 read_only 标记，错误判断数据库代理不可写的问题。 修复了部分场景下数据库实例 HA 引起数据库代理节点异常的问题。
1.3.5	TDSQL-C MySQL 版 5.7 ≥ 2.0.20/2.1.6TDSQL-C MySQL 版 8.0 ≥ 3.1.6	问题修复 优化了在高并发场景下只读实例的读性能出现下降波动的问题。
1.3.4	TDSQL-C MySQL 版 5.7 ≥ 2.0.20/2.1.6TDSQL-C MySQL 版 8.0 ≥ 3.1.6	问题修复 修复了 show processlist 返回数据不全的问题。
1.3.3	TDSQL-C MySQL 版 5.7 ≥ 2.0.20/2.1.6TDSQL-C MySQL 版 8.0 ≥ 3.1.6	问题修复 修复会话连接池在复用连接时，向后端发送 change_user 报错，数据库代理异常处理，新建连接后，未正确处理 prepare 语句的问题。 修复了 execute 语句没有参数类型的问题。

1.2.1	-	功能更新 支持 MySQL 5.7/8.0 版本。 支持集群部署，一个数据库代理下部署多个实例。 支持读写分离与读写分离下的权重配置。 支持故障转移功能，在只读实例异常时，会将读请求发送至读写实例。 支持负载均衡功能，应对各代理节点连接数不均衡的场景。 支持 hint 语法指定路由节点。 支持会话级连接池功能，应对短连接业务下，频繁和数据库建立连接的场景。 数据库代理会将连接进行保存，在下一次建连时复用连接。 支持热加载，配置均可在线更改，无需重启数据库专属代理。 支持了只读实例的重连功能。在长连接场景下，当只读实例发生重启，或者添加了新的只读实例，数据库代理将自动对只读实例重新建立连接恢复路由节点。

版权所有：腾讯云计算（北京）有限责任公司

内核优化版本
编译优化高性能版本

最近更新时间：2024-04-25 09:43:22

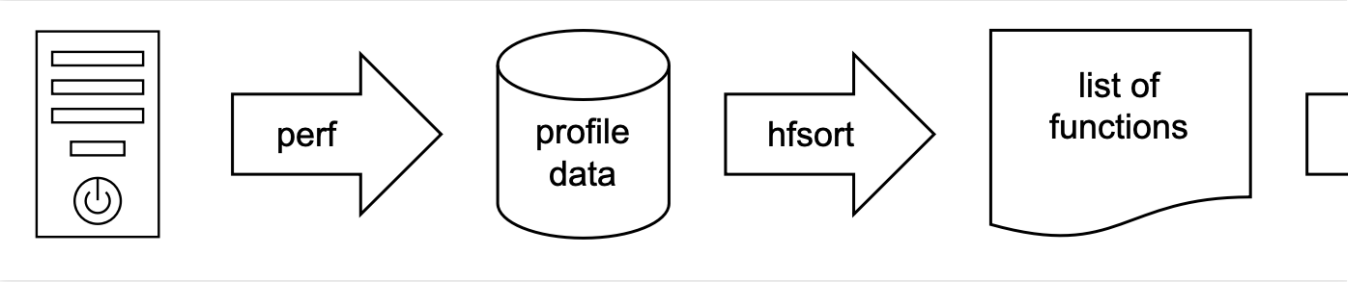
TDSQL-C MySQL 版的 TXSQL 内核支持编译优化高性能版本，可以在保持原有兼容性、不更改内核内部实现逻辑的前提下，借助动态编译器优化手段考虑用户可能的输入行为，使得数据库内核在业务通用场景下表现出更强的性能、同时降低其使用功耗。本文为您介绍 TDSQL-C MySQL 版的编译优化高性能版本。

支持版本

内核版本 TDSQL-C MySQL 版5.7 2.1.11及以上。
内核版本 TDSQL-C MySQL 版8.0 3.1.12及以上。
说明：
编译优化高性能版本当前为灰度发布中，如需提前体验，请在满足以上数据库内核版本的前提下 提交工单 申请使用。

背景

随着现代 CPU 内部实现的日益复杂，云数据库默认的编译、配置和运行方法很难充分发挥 CPU 的性能潜力，导致大量 CPU 周期空转。这种现象在规模效应叠加的情况下，不仅造成硬件资源的浪费，而且还会浪费大量的功耗。因此，需要对云数据库进行优化，以最大限度地发挥 CPU 的性能潜力，减少 CPU 周期的空转，提高硬件资源的利用率，并降低功耗的浪费。
TDSQL-C MySQL 版在不更改数据库内核业务逻辑代码前提下，借助动态编译器优化手段，以最小的代价完成了内核性能提升和功耗减少。通过收集云数据库在典型/真实业务场景下的行为和性能消耗数据，解决默认编译方式对业务行为无感知的问题；分析数据库运行时的行为特征，结合 CPU 微架构的特点，利用编译优化技术使得优化之后的版本对 CPU 微架构更友好，可充分发挥 CPU 的性能潜力；通过大量的场景测试以保证优化效果在各种条件下不劣化。
通过以上技术手段，此版本实现了如下优化：
1. 基于数据库运行行为数据实现反馈优化，提升函数内联/函数重排/基本库重排的优化能力，从而大幅降低数据库 CPU ICache/ITLB 失效，提升性能。
2. 通过“链接时优化技术”增加编译优化视野，从单文件单函数扩展到跨文件/全二进制文件范围，可极大提升内联的优化空间，降低指令数。
3. 在实践中摸索出一套高效验证和分析的方法，确保效果接近理论极限和保证在各种场景下不劣化。



什么是编译优化

编译优化是指在编译代码时，通过对代码进行优化和调整编译参数，以提高程序的执行效率和性能的过程。

优化原理

TDSQL-C MySQL 版的编译优化高性能版本使用 PGO（profile guided optimization） 技术进行优化，PGO 技术能够解决传统编译器在执行优化时，只基于静态代码信息，而不去考虑用户可能的输入，从而无法对代码进行有效优化的问题。

PGO 技术主要分为以下三个阶段：

1. instrument：在 instrument 阶段，先对应用做一次编译，在此编译中，编译器会向代码中插入指令，以便下一阶段可以收集数据。这些指令分为三种类型，分别用于统计每个函数被执行了多少次、每个分支被执行了多少次（例如 if-else 的场景）以及某些变量的值（主要用于 switch-case 的场景）。
2. train：在 train 阶段，用户需要使用最常用的输入来运行上一阶段编译生成后的应用。由于上一阶段已经做好了收集数据的准备，所以在经过 train 阶段之后，该应用最常见的使用场景对应的数据就会被收集下来。
3. optimization：在 optimization 阶段中，编译器会利用上一阶段收集到的数据，对应用进行重新编译。由于上一阶段的数据来自于用户输入的最常见的用户场景，所以最后优化得到的结果就能在该场景下有更好的优化。

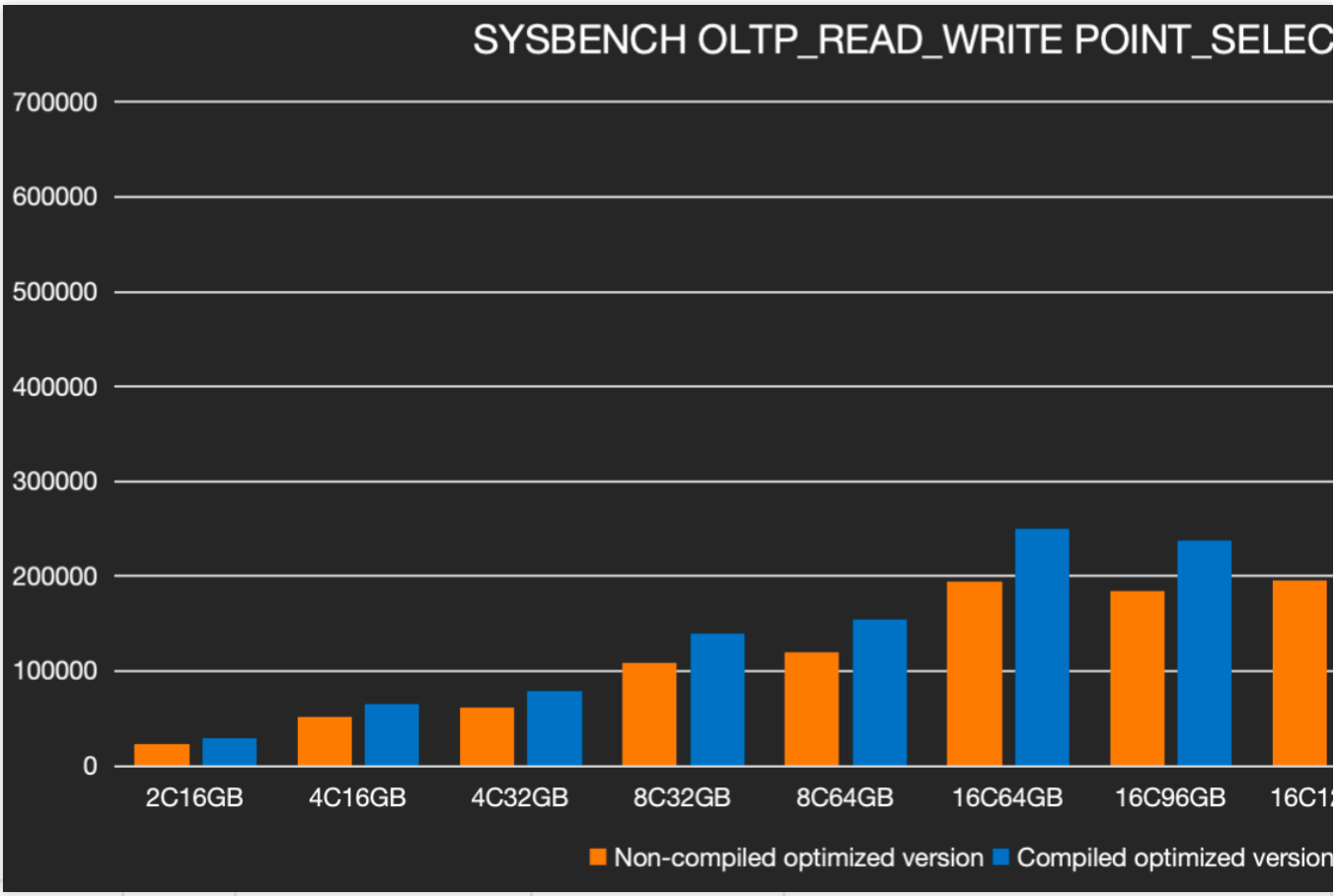
通过以上三个阶段的优化，TDSQL-C MySQL 版的编译优化高性能版本就能更好地适应用户的需求，提高应用的性能和效率。

性能测试

测试场景

混合读写（point select）测试场景，主要测试数据库在同时进行读写操作时的性能表现，可以帮助评估数据库在实际应用场景下的性能表现，包括并发读写操作的处理能力、响应时间、吞吐量等指标。

测试结果



规格	并发	单表数据量（table_size）	表总数（tables）	QPS	
				使用编译优化版本	提升百分比
2核16GB	64	800000	150	29207	27%
4核16GB	256	800000	300	65562	27%
4核32GB	256	800000	300	78973	27%
8核32GB	256	800000	300	139845	28%
8核64GB	256	800000	450	154894	28%
16核64GB	256	800000	450	249954	29%
16核96GB	256	800000	600	238061	29%
16核128GB	512	5000000	300	253848	29%
32核128GB	512	5000000	300	399647	30%
32核256GB	512	5000000	400	402105	30%

版权所有：腾讯云计算（北京）有限责任公司

64核256GB	512	6000000	450	596706	31%

功能类特性
自动 kill 空闲事务

最近更新时间：2023-11-01 16:49:27

功能介绍

Kill 超过一定时长的空闲事务，及时释放资源。

支持版本

内核版本 TDSQL-C MySQL 版 5.7 2.0.23/2.1.9 及以上。
内核版本 TDSQL-C MySQL 版 3.1.10 及以上。

适用场景

对于处于开启事务状态的连接（显示使用 begin、start transaction 或者隐式开启事务），如果超时时间内没有下一条语句执行，则进行 kill 连接。

使用说明

通过参数 cdb_kill_idle_trans_timeout 控制是否开启该功能，0为不用，非0为启用，与 session 的 wait_timeout 值相比取较小值。

参数名	动态	类型	默认	参数值范围	说明
cdb_kill_idle_trans_timeout	YES	ulong	0	[0, 31536000]	0代表关闭该功能，否则代表会 kill 掉 cdb_kill_idle_trans_timeout 秒的空闲事务

Instant DDL 功能介绍

最近更新时间：2024-04-25 10:55:18

适用场景

该功能主要针对在线业务的超大表 DDL 操作实现秒级变更。

功能介绍

通过 instant 算法来避免数据拷贝，进而实现大表快速修改列的功能，不拷贝数据，不占用磁盘空间和磁盘 I/O，业务高峰期可以实现秒级变更。

版本限制

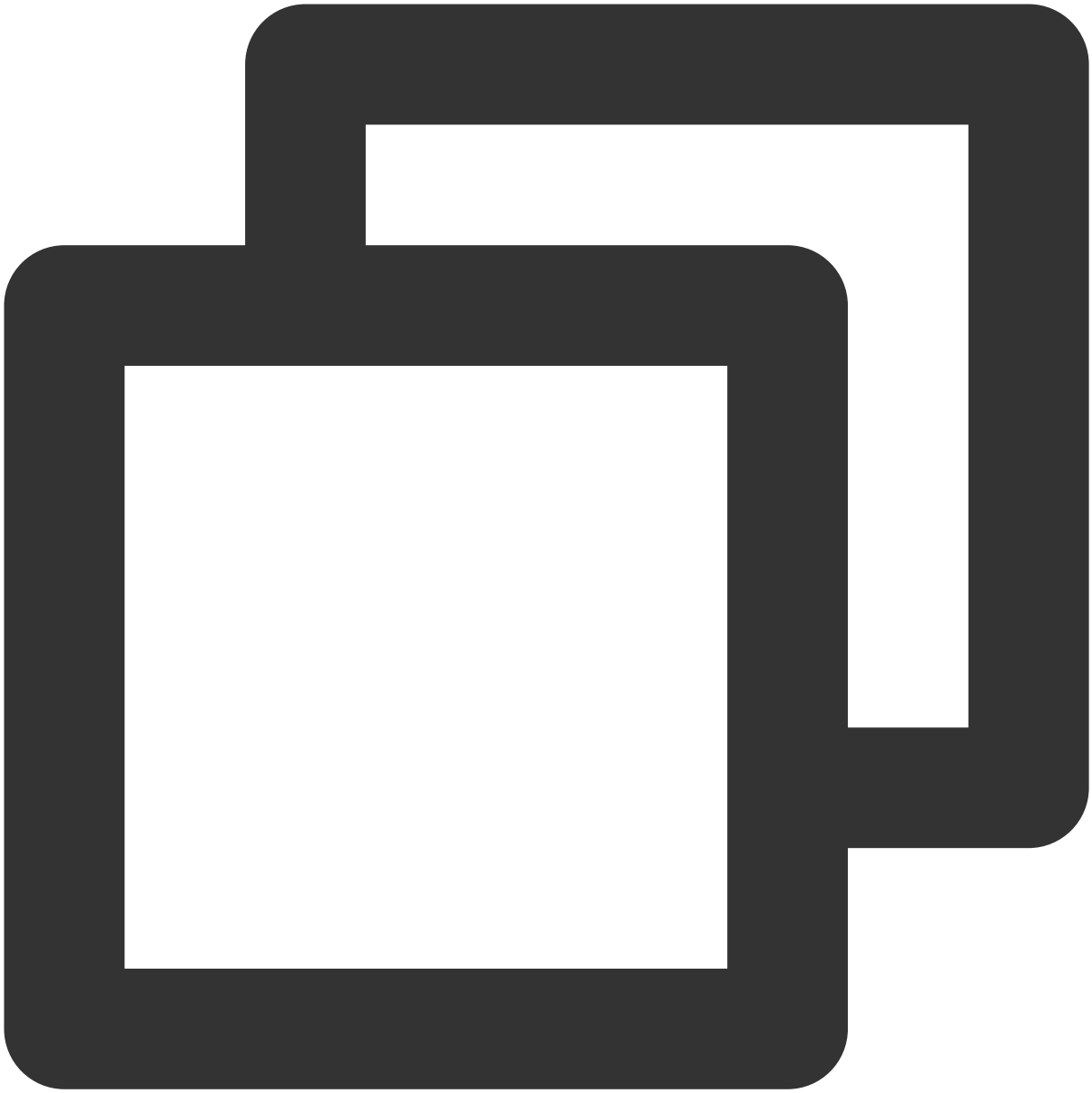
MySQL 5.7 内核小版本2.1.3及以上版本支持此功能。
MySQL 8.0 内核小版本3.1.1及以上版本支持此功能。

使用说明

目前 Instant DDL 支持的操作为 ADD COLUMN。

Instant Add Column 操作说明

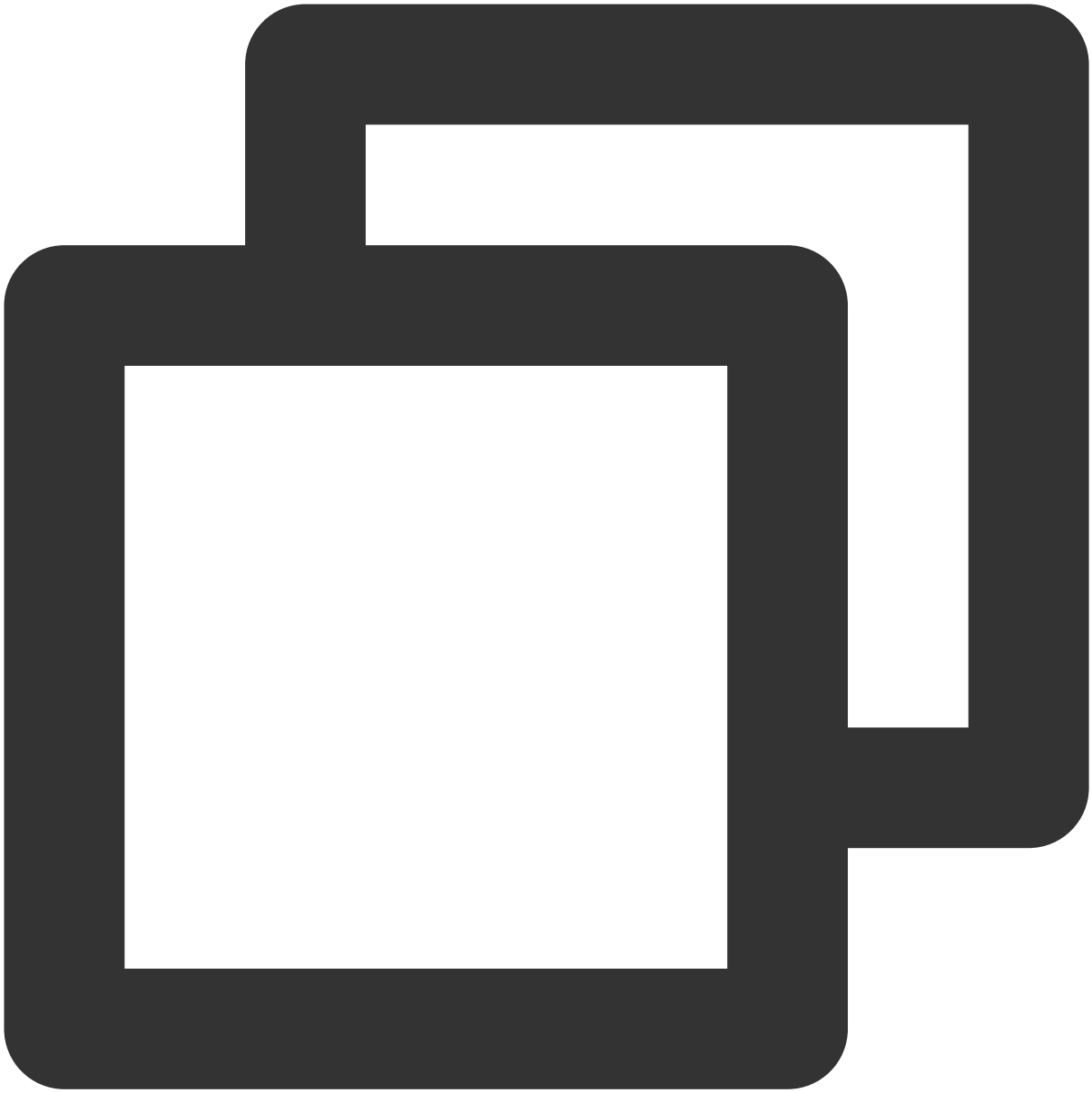
MySQL 8.0 内核小版本3.1.1至3.1.10操作说明
MySQL 8.0 内核小版本3.1.12及以上操作说明
Instant Add Column 的语法。
Alter Table 新增 algorithm = instant 子句，加列操作可通过如下语句进行：



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

新增参数 innodb_alter_table_default_algorithm，其可以设置为 inplace，instant，默认值为 instant。

加列操作可通过如下语句进行：



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

Instant Add Column 限制

一条语句中只有加列操作，不支持有其他的操作在同一条语句的情况。

新增列将会放到最后，不支持改变列的顺序。

不支持在行格式为 COMPRESSED 的表上快速加列。

不支持在已经有全文索引的表上快速加列。

不支持在临时表上快速加列。

动态线程池

最近更新时间：2024-04-25 10:58:43

功能介绍

线程池（Thread_pool）采用一定数量的工作线程来处理连接请求，通常比较适应于 OLTP 工作负载的场景。但线程池的不足在于当请求偏向于慢查询时，工作线程阻塞在高时延操作上，难以快速响应新的请求，导致系统吞吐量反而相较于传统 one-thread-per-connection（Per_thread）模式更低。
Per_thread 模式与 Thread_pool 模式各有优缺点，系统需要根据业务类型灵活地进行切换。遗憾的是，当前两种模式的切换必须重启服务器才能完成。通常而言，两种模式相互转换的需求都是出现在业务高峰时段，此时强制重启服务器将会对业务造成严重影响。
为了提高 Per_thread 模式与 Thread_pool 模式切换的灵活程度，TDSQL-C MySQL 版提出了线程池动态切换的优化，即在不重启数据库服务的情况下，动态开启或关闭线程池。

支持版本

内核版本 TDSQL-C MySQL 版5.7 2.0.23/2.1.9及以上。
内核版本 TDSQL-C MySQL 版8.0 3.1.10及以上。

适用场景

对性能敏感，需要根据业务类型灵活调整数据库工作模式的业务。

性能影响

pool-of-threads 切换为 one-thread-per-connection 过程本身不会带来 query 堆积，以及性能影响。
one-thread-per-connection 切换为 pool-of-threads 过程由于之前线程池处于休眠状态，在 QPS 极高并且有持续高压的情况下，可能存在一定的请求累积。解决方案如下：
方案1：适当增大 thread_pool_oversubscribe，并适当调小 thread_pool_stall_limit，快速激活线程池。待消化完堆积 SQL 再视情况还原上述修改。
方案2：出现 SQL 累积时，短暂暂停或降低业务流量几秒钟，等待 pool-of-threads 完成激活，再恢复持续高压业务流量。

使用说明

新增 thread_handling_switch_mode 用于控制线程池动态切换功能，可选值及其含义如下：

可选值	含义
disabled	禁止模式动态迁移
stable	只有新连接迁移
fast	新连接 + 新请求都迁移，默认模式
sharp	kill 当前活跃连接，迫使用户重连，达到快速切换的效果

在 `show threadpool status` 中新增如下状态：
connections_moved_from_per_thread 表示从 Per_thread 迁移至 Thread_pool 的 connections 数量。
connections_moved_to_per_thread 表示从 Thread_pool 迁移至 Per_thread 的 connections 数量。
events_consumed 表示每个线程池工作线程组消费的 events 总数，当 Thread_pool 迁移至 Per_thread 后，events 总数不再增加。
average_wait_usecs_in_queue 表示每个 event 平均在 queue 中等待的时间。
在 `show full processlist` 中新增如下状态：
Moved_to_per_thread 表示该连接迁移到 Per_thread 的次数。
Moved_to_thread_pool 表示该连接迁移到 Thread_pool 的次数。

参数说明

线程池相关参数的介绍：

参数名	动态	类型	默认	参数值范围	说
thread_pool_idle_timeout	Yes	uint	60	[1, UINT_MAX]	wo 的 间
thread_pool_oversubscribe	Yes	uint	3	[1,1000]	在 个
thread_pool_size	Yes	uint	当前机器 CPU 个数	[1,1000]	线
thread_pool_stall_limit	Yes	uint	500	[10, UINT_MAX]	每 tim 所

					当 优 的 组 负 线
thread_pool_max_threads	Yes	uint	100000	[1,100000]	线 总
thread_pool_high_prio_mode	Yes, session	enum	transactions	transactions\\statement\\none	高 三 tra 开 thr 不 队 thr 池 到 sta 入 no 有 中
thread_pool_high_prio_tickets	Yes, session	uint	UINT_MAX	[0, UINT_MAX]	tra 每
threadpool_workaround_epoll_bug	Yes	bool	false	true/false	是 bu

`show threadpool status` 命令展示的相关状态介绍：

状态名	说明
groupid	线程组 ID
connection_count	线程组用户连接数
thread_count	线程组内工作线程数
havelistener	线程组当前是否存在 listener
active_thread_count	线程组内活跃 worker 数量
waiting_thread_count	线程组内等待中的 worker 数量（调用 wait_begin 的 worker）

waiting_threads_size	线程组中无网络事件需要处理，进入休眠期等待被唤醒的 worker 数量（等待 thread_pool_idle_timeout 秒后自动销毁）
queue_size	线程组普通优先级队列长度
high_prio_queue_size	线程组高优先级队列长度
get_high_prio_queue_num	线程组内事件从高优先级队列被取走的总次数
get_normal_queue_num	线程组内事件从普通优先级队列被取走的总次数
create_thread_num	线程组内创建的 worker 线程总数
wake_thread_num	线程组内从 waiting_threads 队列中唤醒的 worker 总数
oversubscribed_num	线程组内 worker 发现当前线程组处于 oversubscribed 状态，并且准备进入休眠的次数
mysql_cond_timedwait_num	线程组内 worker 进入 waiting_threads 队列的总次数
check_stall_nolistener	线程组被 timer 线程 check_stall 检查中发现没有 listener 的总次数
check_stall_stall	线程组被 timer 线程 check_stall 检查中被判定为 stall 状态的总次数
max_req_latency_us	线程组中用户连接在队列等待的最长时间（单位毫秒）
conns_timeout_killed	线程组中用户连接因客户端无新消息时间超过阈值（net_wait_timeout）被 killed 的总次数
connections_moved_in	从其他线程组中迁入该线程组的连接总数
connections_moved_out	从该线程组迁出到其他线程组的连接总数
connections_moved_from_per_thread	从 one-thread-per-connection 模式中迁入该线程组的连接总数
connections_moved_to_per_thread	从该线程组中迁出到 one-thread-per-connection 模式的连接总数
events_consumed	线程组处理过的 events 总数
average_wait_usecs_in_queue	线程组内所有 events 在队列中的平均等待时间

版权所有：腾讯云计算（北京）有限责任公司

支持 NOWAIT 语法

最近更新时间：2024-04-25 11:01:04

功能介绍

DDL 支持 NO_WAIT 和 WAIT 选项。对于 DDL 操作，可通过 WAIT 设置等待 MDL LOCK 的秒数，如果在设定时间内未能获取到 MDL LOCK 则直接返回，也可指定 NO_WAIT 选项，未能获取到 MDL LOCK 直接返回。
SELECT FOR UPDATE 支持 NOWAIT 和 SKIP LOCKED 选项。在原有的 SELECT FOR UPDATE 逻辑下，如果目标行数据被另一个事务加了锁，则需要等待该事务释放锁，但在某些场景中，如秒杀，并不希望等待锁，通过 SKIP LOCKED 和 NOWAIT 选项提供一种不需要等待锁的功能。SKIP LOCKED 语句会跳过已经被加锁的行，这些行不会出现在结果集中；NOWAIT 语句遇到被加锁的行不会等待，同时会报错。
需要注意的是这两种 NO WAIT 使用的关键字是不一样的。

支持版本

内核版本 TDSQL-C MySQL 版5.7 2.0.23/2.1.9及以上。
内核版本 TDSQL-C MySQL 版8.0 3.1.10及以上。

适用场景

DevAPI/XPlugin 暂不支持 SELECT FOR UPDATE/SHARE 语句中使用 SKIP LOCKED 和 NOWAIT 选项。由于历史原因，DDL 的 NO_WAIT 关键字和 SELECT FOR UPDATE 的 NOWAIT 关键字是两个不同的关键字，需要注意区分。

版权所有：腾讯云计算（北京）有限责任公司

支持 returning

最近更新时间：2024-04-25 11:03:38

功能介绍

在某些使用场景下，需要在 DML 操作后返回刚操作的数据行。实现这个需求一般有两种办法：
一是在开启事务之后，在 DML 语句后面紧跟一条 SELECT 语句。
二是使用触发器等较为复杂的操作实现。
前者主要会增加一条 SELECT 语句的开销，后者则会令 SQL 的实现变得更加复杂并且不够灵活（需要创建触发器）。
因此，RETURNING 语法的设计主要针对该场景的优化，通过在 DML 语句后增加 RETURNING 关键字可以灵活高效地实现上述的需求。

支持版本

内核版本 TDSQL-C MySQL 版5.7 2.0.23/2.1.9及以上。
内核版本 TDSQL-C MySQL 版8.0 3.1.10及以上。

适用场景

在目前 TDSQL-C MySQL 版 5.7 2.0.23/2.1.9 及以上的内核版本中，分别支持：INSERT ... RETURNING、REPLACE ... RETURNING、DELETE ... RETURNING。该语法允许返回所有被 INSERT/REPLACE/DELETE 语句操作过的行（statement 为单位）。同时，RETURNING 也支持在 prepared statements，存储过程中使用。
在目前 TDSQL-C MySQL 版 3.1.10 及以上的内核版本中，分别支持：DELETE ... RETURNING、INSERT ... RETURNING、REPLACE ... RETURNING、UPDATE ... RETURNING 语法，可以返回该 statement 所操作的数据行。
在使用该功能时，需要注意以下几点：
1. 在使用 RETURNING 时，DELETE...RETURNING 语句返回前镜像数据，INSERT/REPLACE...RETURNING 返回后镜像数据。
2. INSERT/REPLACE 场景下，外层表的列对 returning 中的子查询语句，暂不具有可见性。
3. INSERT/REPLACE 的 RETURNING 语句若需要返回 last_insert_id()，则该 last_insert_id() 的值为该语句执行成功之前的值。若需要获得精确的 last_insert_id() 值，建议使用 RETURNING 直接返回该表的自增列 ID。

使用说明

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)
```

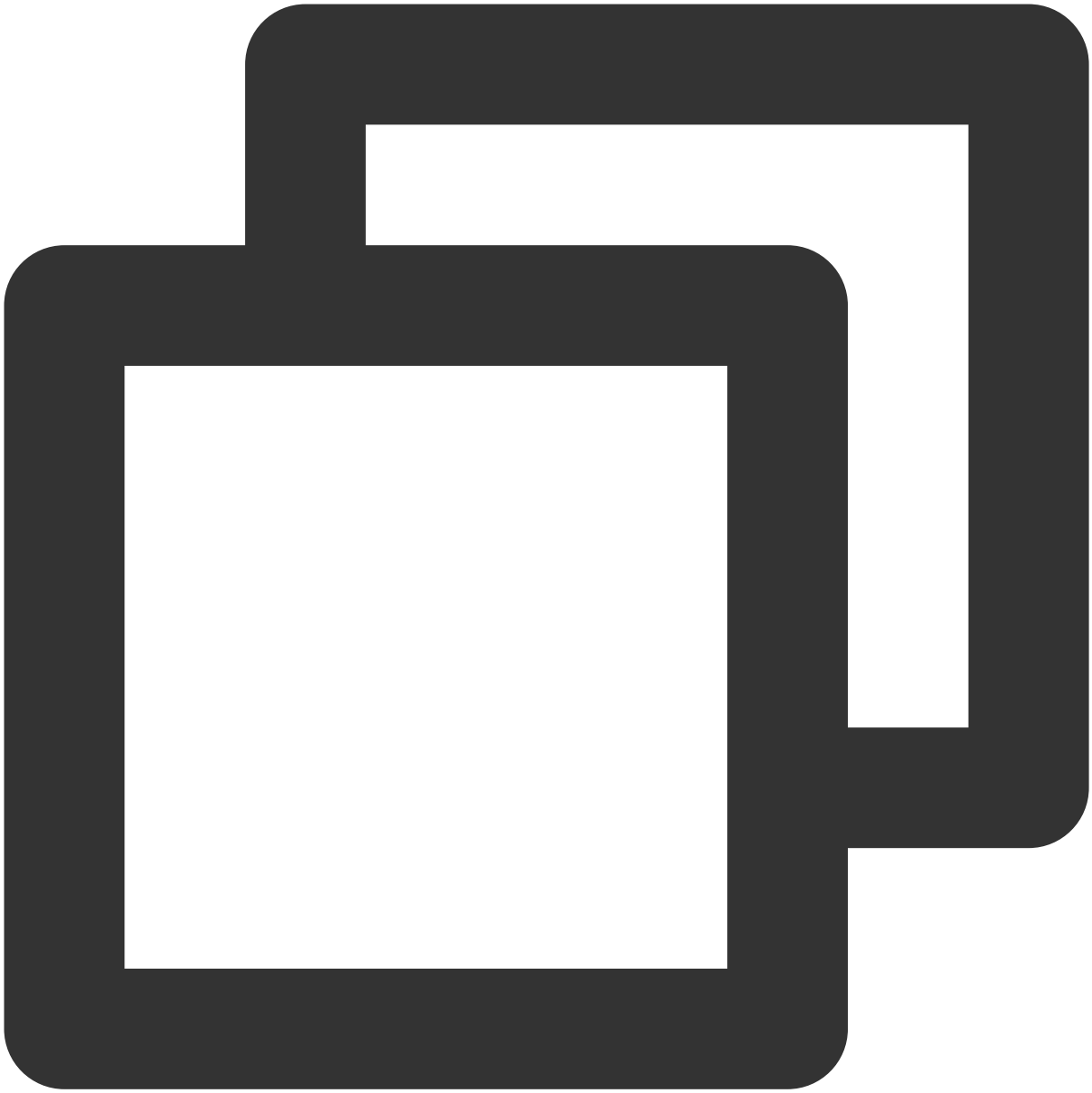
```
MySQL [test]> INSERT INTO t1 (id1) values (1) returning *, id1 * 2, id1 + 1, id1 *
+------+---------+---------+-------+--------------------+
| id1  | id1 * 2 | id1 + 1 | alias | (select * from t2) |
+------+---------+---------+-------+--------------------+
|    1 |       2 |       2 |     1 |   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

闪回查询

最近更新时间：2024-04-25 11:06:12

功能介绍

在数据库运维过程中可能会发生误操作的情况，这些误操作可能会给业务带来严重的影响，因误操作导致业务受到影响时，常见的恢复手段有回档、克隆等操作，但对于少量的数据变更以及紧急故障修复而言，容易出错且耗时较长，在数据量较大时恢复时间不可控。
TXSQL 团队在 Innodb 引擎上设计和实现了闪回查询功能，仅需通过简单的 SQL 语句即可查询误操作前的历史数据，通过特定的 SQL 语法查询指定时间点的数据，节省大量的数据查询和恢复时间，使得误操作后的数据能够快速恢复，从而保障业务快速恢复运行。

支持版本

内核版本 TDSQL-C MySQL 版8.0 3.1.10及以上。

适用场景

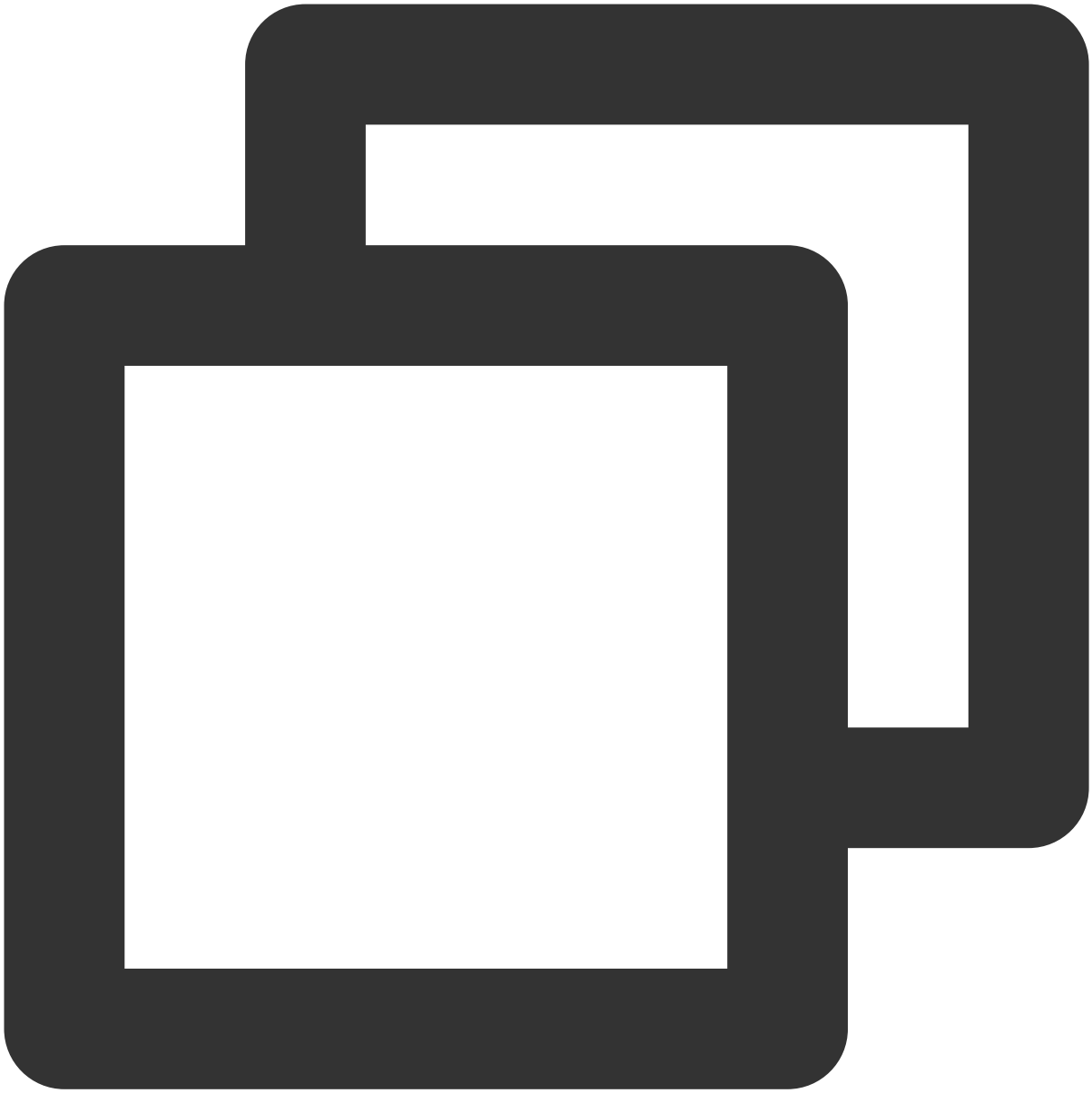
闪回查询功能用于在数据库运维过程中，误操作后进行快速的查询历史数据。
在使用该功能时，需要注意以下几点：
仅支持 Innodb 物理表，不支持 view 及其它引擎，不支持 last_insert_id() 等没有实际列对应的函数。
仅支持秒级的闪回查询，不保证百分之百准确，如果一秒之内有多个改动，可能会查询到其中任何一个。
闪回查询仅支持主键（或者 GEN_CLUST_INDEX）。
不支持在 prepared statement 和 stored procedure 中使用。
不支持 DDL，如果对表进行 DDL（如 truncate table，这种建议通过回收站进行恢复），闪回查询得到的结果可能不符合预期。
同一个语句中，同一张表如果指定了多个闪回查询时间，会选择离当前查询时间最远的时间。
由于主从实例存在时间差，指定相同时间进行闪回查询，主从实例获得的结果可能不一样。
开启闪回查询后会延迟 undo 日志清理以及增加内存占用，不建议 Innodb_backquery_window 设置过大（建议设置在900至1800之间），尤其是业务访问繁忙的实例。
如果数据库实例重启或者 crash，将不能查询到重启或 crash 之前的历史信息。指定的时间需要在支持的范围之内（支持范围可通过状态变量 Innodb_backquery_up_time 和 Innodb_backquery_low_time 查看，执行 `show status like '%backquery%'` ）。

使用说明

闪回查询提供了全新的 AS OF 语法，在参数设置中将 Innodb_backquery_enable 参数设置为 ON，打开闪回查询功能，通过特定语法查询指定时间的数据。语法如下：

```
SELECT ... FROM <表名>
AS OF TIMESTAMP <时间>;
```

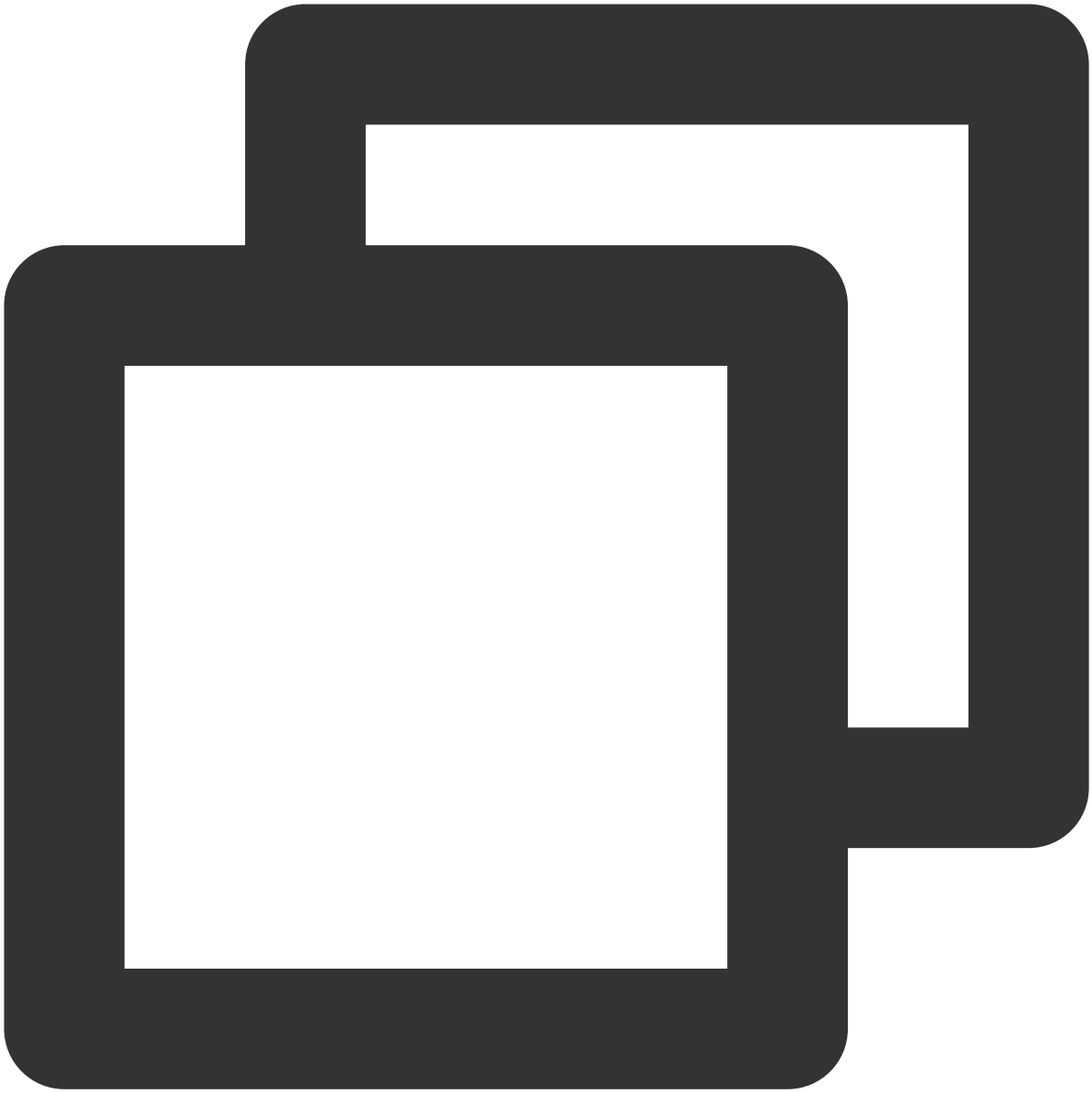
查询指定时间参考示例



```
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()               |
+---------------------+
```

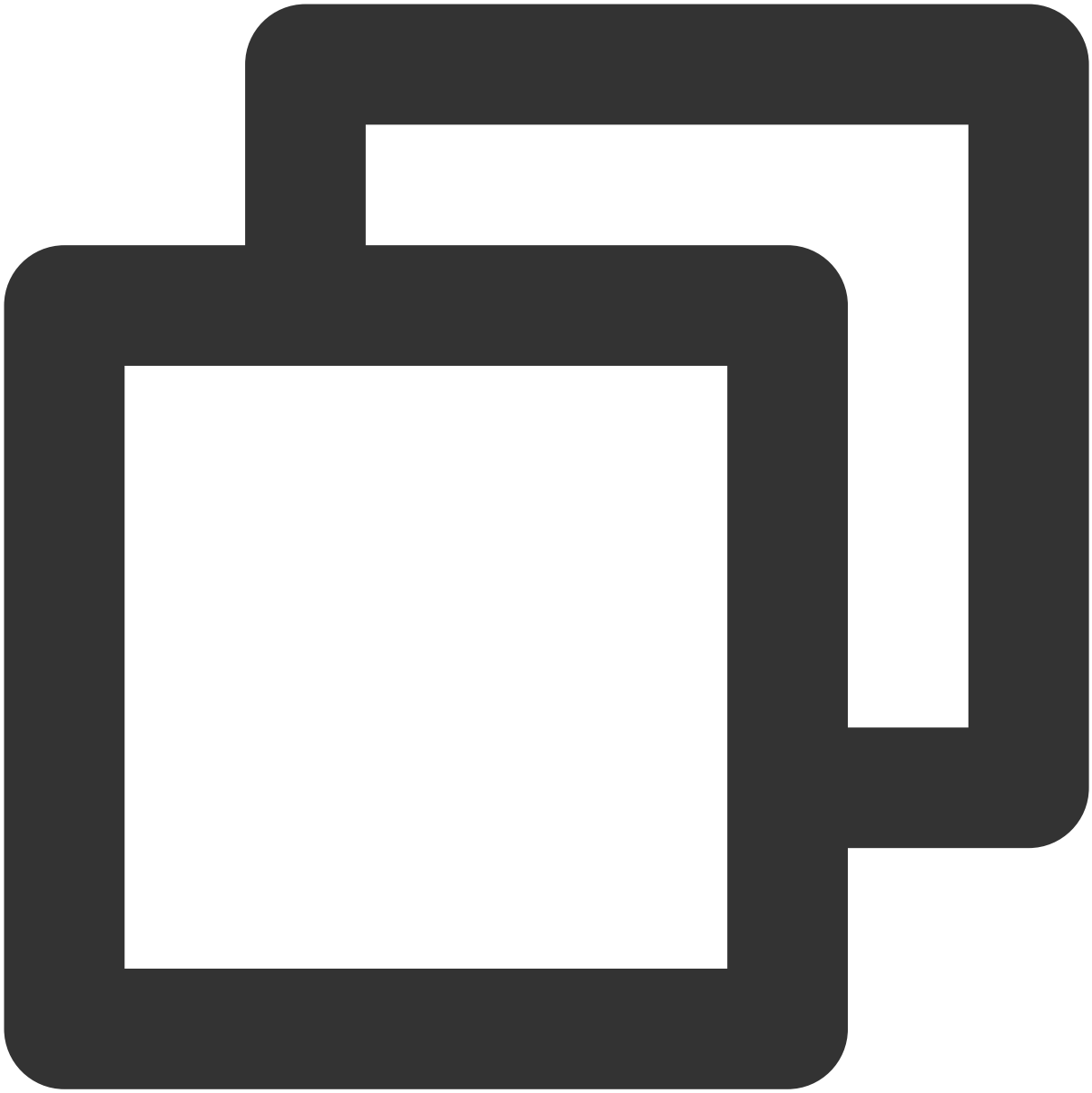
```
| 2023-08-17 15:50:01 |
+---------------------+
1 row in set (0.00 sec)

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)
```

通过历史数据创建表示例



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

插入历史数据至表中示例



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

参数说明

下列表中列举闪回查询功能可配置的参数说明。

参数名	参数	类型	默认值	取值范围	是否	说明

	范围				需重启	
innodb_backquery_enable	全局参数	Boolean	OFF	ON/OFF	否	闪回查询功
innodb_backquery_window	全局参数	Integer	900	1 - 86400	否	支持闪回查 单位：秒， 大，闪回查 据查询时间 undo 表空 间也会上升
innodb_backquery_history_limit	全局参数	Integer	8000000	1 - 9223372036854476000	否	undo 的历 制，超过设 Innodb_ba 触发 purge 长度低于设

性能类特性
计划缓存点查优化

最近更新时间：2024-04-25 11:10:22

功能介绍

TDSQL-C MySQL 数据的 SQL 执行步骤主要包括解析、准备、优化和执行四个阶段。执行计划缓存能力在 prepare statement 模式起作用，prepare statement 模式在 execute 时省略了解析和准备两个阶段，执行计划还会省略优化阶段，将性能进一步提升。

支持版本

内核版本 TDSQL-C MySQL 版8.0 3.1.10及以上。

适用场景

对于线上短小点查询较多，且使用 prepare statement 模式时，应用有性能上的提升。具体性能提升的幅度根据线上业务而定。

使用说明

新增 cdb_plan_cache 开关控制是否打开计划缓存，新增 cdb_plan_cache_stats 开关控制观察缓存命中状态。

参数名	状态	类型	默认	参数值范围	说明
cdb_plan_cache	yes	bool	false	true/false	功能开关，是否打开计划缓存

说明

用户目前无法直接修改以上参数的参数值，如需修改可 提交工单 进行修改。

cdb_plan_cache_stats 参数开关开启后，才可以通过 `show cdb_plan_cache_stat` 命令查看相关数据。

新增 `show cdb_plan_cache_stat` 命令查看计划缓存命中状态，字段意思如下：

字段名	说明
sql	SQL 语句，这里是带有?的 SQL 语句，代表此条 SQL 的执行计划已经被缓存

mode	SQL 缓存的模式，现只支持 prepare 模式
hit	本会话命中的次数

注意：

当 cdb_plan_cache_stats 开关打开时，相当于信息记录，将会对性能产生影响。

支持自增列持久化

最近更新时间：2023-11-01 16:43:51

功能介绍

实现将自增列持久化到数据页中，避免出现自增值重复的问题。

支持版本

内核版本 TDSQL-C MySQL 版 5.7 2.0.23/2.1.9 及以上。

适用场景

不希望自增值出现重复的场景，如历史数据归档。

使用说明

内核默认开启。

Invisible Index

最近更新时间：2024-04-25 11:14:46

功能介绍

许多用户要求能够使索引不可见，以确定是否可以删除它。通过 invisible index 这种方式，用户可以在做出删除该索引的最终决定之前，来查看是否有任何应用程序或数据库用户实际使用它（是否生成/报告了任何错误），该能力从8.0移植至5.7版本中。

支持版本

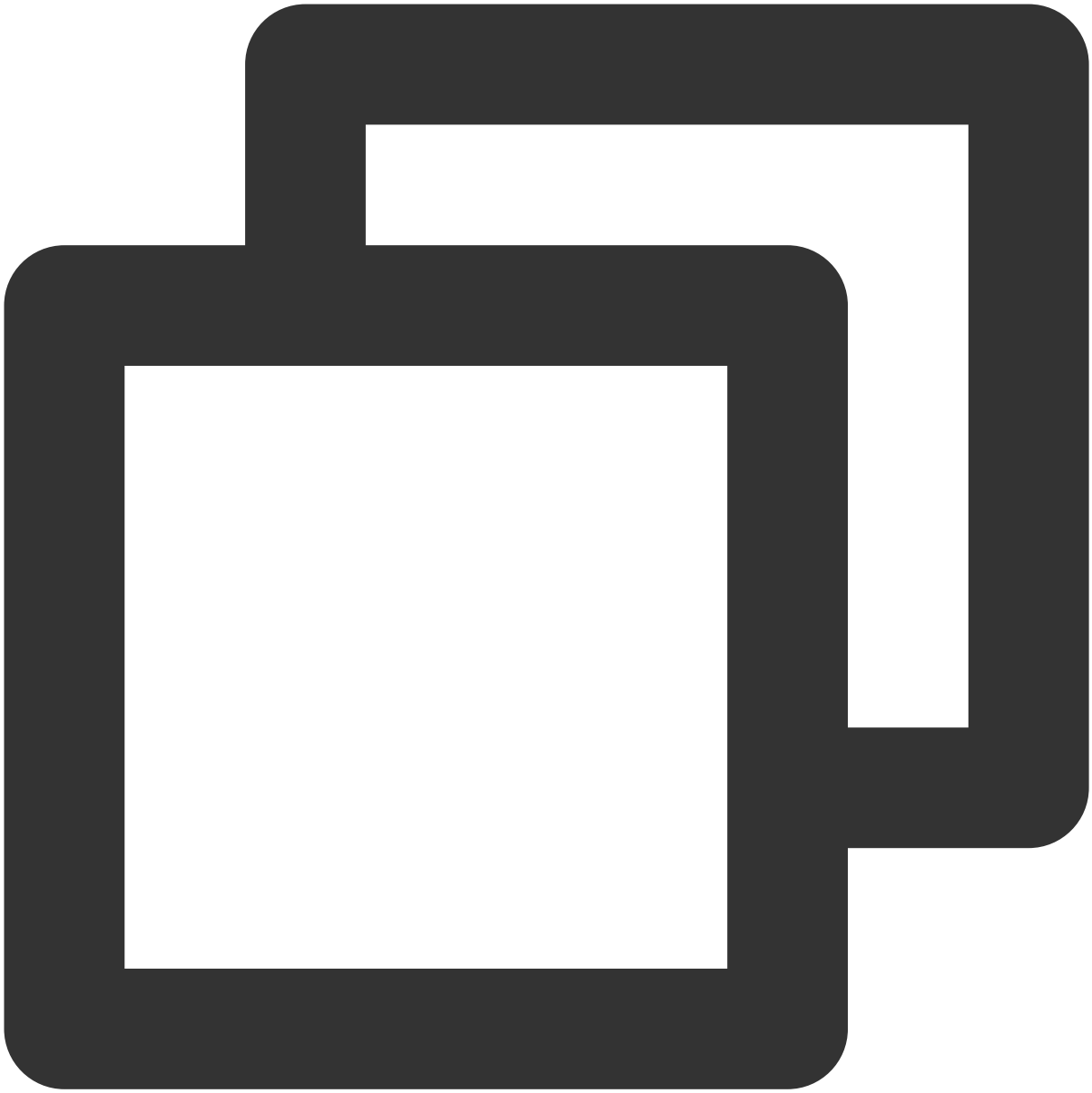
内核版本 TDSQL-C MySQL 版5.7 2.0.23/2.1.9及以上。
内核版本 TDSQL-C MySQL 版8.0 3.1.10及以上。

适用场景

在删除索引前，可以将该索引设置为 invisible，来确认该索引是否有用，这样可以安全地删除该索引。

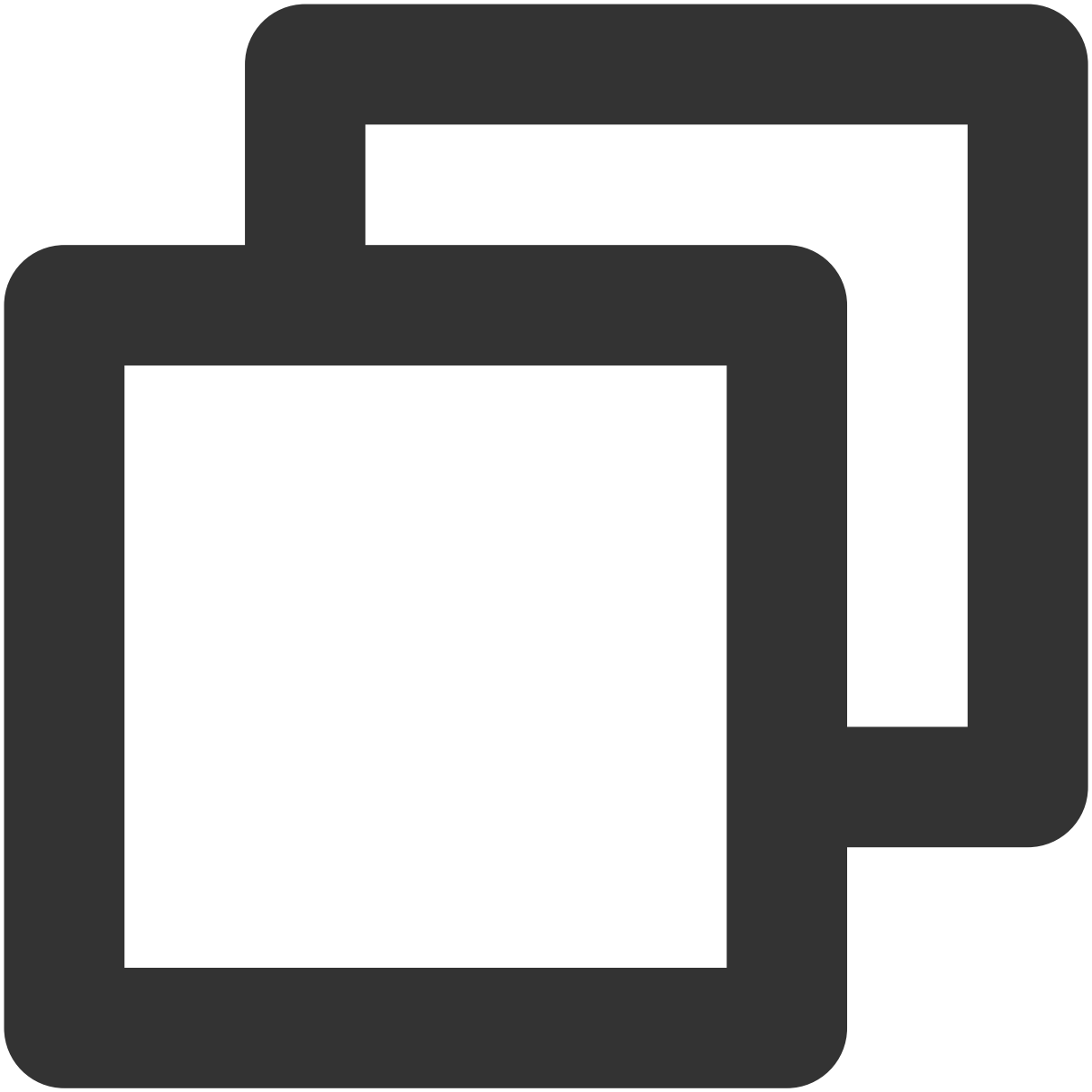
使用说明

可以使用如下语句来创建 invisible 索引和将某个索引改变为 invisible 索引：



```
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;
```

可以使用如下语句将其改变为可见索引：



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

计算下推

最近更新时间：2023-11-01 16:47:15

功能介绍

该功能将单表查询的 LIMIT/OFFSET 或 SUM 操作下推到 InnoDB，有效降低查询时延。
LIMIT/OFFSET 下推到二级索引时，该功能将避免“回表”操作，有效降低扫描代价。
SUM 操作下推到 InnoDB 时，在 InnoDB 层进行计算返回“最终”结果，节省 Server 层和 InnoDB 引擎层多次迭代“每行”记录的代价。

支持版本

内核版本 TDSQL-C MySQL 版 5.7 2.0.23/2.1.9 及以上。

适用场景

该功能主要针对单表查询下存在 LIMIT/OFFSET 或 SUM 的场景，如 “Select *from tbl Limit 10”、“Select* from tbl Limit 10,2”、“Select sum(c1) from tbl” 等语句。
无法优化的场景：
查询语句存在 distinct、group by、having。
存在嵌套子查询。
使用了 FULLTEXT 索引。
存在 order by 并且优化器不能利用 index 实现 order by。
使用多范围的 MRR。
存在 SQL_CALC_FOUND_ROWS。

参数说明

执行 SQL 过程中，根据相应功能控制参数的开关情况，查询优化器自动改写查询计划来完成计算下推的优化。
参数如下：

参数名	动态	类型	默认	参数值范围	说明
cdb_enable_offset_pushdown	Yes	bool	ON	{ON,OFF}	控制 LIMIT/OFFSET 下推，默认开启

cdb_enable_sumagg_pushdown	Yes	bool	OFF	{ON,OFF}	控制 SUM 下推，默认关闭

说明

用户目前无法直接修改以上参数的参数值，如需修改可 提交工单 进行修改。

InnoDB buffer pool 并行初始化

最近更新时间：2024-06-17 16:22:07

功能介绍

加快 buffer pool 初始化速度，降低数据库实例启动耗时。

支持版本

内核版本 TDSQL-C MySQL 版 5.7 2.0.23/2.1.9 及以上。

适用场景

用于提高数据库启动的速度。

性能测试数据

在给定8个 instance 的情况下，性能测试数据：

buffer_pool_size	buffer pool 初始化时间(优化前)	buffer pool 初始化时间(优化后)	速度提升
50GB	2.55s	0.13s	1962%
200GB	10.28s	0.52s	1977%
500GB	25.72s	1.32s	1948%

版权所有：腾讯云计算（北京）有限责任公司

稳定性特性

statement outline

最近更新时间：2024-04-25 11:19:31

功能介绍

SQL 调优是数据库性能优化中非常重要的一环。为了避免优化器无法选择合适的执行计划所带来的影响，TXSQL 提供了 OUTLINE 功能，供用户绑定执行计划。TDSQL-C MySQL 数据库有通过 HINT 来人为绑定执行计划的功能，HINT 信息包含 SQL 采用何种优化规则，执行何种算法，数据扫描采用何种索引等。OUTLINE 主要依靠 HINT 来指定查询计划，我们提供系统表 mysql.outline 让用户添加计划绑定规则，通过开关（cdb_opt_outline_enabled）控制是否开启该功能。

支持版本

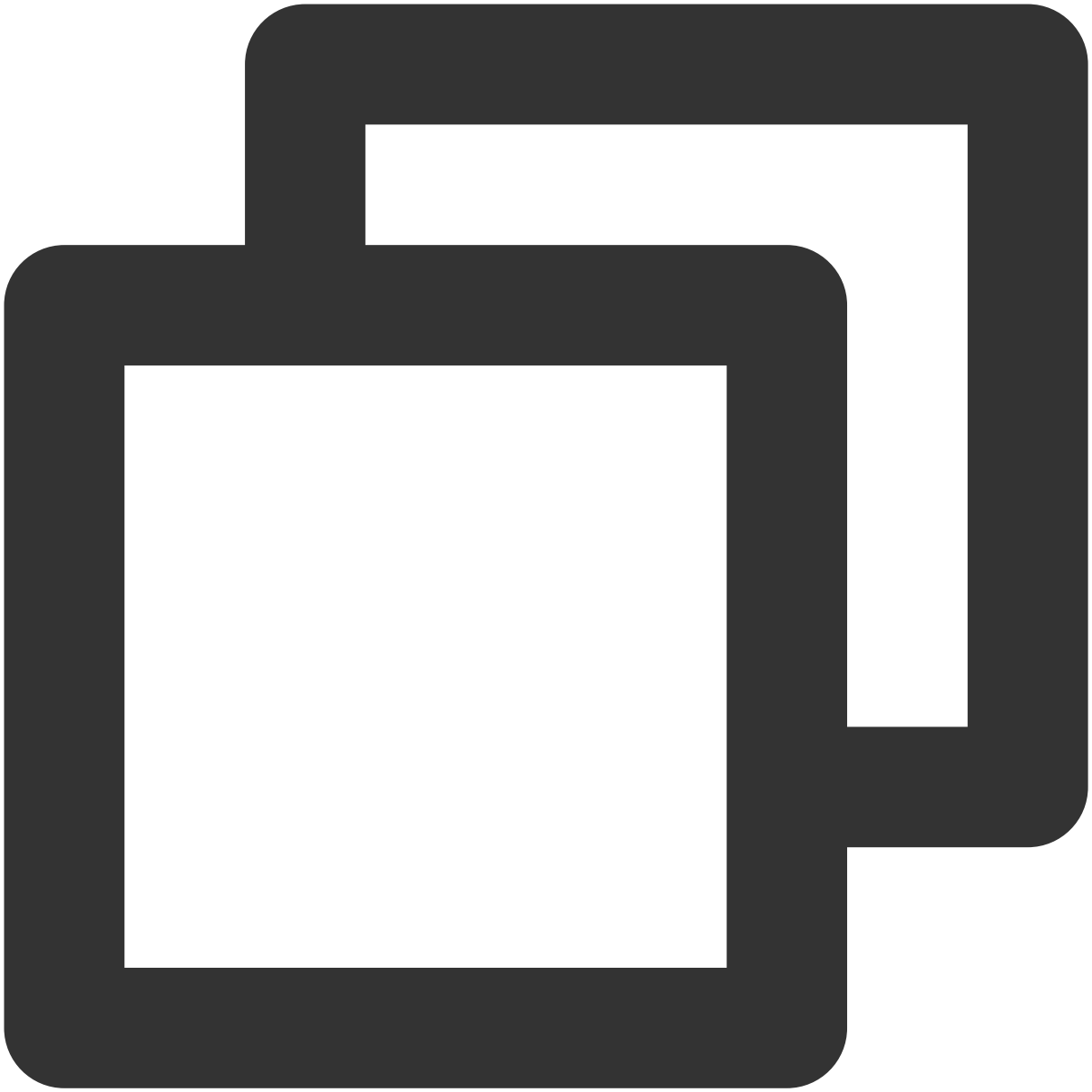
内核版本 TDSQL-C MySQL 版8.0 3.1.10及以上。

适用场景

当线上执行计划走错，如线上执行计划选错索引，但业务又不想修改 SQL 重新发版本解决的场景。

使用说明

OUTLINE 语法设置采用新的语法形式：
设置 OUTLINE 信息：`outline "sql" set outline_info "outline";`
清空 OUTLINE 信息：`outline reset ""; outline reset all;`
刷新 OUTLINE 信息：`outline flush;`
下面介绍 OUTLINE 的主要使用方法，用以下 schema 为例说明：



```
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));
```

参数名	动态	类型	默认	参数值范围	说明
cdb_opt_outline_enabled	yes	bool	false	true/false	是否打开 outline 功能

说明：

用户目前无法直接修改以上参数的参数值，如需修改可 提交工单 进行修改。

绑定 OUTLINE

直接绑定 OUTLINE 的方式是将一条 SQL 替换成另一条，SQL 的语义没有改变，仅是加入了一些 HINT 信息告知优化器如何去执行。

语法形式为： `outline "sql" set outline_info "outline";` ，注意 outline_info 后的字符串应该以 "OUTLINE:" 开头，"OUTLINE:" 之后为加入 HINT 之后的 SQL。如给 `select *from t1, t2 where t1.a = t2.a` 这条 SQL 的 t2 表加上 a 列上的索引。

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

绑定 optimizer hint

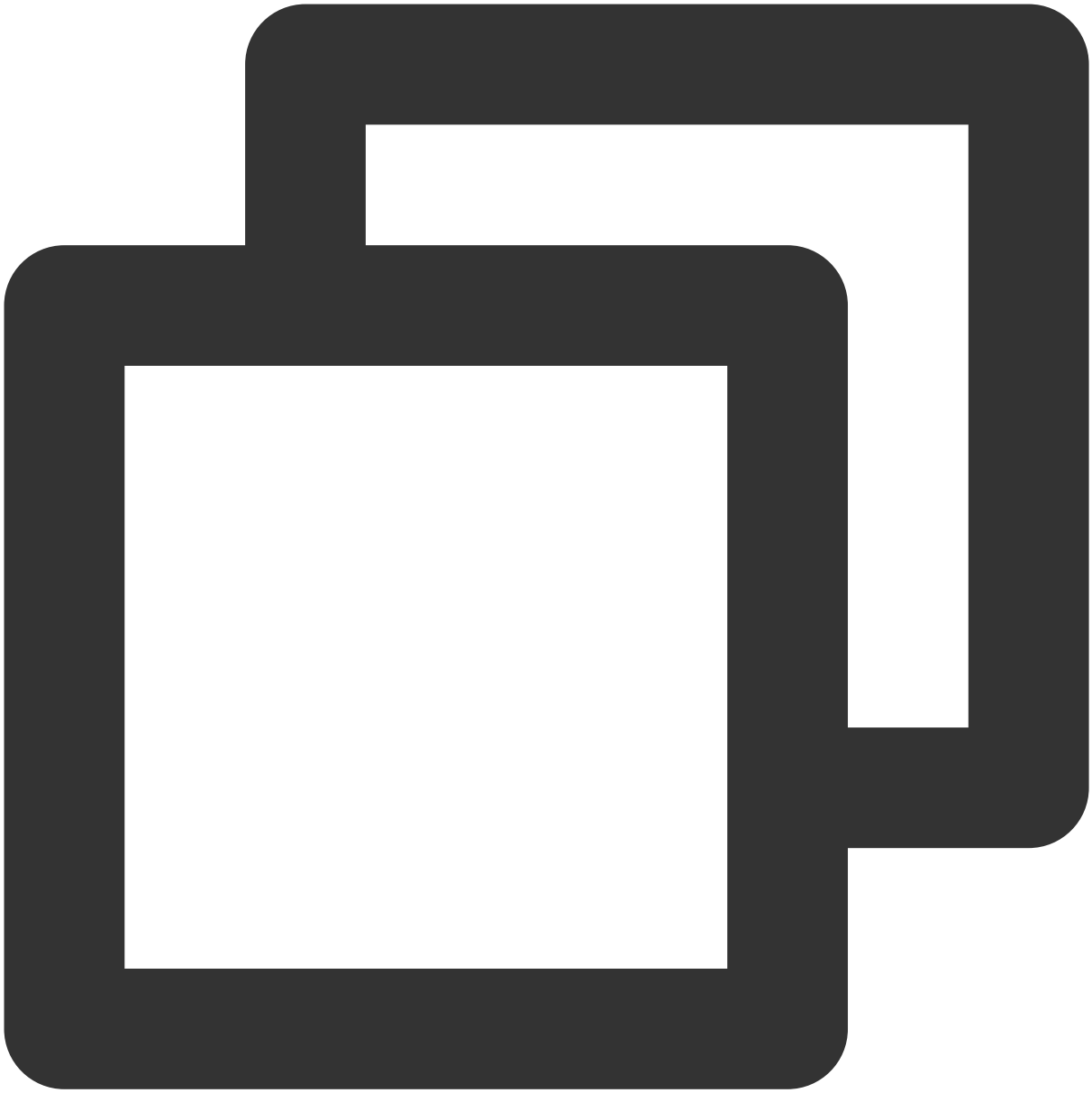
为了功能更加灵活，TXSQL 允许向 SQL 中增量添加 optimizer hint，同样的功能也可以通过直接绑定 outline 实现。语法形式为：`outline "sql" set outline_info "outline";`，注意 outline_info 后的字符串应该以 "OPT:" 开头，"OPT:" 之后为需要加入的 optimizer hint 信息。如给 `select *from t1 where t1.a in (select b from t2)` 这条 SQL 指定 MATERIALIZATION/DUPSWEEDOUT 的 SEMIJOIN。

```
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
```

向原始 SQL 语句中添加 OPTIMIZER 的 HINT，仅支持一次添加一个 HINT，语法上需注意三点：
OPT 关键字应该紧随在"之后。
需要绑定的新语句前必须是':'。
需要添加两个字段（query block 的号码 #optimizer hint 的字符串），中间必须用#分割（`ie. "OPT:1#max_execution_time(1000)"`）。

绑定 index hint

为了功能更加灵活，TXSQL 允许向 SQL 中增量添加 index hint，同样的功能也可以通过直接绑定 outline 实现。
语法形式为：`outline "sql" set outline_info "outline";`，注意 outline_info 后的字符串应该以 "INDEX:" 开头，"INDEX:" 之后为需要加入的 index hint 信息。
下面举个例子：给 `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))` 这条 SQL 的 query block 3 上数据库 test 下 t1 表增加 USE INDEX 的索引 idx1，类型为 FOR JOIN。



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

向原始 SQL 语句中添加 INDEX 的 HINT，仅支持一次添加一个 HINT，语法上注意四点：

INDEX 关键字应该紧随"之后。

需要绑定的新语句前必须是':'。

需要添加五个字段（query block 的号码 #db_name#table_name#index_name#index_type#clause）。

其中 index_type 还有三个值（0为 INDEX_HINT_IGNORE、1为 INDEX_HINT_USE、2为 INDEX_HINT_FORCE），其中 clause 有三个值（1为 FOR JOIN、2为 FOR ORDER BY、3为 FOR GROUP

BY），中间必须用#分割（`ie. "INDEX:2#test#t2#idx2#1#1"` 表示将第2个 query block 中的 test.t2 表中绑定类型为 USE INDEX FOR JOIN 的 idx1 索引）。

删除某条 SQL 对应的 OUTLINE 信息

TXSQL 允许用户删除某一条 SQL 语句的 OUTLINE 绑定信息。
语法为：`outline reset "sql";`，如将 `select *from t1, t2 where t1.a = t2.a` 的 outline 信息删除：`outline reset "select* from t1, t2 where t1.a = t2.a";`。

清空所有 OUTLINE 信息

TXSQL 允许用户删除内核中所有 OUTLINE 绑定信息。语法为：`outline reset all`，执行语句为：`outline reset all;`。
线上业务中有时候会有一些非常特定的问题，需要强制绑定索引，这里可以直接设置 OUTLINE 去绑定。
需要分析设置 OUTLINE 后可能导致的性能回退，在可接受的性能回退范围下做绑定，必要时和内核人员商议。

热点更新保护

最近更新时间：2024-04-25 11:21:43

功能介绍

针对于频繁更新或秒杀类等业务场景，大幅度优化对于热点行数据的 update 操作性能。当开启热点更新自动探测时，系统会自动探测是否有单行的热点更新，如果有，则会让大量的并发 update 排队执行，以减少由于大量行锁造成的并发性能下降。

支持版本

内核版本 TDSQL-C MySQL 版5.7 2.0.23/2.1.9及以上。
内核版本 TDSQL-C MySQL 版8.0 3.1.5及以上。

适用场景

适用于单点或多点带主键的更新压力非常大的场景（秒杀场景）。

参数说明

参数名	动态	类型	默认	参数值范围	说明
cdb_sql_filter_enable	yes	bool	off	on/off	是否打开热点更新功能

使用说明

热点更新保护