云数据库 MySQL

自研内核 TXSQL

产品文档

腾讯云

文档目录

自研内核 TXSQL
TXSQL 内核概述

最近更新时间：2023-04-10 16:09:48

TXSQL 是腾讯云数据库团队维护的 MySQL 内核分支，100%兼容原生 MySQL 版本，TXSQL 提供了类似于 MySQL 企业版的诸多功能，如企业级透明数据加密、审计、动态线程池、加密函数、备份恢复、并行查询等功能。

TXSQL 不仅对 InnoDB 存储引擎、查询优化、复制性能等方面进行了大量优化，同时提升了云数据库 MySQL 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、审计等高级特性。

有关 TXSQL 内核的更多信息：

- 云数据库 MySQL 内核版本更新动态，请参见 内核版本更新动态。
- 云数据库 MySQL 支持自动或手动升级内核小版本，请参见 升级内核小版本。
- 云数据库 MySQL 支持通过云服务器，登录到 MySQL 实例查看内核小版本，请参见 查看内核小版本。

内核版本更新动态
TXSQL 内核更新动态

最近更新时间：2023-07-05 16:42:03

本文为您介绍 TXSQL 的内核版本更新说明。

> 说明：
>
> - 如何升级 MySQL 实例的内核小版本，请参见 升级内核小版本。
> - 升级小版本时，可能会存在部分小版本维护中，无法选取的情况，请以控制台可选小版本为准。

- MySQL 8.0 内核版本更新说明
- MySQL 5.7 内核版本更新说明
- MySQL 5.6 内核版本更新说明

小版本	说明
20220831	• 新特性 ◦ 支持动态设置 MySQL 版本。 ◦ 透明列加密。建表对 varchar 字段指定 encryption 属性，存储侧会对该列进行加密。该能力预计在2023年进行产品化。 ◦ 解决使用第三方数据订阅工具时，因订阅到内部数据一致性对比 SQL 导致数据订阅工具异常的问题。 说明 数据库实例在迁移、升级或故障重建后系统会进行数据一致性对比以确保数据的一致性，数据库对比 SQL 为 `statement` 模式，在部分第三方订阅工具中对 `statement` 模式 SQL 的处理容易出现异常。升级至该版本内核后，第三方订阅工具不会订阅到内部数据一致性对比的 SQL。 ◦ 支持 DDL 操作加 NO_WAIT | WAIT [n] 功能。使 DDL 能够在无法获得 MDL 锁且必须等待之前立即回滚；或者，如果在等待 MDL 锁上花费了指定的时间，则回滚 DDL。 ◦ 支持 Fast Query Cache 功能。针对场景是读多写少。如果写多读少，数据的更新非常频繁，或者查询的结果集非常大，不建议开启。 ◦ 支持 mts 死锁检测增强功能。 ◦ 支持 并行查询。开启并行查询功能，就可以自动识别大查询，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。

小版本	说明
	• 性能优化 ◦ 优化事务系统取快照开销。采用了 Copy Free Snapshot 方式，事务延迟从全局活跃事务 hash 中删除，取快照优化为确定取快照事件的逻辑时间戳。sysbench 测试 read-write 场景极限性能提升11%。 ◦ 优化 prepared statement 过程中的权限校验。全局用一个变量表示权限版本号，prepared statement 在 prepare 完成后记录版本号，执行的时候对比权限版本号是否发生变化，如果没有权限变更，跳过权限检查，否则重新检查权限并记录版本号。 ◦ 优化线程池中时间获取精度。 ◦ 优化记录 offset 获取。为每个 index 缓存一份记录的 offset，当满足条件时，直接使用缓存的 offset 从而省去调用 rec_get_offsets() 函数的计算开销。 ◦ 优化 Parallel DDL。 1. 在索引字段较小时，减少采样内存大小，从而减少采样的频率。 2. 通过 K 路归并来进行排序，能够有效减少归并排序的轮数，从而减少 IO 的次数。 3. 读取记录时将定长 offset 缓存，从而避免每次为每条记录生成一次 offsets。 ◦ 优化 undo log 信息记录逻辑，提升 insert 性能。 ◦ 优化半同步开启时的性能，提升启用半同步时的性能。 ◦ 审计性能优化，降低系统开销。 • bug 修复 ◦ 修复 Thread_memory 有时显示值异常的问题。 ◦ 修复审计批量语句场景中 timestamp 不准确的问题。 ◦ 修复秒级修改列相关问题。 ◦ 修复 CREATE TABLE t1 AS SELECT ST_POINTFROMGEOHASH("0123", 4326); 语句导致主从中断的问题。 ◦ 修复表级别并行时 slave 重试异常的问题。 ◦ 修复执行 show slave hosts 出现 Malformed packet 报错的问题。 ◦ 修复回收站相关问题。 ◦ 修复在 ARM 机型下，jemalloc 分配机制易触发 OOM 的问题。 ◦ 修复 truncate pfs accout table 导致统计失效的问题。 ◦ 修复回收站有外键约束时先恢复子表再恢复父表出现异常的情况。 ◦ 修复日志中有关 sql_mode 日志的问题。 ◦ 修复低概率执行 CREATE DEFINER 存储过程异常问题。 ◦ 修复官方 Copy Free Snapshot 相关问题。 ◦ 修复线程池性能抖动问题。 ◦ 修复 hash join+union 结果可能为空的问题。 ◦ 修复内存相关问题。
20220401	• bug 修复 ◦ 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有可能引发 crash 的问题。

小版本	说明
20220331	• bug 修复 ◦ 修复线程池中野指针被解引用导致 crash 的问题。

20220330	• 新特性
	  ◦ 默认打开 writeset 并行复制。
	  ◦ 支持扩展资源组，可对 IO、内存使用占比以及 SQL 超时策略以用户为单位进行控制。
	  ◦ 支持闪回查询能力，可以查询 UNDO 时间范围内的任意时间点数据。
	  ◦ 支持 delete/insert/replace/update 的 returning 语法，可以返回该 statement 所操作的数据行。
	  ◦ 支持 row 模式 gtid 复制功能扩展。
	  ◦ 支持事务锁优化功能。
	  ◦ 回收站增强，支持 truncate table 和自动清理回收站中的表。
	  ◦ 支持 parallel ddl 能力，通过三阶段并行的方式加速需要创建索引的 DDL 操作。
	  ◦ 支持快速修改索引列功能。
	  ◦ 支持自动统计信息收集和跨机统计信息收集。
	• 性能优化
	  ◦ 优化关闭 binlog_order_commits 时事务提交时 gtid 的锁冲突。
	  ◦ 通过将 InnoDB 启动阶段将单线程创建 rsegs 更改为多线程创建，优化 MySQL 启动时间。
	• bug 修复
	  ◦ 修复死锁或被锁等待后连接断开事务不结束的问题。
	  ◦ 修复 innodb_row_lock_current_waits 等统计值异常的问题。
	  ◦ 修复审计插件没有 use database下 sql type 错误的问题。
	  ◦ 修复大表异步删除功能中，小于 innodb_async_table_size 的表也会被 rename 的问题。
	  ◦ 修复审计插件转义字符错误的出问题。
	  ◦ 修复快速修改列后回滚的问题。
	  ◦ 修复 trx_sys close 时带 xa 场景可能出现 crash 的问题。
	  ◦ 修复 merge derived table 的时候出现 crash的问题。
	  ◦ 修复 writeset 开启后修改 binlog_format 的问题。
	  ◦ 修复 hash scan 对同一行进行 A->B->A->C 更新时1032问题。
	  ◦ 修复 Prepared Statement 模式下排序索引可能失效的问题。
	  ◦ 修复消费物化结果的算子可能被并入物化算子返值路径，导致的执行计划理解和显示问题。
	  ◦ 大表异步删除修复极端情况下的异常行为。
	  ◦ 修复设置 sql filter 时错误信息提示异常的问题。
	  ◦ 修复解析存储过程语法报错问题。
	  ◦ 修复不能应用历史直方图问题。
	  ◦ 修复 SHOW SLAVE HOSTS(show replicas) 显示 Role 列显示带来的兼容性问题。
	  ◦ 修复 Item_in_subselect::single_value_transformer 在列数目出错的情况下，会 crash 的问题。
	  ◦ 修复子表存在 virtual column 和外键列，级联更新的过程中存在内存泄漏导致实例 crash 问题。

小版本	说明
20211202	• 新特性 ◦ 支持快速修改列功能。 ◦ 支持直方图历史版本能力。 ◦ 支持 SQL2003 TABLESAMPLE （单表）采样控制语法，用于获取物理表的随机样本。 ◦ 新增非保留关键字:TABLESAMPLE BERNOULLI。 ◦ 新增 HISTOGRAM() 函数，对于给定输入字段构建直方图。 ◦ 支持 compressed 直方图。 ◦ 支持 SQL 限流功能。 ◦ 支持 MySQL 集群角色设置功能，默认为角色为 CDB_ROLE_UNKNOWN。 ◦ show replicas 命令展示结果新增Role列，用于展示角色。 ◦ 支持 proxy。 • 性能优化 ◦ 优化了由 insert on duplicate key update 引发热点的更新问题。 ◦ 通过聚合 event 多个相同的 binlog event 来提升 hash scan 的应用速度。 ◦ 在 plan cache 打开的情况下，线程池模式下，prepare 语句在点查的内存大量减小。 • bug 修复 ◦ 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而导致的执行计划改变导致的性能问题。 ◦ 修复 query 长时间处于 query end 状态的 bug。 ◦ 修复长记录下，统计信息被严重低估的 bug。 ◦ 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 bug。 ◦ 修复 json_table 函数列名称大小写敏感的问题。 ◦ 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。 ◦ 修复主从 bp 同步在极端情况下（宿主机文件系统损坏的情况）泄漏文件句柄的问题。 ◦ 修复 index mapping 问题。 ◦ 修复统计信息缓存同步问题。 ◦ 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。

小版本	说明

20210830	• 新特性
	  ◦ 支持预加载行数限制功能。
	  ◦ 支持计划缓存点查优化功能。
	  ◦ 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。
	• bug 修复
	  ◦ 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。

20210330	• 新特性
	  ◦ 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 FAST DDL 功能。
	  ◦ 支持用户侧查询 character_set_client_handshake 参数显示当前值功能。
	• 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。
	• 官方 bug 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。

小版本	说明

20201230	• 新特性
	  ◦ 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。
	• 性能优化
	  ◦ 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。
	  ◦ 审计性能优化。
	• 官方 bug 修复
	  ◦ 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。

20200630	• 新特性
	  ◦ 支持异步删除大表：异步、缓慢地清理文件，进而避免因删除大表导致业务性能出现抖动情况，该功能需 提交工单 申请开通。
	  ◦ 支持自动 kill 空闲任务，减少资源冲突，该功能需 提交工单 申请开通。
	  ◦ 支持透明数据加密功能。
	• 官方 bug 修复
	  ◦ 修复由于 relay_log_pos & master_log_pos 位点不一致导致切换失败的问题。
	  ◦ 修复异步落盘所引起的数据文件出错的问题。
	  ◦ 修复 fsync 返回 EIO，反复尝试陷入死循环的问题。
	  ◦ 修复全文索引中，词组查找（phrase search）在多字节字符集下存在的崩溃问题。

TXRocks 内核更新动态

最近更新时间：2023-09-13 15:41:49

本文为您介绍 TXRocks 的内核版本更新说明。

说明

如何升级 MySQL 实例的内核小版本，请参见 升级内核小版本。

升级小版本时，可能会存在部分小版本维护中，无法选取的情况，请以控制台可选小版本为准。

MySQL 8.0 内核版本更新说明

MySQL 5.7 内核版本更新说明

小版本	说明
20230401	新特性： 默认将建表语句中 TokuDB 引擎转为 RocksDB 引擎。

小版本	说明
20230401	新特性： 默认将建表语句中 TokuDB 引擎转为 RocksDB 引擎。

数据库代理内核更新动态

最近更新时间：2023-10-10 10:35:07

本文介绍云数据库 MySQL 数据库代理的内核版本更新说明。

> 说明：
> 如不满足云数据库 MySQL 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。

数据库代理版本	MySQL 内核版本要求	说明
1.3.7	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211202及以上	**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。
1.3.4	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211202及以上	**问题修复** 修复了 show processlist 返回数据不全的问题。
1.3.3	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211202及以上	**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。

数据库代理版本	MySQL 内核版本要求	说明
1.3.2	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211202及以上	**问题修复** 修复了 execute 语句没有参数类型的问题。
1.3.1	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211202及以上	**功能更新** • 当数据库代理下所有有效实例的权重均为0时，权重为0的实例也会分担读请求。 • 支持多可用区部署架构，可挂载跨可用区只读实例。 • 提供只读模式。 • 支持事务拆分能力。 • 支持防闪断的功能，即连接保持，在计划内任务导致的数据库实例 HA 切换，客户端连接不断开。
1.2.1	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211202及以上	**功能更新** • 支持了 db lower_case_table_names 参数，默认不校验大小写。 • 数据库代理建连阶段发生错误时在查询阶段会进行报错信息返回。
1.1.3	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211202及以上	**功能更新** 支持了在 MySQL 预处理的 COM_PREPARE 报文中使用 hint 路由信息，在 PREPARE 中使用 hint 指定路由目标后，后续的 execute 报文将会发送到指定的后端节点上。 **问题修复** • 数据库代理上的主实例进行主从切换后，前端连接立即重置。 • 修复了只读实例超过延迟阈值后负载均衡可能失效的问题。现在当只读实例延迟回落到阈值以内后，路由会正常恢复。 • 修复了对于 MySQL 8.0 可能返回错误握手信息导致建连失败的问题。

数据库代理版本	MySQL 内核版本要求	说明
1.1.2	• MySQL 5.7 20211030及以上 • MySQL 8.0 20211130及以上	**功能更新** • 支持 MySQL 8.0 版本。 • 支持连接级连接池功能，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。 • 支持了只读实例的重连功能。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，数据库代理将自动重新对只读实例建连恢复路由。 • 更新了内部内存管理机制，新版本将会有更低的内存消耗。 **问题修复** • 修复了后端连接超时断开后，客户端连接仍未断开的问题。 • 修复了内部缓存可能会导致内存过快增长的问题。 • 修复了小概率情况下可能会返回格式不正确报文的问题。
1.0.1	MySQL 5.7 20201230及以上	**功能更新** • 支持 MySQL 5.7 版本。 • 支持读写分离。 • 支持读写分离下的读权重配置。 • 支持主从复制延迟阈值设置，只读实例延迟达到阈值以上后将从路由中剔除，延迟回落到阈值以下后恢复路由。如果主从复制中断则直接剔除中断的只读实例。 • 支持最小保留数设置，当发生只读实例剔除时，如果设置了最小保留数为 n，则至少会保留 n 个只读实例在路由里。 • 支持故障转移设置，默认开启。如果故障转移关闭，且主实例读权重为0，则所有只读实例均剔除后，读请求会报错。如果故障转移开启，或主库权重不为0，则会将请求路由到主实例。 • 支持 hint 语法指定路由节点。

功能类特性
自动 kill 空闲事务

最近更新时间：2023-07-06 16:37:44

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 MySQL 5.6 20180915 及以上
- 内核版本 MySQL 5.7 20180918 及以上
- 内核版本 MySQL 8.0 20200630 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，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

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。

并行复制

最近更新时间：2023-07-06 16:36:49

功能介绍

官方 MySQL 5.6 以下的版本在 slave 节点进行回放，master 节点同步 binlog 时，均为单线程模式，5.6 及之后的版本变更为并行模式。但官方的并行是基于 database 和 logical clock，并行粒度太大，导致很多情况下并行效果不理想。

腾讯云 TXSQL 内核团队针对并行复制功能进行了优化，支持按 table 并行，相当于将其粒度拆分至表，提升了并行度，从而减少主从延迟。

支持版本

- 内核版本 MySQL 8.0 20201230 及以上
- 内核版本 MySQL 5.7 20180530 及以上
- 内核版本 MySQL 5.6 20170830 及以上

适用场景

该功能主要针对部分负载能提升 slave 机重放 binlog 速度，减少主从的 delay。

使用说明

MySQL 5.6、5.7 版本可通过将参数 slave_parallel_type 设置为新增加的值 TABLE 来打开这个功能，MySQL 8.0 版本不支持 TABLE 模式。
另外 information_schema 下新增加了状态表 cdb_slave_thread_status，用以展示状态信息。

- MySQL 5.6 版本相关参数说明
- MySQL 5.7 版本相关参数说明
- MySQL 8.0 版本相关参数说明

参数名	动态	类型	默认	参数值范围	说明

参数名	动态	类型	默认	参数值范围	说明
slave_parallel_type	YES	char*	SCHEMA	SCHEMA/TABLE	从机并行复制级别： • SCHEMA 为对象级别复制，不同对象的复制事件可以并行执行。 • TABLE 为表级别复制，不同表的复制事件可以并行执行。

动态线程池

最近更新时间：2024-02-18 11:50:38

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，云数据库 MySQL 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 MySQL 8.0 20201230 及以上。
内核版本 MySQL 5.7 20201230 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。

相关参数状态说明

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	高稳定参数模板：10 高性能参数模板：16	[3,32]	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

支持 NOWAIT 语法

最近更新时间：2022-12-06 11:52:46

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。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，但某些场景，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 MySQL 5.7 20171130 及以上，DDL 语句支持 NO_WAIT 和 WAIT 选项。
- 内核版本 MySQL 5.7 20200630 及以上（该功能从 MySQL 8.0 移植，因此8.0版本原生支持），SELECT FOR UPDATE 语句支持 NOWAIT 和 SKIP LOCKED 选项。

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。

使用说明

SELECT FOR UPDATE NOWAIT/SKIP LOCKED

```
###############session 1###############
MySQL [test]> create table t1(seat_id int, state int, primary key(seat_id)) engin
e=innodb;
Query OK, 0 rows affected (0.03 sec)
```

```
MySQL [test]> INSERT INTO t1 VALUES(1,0), (2,0), (3,0), (4,0);
Query OK, 4 rows affected (0.01 sec)
Records: 4 Duplicates: 0 Warnings: 0

MySQL [test]> begin;
Query OK, 0 rows affected (0.01 sec)

MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR SHARE;
+---------+-------+
| seat_id | state |
+---------+-------+
| 1 | 0 |
| 2 | 0 |
+---------+-------+
2 rows in set (0.00 sec)

###############session 2###############
MySQL [test]> SET SESSION innodb_lock_wait_timeout=1;
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR UPDATE;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR UPDATE NOWAIT;
ERROR 5010 (HY000): Do not wait for lock.
MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR UPDATE SKIP LOCKED;
+---------+-------+
| seat_id | state |
+---------+-------+
| 3 | 0 |
| 4 | 0 |
+---------+-------+
2 rows in set (0.00 sec)

MySQL [test]> SELECT * FROM t1 WHERE seat_id > 0 LIMIT 2 FOR UPDATE NOWAIT;
ERROR 5010 (HY000): Do not wait for lock.
MySQL [test]> SELECT * FROM t1 WHERE seat_id > 0 LIMIT 2 FOR UPDATE SKIP LOCKED;
+---------+-------+
| seat_id | state |
+---------+-------+
| 3 | 0 |
| 4 | 0 |
+---------+-------+
2 rows in set (0.00 sec)

MySQL [test]> commit;
Query OK, 0 rows affected (0.00 sec)
```

SELECT FOR SHARE NOWAIT/SKIP LOCKED

```
###############session 1###############
MySQL [test]> begin;
Query OK, 0 rows affected (0.01 sec)

MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR UPDATE;
+---------+-------+
| seat_id | state |
+---------+-------+
| 1 | 0 |
| 2 | 0 |
+---------+-------+
2 rows in set (0.00 sec)

###############session 2###############
MySQL [test]> SET SESSION innodb_lock_wait_timeout=1;
Query OK, 0 rows affected (0.00 sec)

MySQL [test]> begin;
Query OK, 0 rows affected (0.00 sec)

MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 LOCK IN SHARE MODE;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR SHARE;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR SHARE NOWAIT;
ERROR 5010 (HY000): Do not wait for lock.
MySQL [test]> SELECT * FROM t1 WHERE state = 0 LIMIT 2 FOR SHARE SKIP LOCKED;
+---------+-------+
| seat_id | state |
+---------+-------+
| 3 | 0 |
| 4 | 0 |
+---------+-------+
2 rows in set (0.00 sec)

MySQL [test]> commit;
Query OK, 0 rows affected (0.00 sec)
```

DDL 语句 NO_WAIT 和 WAIT 选项

```
ALTER TABLE `table` [NO_WAIT | WAIT [n]] `command`;
DROP TABLE `table` [NO_WAIT | WAIT [n]];
TRUNCATE TABLE `table` [NO_WAIT | WAIT [n]];
OPTIMIZE TABLE `table` [NO_WAIT | WAIT [n]];
```

```
RENAME TABLE `table_src` [NO_WAIT | WAIT [n]] TO `table_dst`;
CREATE INDEX `index` ON `table.columns` [NO_WAIT | WAIT [n]];
CREATE FULLTEXT INDEX `index` ON `table.columns` [NO_WAIT | WAIT [n]];
CREATE SPATIAL INDEX `index` ON `table.columns` [NO_WAIT | WAIT [n]];
DROP INDEX `index` ON `table` [NO_WAIT | WAIT [n]];
```

支持 returning

最近更新时间：2022-04-22 17:00:48

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：

- 一是在开启事务后在 DML 语句后紧跟一条 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 MySQL 5.7 20210330 及以上

适用场景

在目前 MySQL 5.7 20210330 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. 暂不支持 UPDATE...RETURNING 语句。
3. 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。
4. 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 as alias, (select * from t2);
+------+---------+---------+-------+--------------------+
| 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)
```

存储过程

```
MySQL [test]> CREATE TABLE `t` (id INT);
Query OK, 0 rows affected (0.03 sec)
MySQL [test]> delimiter $$
MySQL [test]> CREATE PROCEDURE test(in param INT)
-> BEGIN
-> INSERT INTO t (id) values (param) returning *;
-> END$$
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> delimiter ;
MySQL [test]> CALL test(100);
+------+
| id |
```

```
+------+
| 100 |
+------+
1 row in set (0.01 sec)
Query OK, 0 rows affected (0.01 sec)
```

列压缩

最近更新时间：2022-08-08 15:27:40

功能介绍

当前有针对行格式的压缩和针对数据页面的压缩，但是这两种压缩方式在处理一个表中的某些大字段和其他很多小字段，同时对小字段的读写很频繁，对大字段访问不频繁的情形中，它在读写访问时都会造成很多不必要的计算资源的浪费。

列压缩功能可以压缩那些访问不频繁的大字段而不压缩那些访问频繁的小字段，此时不仅能够减少整行字段的存储空间，而且可以提高读写访问的效率。

例如，一张员工表：`create table employee(id int, age int, gender boolean, other varchar(1000) primary key (id))`，当对 `id,age,gender` 小字段访问比较频繁，而对 `other` 大字段的访问频率比较低时，可以将 `other` 列创建为压缩列。一般情况下，只有对 `other` 的读写才会触发对该列的压缩和解压，对其他列的访问并不会触发该列的压缩和解压。由此进一步降低了行数据存储的大小，使得对访问频繁的小字段能够实现更快访问，对访问频率比较低的大字段的存储空间能够实现进一步降低。

支持版本

内核版本 MySQL 5.7 20210330 及以上

> 说明：
> 列压缩功能默认为关闭状态，如需使用请 提交工单 开启。

适用场景

表中有某些大字段和其他很多小字段，同时对小字段的读写很频繁，对大字段访问不频繁的情形中，可以将大字段设置为压缩列。

使用说明

支持的数据类型

1. `BLOB`（包含 `TINYBLOB`、`MEDIUMBLOB`、`LONGBLOB`）
2. `TEXT`（包含 `TINYTEXT`、`MEDIUMTEXT`、`LONGTEXT`）
3. `VARCHAR`
4. `VARBINARY`

> 注意：
> 其中 `LONGBLOB` 和 `LONGTEXT` 的长度最大只支持$2^{32}-2$，相比官方 String Type Storage Requirements 支持的$2^{32}-1$少一个字节。

支持的 DDL 语法类型

相对官方的 建表语法，其中 `column_definition` 的 `COLUMN_FORMAT` 定义有所变动。同时列压缩只支持 Innodb 存储引擎类型的表。

```
column_definition:
data_type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT 'string']
[COLLATE collation_name]
[COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}|COMPRESSED=[zlib]] # COMPRESSED 压缩列关键字
[STORAGE {DISK|MEMORY}]
[reference_definition]
```

一个简单的示例如下：

```
CREATE TABLE t1(
id INT PRIMARY KEY,
b BLOB COMPRESSED
);
```

此时省略了压缩算法，默认选择 `zlib` 压缩算法，您也可以显示指定压缩算法关键字，目前只支持 `zlib` 压缩算法。

```
CREATE TABLE t1(
id INT PRIMARY KEY,
b BLOB COMPRESSED=zlib
);
```

支持的 DDL 语法总结如下：

create table 方面：

DDL	是否继承压缩属性
`CREATE TABLE t2 LIKE t1;`	Y
`CREATE TABLE t2 SELECT * FROM t1;`	Y
`CREATE TABLE t2(a BLOB) SELECT * FROM t1;`	N

alter table 方面：

DDL	描述
`ALTER TABLE t1 MODIFY COLUMN a BLOB;`	将压缩列变为非压缩
`ALTER TABLE t1 MODIFY COLUMN a BLOB COMPRESSED;`	将非压缩列变为压缩

新增变量说明

参数名	动态	类型	默认	参数值范围	说明
innodb_column_compression_zlib_wrap	Yes	bool	TRUE	true/false	如果打开，将生成数 zlib 头和 zlib 尾并做 校验
innodb_column_compression_zlib_strategy	Yes	Integer	0	[0,4]	列压缩使用的压缩策 小值为：0，最大值 4分别和 zlib 中的压 Z_DEFAULT_STRA Z_FILTERED、 Z_HUFFMAN_ONL Z_RLE、Z_FIXED 应。 一般来说， Z_DEFAULT_STRA 对于文本数据常是最 Z_RLE 对于图像数 最佳的
innodb_column_compression_zlib_level	Yes	Integer	6	[0,9]	列压缩使用的压缩级 小值：0，最大值： 表不压缩，该值越大 缩后的数据越小，但 间也越长

参数名	动态	类型	默认	参数值范围	说明
innodb_column_compression_threshold	Yes	Integer	256	[0, 0xffffffff]	列压缩使用的压缩阈值为：1，最大值为0xffffffff，单位：字节长度大于或等于该值会被压缩，否则原数不变，只是添加压缩
innodb_column_compression_pct	Yes	Integer	100	[1, 100]	列压缩使用的压缩率值：1，最大值：10位：百分比。只有**压据大小 / 压缩前数据**于该值时，数据才会缩，否则原数据保持只是添加压缩头

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。

新增状态说明

名称	类型	说明
`Innodb_column_compressed`	Integer	列压缩的压缩次数，包括非压缩格式和压缩格式两种状态的压缩
`Innodb_column_decompressed`	Integer	列压缩的解压次数，包括非压缩格式和压缩格式两种状态的解压缩

新增错误说明

名称	范围	说明
`Compressed column '%-.192s' can't be used in key specification`	指定压缩的列名	不能对有索引的列指定压缩属性
`Unknown compression method: %s"`	在 DDL 语句中指定的压缩算法名	在 `create table` 或者 `alter table` 时指定 `zlib` 之外非法的压缩算法

名称	范围	说明
`Compressed column '%-.192s' can't be used in column format specification`	指定压缩的列名	在同一个列中，已经指定 `COLUMN_FORMAT` 属性就不能再指定压缩属性，其中 `COLUMN_FORMAT` 只在 NDB 中被使用
`Alter table ... discard/import tablespace not support column compression`	\	带有列压缩的表不能执行 `Alter table ... discard/import tablespace` 语句

性能

整体性能分为 DDL 和 DML 两方面：
DDL 方面，使用 sysbench 进行测试：

- 列压缩对 COPY 算法的 DDL 有较大的性能影响，压缩后性能表现比之前慢7倍 - 8倍。
- 对于 inplace 的影响则取决于压缩后的数据量大小，如果采用压缩后，整体数据大小有降低，那么 DDL 的性能是有提升；反之，性能会有一定的降幅。
- 对于 instant 来说，列压缩对该类型的 DDL 基本没有影响。

DML 方面：考虑最常见的压缩情形（压缩比1:1.8），此时有8个列的表，表中有一个大的 varchar 类型的列，其插入数据长度在1 - 6000内均匀随机，插入的字符在0 -9 、a - b内随机，其他几个列数据类型为 char(60) 或 int 类型。此时其对非压缩列插入、删除和查询都有10%以内的提升，但对于非压缩列的更新则有10%以内的下降，对于压缩列的更新则有15%以内的性能跌幅。这是因为在更新过程中，MySQL 会先读出该行的值然后再写入该行更新之后的值，整个更新过程会触发一次解压和压缩而插入和查询只会进行一次压缩或者解压。

注意事项

1. 逻辑导出方面，逻辑导出时 create table 还是会附有 Compressed 相关的关键字。因此导入时在云数据库 MySQL 内部是支持的。其他 MySQL 分支以及官方版本：
   - 官方版本号小于5.7.18，可以直接导入。
   - 官方版本号大于或等于5.7.18，需要在逻辑导出之后，去掉压缩关键字。
2. DTS 导出其他云或是用户时，在 binlog 同步过程中可能会出现不兼容的问题，可以跳过带压缩关键字的 DDL 语句。

闪回查询

最近更新时间：2023-02-22 16:13:27

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 MySQL 5.7 20220715 及以上。
- 内核版本 MySQL 8.0 20220331 及以上。

如何查看或升级内核小版本，请参见 升级内核小版本。

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() |
+---------------------+
| 2022-02-17 16:01:01 |
+---------------------+
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 '2022-02-17 16:01:01';
```

```
+------+------+
| 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 '2022-02-17 16:01:01';
```

插入历史数据至表中示例

```
insert into t4 select * from t1 as of timestamp '2022-02-17 16:01:01';
```

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	参数范围	类型	默认值	取值范围	是否需重启
Innodb_backquery_enable	全局参数	Boolean	OFF	ON\OFF	否
Innodb_backquery_window	全局参数	Integer	900	1 - 86400	否
Innodb_backquery_history_limit	全局参数	Integer	8000000	1 - 9223372036854476000	否

性能类特性
并行查询
简介

最近更新时间：2023-03-13 12:12:23

云数据库 MySQL 支持并行查询能力，开启并行查询，可以自动识别大查询，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。

概念

并行查询（Parallel Query，PQ）指利用更多计算资源完成查询工作。传统的查询方法对于较小的数据量（几百 GB）是比较友好的，但随着业务不断发展，很多用户的数据量开始到达了 TB 级别，这已经超过了传统数据库的处理能力，而并行查询正是为了应对这种场景，查询时，在存储层将数据下分到不同的线程上，单个节点内多个线程并行计算，将结果流水线汇总到总线程，最后总线程做简单归并返回给用户，以提高查询效率。

功能背景

云数据库 MySQL 对比于传统 MySQL 数据库在计算性能、存储能力、容灾能力和弹性扩展能力上的痛点问题已经解决并有所突破，但仍然存在以下痛点问题：

- 随着互联网的发展，数据库对存储能力和存储量级的提升逐渐增强，数据量的表单出现的频次越来越高，对于大表查询能力，现有的技术瓶颈导致 SQL 语句响应过慢，影响业务流程。
- 现行的市场环境中，业务上出现越来越多报表统计或者其他分析查询，这些查询虽然不多，但通常要处理比较大的数据量且对查询时间要求很高。一定的数据分析能力，异构数据处理能力开始成为标配能力。

以上两个痛点问题的产生原因主要是：在 MySQL 生态里，各开源发行版只支持传统的单线程查询处理模式，即单条 SQL 处理涉及到的解析、优化和执行等阶段，都是在一个线程（称为用户线程）中完成的，这种技术实现模式无法充分利用现代多核 CPU 与大内存的硬件资源，导致一定程度的资源浪费。

因此，需要简化复杂分析的使用并且提升分析性能，基于同一份数据，调动多核服务于大查询（查询内并行），无疑是查询加速和降本增效的重要措施。

功能优势

- **零成本性能提升**：内核能力升级、无需付费支付额外附加成本，将充分调动您的实例 CPU 计算能力，加快语句响应速度，计算性能大幅提升。
- **常用语句全面支持**：兼容大部分常用 SQL 语句，支撑多种业务场景，保证业务流畅加速。
- **灵活参数设置**：提供多种参数帮助您控制并行查询的启停条件，让查询更智能，灵活适配您的业务场景，无需改造即可使用该能力。

支持的语句场景和受限场景

最近更新时间：2023-03-13 12:12:23

本文介绍并行查询能力支持的语句场景和受限场景。

兼容语句场景

云数据库 MySQL 已经实现了具备如下特征的 SQL 语句的并行查询处理，并在逐渐完善更多的功能场景。

- 对于单表扫描：支持全表扫描、索引扫描、索引范围扫描、索引 REF 查询等扫描类型的正序、逆序扫描。
- 对于多表连接：支持 Nested Loop Join 算法以及 Semi Join、Anti Join、Outer Join 等连接类型。
- 对于子查询：支持 derived table 的并行。
- 对于数据类型：支持带多种数据类型的查询，包括整型数据、字符型数据、浮点型数据、时间类型数据、以及（有运行时大小限制的）溢出类型数据。
- 普通运算符和函数原则上不限。
- 聚合函数支持 COUNT/SUM/AVG/MIN/MAX。
- 支持 UNION/UNION ALL 查询。
- 支持 traditional（默认格式）、json 和 tree 三种 EXPLAIN 格式。

受限场景

云数据库 MySQL 并行查询能力不支持的场景如下。

限制项	限制说明
语句兼容性限制	非查询语句不支持并行查询，包括 INSERT ... SELECT 和 REPLACE ... SELECT。
	stored program 中的查询语句无法并行。
	prepared statement 中的查询语句无法并行。
	串行化隔离级别事务内的查询语句无法并行。
	加锁读语句无法并行，如 select for update/share lock。
	CTE 无法并行。
表/索引兼容性限制	查询表为系统表/临时表/非 Innodb 表时无法并行。
	空间索引无法并行。

限制项	限制说明
	全文索引无法并行。
	分区表无法并行。
	扫描方式为 index_merge 的表无法并行。
表达式/ Field 兼容性限制	包含 Generated Column 、BLOB、TEXT、JSON、BIT 和 GEOMETRY 字段的表无法并行。
	BIT_AND、BIT_OR、BIT_XOR 类型的聚合函数无法并行。
	aggregation(distinct)，如 SUM(DISTINCT)、COUNT(DISTINCT) 等聚合函数无法并行。
	GIS 相关函数（如 SP_WITHIN_FUNC、ST_DISTANCE 等）无法并行。
	用户自定义函数无法并行。
	json 相关的函数无法并行（如 json_length，json_type，JSON_ARRAYAGG 等）。
	XML 相关函数无法并行（xml_str）。
	用户锁相关的函数无法并行（is_free_lock，is_used_lock，release_lock，release_all_locks，get_lock）。
	sleep 函数、random 函数、GROUP_CONCAT 函数、set_user_var 函数、weight_string 函数无法并行。
	部分统计相关函数（STD/STDDEV/STDDEV_POP，VARIANCE/VAR_POP/VAR_SAMP）无法并行。
	子查询无法并行。
	窗口函数无法并行。
	rollup 无法并行。

除了通过 并行查询兼容语句场景 可以查询到语句是否被执行并行查询外，您还可以通过查看并行查询执行计划与查看线程工作状态查询，详情参见 查看并行查询。

开启或关闭并行查询

最近更新时间：2023-04-10 16:09:48

云数据库 MySQL 支持并行查询能力，您可通过控制台或命令行调整相关参数，为实例开启或关闭并行查询功能。

前提条件

数据库版本：MySQL 8.0 内核版本20220831及以上。

参数说明

> 说明：
> 主实例与只读实例均支持开启并行查询功能，但实例 CPU 核数需大于等于4。

您可通过控制台或命令行调整参数 txsql_max_parallel_worker_threads 和 txsql_parallel_degree 不为0，来开启当前实例的并行查询功能。参数的相关信息和具体设置建议如下。

参数信息

参数	变量类型	作用域	默认值	取值范围
txsql_max_parallel_worker_threads	Integer	Global	{MIN(DBInitCpu,0)}	0-{MAX(DBInitCpu-2,2)}

参数	变量类型	作用域	默认值	取值范围
txsql_parallel_degree	Integer	Global/session	4	0 - 64

设置建议

- 并行度规格限制：txsql_parallel_degree 参数的数值表明单条语句并行查询使用的最大线程数，即并行查询的默认并行度，建议并行度不要超过实例 CPU 核数的二分之一。为保证稳定性，CPU 核数小于4的小规格集群将禁用并行查询功能，您将无法在控制台或使用命令行调整并行查询相关参数。
- SQL 语句在执行并行查询时将默认使用 txsql_parallel_degree 所设置的并行度，但用户可通过 hint 语句调整单条 SQL 语句的并行查询并行度，详细说明请参见 hint 语句控制。
- txsql_max_parallel_worker_threads 参数的值表明并行查询中实例可用于并行查询的线程数，txsql_max_parallel_worker_threads / txsql_parallel_degree 的值表明最多同时有多少条 SQL 语句可以执行并行查询。
- txsql_max_parallel_worker_threads 与 txsql_parallel_degree 共同控制并行查询功能的开启与关闭，当任意一个参数设置为0时，表示关闭并行查询功能。

云数据库 MySQL 也提供了多种参数对并行查询的执行条件进行设置，方便您对业务进行个性化适配，保证业务稳定运行。设置后，数据库将会对语句的执行代价，表的行数，单条语句执行并行计划时所使用的内存等条件进行判断，确认每一条 SQL 语句是否允许执行并行查询。相关参数说明如下：

参数	变量类型	作用域	默认值

参数	变量类型	作用域	默认值
innodb_txsql_parallel_partitions_per_worker	Integer	Global/Session	13
txsql_optimizer_context_max_mem_size	Integer	Global/Session	{MIN(DBInitMemory*52429,8388
txsql_parallel_cost_threshold	Integer	Global/Session	50000

参数	变量类型	作用域	默认值
txsql_parallel_exchange_buffer_size	Integer	Global/Session	1048576
txsql_parallel_table_record_threshold	Integer	Global/Session	5000

注意：

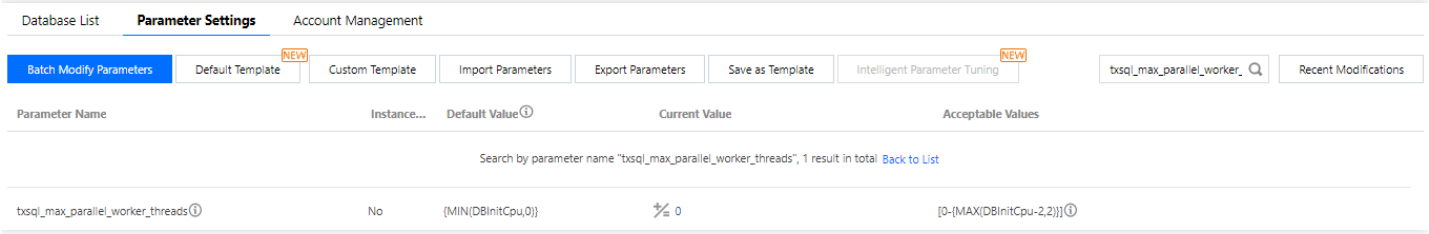
- 并行查询等所有参数均无需重启实例，设置后可即时生效。
- 标有 session 作用域的参数表明该参数支持对单条语句进行设置。

通过控制台开启或关闭并行查询

您可通过 MySQL 控制台进入实例参数设置页面通过设置相关参数开启或关闭功能。

- 设置 txsql_max_parallel_worker_threads 和 txsql_parallel_degree 不为0表示开启并行查询能力。
- 设置 txsql_max_parallel_worker_threads 和 txsql_parallel_degree 任意一个为0表示关闭并行查询能力。

也可设置相关执行条件参数。控制台参数设置页面如下图所示。详细操作方法请参考 设置实例参数。



通过 hint 语句对单条 SQL 语句指定并行执行方式

云数据库 MySQL 支持对单条 SQL 语句进行指定并行执行方式。对单条 SQL 语句进行设置时，将采用 hint 语句的方式进行操作，详细方法请参照 hint 语句控制。

相关文档

- 查看并行查询
- hint 语句控制

hint 语句控制

最近更新时间：2023-03-13 12:12:23

云数据库 MySQL 支持通过调整相关参数开启或关闭并行查询功能，通过控制台可实现对整个 SQL 语句开启或关闭并行查询能力、设置执行条件参数，也支持使用 hint 语句对单条 SQL 语句进行指定并行执行方式。

> 说明：
> hint 语句可以指定 SQL 语句是否执行，并对指定 SQL 语句可以应用 session 级参数。hint 语句同时支持查询指定的并行表。

hint 语句使用范例

功能	命令行	说明
开启并行查询	`SELECT /*+PARALLEL(x)*/ ... FROM ...;`	x 需大于0，x 表示该条 SQL 语句所使用的并行查询并行度。
关闭并行查询	`SELECT /*+PARALLEL(x)*/ ... FROM ...;`	x 设置为0，表示关闭并行查询能力。
指定并行表	可通过以下两种方式指定允许哪些表执行或不执行并行查询计划： • 通过 PARALLEL 可指定表执行并行查询计划 `SELECT /*+PARALLEL(t)*/ ... FROM ...;` • 通过 NO_PARALLEL 可以指定表禁止执行并行查询计划 `SELECT /*+NO_PARALLEL(t)*/ ... FROM ...;`	t 为表的名称。
同时指定并行表与并行查询并行度	`SELECT /*+PARALLEL(tx)*/ * ... FROM ...;`	x 需大于0，x 表示该条 SQL 语句所使用的并行查询并行度，t 为表的名称。
通过 hint 语句设置 session 级参数，仅对指定 SQL 语句生效	`SELECT /*+SET_VAR(var=n)*/ * ... FROM ...;`	var 为支持 session 作用域的并行查询参数。

hint 语句使用场景示例

场景一：

```
select /*+PARALLEL () */ * FROM t1, t2;
```

强制并行度为 txsql_parallel_degree 所设置的数值（默认并行度）执行并行查询，当语句不符合并行查询执行条件时，将回退为串行查询。

场景二：

```
select /*+PARALLEL (4) */ * FROM t1, t2;
```

无论系统默认并行度数值为多少，强制该条语句使用并行度为4执行并行查询，设置该条语句的 txsql_parallel_degree = 4，当语句不符合并行查询执行条件时，将回退为串行查询。

场景三：

```
select /*+PARALLEL (t1) */ * FROM t1, t2;
```

选择 t1 表执行并行查询，并行度为系统默认并行度，当 t1 表小于 txsql_parallel_table_record_threshold 所设置的值时，将回退为串行查询。

场景四：

```
select /*+PARALLEL (t1 8) */ * FROM t1, t2;
```

选择 t1 表执行并行查询，并行度为8，当 t1 表小于 txsql_parallel_table_record_threshold 所设置的值时，将回退为串行查询。

场景五：

```
select /*+NO_PARALLEL (t1) */ * FROM t1, t2;
```

选择 t1 表禁止执行并行查询，当 t1 表大于 txsql_parallel_table_record_threshold 所设置的值时，将回退为串行查询。

场景六：

```
select /*+SET_VAR(txsql_parallel_degree=8)*/ * FROM t1, t2;
```

无论系统默认并行度数值为多少，强制该条语句使用并行度为8执行并行查询，设置该条语句的 txsql_parallel_degree = 8。

场景七：

```
select /*+SET_VAR(txsql_parallel_cost_threshold=1000)*/ * FROM t1, t2
```

设置该条语句的 txsql_parallel_cost_threshold=1000，当该条语句的执行代价大于1000时，即可使用并行查询。

场景八：

```
select /*+SET_VAR(txsql_optimizer_context_max_mem_size=500000)*/ * FROM t1, t2
```

设置单条语句的 txsql_optimizer_context_max_mem_size=500000，该条语句可申请的并行查询计划环境最大内存限制调整为500000。

相关文档

- 开启或关闭并行查询
- 查看并行查询

查看并行查询

最近更新时间：2023-03-13 12:12:23

云数据库 MySQL 支持查看并行查询的执行计划，以及查看线程中哪些线程在执行并行查询计划。您可清晰了解到并行查询是如何在数据库中稳定生效，也可在并行查询执行过程中遇到问题时，帮助快速定位问题。
本文为您介绍查看并行查询的两种常用方法。

方法一：使用 EXPLAIN 语句

示例 SQL 语句：

```
SELECT l_returnflag, l_linestatus, sum(l_quantity) as sum_qty
FROM lineitem
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

本示例为 TPC-H Q1 的简化形式，是典型的报表运算。

执行计划打印语句（EXPLAIN）：

```
EXPLAIN SELECT l_returnflag, l_linestatus, sum(l_quantity) as sum_qty
FROM lineitem
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

查询结果:

```
MySQL [tpch100g]> explain SELECT l_returnflag, l_linestatus, sum(l_quantity) as s
um_qty FROM lineitem WHERE l_shipdate <= '1998-09-02' GROUP BY l_returnflag, l_li
nestatus ORDER BY l_returnflag, l_linestatus;
+----+-------------+------------+-----------+------+---------------+------+----
-----+------+----------+----------+-----------------------------------------------
--------------+
| id | select_type | table | partitions | type | possible_keys | key | key_len |
ref | rows | filtered | Extra |
+----+-------------+------------+-----------+------+---------------+------+----
-----+------+----------+----------+-----------------------------------------------
--------------+
| 1 | SIMPLE | lineitem | NULL | ALL | i_l_shipdate | NULL | NULL | NULL | 593184
480 | 50.00 | Parallel scan (4 workers); Using where; Using temporary |
```

```
| 1 | SIMPLE | <sender1> | NULL | ALL | NULL | NULL | NULL | NULL | 0 | 0.00 | Se
nd to (<receiver1>) |
| 1 | SIMPLE | <receiver1> | NULL | ALL | NULL | NULL | NULL | NULL | 0 | 0.00 |
Receive from (<sender1>); Using temporary; Using filesort |
+----+-------------+-------------+------------+------+---------------+------+----
-----+------+-----------+----------+----------------------------------------------
--------------+
3 rows in set, 1 warning (0.00 sec)
```

树状执行计划打印语句（EXPLAIN format=tree）：

```
EXPLAIN format=tree query SELECT l_returnflag, l_linestatus, sum(l_quantity) as s
um_qty
FROM lineitem
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

查询结果:

```
MySQL [tpch100g]> explain format=tree SELECT l_returnflag, l_linestatus, sum(l_qu
antity) as sum_qty FROM lineitem WHERE l_shipdate <= '1998-09-02' GROUP BY l_retu
rnflag, l_linestatus ORDER BY l_returnflag, l_linestatus\G
*************************** 1. row ***************************
EXPLAIN: -> Sort: lineitem.L_RETURNFLAG, lineitem.L_LINESTATUS
-> Table scan on <temporary>
-> Final Aggregate using temporary table
-> PX Receiver (slice: 0; workers: 1)
-> PX Sender (slice: 1; workers: 4)
-> Table scan on <temporary>
-> Aggregate using temporary table
-> Filter: (lineitem.L_SHIPDATE <= DATE'1998-09-02') (cost=65449341.10 rows=29659
2240)
-> Parallel table scan on lineitem (cost=65449341.10 rows=593184480)

1 row in set (0.00 sec)
```

由上述结果可以看出：

- 并行查询计划将语句分布给了4个工作线程进行运算。
- 将聚合运算拆分为了上下段，用户线程和并行线程分别执行。
- 对 lineitem 表采用了并行扫描算子。
- 实例中树状执行计划打印（EXPLAIN format=tree query）相较于传统执行计划打印（EXPLAIN）效果更好。

方法二：线程列表查看

show processlist 命令的输出结果显示了有哪些线程在运行，不仅可以查看当前所有的连接数，还可以查看当前的连接状态帮助识别出有问题的查询语句等。

基于 show processlist 命令，云数据库 MySQL 自研了 show parallel processlist 语句，帮助您过滤线程中非并行查询的线程，使用该命令行后，将只展示与并行查询有关的线程。

示例 SQL 语句：

```
SELECT l_returnflag, l_linestatus, sum(l_quantity) as sum_qty
FROM lineitem
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

本示例为 TPC-H Q1 的简化形式，是典型的报表运算。

show processlist 查询结果：

```
mysql> show processlist;
+--------+-------------+-----------------+-----------+---------+-------+---------
---+-----------------------------------------------------------------------------
-------------------------+
| Id | User | Host | db | Command | Time | State | Info |
+--------+-------------+-----------------+-----------+---------+-------+---------
---+-----------------------------------------------------------------------------
-------------------------+
| 7 | tencentroot | 127.0.0.1:49238 | NULL | Sleep | 0 | | NULL |
| 11 | tencentroot | 127.0.0.1:49262 | NULL | Sleep | 0 | | NULL |
| 13 | tencentroot | 127.0.0.1:49288 | NULL | Sleep | 1 | | NULL |
| 237062 | tencentroot | localhost | tpch100g | Query | 24 | Scheduling | SELECT
l_returnflag, l_linestatus, sum(l_quantity) as sum_qty FROM lineitem WHERE l_ship
date <= '199 |
| 237107 | tencentroot | localhost | NULL | Query | 0 | init | show processlist |
+--------+-------------+-----------------+-----------+---------+-------+---------
---+-----------------------------------------------------------------------------
-------------------------+
6 rows in set (0.00 sec)
```

show parallel processlist 查询结果：

```
mysql> show parallel processlist;
+--------+-------------+-----------+----------+---------+------+-------------+---
-----------------------------------------------------------------------------
------------------+
| Id | User | Host | db | Command | Time | State | Info |
```

```
+--------+-------------+-----------+----------+---------+------+-------------+---
-------------------------------------------------------------------------------
------------------+
| 237062 | tencentroot | localhost | tpch100g | Query | 18 | Scheduling | SELECT
l_returnflag, l_linestatus, sum(l_quantity) as sum_qty FROM lineitem WHERE l_ship
date <= '199 |
| 237110 | | | | Task | 18 | Task runing | connection 237062, worker 0, task 1 |
| 237111 | | | | Task | 18 | Task runing | connection 237062, worker 1, task 1 |
| 237112 | | | | Task | 18 | Task runing | connection 237062, worker 2, task 1 |
| 237113 | | | | Task | 18 | Task runing | connection 237062, worker 3, task 1 |
+--------+-------------+-----------+----------+---------+------+-------------+---
-------------------------------------------------------------------------------
------------------+
5 rows in set (0.00 sec)
```

由上述结果可以看出：

- 上述查询由并行计划分布给四个 work 线程进行执行：user 仅有一行有显示，表明 ID 237062 为用户线程，将 SQL 语句执行计划下推至下面四个 work 线程中进行，通过 info 列可看到，这四个工作线程均在执行 task1。
- 每个线程均可查询出来，精准进行定位。
- show parallel processlist 相较于 show processlist 可以精准查询到所有进行并行查询的线程，不被其余线程影响。

相关文档

- 开启或关闭并行查询
- hint 语句控制

大事务复制

最近更新时间：2022-03-02 22:25:55

功能介绍

在 row 模式下，单个语句更新多行的大事务每行生成1个 event，一方面产生大量 binlog，另一方面复制时备库 apply 的时候也比较慢，导致备库复制延迟。
腾讯云内核团队通过对大事务复制场景的分析和优化，开发了该能力，大事务复制优化功能将自动识别大事务，并将 row 模式的 binlog 转化为 statement 格式的 binlog，从而减少 binlog 并提升复制效率。

支持版本

- 内核版本 MySQL 5.6 20210630 及以上
- 内核版本 MySQL 5.7 20200630 及以上
- 内核版本 MySQL 8.0 20200830 及以上

适用场景

- 该功能主要提升 row 模式下无主键表的大事务回放速度，在确定是由无主键回放慢导致延迟时可以打开。
- 该功能主要针对 row 模式下存在大事务，出现复制较慢的场景。

性能数据

update 场景复制时间减少85%，insert 场景减少约30%。

使用说明

大事务复制优化功能是基于 SQL 历史执行的统计情况去判断它是否有可能是大事务；当识别为大事务时并且有可能被优化时，会自动将它的隔离级别提升至 RR（可重复读）的级别，将 binlog 落成 Statement 格式，以达到缩短大事务备库执行的时间。 其中：

- cdb_optimize_large_trans_binlog 为该功能的开关。
- cdb_sql_statistics 为 SQL 运行情况进行统计的开关。

- cdb_optimize_large_trans_binlog_last_affected_rows_threshold 和 cdb_optimize_large_trans_binlog_aver_affected_rows_threshold 共同组成了大事务的阈值条件。
- cdb_sql_statistics_info_threshold 为内存中保持中的历史统计数据条数。

为了更好的监控事务运行情况，还增加了 information_schema 库下的表 CDB_SQL_STATISTICS 用于查询当前事务的统计情况。

新增参数

名称	状态	类型	默认	说明
cdb_optimize_large_trans_binlog	true	bool	false	binlog 大事
cdb_optimize_large_trans_binlog_last_affected_rows_threshold	true	ulonglong	10000	大事务优化 影响行数的
cdb_optimize_large_trans_binlog_aver_affected_rows_threshold	true	ulonglong	10000	大事务优化 影响行数的
cdb_sql_statistics	true	bool	false	是否开始对 况进行统计
cdb_sql_statistics_info_threshold	true	ulonglong	10000	CDB_SQL_ 的 map 里保 计的 SQL 个

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。

新增 information_schema.CDB_SQL_STATISTICS 表

名称	类型	说明
DIGEST_MD5	MYSQL_TYPE_STRING	该条 SQL 的 digest 换算出来的 MD5
DIGEST_TEXT	MYSQL_TYPE_STRING	SQL 的 digest 文本格式
SQL_COMMAND	MYSQL_TYPE_STRING	SQL 命令的类型
FIRST_UPDATE_TIMESTAMP	MYSQL_TYPE_DATETIME	该条统计信息第一次产生的时间

名称	类型	说明
LAST_UPDATE_TIMESTAMP	MYSQL_TYPE_DATETIME	该条统计息上一次更新的时间
LAST_ACCESS_TIMESTAMP	MYSQL_TYPE_DATETIME	该条统计信息上一次被访问的时间
EXECUTE_COUNT	MYSQL_TYPE_LONGLONG	这类 SQL 被执行次数
TOTAL_AFFECTED_ROWS	MYSQL_TYPE_LONGLONG	总影响的行数
AVER_AFFECTED_ROWS	MYSQL_TYPE_LONGLONG	平均影响的行数
LAST_AFFECTED_ROWS	MYSQL_TYPE_LONGLONG	上次影响的行数
STMT_BINLOG_FORMAT_IF_POSSIBLE	MYSQL_TYPE_STRING	这类 SQL 是否可以落成 statement 格式的 binlog，TRUE 或者 FALSE

计划缓存点查优化

最近更新时间：2023-07-06 16:35:12

功能介绍

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。

MySQL 8.0 20210830 版本只对（UK&PK）点查起作用，我们将会在后面的版本中开放更大的功能范围。

支持版本

内核版本 MySQL 8.0 20210830 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。

性能影响

- 对于点查（UK&PK）的 SQL，延迟性能提升20% - 30%，吞吐性能提升20% - 30%（sysbench 中的 point_select.lua 测试）。
- 对于内存开销，打开计划缓存开关的情况下，内存使用较不打开将有所提升。

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，以上参数为 tencentroot 级别。

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。

新增 `show cdb_plan_cache` 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

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。

相关状态说明

在通过 show profile 查看 SQL 执行各阶段状态时，当执行 SQL 命中计划缓存，optimizing、statistics 和 preparing 状态将被省略。

支持使用 fdatasync

最近更新时间：2021-12-22 16:25:13

功能介绍

redo 日志文件目前采用 fsync 系统调用来落盘，包括文件元数据落盘和文件数据落盘。文件元数据信息包括最后修改时间等不是非常重要的信息，在一些 redo 落盘场景可以避免总是刷文件元数据到存储设备上（通过 fdatasync 系统调用），来减少开销。

支持版本

- 内核版本 MySQL 5.7 20201230 及以上
- 内核版本 MySQL 8.0 20201230 及以上

适用场景

主要适用于写入压力比较大的场景。

性能数据

在 sysbench-write-only 高并发持续写入场景下，TPS 性能有10%左右提升。

使用说明

通过设置参数 innodb_flush_redo_using_fdatasync=true/false 来控制是否用 fdatasync 来避免 redo 日志文件元数据实时落盘。缺省为 false。

参数名	动态	类型	默认	参数值范围	说明
innodb_flush_redo_using_fdatasync	yes	bool	false	true/false	是否使用 fdatasync 方式刷 redo

支持自增列持久化

最近更新时间：2021-12-16 11:09:24

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 MySQL 5.7 20190830 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。

bufferpool 初始化

最近更新时间：2021-12-22 16:25:13

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 MySQL 5.6 20200915 及以上
- 内核版本 MySQL 5.7 20200630 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%

使用说明

内核默认实现。

FAST DDL

最近更新时间：2023-07-05 16:42:03

功能介绍

该功能优化二级索引创建过程的耗时。开启该功能会使用多线程并发对二级索引数据进行外部排序，同时优化 flush bulk loading 阶段对 flush list 的加锁操作，有效降低 CREATE INDEX 的耗时和对并发 DML 的影响。

支持版本

- 内核版本 MySQL 8.0 20210330 及以上
- 内核版本 MySQL 5.7 20210331 及以上

适用场景

数据库经常会执行 DDL 操作，也经常会遇到 DDL 相关的问题，例如：

- 为什么加索引会造成实例的抖动，影响正常的业务读写？
- 为什么不到1GB的表执行 DDL 有时需要十几分钟？
- 为什么使用了临时表的连接退出时会造成实例抖动？

针对以上常见问题，TXSQL 内核团队经过多场景深入分析以及测试，优化 flush bulk loading 阶段对 flush list 的加锁操作，有效降低 CREATE INDEX 的耗时和对并发 DML 的影响，降低了 DDL 操作带来的影响。

性能数据

sysbench 测试导入20亿行数据，数据量约453GB，开启 FAST DDL 功能。

```
mysql> set global innodb_fast_ddl=ON;
Query OK, 0 rows affected (0.00 sec)
```

开启前耗时4395秒，开启后耗时2455秒。

使用说明

通过参数 innodb_fast_ddl 开启或关闭该功能。

参数名	动态	类型	默认	参数值范围	说明
innodb_fast_ddl	Yes	bool	OFF	{ON,OFF}	开启或关闭 FAST DDL

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。

invisible index

最近更新时间：2021-12-22 16:25:14

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 MySQL 5.7 20180918 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
```

CATS 事务调度

最近更新时间：2022-03-07 17:29:50

功能介绍

TXSQL 新增一种新的并发事务调度算法（Contention-Aware Transaction Scheduling，CATS），可自动感知事务直接锁冲突并根据事务的优先级来调度事务执行。
MySQL 原有的并发事务是采用 FIFO（First-In-First-Out）规则来决定事务执行顺序的，而 CATS 事务调度算法的主要原理是根据事务持有锁的情况来判断并发事务的冲突情况，并决定事务执行的优先级，而后，根据优先级来安排事务的执行顺序，从而提升系统事务处理的吞吐量。

支持版本

- 内核版本 MySQL 5.7 20190230 及以上
- 内核版本 MySQL 8.0 20200630 及以上

适用场景

主要适用于高并发并且锁冲突比较严重的场景。

性能数据

高并发，锁冲突严重的场景下有50%以上的 TPS 性能提升。

- 测试方法：sysbench-oltp_read_write 场景（RR 隔离级别，8张表10MB条数据，pareto 随机模式）
- 测试环境：32核128GB线上实例

线程数	FCFS(FIFO)	CATS	性能提升
128	11999	12005	0%
256	6609	10137	53%
512	3453	9365	171%
1024	2196	7015	219%

使用说明

MySQL 5.7 版本可以通过全局参数 innodb_trx_schedule_algorithm 来指定事务调度算法，该参数缺省值是 auto。其中，算法有三种：

- auto：自动，根据当前系统状况自动调整。当锁等待线程数超过32个时采用 CATS 调度算法，否则采用 FCFS 算法。
- fcfs：先来先服务算法。
- cats: 冲突感知调度算法。

参数名	动态	类型	默认	参数值范围	说明
innodb_trx_schedule_algorithm	yes	string	auto	[auto,fcfs,cats]	事务等待调度算法

> 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。

MySQL 8.0 版本固定采用 auto 算法，不可设置。

计算下推

最近更新时间：2022-03-02 22:31:31

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

- LIMIT/OFFSET 优化对应内核版本 MySQL 5.7 20180530
- SUM 操作下推优化对应内核版本 MySQL 5.7 20180918

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。

性能数据

sysbench 导入一百万行数据后：

- 执行 `select * from sbtest1 limit 1000000,1;` 的时间从6.3秒下降到2.8秒。
- 执行 `select sum(k) from sbtest1;` 的时间从5.4秒下降到1.5秒。

使用说明

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。

安全类特性
透明数据加密

最近更新时间：2021-12-16 11:09:24

功能介绍

在 TXSQL 中，我们沿用 MySQL 的透明加密体系，提供 KEYRING 插件的另外一种实现：KEYRING_KMS，将 KEYRING 与腾讯云企业级的 密钥管理服务 KMS 集成。

KMS 是腾讯云一项保护数据及密钥安全的密钥服务，服务涉及的各个流程均采用高安全性协议通信，保证服务高安全，提供分布式集群管理和热备份，保证服务高可靠和高可用。

KMS 采用的是两层密钥体系，涉及两类密钥，即用户主密钥（CMK）与数据密钥（Datakey）。用户主密钥用于加密数据密钥或密码、证书、配置文件等小包数据（最多4KB）。数据密钥用于加密业务数据。海量的业务数据在存储或通信过程中使用数据密钥以对称加密的方式加密，而数据密钥又通过用户主密钥采用非对称加密方式加密保护。通过两层密钥体系，确保数据在内存和文件中都进行加密。

支持版本

- 内核版本 MySQL 5.7 20171130 及以上
- 内核版本 MySQL 8.0 20200630 及以上

适用场景

透明数据加密指数据的加解密操作对用户透明，支持对数据文件进行实时 I/O 加密和解密，在数据写入磁盘前进行加密，从磁盘读入内存时进行解密，可满足静态数据加密的合规性要求。

使用说明

开启透明数据加密功能，请参见 透明数据加密。

审计

最近更新时间：2023-05-19 11:38:37

功能介绍

腾讯云为云数据库 MySQL 实例提供数据库审计能力，记录对数据库的访问及 SQL 语句执行情况（包括语句开启时间、扫描行数、锁等待时间、 CPU 使用时间、客户端 IP、用户名、SQL 内容等），帮助企业进行风险控制，提高数据安全等级。

适用场景

该功能适用于需要对数据库遭受到的风险行为进行告警，针对数据库 SQL 注入、异常操作等数据库风险行为进行记录与告警的场景。

性能影响

审计分为同步审计和异步审计两种模式。同步审计同步记录所有审计日志，平均性能影响小于6%；异步审计能确保实例性能近乎不受影响，性能损耗最高仅有不到3%，业界领先。

使用说明

开启 MySQL 审计功能，参见审计操作。

稳定类特性
秒级加列

最近更新时间：2022-10-28 10:15:27

功能介绍

快速加列功能是通过只修改数据字典的方法来实现大表快速加列，避免之前加列操作必须做的数据拷贝，从而大幅缩小大表加列所需的时间，减少对系统的影响。

支持版本

- 内核版本 MySQL 5.7 20190830 及以上
- 内核版本 MySQL 8.0 20200630 及以上

适用场景

适用于需要对数据量大的表进行增加列操作的场景。

性能数据

通过对5GB数据量的表进行测试，增加一列操作从40秒降到1秒以内。

使用说明

- Instant Add Column 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，可以设置为 inplace、instant。
  该参数默认为 inplace，可通过设置该参数来调整 Alter Table 的默认算法，如：

```
SET @@global.innodb_alter_table_default_algorithm=instant;
```

通过该参数指定了缺省算法后，在不指明算法的情况下，将使用默认算法来进行 Alter Table 操作。

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。

异步删除大表

最近更新时间：2023-07-05 11:40:24

功能介绍

该功能主要用于删除数据文件很大的表，避免 IO 的抖动。

DROP TABLE 会将原数据库文件 (.ibd) 重命名为一个新的临时文件并返回成功，临时文件存放在 innodb_async_drop_tmp_dir 指定的目录下，并在后台分批次 truncate，每次 truncate 的文件大小由 innodb_async_truncate_size 控制，异步删表功能的开启由：innodb_async_truncate_work_enabled（MySQL 5.6）、innodb_table_drop_mode（MySQL 5.7、8.0）参数控制。

该功能无需用户操作，由内核自动完成，其原理是在删除表时，为表的数据文件在另外一个目录中创建一个硬连接。当执行 drop table 后，删除的只是该文件的一个硬连接。之后后台线程扫描到硬连接目录中有需要删除的文件，自动在后台 truncate 前面 drop 掉表数据文件。

支持版本

- 内核版本 MySQL 5.6 20200303 及以上
- 内核版本 MySQL 5.7 20220715 及以上
- 内核版本 MySQL 8.0 20200630 及以上

适用场景

该功能适用于需要删除的表数据文件很大的场景。

使用说明

- MySQL 5.6 版本，通过设置 innodb_async_truncate_work_enabled 为 ON，DROP TABLE 即会变成异步模式，默认值为 OFF。
- MySQL 5.7、8.0 版本，通过设置 innodb_table_drop_mode 为 ASYNC_DROP，DROP TABLE 即会变成异步模式，默认值为 SYNC_DROP。
- 每次 truncate 的文件大小由 innodb_async_truncate_size 控制（MySQL 5.6 版本暂不支持）。
- 打开 innodb_fast_ddl 参数后，可以让异步删除大表能力更高效。

- MySQL 5.6 相关参数说明
- MySQL 5.7 相关参数说明
- MySQL 8.0 相关参数说明

参数名	动态	类型	默认	参数值范围	说明
innodb_async_truncate_work_enabled	Yes	string	OFF	ON/OFF	是否开启异步清理大表。

热点更新

最近更新时间：2021-12-16 11:09:24

功能介绍

针对于频繁更新或秒杀类业务场景，大幅度优化对于热点行数据的update操作的性能。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，以减少大量行锁造成的并发性能下降。

支持版本

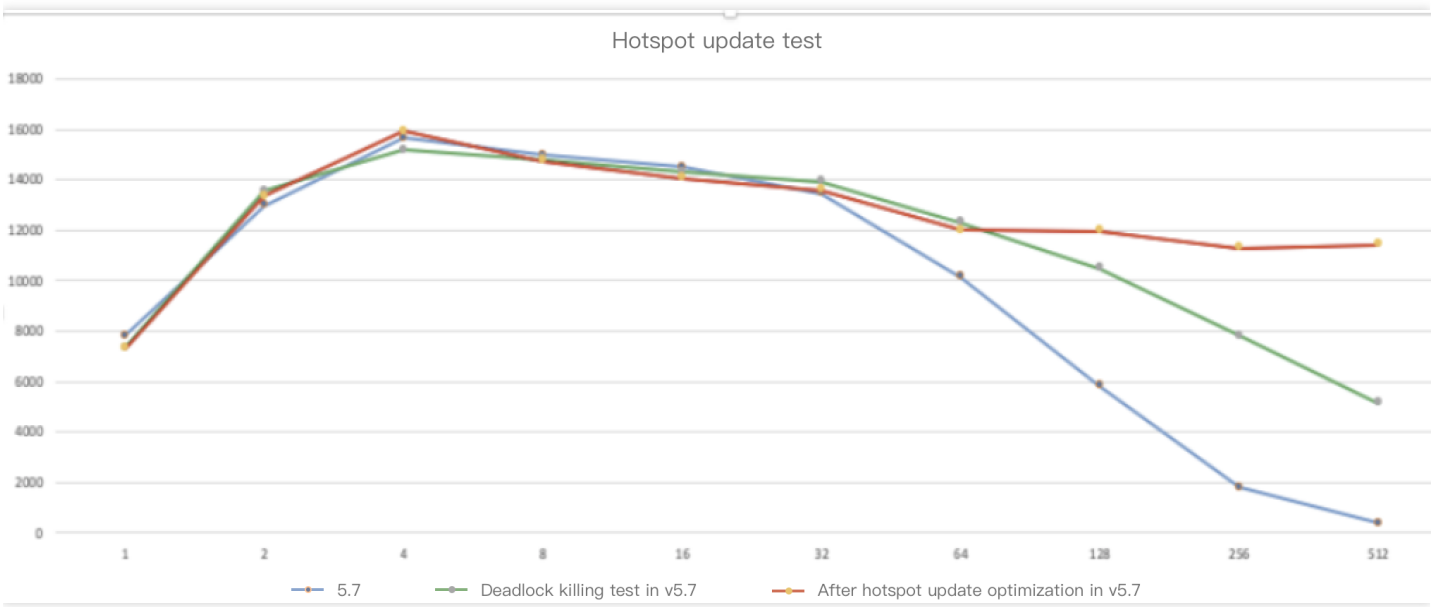
- 内核版本 MySQL 5.7 20200630 及以上
- 内核版本 MySQL 8.0 20200830 及以上

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）。

性能数据

高并发下，带主键条件的单点并发update有10倍以上的性能提升。



使用说明

热点更新保护

SQL 限流

最近更新时间：2024-03-25 17:00:07

功能介绍

通过关键字的设置，限制特定 SQL 同一时间内可并发执行的并发度。

支持版本

内核版本 MySQL 5.7 20200330 及以上
内核版本 MySQL 5.6 20200915 及以上

适用场景

针对某些并发量大，占用大量资源，导致系统性能下降的 SQL。

使用示例

SQL 限流

statement outline

最近更新时间：2022-03-07 11:04:03

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。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 MySQL 8.0 20201230 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。

性能影响

- 当 cdb_opt_outline_enabled 开关打开的情况下，不命中 outline 的 SQL 执行效率将不受影响。
- 命中 outline 的 SQL 执行效率将不及正常执行，但是一般 outline 绑定的提升将比之前的计划性能提升数倍。
- 使用此开关时需要咨询运维或者内核人员，防止可能发生的绑定失误，导致性能回退。

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	fasle	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
* from t1, t2 use index(idx2) where t1.a = t2.a";
```

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 s `elect *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#qb_name(qb2)";
outline "select * from t1 where t1.a in (select b from t2)" set outline_info "OP
T:1#SEMIJOIN(@qb2 MATERIALIZATION, DUPSWEEDOUT)";
```

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 t1.a from t1 left join t2 on t1.a = t2.a))" set outline_info "INDEX:3#test#t1#idx1#1#0";
```

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#0"` 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。

相关参数状态说明

TXSQL 提供多种方式可以查看用户 SQL 的 OUTLINE 绑定，首先可以通过 mysql.outline 表来查看用户设置 OUTLINE 的情况。然后可以通过 show cdb_outline_info 和 select * from information_schema.cdb_outline_info 两个接口查看内存中的 OUTLINE 信息，输入 SQL 是否能被改变，取决于内存中是否有 OUTLINE 信息，所以用户可以用以上两个接口调试。

新增 mysql.outline 系统表，用户设置的 OUTLINE 信息记录存放于此表中，该表字段如下：

字段名	说明
Id	OUTLINE 设置信息编号
Digest	原始 SQL 语句的哈希值
Digest_text	原始 SQL 语句的指纹信息文本
Outline_text	绑定 OUTLINE 之后的 SQL 语句的指纹信息文本

通过 show cdb_outline_info 或者 select * from information_schema.cdb_outline_info 也可以查看内存中的记录，执行 SQL 会命中其中的 OUTLINE 记录绑定计划，参数如下：

字段名	说明
origin	原始 SQL 语句指纹
outline	绑定 OUTLINE 之后的 SQL 语句指纹

TXRocks 引擎
TXRocks 概述

最近更新时间：2022-09-16 14:14:23

TXRocks 概述

RocksDB 是一个非常流行的高性能持久化 KV（key-value）存储，TXRocks 是腾讯 TXSQL 团队基于此开发的事务型存储引擎。

为什么要使用 TXRocks 存储引擎

TXRocks 事务型存储引擎得益于 RocksDB LSM Tree 存储结构，既减少了 InnoDB 页面半满和碎片浪费，又可以使用紧凑格式存储，因此 TXRocks 在保持与 InnoDB 接近的性能前提下，存储空间相比 InnoDB 可以节省一半甚至更多，更适合对事务读写性能有要求，且数据存储量大的业务。

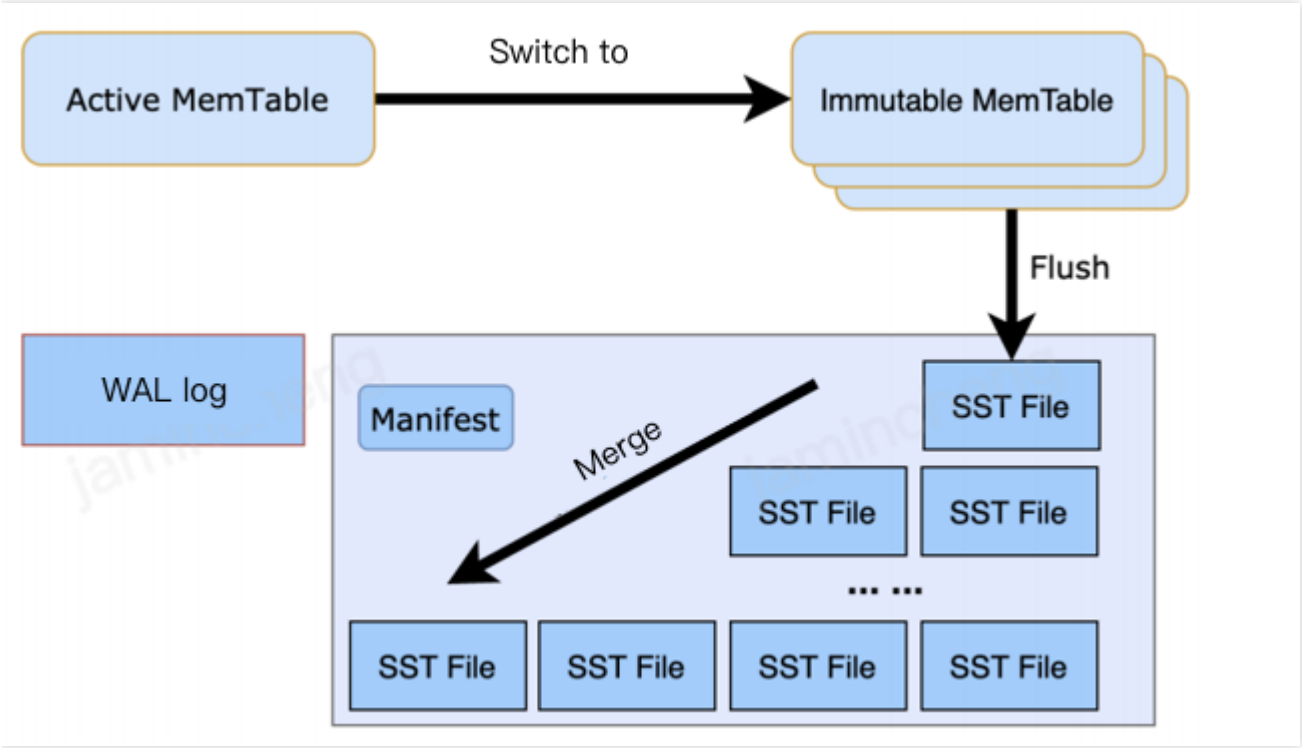
RocksDB 的 LSM Tree 架构

RocksDB 使用 LSM Tree 存储结构，数据组织为一组在内存中的 MemTable 和磁盘上若干层的 SST 文件。
写入请求先将新版本记录写入 Active MemTable，同时写 WAL 日志持久化。写入请求写完 MemTable 和 WAL 就可以返回。
当 Active MemTable 写满到一定程度，将 Active MemTable 切换为冻结的 Immutable MemTable。后台线程将 Immutable MemTable 刷到硬盘，生成对应的 SST 文件。SST 按照刷新的次序分层，通常分为L0层 - L6层。L1层 - L6层，每层内的 SST 中的记录都是有序的，SST 文件之间不会有记录范围的交叠。L0为了支持尽快将 Immutable MemTable 占用的内存空间释放出来，允许 Flush 生成的 L0层的 SST 出现记录范围交叠。

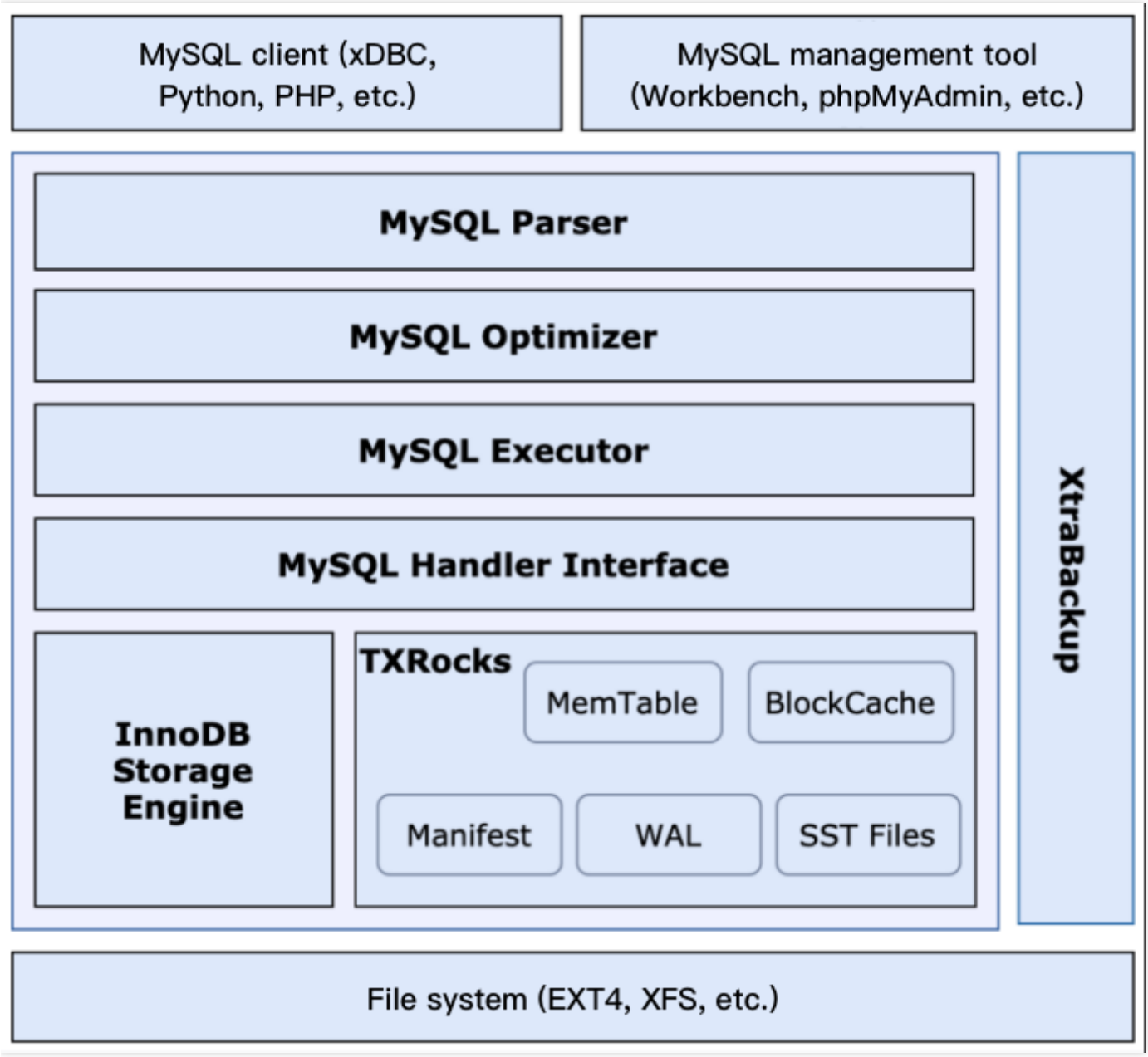
当读取一行记录时，按照新旧，依次从 Active MemTable、Immutable MemTable、L0、L1 - L6各个组件查找这一行，从任一组件找到，就表明找到了最新的版本，可以立刻返回。

当执行范围扫描时，对包含每层 MemTable 在内的各层数据，分别生成一个迭代器，这些迭代器归并查找下一条记录。从读的流程可以看到，如果 LSM Tree 层数太多，则读性能，尤其是范围扫描的性能会明显下降。所以，为了维

持一个更好的 LSM Tree 形状，后台会不断地执行 compaction 操作，将低层数据合并到高层数据，减少层数。



TXRocks 架构



TXRocks 存储引擎的优势

更节省存储空间

相比 InnoDB 使用的 B+Tree 索引结构，LSM Tree 可以节省相当比例的存储空间。
InnoDB 的 B+Tree 分裂通常会导致页面半满，页面内空闲、空间浪费，页面有效利用率比较低。

TXRocks 的 SST 文件一般设置为 MB 量级或者更大，文件要4K对齐产生的浪费比例很低，SST 内部虽然也划分为 Block，但 Block 是不需要对齐的。

另外，TXRocks 的 SST 文件采用前缀压缩，相同的前缀只会记录一份，同时 TXRocks 不同层的 SST 可以采用不同的压缩算法，进一步降低存储空间开销。 事务 overhead 方面，InnoDB 的记录上要包含 trx id，roll_ptr 等字段信息，TXRocks 最底层的 SST 文件（包含绝大部分比例的 数据）上，数据不需要存放其他事务开销，例如记录上的版本号在经过足够长的时间后就可以抹掉。

写放大更低

InnoDB 采用 In-Place 的修改方式，即使仅修改一行记录也可能要刷盘一整个页面，导致比较高的写入放大和随机写。

TXRocks 采用 Append-Only 方式，相比而言写入放大更低。因此 TXRocks 对于擦写次数有限的 SSD 等产品更友好。

适用场景

TXRocks 非常适合对存储成本比较敏感，写多读少但对事务读写性能有要求的，数据存储量大的业务场景。

如何使用 TXRocks 存储引擎

请参见 TXRocks 引擎使用须知。

优化和后续发展

TXRocks 根据业务需求做了部分优化，例如优化 sum 算子下推优化，将 sum 查询性能优化了30多倍。同时 TXRocks 也在积极探索与新硬件结合，利用 AEP 做二级缓存，大幅提升性能，提升性价比。

TXRocks 作为 MySQL 的存储引擎，后续会针对在业务使用中遇到的问题逐渐优化和改进，并计划针对新硬件去做一些新的技术探索，作为 InnoDB 的重要补充，TXRocks 存储引擎会在更多重要业务中上线并平稳运行。

TXRocks 引擎使用须知

最近更新时间：2024-03-25 17:15:44

TXRocks 是腾讯 TXSQL 团队基于 RocksDB 开发的事务型存储引擎，兼具更加节省存储空间和写入放大更低的优势。

产品介绍

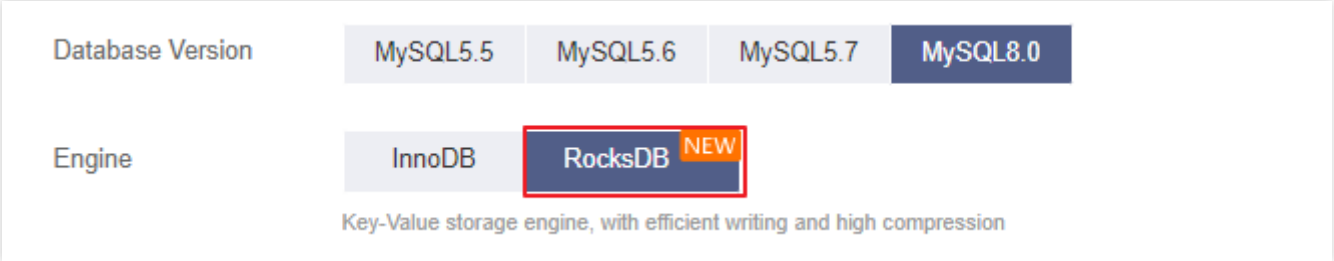
TXRocks 是腾讯 TXSQL 团队基于 RocksDB 的事务型存储引擎，得益于 RocksDB LSM Tree 存储结构，既减少了 InnoDB 页面半满和碎片浪费，又可以使用紧凑格式存储，因此 TXRocks 在保持与 InnoDB 接近的性能的前提下，存储空间相比 InnoDB 可以节省一半甚至更多，非常适合对事务读写性能有要求，且数据存储量大的业务。

前提条件

数据库版本须为 MySQL 5.7、8.0，架构为双节点。

购买云数据库 MySQL 实例（RocksDB 引擎）

您可以在云数据库 MySQL 购买页 购买实例时，选择引擎为 RocksDB，其他参数项可参考 创建 MySQL 实例。

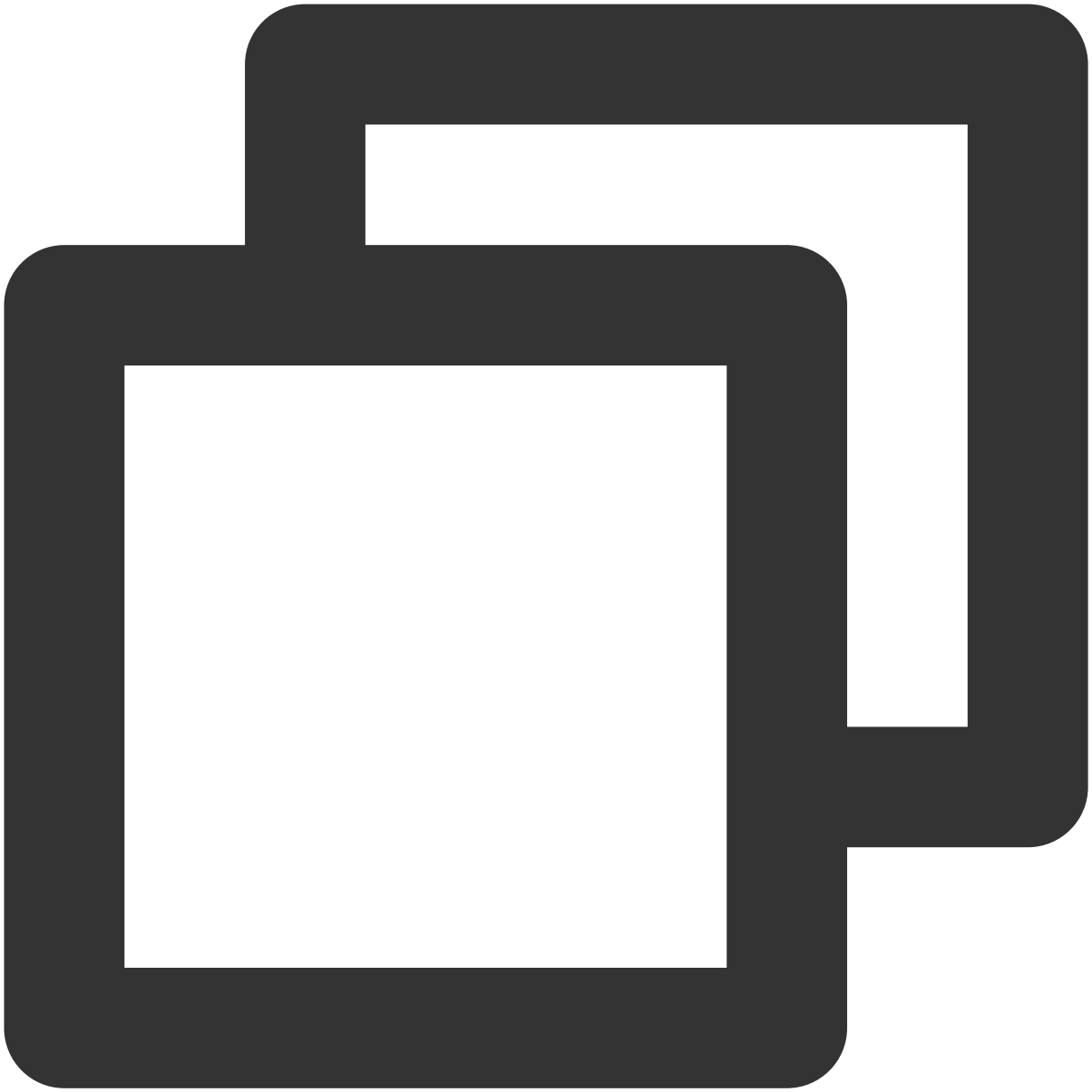


说明：

RocksDB 是 key-value 存储引擎，以高效写入能力与高压缩存储著称，目前暂时仅支持 MySQL 5.7、8.0 版本可选择引擎为 RocksDB。

创建 RocksDB 表

如果创建实例时设置了默认引擎为 RocksDB，则建表时默认引擎就是 RocksDB。您可以通过如下命令查看默认引擎：



```
show variables like '%default_storage_engine%';
```

```
mysql> show variables like '%default_storage_engine%';
+------------------------+---------+
| Variable_name          | Value   |
+------------------------+---------+
| default_storage_engine | RocksDB |
+------------------------+---------+
1 row in set (0.00 sec)
```

当默认引擎是 RocksDB 时，建表语句不许指定存储引擎。

```
mysql> create table tencent_db (id int primary key,c1 varchar(30),c2 varcha
Query OK, 0 rows affected (0.00 sec)

mysql> show create table tencent_db;
+------------+-----------------------------------------------------------------
-----------------+
| Table      | Create Table
                |
+------------+-----------------------------------------------------------------
-----------------+
| tencent_db | CREATE TABLE `tencent_db` (
  `id` int(11) NOT NULL,
  `c1` varchar(30) DEFAULT NULL,
  `c2` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=RocksDB DEFAULT CHARSET=utf8 |
+------------+-----------------------------------------------------------------
-----------------+
1 row in set (0.00 sec)
```

表创建成功后，后续的使用方法与 InnoDB一样，数据会存储在 RocksDB 引擎。

引擎功能限制

TXRocks 在引擎功能上有一些限制，具体如下表所示：

功能分类	功能项	TXRocks 限制
DDL	Online DDL	不支持，例如不支持 ALTER TABLE ... ALOGRITHM=INSTANT 功能，Partition 管理操作仅支持 COPY 算法
SQL 功能	外键	不支持外键（Foreign Key）
	分区表	不支持分区表 （Partition）
	生成列	不支持生成列 （Generated Columns）
	显式 Default 表达式	不支持，如 CREATE TABLE t1（c1 FLOAT DEFAULT(RAND())）

		ENGINE=ROCKSDB; 会失败，报错 'Specited storage engine' is not supported for default value expressions.
	加密表	不支持加密表
索引	空间索引	不支持空间索引（Spatial Index）、空间数据类型（如 GEOMETRY、POINT等）
	全文索引	不支持全文索引（Fulltext Index）
	多值索引	不支持多值索引（multi-valued index）
复制	组复制	不支持组复制（Group Replication）
	binlog 格式	仅支持 ROW 格式，不支持 stmt 或者 mixed 格式
	克隆插件	不支持克隆插件（Clone Plugin）
	可传输表空间	不支持可传输表空间（Transportable Tablespace）
事务与锁	LOCK NOWAIT 和 SKIP LOCKED	不支持 LOCK NOWAIT 和 SKIP LOCKED
	间隙锁	不支持间隙锁（Gap Lock）
	Savepoint	不支持 Savepoint
	部分更新 LOB 字段	不支持部分更新 LOB 字段
	XA 事务	不建议使用

参数说明

说明：

在创建云数据库 MySQL 实例时，可以选择 RocksDB 为默认存储引擎，也可以根据下表的参数说明调整参数模板以便适应自身业务。

MySQL 5.7 相关参数列表

参数名称	是否需要重启	默认值	允许值	描述

rocksdb_use_direct_io_for_flush_and_compaction	是	ON	ON/OFF	compaction 时是否使用 DIC
rocksdb_flush_log_at_trx_commit	否	1	0/1/2	控制何时将日志写入磁盘。类似于 innodb_flush_log_at_trx_co 事务提交时是否进行同步。 等于0时，事务提交时不同步 等于1时，每次事务提交时都 步； 等于2时，每1秒同步一次。
rocksdb_lock_wait_timeout	否	1	1-1073741824	锁等待超时时间，单位秒。
rocksdb_deadlock_detect	否	ON	ON/OFF	死锁检测开关，开启后，有 死锁的信息将记录在 mysqld 日志中。
rocksdb_manual_wal_flush	是	ON	ON/OFF	rocksdb_max_total_wal_siz WAL 超过这个大小，Rocks 开始强制列族落盘，以保证 老的 WAL 文件。

MySQL 8.0 相关参数列表

参数名称	是否需要重启	默认值	允许值	
rocksdb_flush_log_at_trx_commit	否	1	0/1/2	
rocksdb_lock_wait_timeout	否	1	1-1073741824	
rocksdb_merge_buf_size	否	524288(=512K)	100-18446744073709551615	

rocksdb_merge_combine_read_size	否	8388608 (=8M)	524288(=512K)-18446744073709551615
rocksdb_deadlock_detect	否	ON	ON/OFF
rocksdb_manual_wal_flush	是	ON	ON/OFF

RocksDB 引擎监控项

下表为 RocksDB 的引擎监控指标。

指标	描述
rocksdb_bytes_read	读磁盘量
rocksdb_bytes_written	写磁盘量
rocksdb_block_cache_bytes_read	读数据块数
rocksdb_block_cache_bytes_write	写数据块数
rocksdb_wal_log_capacity	写 WAL 日志大小

TXRocks 性价比

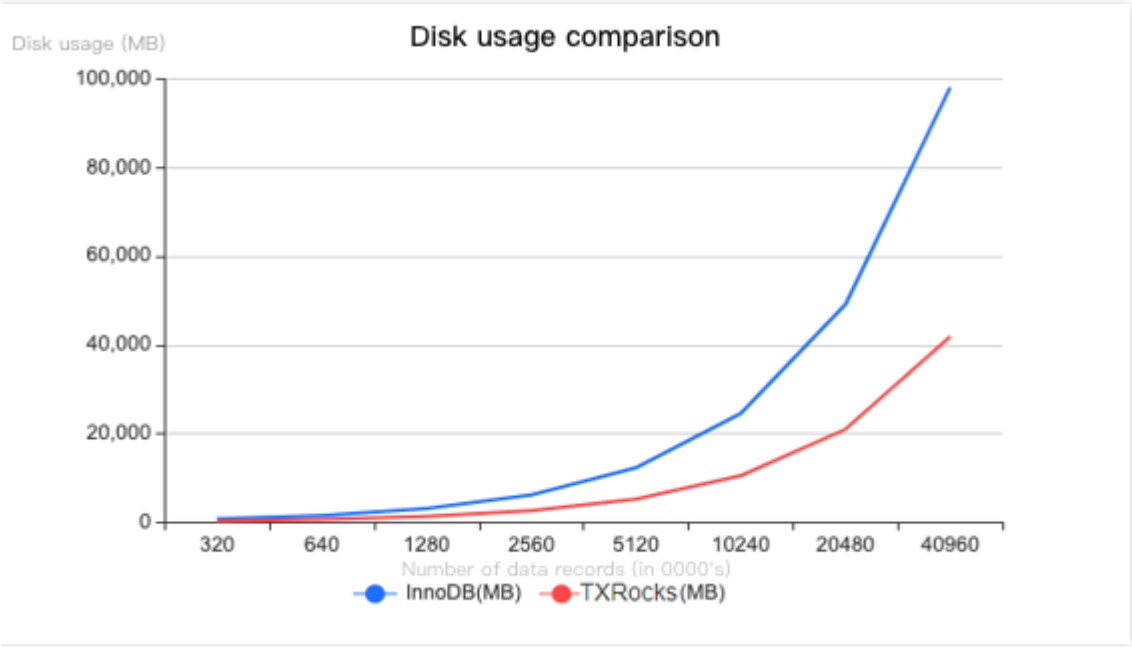
最近更新时间：2022-06-13 15:37:15

TXRocks 的性能与 InnoDB 接近，但由于 LSM Tree 存储结构，减少了 InnoDB 页面半满和碎片浪费，相比 InnoDB，TXRocks 的存储空间可以节省更多，因此具备超高性价比。

背景信息

在腾讯云数据库产品中，TXRocks 为 InnoDB 的重要补充，在性能相近的基础上，TXRocks 做了部分优化和改进，在存储空间上，相比 InnoDB 更为节省，下文将从空间占用和性能来对比两个引擎。

TXRocks 空间占用比 InnoDB 更低

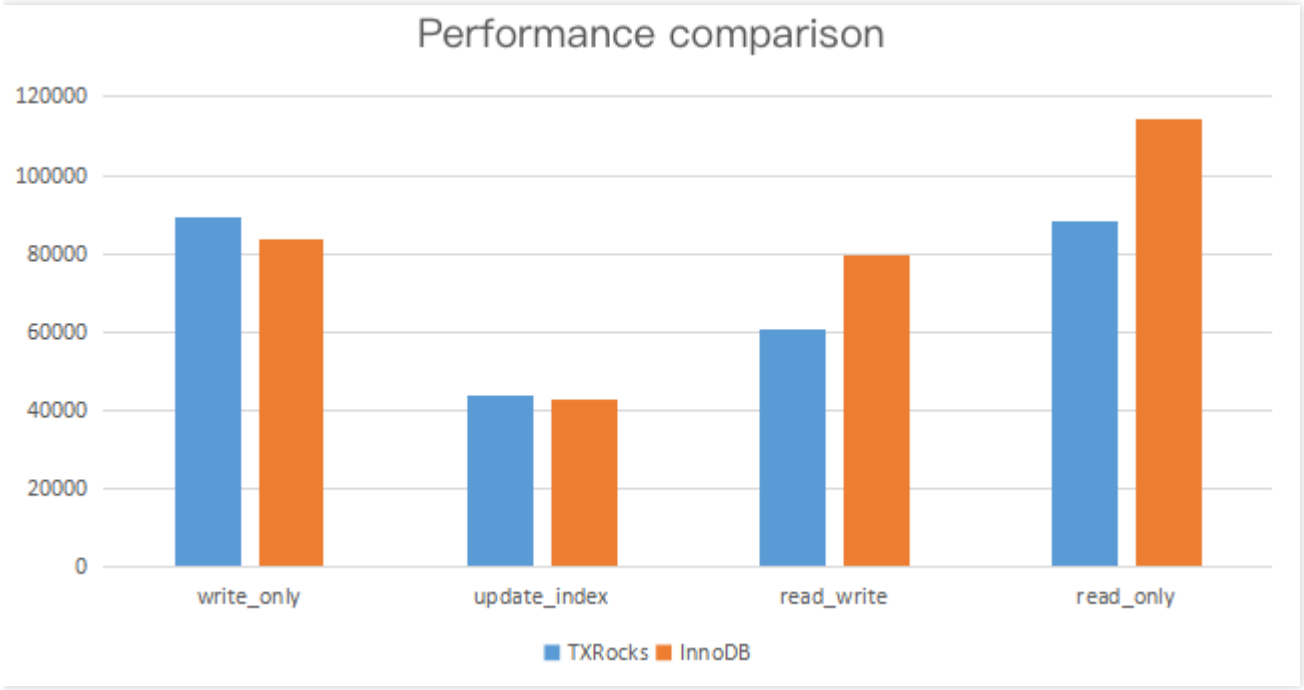


测试场景：两种存储引擎均使用默认配置，使用 SysBench 的默认表结构，每张表包含80万条记录，表总数从4张逐渐增长到512张。

上图为测试条件下分别使用 TXRocks 和 InnoDB 存储引擎时的空间占用情况，左侧为使用 InnoDB 和 TXRocks 存储引擎时的硬盘使用情况。

实测数据显示，随着数据量的逐渐增长，TXRocks 引擎的硬盘占用的增长更慢，节省的空间越多，最多时仅为 InnoDB 的42.71% 。对于记录前缀重复率较高的数据，TXRocks 具备更高的压缩率，具备更高的存储性价比。

TXRocks 性能与 InnoDB 基本持平



测试场景：实例8核32GB场景，6张表500万行数据，每个测试均重启后冷启动测试，每个 case 跑1200秒。
上图为测试场景条件下分别使用 TXRocks 和 InnoDB 存储引擎时的性能对比，通过对比可以发现 TXRocks 和 InnoDB 性能相近。
sysbench 命令关键参数：

```
sysbench --table-size=5000000 --tables=6 --threads=32 --time=1200
```

总结

TXRocks 是一款性能与 InnoDB 相似，但是空间占用较低的腾讯云数据库 MySQL 存储引擎产品。保证了业务性能需要的同时还能降低存储成本，关于 TXRocks 的详细介绍请参见 TXRocks 概述。

TXRocks 最佳实践

最近更新时间：2022-07-12 14:31:13

本文为您介绍使用 TXRocks 的最佳实践 - 大量数据导入如何提升导入速度。

背景

- **场景**：将大量数据导入 TXRocks 引擎的数据库中，需对导入速度进行提升。
- **影响**：导入海量数据时，有可能出现 `Rows inserted during bulk load must not overlap existing rows` 错误。

处理方法1

1. 先删除二级索引（只保留主键索引）。
2. 根据规格和用户数据量调整内存相关参数。

> 说明：
> 需要根据规格和数据量，对参数 rocksdb_merge_buf_size 和 rocksdb_merge_combine_read_size 适当调大。

- rocksdb_merge_buf_size 表示建索引过程中多路归并时每路数据量，rocksdb_merge_combine_read_size 表示多路归并时，各路占用总内存。
- rocksdb_block_cache_size 表示 rocksdb_block_cache 大小，在多路归并时，建议临时调小。

3. bulk load 方式导数据。

```
SET session rocksdb_bulk_load_allow_unsorted=1;
SET session rocksdb_bulk_load=1;
...
导入数据
...
SET session rocksdb_bulk_load=0;
SET session rocksdb_bulk_load_allow_unsorted=0;
```

说明：
如果导入的数据本身有序，则不需要设置 rocksdb_bulk_load_allow_unsorted。
4. 重建二级索引，可以在全部数据导入完成后，逐个重建二级索引。
!

- 二级索引创建过程中涉及多路归并，rocksdb_merge_buf_size 为每路数据量大小，rocksdb_merge_combine_read_size 为合并过程中多路合并所使用的总内存大小。
- 例如，建议 rocksdb_merge_buf_size 设置为64MB以上，rocksdb_merge_combine_read_size 设置为1GB以上，为避免 OOM，导入数据全部完成后务必改回原参数值。
- 此外，每个二级索引创建过程都会消耗较多内存，建议不要同时创建较多的二级索引。

处理方法2

导入数据过程中，关闭 unique_check，可以提升导入性能。

```
SET unique_checks=OFF;
...
导入数据
...
SET unique_checks=ON;
```

注意：
处理完成后，务必将 unique_checks 改回 ON，否则后续正常事务写入的 insert 操作不会进行唯一性检查。