TDSQL-C MySQL 版

自研内核

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

自研内核
内核概述

最近更新时间：2025-07-02 15:03:42

TDSQL-C MySQL 版支持多引擎功能。其中 TXSQL 引擎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。TXSQL 引擎为默认引擎，读写实例和只读实例均可支持使用 TXSQL 引擎。LibraDB 引擎同样兼容支持绝大部分 MySQL 语法，通过只读实例的形态对用户提供服务。
TXSQL 引擎内核还提供了多种 MySQL 企业版功能，如数据库审计、线程池、加密函数、备份恢复等功能。
TXSQL 引擎不仅对 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、数据库审计等高级特性。
而另外一个引擎 LibraDB，则是面向实时分析场景而设计的自研引擎。LibraDB 引擎支持了大规模并行、向量化执行引擎、列式存储等能力，可提供高性能的数据复杂分析查询，同时基于只读实例的形式，可支撑实时的数据查询，避免运维复杂的 ETL 组件，也无需忍受超长的数据时延。LibraDB 引擎作为一个可插拔式引擎，支持在 TDSQL-C MySQL 版5.7版本与8.0版本的集群（实例形态为预置资源的集群）中进行创建。

如何查询数据库内核版本（TXSQL）？

1. 页签视图在**集群管理**页，**数据库版本**后查询。

2. 列表视图在集群详情页的**配置信息** > **数据库版本**后查询。

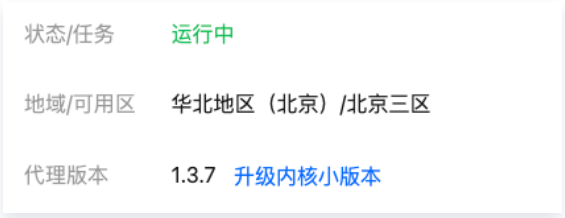
如何查询数据库内核版本（LibraDB）？

1. 页签视图在**集群管理**页，实例信息下，找到分析引擎实例，单击**详情**，可在**基本信息**下查询。

2. 列表视图在**集群管理**页，选择**实例列表**页，然后选择**只读分析引擎**，在列表下单击目标**实例 ID**，进入实例详情页，在基本信息下查询。

如何查询数据库代理版本?

前往**数据库代理**页，在**概览** > **基本信息** > **代理版本**后查询。



TXSQL 引擎内核版本的更多信息

- 关于 TXSQL 引擎内核版本更新动态，请参见 TXSQL 引擎内核版本更新动态 。
- 支持自动或手动升级内核版本，操作及说明请参见 升级内核小版本 。

LibraDB 引擎内核版本的更多信息

关于 LibraDB 引擎内核版本更新动态，请参见 LibraDB 引擎内核版本更新动态 。

数据库代理版本的更多信息

- 关于数据库代理版本更新动态，请参见 数据库代理版本更新动态 。
- 支持升级数据库代理版本，请参见 升级数据库代理小版本 。

TXSQL 引擎内核版本发布时间

TXSQL 引擎8.0版本

数据库内核版本	发布时间
3.1.16.001	2025年05月

3.1.15.006	2025年03月
3.1.15.005	2025年02月
3.1.15.004	2024年11月
3.1.15.003	2024年10月
3.1.15.002	2024年10月
3.1.15.001	2024年10月
3.1.15	2024年08月
3.1.14	2024年05月
3.1.13	2024年04月
3.1.12	2024年01月
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

TXSQL 引擎5.7版本

数据库内核版本	发布时间
2.1.13.002	2025年02月

2.1.13.001	2024年10月
2.1.12	2024年01月
2.1.11	2024年01月
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

LibraDB 引擎内核版本发布时间

只读分析引擎 - LibraDB 引擎版本	发布时间
2.2410.8.0	2025年06月
2.2410.7.0	2025年06月
2.2410.6.0	2025年05月
2.2410.5.0	2025年04月
2.2410.4.1	2025年03月
2.2410.4.0	2025年03月

2.2410.3.0	2025年03月
2.2410.2.0	2025年03月
2.2410.1.0	2025年02月
1.2404.24.0	2025年05月
1.2404.23.0	2025年04月
1.2404.22.1	2025年03月
1.2404.22.0	2025年03月
1.2404.21.0	2025年02月
1.2404.20.0	2025年01月
1.2404.19.1	2024年12月
1.2404.19.0	2024年12月
1.2404.17.0	2024年12月
1.2404.16.0	2024年11月
1.2404.15.2	2024年11月
1.2404.15.1	2024年10月
1.2404.10.0	2024年09月
1.2404.7	2024年08月01日

数据库代理版本发布时间

数据库代理版本	发布时间
1.4.4	2025年03月
1.3.17	2025年05月
1.3.16	2025年03月
1.3.15	2024年12月
1.3.14	2024年11月

1.3.13	2024年08月
1.3.12	2024年06月
1.3.10	2024年03月
1.3.8	2023年10月
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

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：2025-07-02 15:03:42

本文为您介绍 TDSQL-C MySQL 版 TXSQL 引擎的内核版本更新动态。

> **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 。

TXSQL 引擎8.0内核更新说明

小版本	说明
3.1.1 6.001	**注意：** 从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到8.0.28及之后版本。详情请参见： • MySQL 8.0.29 Character Set Support • Changes in MySQL Connector/NET 8.0.28 • 问题修复 ○ 读写/只读实例日志线程的 Socket Buffer 支持动态调整。 ○ 修复第三方订阅工具（例如 Canal、Flink CDC 等）通过只读实例订阅 Binlog 失败的问题。 ○ 物理日志支持并行复制，修复写入量过大导致只读实例被踢出的问题。 ○ 修复调用 System 系统函数修改线程优先级耗时过长的问题。 ○ 兼容了原生基于时间戳的数据订阅行为。 ○ 修复 log_checkpointer 线程长时间拿不到 Log Writer Mutex 造成线程阻塞的问题。 ○ 修复使用 Rewriter 插件导致只读实例启动 Crash 的问题。 ○ 修复使用并行查询时，内存泄漏的问题。

- 修复列压缩在字符串长度为255字节时溢出的问题。
- 合并 MySQL 社区关于 Instant DDL 的修复。
  - 修复 MySQL 5.7升级 MySQL 8.0后有外键的表做 Instant DDL 后数据损坏的问题。
  - 修复执行 Instant DDL 导致后台 Rollback 线程 Crash 的问题。
  - 修复 Instant 修改列位置后 Redo 中未记录 Instant 信息导致 Redo 应用时出错的问题。
  - 修复因8029基线版本前的实例系统表存在 Instant Add Column，导致升级8029后 Crash 的问题。
- 修复只读实例对总空间容量检查的问题，修复后只读实例上取消对总空间容量的检查。
- 修复全文索引大事务 Insert 阻塞其它事务写入的问题。
- 修复全文索引在没有只读实例时后台 Optimize 线程不工作的问题。
- 修复页分裂与降序查询并发时发生死锁的问题。
- 修复只读实例上由于统计信息逻辑错误导致打开分区表慢的问题。
- 修复了数据库代理的 Change User 和 Show Processlist 并发执行导致的死锁问题。

3.1.15.006

> **注意：**
> 从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28 之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到 8.0.28 及之后版本。详情请参见：
> - MySQL 8.0.29 Character Set Support
> - Changes in MySQL Connector/NET 8.0.28

- 问题修复
  - 修复在只读实例上由于误收集 CSI 统计信息，导致打开分区表慢的问题。

3.1.15.005

> **注意：**
> 从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28 之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到 8.0.28 及之后版本。详情请参见：

	⚠ 注意： 从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28 之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到 8.0.28 及之后版本。详情请参见： • MySQL 8.0.29 Character Set Support • Changes in MySQL Connector/NET 8.0.28 • 功能更新 ○ 在只读实例上 binlog 支持了2种订阅方式：1. 基于 file_name 和 pos 的位点方式；2. 基于时间戳的方式。 • 问题修复 ○ 修复调用 system 系统函数修改线程优先级耗时过长的问题。 ○ 修复透明加密不生效的问题。 ○ 修复并行查询执行时有概率导致页面锁超时的问题。
3.1.15.004	⚠ 注意： 从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28 之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到 8.0.28 及之后版本。详情请参见： • MySQL 8.0.29 Character Set Support • Changes in MySQL Connector/NET 8.0.28 • 功能更新 ○ 新增 query cache 功能。 • 问题修复 ○ 修复部分 binlog 开启 binlog checksum 时，订阅工具通过 cdc 订阅可能遇到‘could not queue event from master’错误的问题。
3.1.15.003	⚠ 注意： 从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28 之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到 8.0.28 及之后版本。详情请参见： • MySQL 8.0.29 Character Set Support • Changes in MySQL Connector/NET 8.0.28

	• 功能优化
	  ○ 加快了 Serverless 实例的唤醒时间。
	• 问题修复
	  ○ 修复了用户通过 flink cdc gtid 订阅遇到的兼容性问题，防止用户通过 flink 订阅数据时出现报错场景。
	  ○ 修复全文索引时执行大事务 INSERT 操作阻塞其它事务写入的问题。
	  ○ 修复全文索引在没有 RO 时后台 optimize 线程不工作的问题。
	  ○ 修复 Parallel DDL 在执行期间由于索引文件较小导致使用单线程排序 crash 的问题。

3.1.15.002	**⚠ 注意：**
	从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28 之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到 8.0.28 及之后版本。详情请参见：
	• MySQL 8.0.29 Character Set Support
	• Changes in MySQL Connector/NET 8.0.28
	• 功能更新
	  ○ 支持 TDE 透明数据加密 。
	  ○ 支持二级缓存功能。
	• 问题修复
	  ○ 修复 MySQL 社区关于 INSTANT DDL 的一些问题，具体如下。
	  ○ 修复数据库版本5.7升级至8.0后，对包含外键的表执行 INSTANT DDL 操作后数据损坏的问题。
	  ○ 修复执行 INSTANT DDL 操作导致后台 rollback 线程 crash 的问题。
	  ○ 修复执行 INSTANT 修改列位置操作后 redo 中未记录 INSTANT 信息导致 redo 应用时出错的问题。
	  ○ 修复了在 MySQL 8.0.29版本之前的实例系统表存在 instant add column，升级到3.1.15版本后 crash 的问题。
	  ○ 修复打开 outline 时不显示并行执行计划的问题。

3.1.15 或 3.1.15.001	**⚠ 注意：** 从 MySQL 8.0.29 开始，Information Schema 中表的查询结果会使用 utf8mb3 替代 utf8。Connector/Net 8.0.28 之前的版本不支持 utf8mb3，遇到 utf8mb3 时会报错：Character Set 'utf8mb3' is not supported by

> .Net Framework。如果应用程序使用了 Connector/Net，请在升级 TDSQL-C MySQL 版的内核版本前，先升级 Connector/Net 到 8.0.28 及之后版本。详情请参见：
> - MySQL 8.0.29 Character Set Support
> - Changes in MySQL Connector/NET 8.0.28

- 功能更新
  - 合并官方 8.0.23 至 8.0.30 的变更。
  - 新增 回收站 功能。
  - 新增 并行 Copy DDL 功能。
  - 新增 COS 外表（数据库直接访问 COS 数据）功能。
  - 新增 列压缩 功能。
  - 审计支持更准确的 Sql_Type。
  - 审计添加字段事务 ID 和表名。
  - 创建账号默认审计。
  - 支持 全局索引 功能。
  - 支持 range/list 二级分区 功能。
  - 支持 slave worker 线程与 sql 线程间的死锁检测功能，防止 crash 或重启后的 relaylog 损坏。
  - 支持 innodb 更新计数值的持久化。
  - 并行查询功能更新：
    - PS 模式并行执行。
    - Hash join（in memory）并行执行。
    - 子查询 derived table 单独并行执行。
    - Nested loop join 内表并行。
    - 聚合函数（std，variance，var_samp，stddev_samp）并行执行。
    - ROLL UP 并行执行。
    - EXPLAIN ANALYZE 并行。
    - 支持分区表作为并行查询并行表。
    - 支持全局聚合优化。
    - 支持 having 条件下推并行。
  - 支持将含有子查询的语句在满足条件的情况下改写为窗口函数。
  - 支持 Nonblocking DDL 功能。
  - 支持 BP 隔离 功能。
  - 增加 TP/AP 负载监测功能，允许检查对应语句扫描行数。
  - 闪回查询支持持久化。
  - subquery in limit 在语法层面被禁止。
  - 支持 关联子查询结果缓存 功能。

- 支持将视图转换为 CTE。
- 支持 column 语法 。
- 支持 exchange subpartition template 。

- 问题修复
  - 修复 cdc 补齐 binlog 期间发生 binlog purge 可能导致 slave binlog 缺失的问题。
  - 修复若干 instant ddl 的 bug。
  - 修复了存储过程中调用 update returning，可能导致客户端断开连接的问题。
  - 修复了包括 sql filter 功能和并行执行的计划等价判断功能中的漏洞。
  - 修复了并行执行下，表结构发生变化时有概率触发 crash 的问题。
  - 修复 prepare 语句使用 WHERE column IN (list) 导致性能下降问题。
  - 修复当表的变更包含自增列的时候使用 Parallel Copy DDL 存在的主键冲突问题。
  - 修复了高并发下 binlog 提交时的互斥锁冲突问题。
  - 优化了在线程池模式下，当配置的 thread_pool_size 配置较小时性能严重回退的问题。
  - 优化 outer join 中禁用 eq_ref cache 导致性能回退的问题。
  - 修复 Parallel Copy DDL 事务回滚时断言失败问题。
  - 修复 EXPLAIN FORMAT=TREE 不打印 HashJoin 条件中的子查询的问题，官方 bug 。
  - 修复 index merge intersect 导致查询结果错误的问题。
  - 修复跨机统计信息收集可能阻塞 shutdown 过程的问题。
  - 修复 check index 持有大量页面锁的问题。
  - 修复了并行查询空 range 的分区流程退出的问题。
  - 修复 Txsql_optimizer_context_prealloc_size 上限溢出为0的问题，官方 bug 。
  - 修复了 pfs_thread_setname 函数影响线程池性能的问题。
  - 修复 partition_id 溢出导致 truncate partition crash 的问题。
  - 修复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。
  - 修复并行 DDL 中获取了错误 offset 的问题。
  - 修复并行 DDL 在有重复数据的列上添加 unique key 时 crash 的问题。
  - 修复并行 hash join debug 断言失败的问题。
  - 修复并行查询代价计算 NDV 为0的问题。
  - 修复并行 DDL 在开启 innodb_disable_sort_file_cache 创建索引失败的问题。
  - 修复并行查询执行 Prepare statement 时 exchange id 未重置的问题。
  - 修复了 FORCE INDEX ORDER BY 语句跳过 index dive 的问题，官方 Bug#34976138。
  - 修复了官方子查询计划重复显示的问题，官方 Bug#112345。
  - 修复了落盘临时表计数不增加的问题，官方 Bug#112556。

	○ 修复了数据库代理的 change user 和 show processlist 并发执行导致的死锁问题。
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。 ○ 包含 BLOB、TEXT、JSON 字段的 SQL 查询支持并行，设置方法请参见 支持的语句场景 。 • 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 #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，详见 查询和下载慢日志 。 ○ 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。 ○ 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 ○ 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。

TXSQL 引擎5.7内核更新说明

小版本	说明
2.1.13.002	• 功能更新 ○ 支持二级缓存功能。 • 问题修复 ○ JSON 字符串解析时支持采用更高的浮点数精度。 ○ 修复 innodb_ncdb_avg_log_rsp_time 统计错误的问题。 ○ 修复当 table 内有 JSON 字段时，`checksum table` 命令得到的结果不稳定的问题。 ○ 优化 binlog 订阅功能。

2.1.13.001	• 功能更新 ○ 支持 回收站 功能。 ○ 支持 闪回查询 功能。 ○ 支持 returning 功能。 ○ 支持 Nonblocking DDL 功能。 ○ 支持 并行 Copy DDL 功能。 ○ 新增 DDL 过程信息。 ○ 支持在子查询中使用 LIMIT & IN/ALL/ANY/SOME。 ○ 创建账号默认审计。 ○ 审计增加端口号 ClientPort 字段。 • 问题修复 ○ 修复 cdc 重试次数过多时引起 reached open_files_limit 的错误。 ○ 修复审计语句内单引号未转义的问题。 ○ 修复 JSON 显示精度问题。 ○ 修复 threadpool 状态计算问题。 ○ 修复开启 cdb_enable_sumagg_pushdown 时，聚合函数 sum() 结果出错的问题。 ○ 修复 gtid_subset 的 null_value 未复位导致查询结果出错的问题。 ○ 修复 GROUP_CONCAT 在打开 DERIVED_MERGE 开关的前提下没有正确设置 USED_TABLES 的问题。 ○ 修复数据库代理在使用不同的用户重用连接时出现错误的问题。 ○ 修复数据库代理对 row count 和 found row 以及 db 设置的返回错误。 ○ 修复 hash scan 时内存占用过多的问题。 ○ 修复在 session track 中获取用户自定义变量字符串错误的问题。 ○ 修复使用*的列名查询错误的问题。 ○ 修复 sqlfilter 产生空指针的问题。 ○ 在涉及多个多表时，禁用 returning 语句。 ○ 修复 show processlist 时 crash 的问题。 ○ 修复 Returning 语法导致跨可用区备机 crash 的问题。 ○ 审计 bug 修复：长日志截断多字节字符导致产生乱码日志内容无法解析。 ○ 审计 bug 修复：审计日志在处理转义字符时频繁申请内存导致性能不佳。 ○ 修复 flink - cdc 时间戳订阅报错的问题。 ○ 修复 tablespace name 校验时，如果 name 为空，在有些环境下会 crash 的问题。
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。

LibraDB 引擎内核版本更新动态

最近更新时间：2025-07-02 15:03:42

本文为您介绍 TDSQL-C MySQL 版 LibraDB 引擎的内核版本更新动态。详细产品功能请查看 LibraDB 引擎功能特性 。

版本号规则

LibraDB 引擎的内核版本可以在实例详情信息中查看到。版本号由四位组成，以点号分隔。不同位代表不同的版本含义，具体如下。
第一位：代表产品的主要版本，当产品当前版本与上一个版本不兼容时，则会对此版本号加1。
第二位：代表产品的迭代版本，当对产品新增了重要特性后，就会基于当前的年份与月份进行此迭代号的定义。
第三位：代表产品的次要版本，当对产品做出了一些优化特性或者 Bug Fix 后，就会对此版本号加1。
第四位：代表产品的 Patch 版，当产品发现了紧急 Bug 需要修复，则会对此版本进行 Patch 升级。

版本更新动态说明

LibraDB 引擎 2.2410.x 更新说明

版本	说明

2.2410.8.0	• 系统优化
	  ○ 优化了数据刷盘并发度默认值，极大提升了全量数据加载的执行效率。
	  ○ 优化了计划搜索生成逻辑，提升了部分查询场景下的性能。
	• 问题修复
	  ○ 解决了数据中存在“艺术字”的特殊字符无法加载为列存的问题。

2.2410.7.0	• 功能更新

	○ 支持了对“读写节点”执行 PT 变更、Gh-ost 变更、阿里云 DMS 的无锁变更场景下的行列同步。 ○ 支持了在外连接场景下的谓词推导能力。 ○ 开放了 block_cache_capacity_mb 参数配置。 ○ 支持了 ON 表达式存在子查询的场景。 ○ 支持了列名中包含%的数据表同步到分析引擎。 • 系统优化 ○ 优化了 Runtime Filter 的执行性能。 ○ 优化了分区表的分区裁剪性能。 ○ 优化了在高并发执行场景下的线程使用。 • 问题修复 ○ 修复了 Tinyint 与 Int 数据在 Union 时的报错问题。 ○ 修复了暂停数据增量同步时有可能遇到数据错误的问题。 ○ 修复了 Create View 中包含 Union 时无法同步的问题。 ○ 修复了 PREPARE 语句执行时遇到的执行报错问题。 ○ 修复了表达式下推导致查询异常的问题。 ○ 修复了 Bitmap 索引在执行时查询出错的问题。
2.2410.6.0	• 功能更新 ○ 支持在添加列的同时设置列的默认值。 • 系统优化 ○ 优化了系统表 SLOW_LOG 显示的慢 SQL 信息，避免过多无效的 SQL 执行干扰用户。 ○ 优化了后端存储做 Compact 的时机选择策略。 ○ 优化了在高并发执行时线程数量膨胀的问题。
2.2410.5.0	• 功能更新 ○ 支持对 Json 数据类型中取出的元素值进行聚合函数计算。 ○ 支持了数据全量加载完成时的状态提示。 ○ 支持对多个 Select 语句结果中长度不同的 Decimal 类型字段进行 Union 操作。 • 系统优化 ○ 优化了数据加载时部分采集指标逻辑，提升了监控指标的准确性。
2.2410.4.1	• 问题修复 ○ 修复了在 Add Column 时可能会出现表结构不一致的问题。 • 系统优化 ○ 优化了通过数据库代理访问到只读分析引擎时产生的连接释放的效率。
2.2410.4.0	• 功能更新

	○ 支持适配全新 数据库代理 1.4.4版本能力。 ○ 支持用户自行创建 列存二级索引 能力。 • 问题修复 ○ 修复了 Sort/TopN 算子无法生成多表 Colocate Join 的问题。 ○ 修复了加载部分 DDL 时的错误与告警信息。 ○ 修复了部分算子无法被 Profiling 的问题。 ○ 修复了数据加载过程中日志打印过多占用存储空间的问题。 ○ 修复了 ETL 回写功能执行后，MySQL 客户端返回的影响行数和 Records 条目不对应的问题。
2.2410.3.0	• 问题修复 ○ 修复了在 Uint64 和 Int64 字段类型之间无法互相 Join 的问题。 ○ 修复了数据加载时遇到 Geometry 数据类型导致的数据加载中断问题。 ○ 修复了在空值列上创建索引时解析错误的问题。 ○ 修复了在数据增量加载时，删除主键后又立即新建主键导致表无法重建的逻辑问题。 • 系统优化 ○ 优化了无主键表在加载数据时的处理策略，使无主键表的加载更加稳定。
2.2410.2.0	• 功能更新 ○ 支持了 Date_Add/Find_In_Set/Weekday 函数。 • 问题修复 ○ 修复了在 OR 表达式中添加子查询时使用 Cross Join 导致内存占用过多的问题。 ○ 修复了在 SQL 语句中存在常量列时，执行命令偶尔报错的问题。 ○ 修复了 IF 表达式下执行命令时偶尔报错的问题。 ○ 修复了部分表达式在处理列长度时出现的执行报错问题。
2.2410.1.0	• 功能更新 存储类特性 ○ 支持了基于 列存的二级索引 能力。索引类型包含：Zonemap Index、Bloom Filter Index 以及 Bitmap Index。 ○ 支持了并发 FlushDMS，提升了数据写入效率，特别是在大数据量写入时的更新效率。 ○ 新增支持了 JSON 类型 与部分 JSON 函数 。 ○ 支持了 Range/List 的分区表加载到 LibraDB 引擎。并支持了对分区的 DDL 语句同步加载到 LibraDB 引擎。包括了（Add/Drop 分区/子分区）详细的分区语法和函数支持情况参考 数据加载限制 。 ○ 支持了无主键表加载为列存。同时支持了对表主键的变更场景。详细请参考 数据加载限制 。 计算类特性

- 支持了 Merge Join 能力，提升了在主键 Join 场景下的查询性能。
- 支持了 Stream Agg 能力，提升了在数据有序场景下的 Group By 性能。
- 支持了自动 收集统计信息 能力，无需用户手动定时收集统计信息。
- 支持了随机采样功能，提升了 收集统计信息 的效率，减少了统计信息对资源的消耗。
- 支持了自适应 Group By（Adaptive Group By）。
- 支持了流式 CTE，详细可参考 CTE 语法 。
- 支持了 Table Dual 下推到存储层执行。
- 支持了 Union All 算子下推到存储层执行。
- 支持了优化器的 Win Magic 改写，通过将关联子查询改写为窗口函数的方式进行解关联，提升查询效率。
- 新增支持了14个 MySQL 兼容的函数。详细请参考 字符串函数支持说明 。
- 支持 Hour、Minute、Second、Microsecond 函数的入参值为字符串类型。
- 支持表主键为 Decimal 类型，且最长长度为256的表加载到列存。
- 支持了 Null Aware Anti Join 特性。在处理集合操作时，该特性能够智能识别集合是否为空或包含空值，从而优化 `NOT IN` 和 `<>ALL` 等操作的执行效率，显著提升相关 SQL 查询性能。

- 系统优化
  - 优化了部分执行错误的返回提示。
  - 优化了 Grace Hash Join 性能，采用了两阶段 bucket 加速，减少了 bucket 的扩容次数。
  - 结果集拆分功能支持 Right Anti Join/ Right Semi Join，极大减少了 Join 结果集过大导致的接收数据等待时间过长的问题。
  - 结果集拆分功能支持 Agg，避免 Agg 输出结果集过大导致的接收数据等待时间过长的问题。
  - Explain 优化，支持在执行计划的 Detail 信息中展示 TableScan 算子对应的表名和别名。
  - Explain 优化，支持了在执行计划中展示使用表的副本信息。
  - 支持了 Unique Key 列中包含空值的场景。
  - 优化了元数据存储机制，极大减少了元数据占用资源过多的问题。
  - 优化了 Apply 算子的执行方式。提升了在相关子查询场景下的查询性能。

LibraDB 引擎 1.2404.x 更新说明

版本	说明

1.2404.24.0	• 功能更新
	  ○ 支持在添加列的同时设置列的默认值。
	• 系统优化
	  ○ 优化了系统表 SLOW_LOG 显示的慢 SQL 信息，避免过多无效的 SQL 执行干扰用户。
	  ○ 优化了后端存储做 Compact 的时机选择策略。
	  ○ 优化了在高并发执行时线程数量膨胀的问题。

1.2404.23.0	• 功能更新
	  ○ 支持了数据全量加载完成时的状态提示。
	• 系统优化
	  ○ 优化了数据加载时部分采集指标逻辑，提升了监控指标的准确性。

1.2404.22.1	• 问题修复
	  ○ 修复了在 Add Column 时可能会出现表结构不一致的问题。
	• 系统优化
	  ○ 优化了通过数据库代理访问到只读分析引擎时产生的连接释放的效率。

1.2404.22.0	此版本为问题修复与优化版本，主要解决了1.2404.21.0（包含）之前的版本缺陷。
	• 问题修复
	  ○ 修复了部分场景下同步 DDL 时导致的数据延时问题。
	  ○ 修复了相关子查询列构建 Datetime 类型时报错的问题。
	  ○ 修复了在 IN 表达式场景中，参数为数组的场景下报错的问题。
	• 系统优化
	  ○ 优化了在大数据量下，使用主键谓词查询时的性能。
	  ○ 优化了慢 SQL 查询视图遇到较长 SQL 时的截断长度。

1.2404.21.0	此版本为问题修复与优化版本，主要解决了1.2404.20.0（包含）之前的版本缺陷。
	• 功能更新
	  ○ 在当前版本中支持了分区表相关 DDL。
	• 问题修复
	  ○ 修复了在主键包含 Timestamp 类型的场景下数据删除失败的问题。
	  ○ 修复了 Set 类型数据同步的数据错误问题。
	  ○ 修复了 Rename Table 场景下的部分错误。

1.2404.20.0	此版本为问题修复与优化版本，主要解决了1.2404.19.1（包含）之前的版本缺陷。
	• 问题修复
	  ○ 修复了在读写实例设置为大小写敏感场景下无法查看到数据加载对象的问题。
	  ○ 修复了部分存量实例无法适配支持数据库代理的问题。

	○ 修复了延迟物化统计信息的显示错误的问题。 ○ 修复了 DMC 无法获取到只读分析引擎用户信息的问题。 ○ 优化了部分内核参数设置的默认值。
1.2404.19.1	此版本为问题修复版本，主要修复了1.2404.19.0之前的版本中的缺陷。 • 问题修复 ○ 修复了在加载视图到只读分析引擎时，视图的 definer 不一致问题。 ○ 修复了 mysql.user 系统表在遇到空密码的用户时的用户同步失败问题。 ○ 修复了在 SQL 执行过程中执行 DDL 导致的执行出错问题。 ○ 修复了“加速 ETL 回写”功能在遇到存在常量列时的写异常问题。
1.2404.19.0	• 功能更新 ○ 当前版本支持通过 数据库代理，同时使用 Hint 语法 访问至“只读分析引擎”。 ○ 支持了通过 DMC 访问至“只读分析引擎”的能力。 • 系统优化 ○ 优化了 Sort 算子以及 Aggregation 算子的内存管理方式，提高了内存使用效率。优化后，Sort 算子和 Aggregation 算子执行结束时就可以提前释放内存，而不需要等到查询执行结束。 ○ 优化了 Join 结果输出的内存使用方式，降低了 Join 算子输出结果集比较大时的内存使用。 ○ 优化了主键场景下延迟物化的分配策略。 • 问题修复 ○ 修复了 Prepare Statement 默认走 Plan Cache 导致多个相似查询，常量参数类型不一致时命中之前的缓存计划导致查询报错的问题。 ○ 修复了在 Union All 场景下输出列涉及 VARCHAR 类型时部分场景的查询报错问题。 ○ 修复了 DMC 连接列存引擎时，因权限不足导致的部分元数据获取报错问题。 ○ 修复了在频繁创建连接的场景下可能存在的内存泄露问题。 ○ 修复了 PT 变更场景下临时表在只读分析引擎中无法查询的问题。 ○ 修复了执行计划在某些场景下延迟物化算子不显示的问题。 ○ 修复了“只读分析引擎”无法删除联合索引中的字段的问题。 ○ 修复了多表 Join 场景下 Runtime Filter 分配越界问题。 ○ 修复了 Runtime filter 过滤数据偶然出现为空的问题。 ○ 修复了 Union All 场景中 NULLABLE 类型与非 NULLABLE 类型不一致时的报错问题。
1.2404.17.0	• 功能更新 ○ 支持了 FROM_UNIXTIME 与 UNIX_TIMESTAMP 函数。 • 系统优化

	○ 优化了 DBearer 等数据库客户端工具访问只读分析引擎时提示的系统表查询权限。 ○ 优化了部分不支持函数报错时的报错信息。 • 问题修复 ○ 修复了在读写实例的 explicit_defaults_for_timestamp 参数值不同的场景下，只读分析引擎加载 Timestamp 字段值的问题。
1.2404.16.0	• 功能更新 ○ 支持使用“只读分析引擎”加速 `INSERT…SELECT…` 中的查询语句执行，并且支持更快速度写入读写实例。请参见：加速 ETL 回写 。 ○ 支持了 BIT、ENUM、SET 类型数据的查询与同步。 • 系统优化 ○ 优化了部分系统表的命名格式，使其更容易理解，如 SLOW_LOG 表。 ○ 当前版本将普通列支持的值大小调整到了16MB，之前版本为5MB。 ○ 优化了当查询计划为空时，StorageTableReader 算子获取列名的逻辑。 • 问题修复 ○ 修复了子查询场景下, 主查询结果为空时，仍然使用 Apply 算子逻辑，导致查询报错的问题。 ○ 修复了 Apply 算子在多线程竞争时导致 Crash 的问题。
1.2404.15.2	此版本为问题修复版本，主要修复了1.2404.15.1之前的版本中的缺陷。 • 问题修复 ○ 解决了加载为列存表后，在读写实例中高并发执行 `UPDATE` 时，执行数据查询时偶发的实例 Crash 问题。
1.2404.15.1	• 系统优化 ○ 优化了数据库日志保留大小的默认参数值。调整为：当实例磁盘规格小于等于500G时，默认最大保留5G日志；当实例磁盘规格大于500G时，默认最大保留10G日志。 ○ 默认打开只读分析引擎功能“延迟物化”，详细关于延迟物化调优的能力，请参见 延迟物化 。 ○ 支持了以“_”符号作为开头的表加载到只读分析引擎。 ○ 优化了只读分析引擎重启的效率。 • 问题修复 ○ 解决了 Navicate 部分版本连接到只读分析引擎出现权限不足而报错的问题。 ○ 解决了在连续执行 DDL 时，只读分析引擎数据加载出错的问题。 ○ 解决了在 `CREATE TABLE` 语句中 DEFAULT (0) 的语法兼容性问题。
1.2404.10.0	• 功能更新 ○ 支持了 DATE_SUB 函数。

	○ 支持用户手动执行表的统计信息收集。详细可参考 收集统计信息 。 ○ 支持了 ENUM 在特殊场景下的数据同步加载。 **说明：** 虽然支持了 ENUM 在特殊场景下加载到只读分析引擎，但是依然不支持 ENUM 字段类型的数据查询。 • 系统优化 ○ 修改内核对存储中过期数据保留的版本数量，避免因为保留太多数据快照导致的磁盘空间占用过多。 ○ 优化了内核在对数据清理过程中所占用的资源比例，避免过多的系统资源占用导致的用户 SQL 执行性能受损。 ○ 修改内核信息日志打印的等级，避免因为过多的日志打印占用过多磁盘空间。 ○ 调整了“存储”这单个字段的最大值大小，之前默认支持的最大值为1MB，现在修改为了5MB。 ○ 优化了在数据未完全在只读分析引擎中加载时进行数据查询的报错信息，现在可以更明确的提示需要等待全量数据加载完成后才可以查询。 ○ 优化了部分不支持的数据类型或者函数在执行时的提示错误信息，现在可以很明确的从错误信息中查看到不支持的项目。 ○ 优化了系统内部关于后台任务的触发频率和选择逻辑，避免因为后台任务的触发影响用户在线 SQL 的执行性能。 ○ 优化了通过 MySQL 客户端访问到只读分析引擎实例中时看到的版本号信息，避免产生理解性问题。
1.2404.7	功能更新 • 支持超大规模数据实时数据加载到只读分析引擎中。 • 支持超高性能的数据复杂查询。 **说明：** 此版本为 TDSQL-C MySQL 版首个全新上线的 LibraDB 引擎内核版本。

数据库代理版本更新动态

最近更新时间：2025-05-20 12:11:22

本文介绍 TDSQL-C MySQL 版数据库代理版本的更新说明。

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 。
- 如需了解各个数据库代理版本发布时间，请参见 数据库代理版本发布时间 。
- 关于数据库代理版本请注意，1.4开头的版本为尝鲜版，1.3开头的版本为稳定版，1.3以下的版本为公测期间的版本且不再更新。

稳定版更新说明

数据库代理版本	TDSQL-C MySQL 版内核版本要求	说明
1.3.17	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 • 修复了开启事务拆分后，开启只读事务有时会报错的问题。 • 修复了开启连接池后，连接复用时可能出现字符集错误的问题。
1.3.16	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 • 修复了在只读地址下执行预编译语句时，可能有小概率出现报错的问题。 • 修复了开启连接池之后可能出现 Capabilities Flags 异常的问题。

1.3.15	TDSQL-C MySQL 版 5.7 ⩾ 2.0.20/2.1.6
	TDSQL-C MySQL 版 8.0 ⩾ 3.1.6
		• 性能优化
		  ○ 优化了高负载下的内存占用，解决了可能出现内存长期处于高位的问题。
		• 问题修复
		  ○ 修复了防闪断有概率失败的问题。

1.3.14	TDSQL-C MySQL 版 5.7 ⩾ 2.0.20/2.1.6	• 功能更新 ○ 现在 SHOW BINLOG EVENTS 和 SHOW BINARY LOGS 语句会被路由到主实例。

	TDSQL-C MySQL 版 8.0 ⩾ 3.1.6	○ 优化了查询返回大报文时可能出现内存占用过多的问题。 • 问题修复 ○ 修复了偶尔出现的握手报文字符集排序与后端数据库不一致的问题。 ○ 修复了读写地址下的主库进行主从切换时，有时会导致只读地址连接中断的问题。 ○ 修复了账户指定 IP 同时命中多个账户时，可能鉴权失败的问题。
1.3.13	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 ○ 现在 SELECT binlog 相关的 系统变量 会发送到主库执行。 ○ 新增了新的高可用探测语句，减少误判故障引起的迁移。 ○ 优化了数据库代理内核在多线程场景下的内存占用。 • 问题修复 ○ 修复了只读事务中执行预编译语句会报错的问题。 ○ 修复了当使用错误用户名频繁建连时引起的性能问题。 ○ 开启 事务拆分 后，COMMIT 和 ROLLBACK 语句会路由到所有节点上，防止 RO 节点上出现悬挂的未提交事务。
1.3.12	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 ○ 现在系统变量的读取会发送到主库上执行。 • 问题修复 ○ 修复了大量连接触发防闪断时可能出现部分连接卡住时间过长的问题。 ○ 修复了大量连接触发重新负载均衡时部分连接未按预期断开的问题。 ○ 修复了客户端发送 COM_RESET_CONNECTION 报文导致一致性失效的问题。
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。 • 支持主实例原地重启及升级时防闪断。
1.3.7	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 • 修复了某些情况下 SELECT...FOR UPDATE 语句路由错误的问题。 • 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 ⩾ 2.0.20/2.1.6 TDSQL-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 ⩾ 2.0.20/2.1.6 TDSQL-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 ⩾ 2.0.20/2.1.6 TDSQL-C MySQL 版 8.0 ⩾ 3.1.6	问题修复 • 修复会话连接池下预编译语句有概率报错的问题。 • 修复了预编译语句有概率报错的问题。
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。

尝鲜版更新说明

数据库代理版本	TDSQL-C MySQL 版内核版本要求	说明
1.4.4	TDSQL-C MySQL 版 5.7 ⩾ 2.0.20/2.1.6 TDSQL-C MySQL 版 8.0 ⩾ 3.1.6 只读分析引擎版本 ⩾ 2.2410.4.0	功能更新 • 支持只读分析引擎 LibraDB 实例参与数据库代理读写分离和故障剔除。 • 支持在 SQL 语句中使用 HINT /* to ap */ 让 SQL 路由到只读分析引擎实例。

内核优化版本
编译优化高性能版本

最近更新时间：2024-10-12 10:54:31

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。

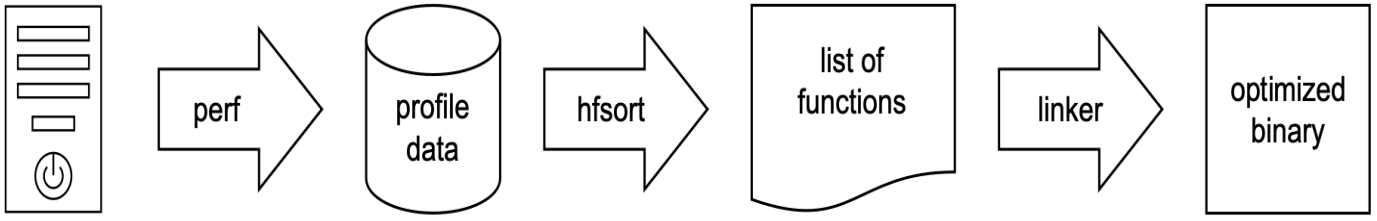
背景

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。

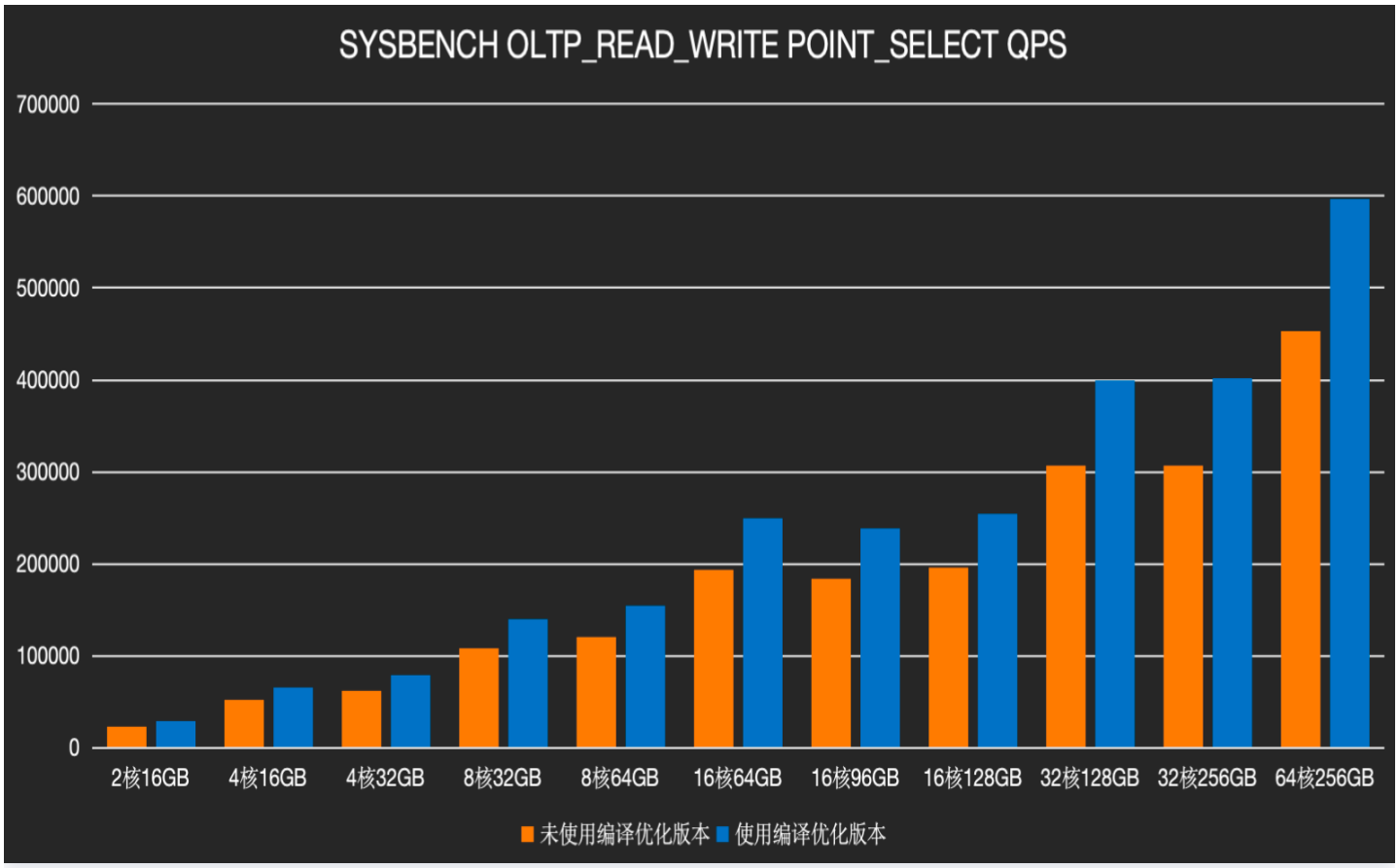
通过以上三个阶段的优化，TDSQL-C MySQL 版的编译优化高性能版本就能更好地适应用户的需求，提高应用的性能和效率。

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%
64核 256GB	512	6000000	450	596706	31%

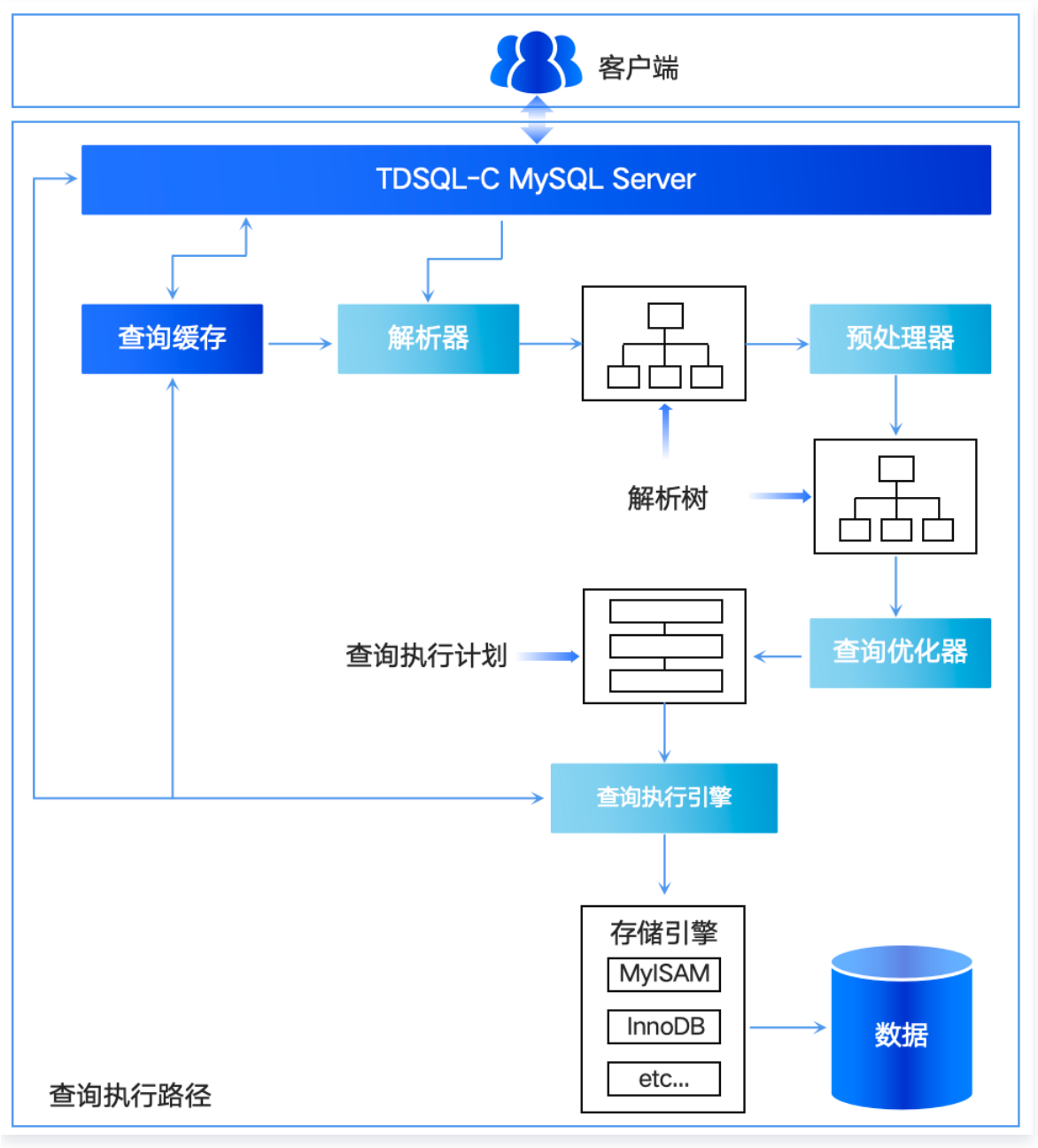
功能类特性

Query Cache

最近更新时间：2025-04-17 17:13:11

功能介绍

查询缓存（Query Cache）是 MySQL 为了提升数据库查询性能而设计的功能。在数据库中执行查询语句时，该功能会把查询文本及其对应结果集存入查询缓存。若之后收到相同查询请求，数据库会先查看查询缓存，若有对应结果，便直接返回，无需再次解析、优化和执行查询，从而大幅缩短查询响应时间。由于 MySQL 的原生查询缓存功能存在性能相关问题，因此 TDSQL-C MySQL 版针对 MySQL 原生查询缓存的不足进行了重新设计，推出 TDSQL-C MySQL 版的查询缓存功能，能够有效提高数据库的查询性能。



MySQL 原生查询缓存功能存在的问题

- 查询缓存的命中率和 Hash 查询性能有限。
- 查询缓存会浪费缓存空间。
- 查询缓存的参数配置困难。

TDSQL-C MySQL 版针对以上问题进行的优化和说明

优化场景	说明	价值
查询缓存分区优化	对 Query Cache/Meta Hash 进行分区处理，将不同类型或范围的查询精准映射至特定分区，降低 Hash 冲突概率，加速查询缓存的定位与检索流程。	提升查询缓存的 Hash 性能与命中率，在高并发查询场景中显著减少 CPU 资源消耗，更高效地响应大量查询请求。
只读节点开表优化	针对只读节点查询缓存有效性验证环节中的开表操作进行深度优化，减少因表操作引发的不必要的缓存失效情形，确保查询缓存在更多场景中维持有效状态。	提升缓存的实际利用率与系统整体查询性能。
基于命中率的智能缓存控制	当查询缓存命中率处于较低水平时，自动启动智能缓存控制机制，限制并发查询数量，合理设置采样频率，让绝大多数查询绕过查询缓存，仅用专用线程负责采样与命中率估算工作，命中率回升后重新启用查询缓存功能。	避免在低命中率状况下大量查询涌入缓存系统导致的资源浪费与性能下滑问题，减少无效查询缓存占用的空间资源，维持系统在不同命中率场景下的稳定高效运行。
缓存换入换出开销优化	优化缓存替换算法与内存管理策略，降低缓存换入换出操作的频率与资源消耗，依据科学合理策略进行缓存替换，优先保留高频次及重要的查询缓存结果。	减少因频繁换入换出导致的系统性能波动，让查询缓存在有限内存资源下发挥最大效能，提供持续稳定的查询加速支持。
LRU 缓存策略优化	依据查询的大小与 Hit 数对 Query Cache LRU 缓存策略进行精细化调整，优先存储高频次及标量查询结果，避免缓存低效查询造成的空间浪费。	提升缓存空间的利用效率与系统在特定负载下的查询响应性能，确保缓存资源分配至最具价值的查询任务上。
动态缓存开关控制	在系统面临写负载压力时及时动态关闭查询缓存，读写负载停止后秒级恢复启用状态。	防止因频繁更新数据导致的查询缓存失效及性能退化问题，提高查询缓存的适应性与灵活性，使系统在复杂读写负载环境中维持良好性能表现。
动态缓存	支持动态 resize query cache size，且调整过程不阻塞当前正在进行的读写请求，根据实时负载需求与资源状况灵活调整缓存大小。	解决固定缓存大小设置不合理的性能瓶颈问题，避免因缓存容量调整引发的性能急剧下降

容量调整		或中断现象，为系统持续稳定高效提供查询加速服务。

支持版本

内核版本 TXSQL 8.0 3.1.15.004及以上。

适用场景

查询缓存（Query Cache）作为一种读缓存系统，适用于有局部热点读或存在较长时间只读的业务负载场景。

> **说明：**
> 如果您的业务负载场景不存在热点读负载，或者数据库中每个表都在频繁进行更新，使用查询缓存功能可能存在一定的性能损耗。

功能使用建议

- 当存在热点读负载时，可以通过状态 Qcache_free_memory（缓存剩余空间）和 Qcache_hit_rate（近期查询命中率）来确定是否需要调整 query_cache_size 以提高读性能。
- 当缓存剩余空间较少，近期查询命中率相对较低时，可以通过适当增大 query_cache_size 来提高缓存系统命中率，从而提高读性能。
- 如果增大 query_cache_size 无法明显提升缓存命中率，则说明当前负载访问范围较大，没有明显热点。也可以通过状态 Qcache_queries_in_cache（缓存查询数量）观察当前系统中缓存的查询数量来确定能否覆盖业务热点读。
- 不建议将 query_cache_size 设置超过实例内存的10%，这是由于较大的 query_cache_size 会带来 OOM 风险。

使用限制

> **注意：**
> 当前版本的查询缓存（Query Cache）和数据库审计不兼容。若已开启查询缓存功能，则无法开启数据库审计，需要先关闭查询缓存功能才能开启数据库审计；若已开启数据库审计，则无法开启查询缓存功能，需要先关闭数据库审计才能开启查询缓存功能。

使用说明

您可通过以下参数，对查询缓存功能进行使用和管理，设置参数的方法请参见 设置实例参数 。

参数名	是否	是否	默认值	取值范围	支持配置的节点	说明

	重启	全局				
query_cache_limit	no	否	1048576	0 – ulong_max*	• 读写实例 • 只读实例	超过该值的 Query 结果集不会被缓存，单位：Bytes。
query_cache_size	no	否	1048576	1048576 – ulong_max*	• 读写实例 • 只读实例	系统中可以用于 Query Cache 内存的大小（实际使用时才会占用内存空间），单位：Bytes，取值必须为1024的倍数。 **说明：** Sysbench Poc 场景建议设置表大小的20%以上作为 query_cache_size。正常场景建议为 Buffer Pool 的5%左右，如果业务没有只读表或者热点不明显，建议降低 query_cache_size 大小。
query_cache_type	详见说明列	否	OFF	OFF/ON/DEMAND	• 读写实例 • 只读实例	• 取值 OFF：表示功能关闭。 • 取值 ON：表示缓存所有结果，除非 SELECT 语句使用 sql_no_cache 禁用查询缓存。 • 取值 DEMAND：表示只缓存 SELECT 语句中通过 sql_cache 指定需要缓存的查询。 **注意：** 如果查询缓存最初是关闭的（配置文件中

						query_cache_type=0），则无法再动态开启，需要配置 my.cnf 重启才能开启查询缓存。但是关闭查询缓存始终都可以动态执行。
query_cache_wlock_invalidate	no	否	OFF	ON/OFF	• 读写实例 • 只读实例	当写锁发生在表上时，是否先失效该表相关的查询缓存。 • ON：是 • OFF：否

*ulong_max = 18446744073709551615

以下是查询缓存新增的 Status，可以通过命令查询 Query Cache 相关 Status，语法如下。

```
SHOW STATUS LIKE 'Qcache%';
```

Status	描述
Qcache_hits	查询缓存命中次数。
Qcache_hit_rate	近期查询命中率，值为最近10000次查询缓存（Qcache_search_times 每增加10000）中，命中查询缓存的比例。
Qcache_search_times	Query 查询 Cache 的次数（被限流以及不满足走 Query Cache 的查询不会计数）。
Qcache_total_times	Query 走 Query Cache 的总次数（限流等情况也会被记录）。
Qcache_free_memory	Query Cache 剩余空间。
Qcache_inserts	成功缓存结果集的次数。
Qcache_not_cached	未能缓存结果集的次数。
Qcache_queries_in_cache	在 Query Cache 中缓存的 Query 数量。

Qcache_lowmem_prunes	被 LRU 驱逐的 Query Cache 数量。

性能测试

测试环境

备节点数量：1个读写实例和1个只读实例。
集群配置：8核16GB。
测试工具：Sysbench 1.0.20。
数据库内核版本：TXSQL 8.0 3.1.15.004。
数据量：

- **全缓存场景**：Sysbench 的数据量约为1.4GB，共25000*250张表。
- **大 IO 场景**：Sysbench 的数据量约为54GB，共800000*300张表。

测试目标、场景、步骤、结果

测试目标	测试场景	测试步骤及结果
查询缓存对读场景性能的提升。	不同数据量下，Query Cache 带来的性能提升。	测试场景一：查询缓存对读场景的性能提升
查询缓存大小（query_cache_size）对性能的影响。	不同 query_cache_size 及数据量下，Query Cache 带来的性能提升。	测试场景二：查询缓存大小对性能的影响

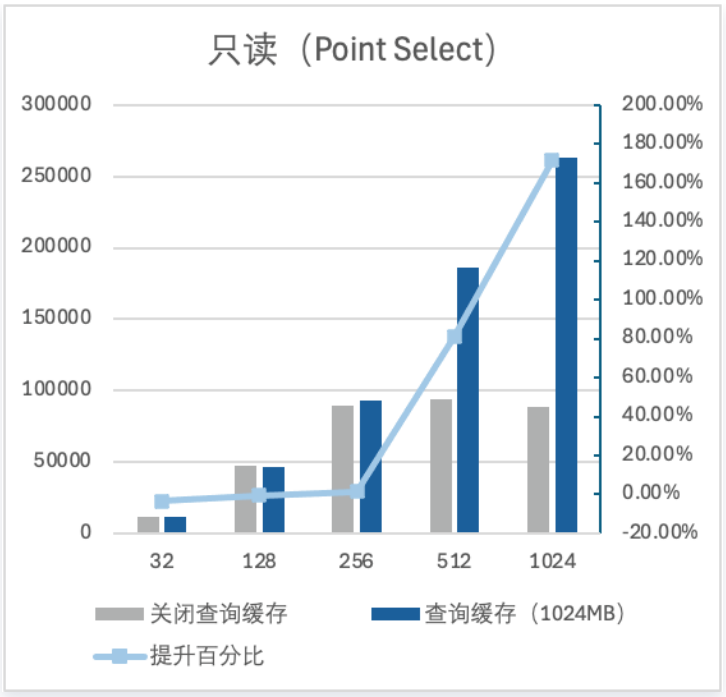
测试场景一：查询缓存对读场景的性能提升

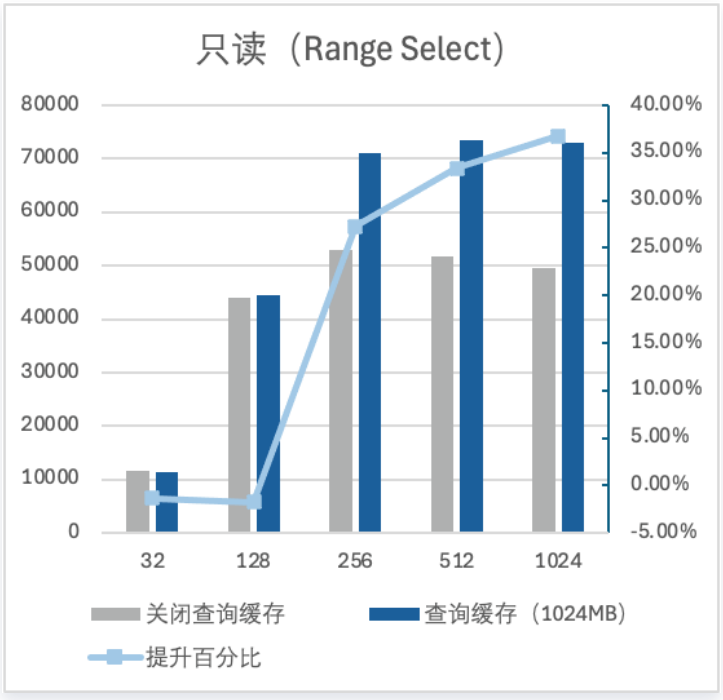
测试步骤

1. 开启查询缓存功能（设置参数 query_cache_type 为 ON），设置 query_cache_size 的值为1073741824（即1024MB）。
2. 在全缓存的场景下，在不同并发度下，记录集群的 QPS。Sysbench 的数据量约为1.4GB，共25000*250张表。
3. 在大 IO 场景下，在不同并发度下，记录集群的 QPS。Sysbench 的数据量约为54GB，共800000*300张表。

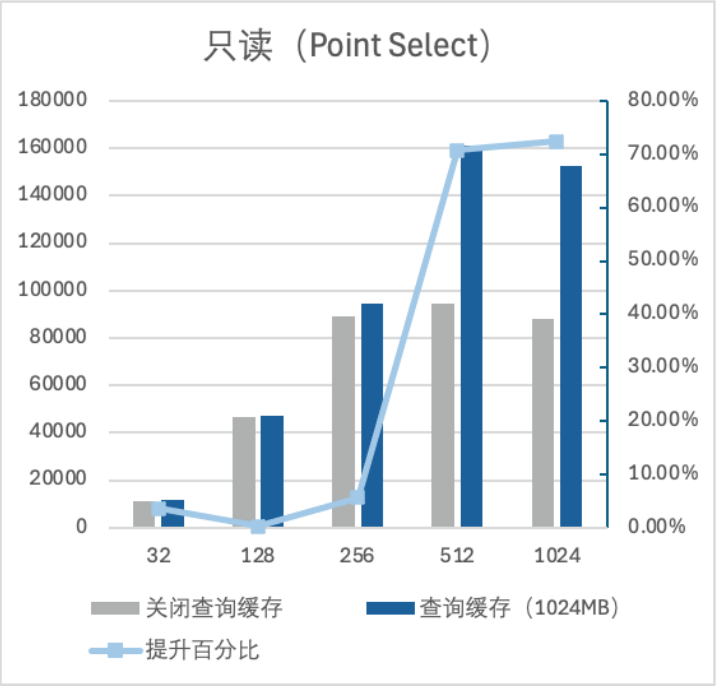
测试结果

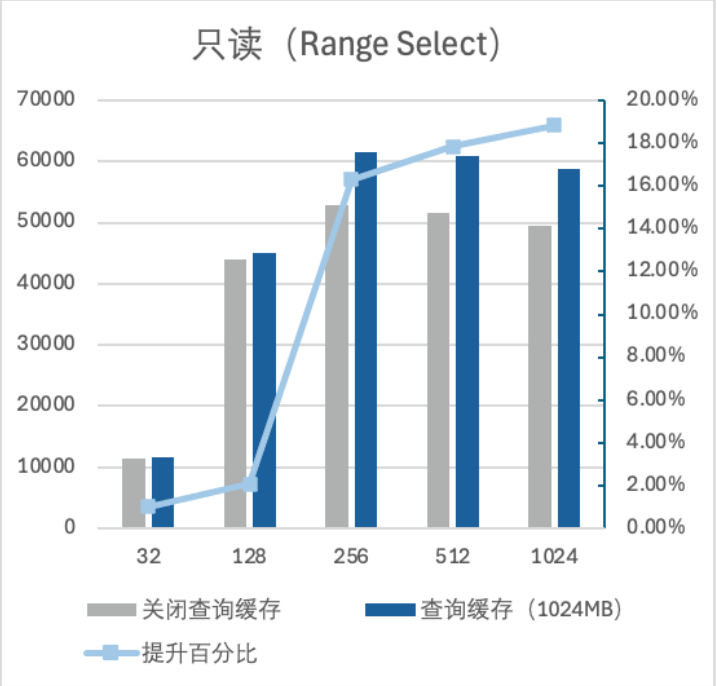
1. 全缓存场景下，机器内存为16GB，数据量大小为1.4GB。开启 Query Cache 后，点查可提升了**171%**左右的性能，范围查可提升了**36%**左右的性能。





2. **大 IO 场景下，机器内存为16GB，数据量大小为54GB。开启 Query Cache 后，点查可提升了36%左右的性能，范围查可提升了10%左右的性能。**





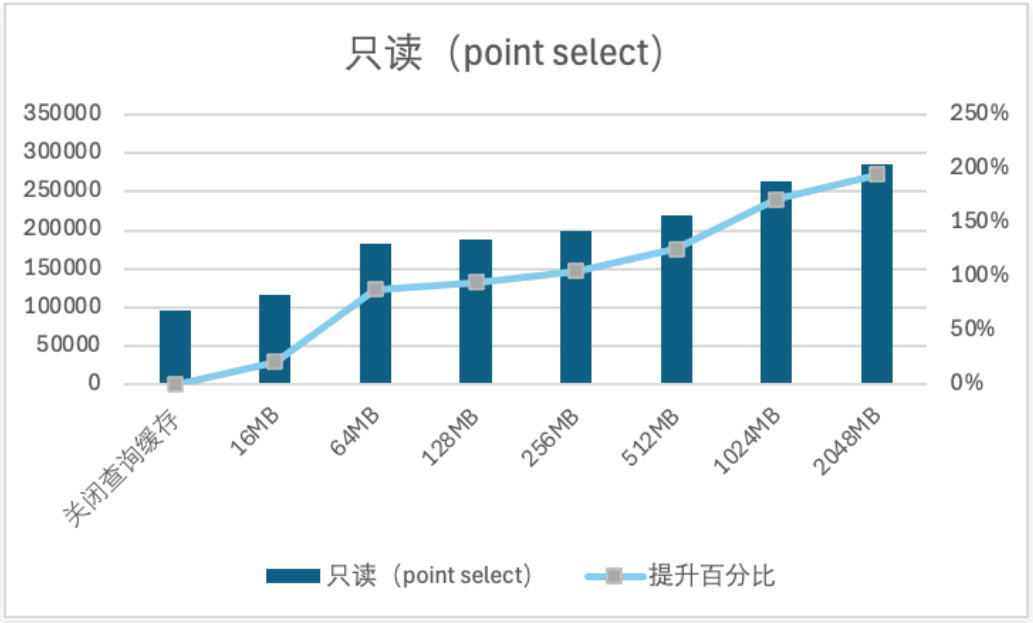
测试场景二：查询缓存大小对性能的影响

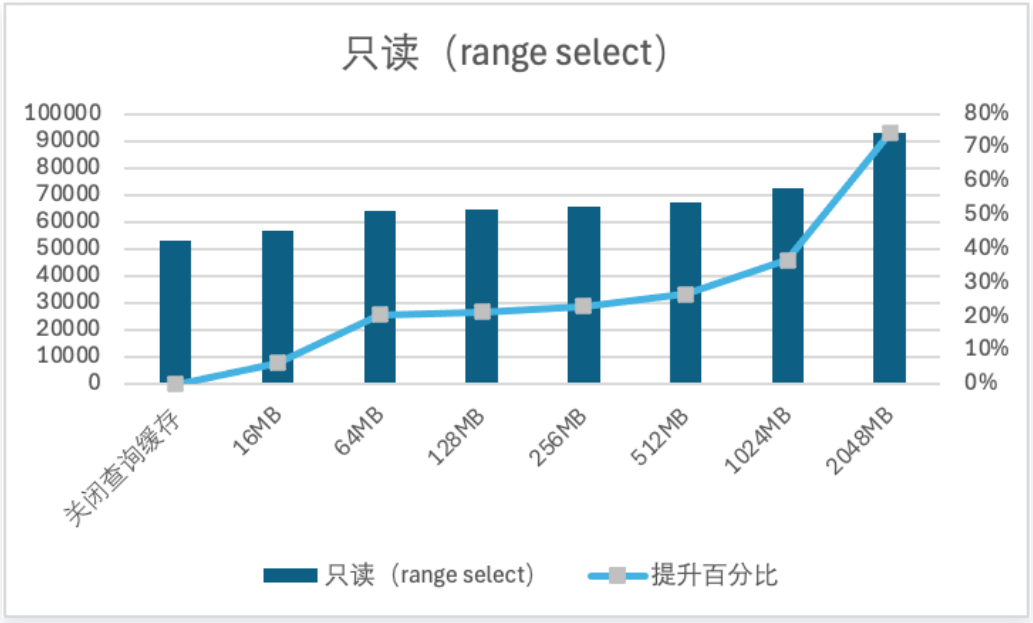
测试步骤

1. 设置并发度为1024，开启查询缓存（设置参数 query_cache_type 为 ON）。
2. 在全缓存场景下，调整不同查询缓存大小 query_cache_size，进行性能测试，记录集群的 QPS。
3. 在大 IO 场景下，调整不同查询缓存大小 query_cache_size，进行性能测试，记录集群的 QPS。

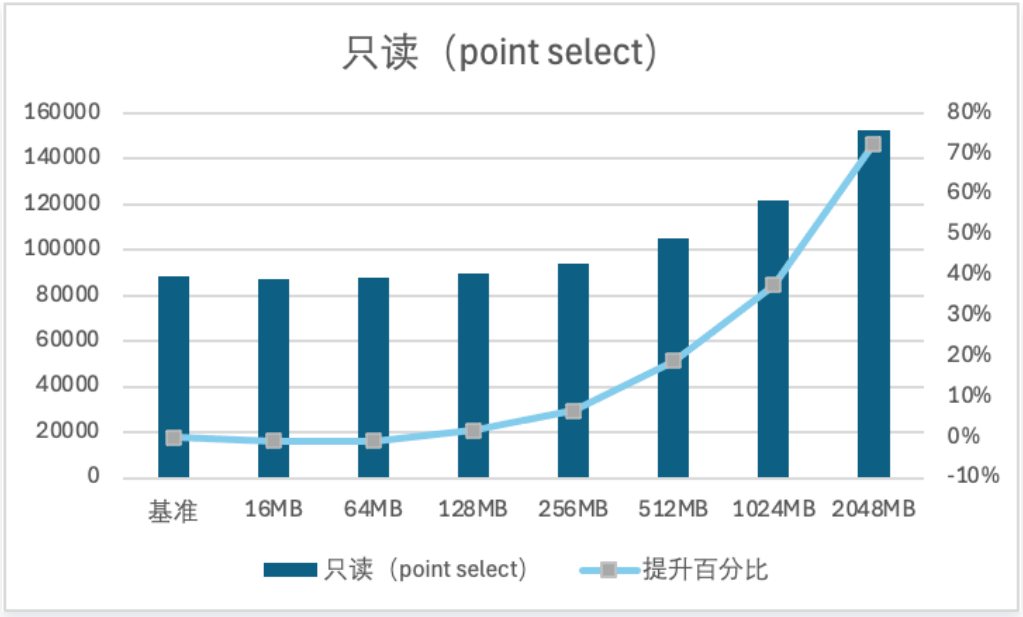
测试结果

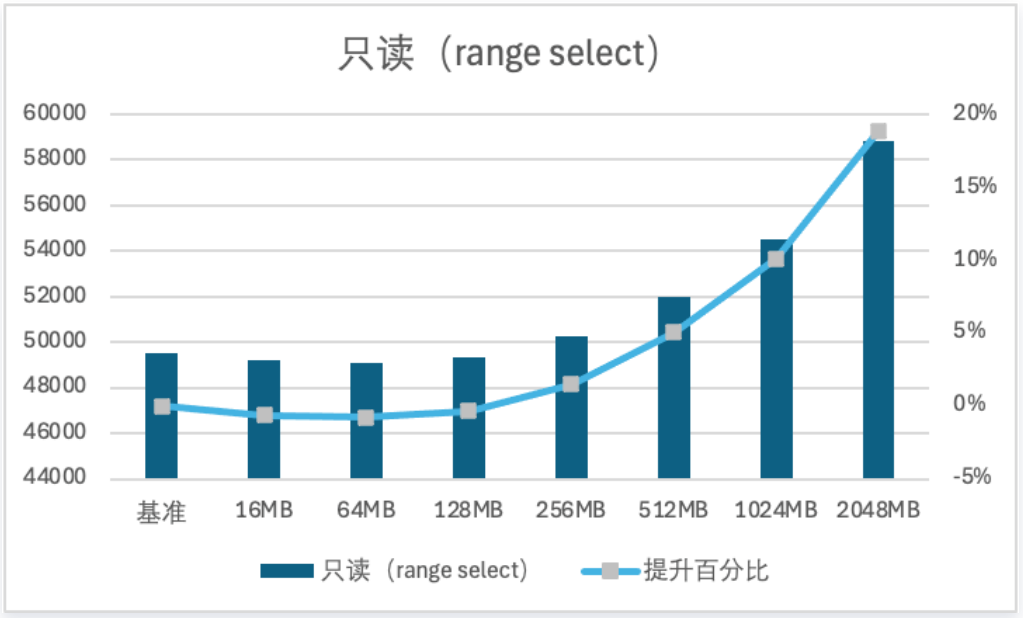
1. 全缓存场景下，随着查询缓存大小提升，性能逐渐上升。在查询缓存为2048MB时，对点查 Point Select 带来的提升约为**194%**，对范围查 Range Select 带来的提升为**74%**。





2. **大 IO 场景下，随着查询缓存大小提升，性能逐渐上升。在查询缓存为2048MB时，对点查 Point Select 带来的提升约为72%，对范围查 Range Select 带来的提升为19%。**


只读（point select）
160000
140000
120000
100000
80000
60000
40000
20000
0
80%
70%
60%
50%
40%
30%
20%
10%
0%
-10%
基准
16MB
64MB
128MB
256MB
512MB
1024MB
2048MB
只读（point select）
提升百分比


只读（range select）
60000
58000
56000
54000
52000
50000
48000
46000
44000
20%
15%
10%
5%
0%
-5%
基准
16MB
64MB
128MB
256MB
512MB
1024MB
2048MB
只读（range select）
提升百分比

列压缩

最近更新时间：2024-12-02 16:02:22

功能介绍

当前存在两种压缩方式：针对行格式的压缩和针对数据页面的压缩。然而，在处理一个表中既包含某些大字段又包含许多小字段的情况下，尤其是当频繁读写小字段而不常访问大字段时，这两种压缩方式都会产生不必要的计算资源浪费。
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，这样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 TXSQL 8.0 3.1.15.002及以上。

> **说明：**
> - 内核版本 TXSQL 8.0 3.1.15.002及以上的列压缩功能为默认开启。
> - 内核版本 TXSQL 5.7 暂不支持列压缩功能。

适用场景

表中如果存在一些大字段和许多小字段，且对小字段的读写操作较为频繁，而对大字段的访问较少，可以考虑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。
5. JSON。

语法如下：

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

或者

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

压缩阈值（Threshold）

压缩阈值通过参数 innodb_min_column_compress_length 控制，默认为256。如果列的原 size 超过此参数的取值，则进行压缩，否则只添加压缩 header，不对数据进行实际压缩。

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;`	N
`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列
`ALTER TABLE t1 MODIFY COLUMN a BLOB COMPRESSED;`	将非压缩列变为压缩列

参数说明

参数名	动态	类型	默认	参数值范围	说明
innodb_zlib_column_c	Ye	UIN	6	[0-9]	zlib 压缩，0表示不压缩，1表示最快的压缩，9表示压缩程度最大的压缩，从1到9压缩速度越

ompressio n_level	s	T			慢，但压缩率越高。
innodb_zst d_column_ compressi on_level	Y e s	UI N T	3	[1-22]	zstd 压缩，1表示最快的压缩，22表示压缩程度最大的压缩，从1到22压缩速度越慢，但压缩率越高。
innodb_mi n_column_ compress_ length	Y e s	UI N T	2 5 6	［1，UINT_MA X32］	控制压缩阈值，单位：字节，如果列的原长度大于或等于此参数的取值，则进行压缩，否则只添加压缩 header，实际不对数据进行压缩。

多算法支持（Multiple Algorithms）

支持 ZLIB、LZ4、ZSTD 三种压缩算法。也可以省略算法，省略后将默认算法为 ZLIB。
语法如下：

```
CREATE TABLE t1(
  id INT PRIMARY KEY,
  b BLOB COMPRESSED ALGORITHM = [ZLIB|LZ4|ZSTD]
);
```

或者

```
CREATE TABLE t1(
  id INT PRIMARY KEY,
  b BLOB COLUMN_FORMAT COMPRESSED ALGORITHM = [ZLIB|LZ4|ZSTD]
);
```

压缩算法与压缩程度选择（Compression Algorithms and Compression Levels）

1. ZLIB：目前提供了 ZLIB 的多压缩级别，参数为 innodb_zlib_column_compression_level，值为0-9，其中0表示不压缩，1表示最快的压缩，9表示压缩程度最大的压缩，默认为6。
2. LZ4：与 MySQL 的 Page 压缩保持一致，不支持 LZ4 的多程度压缩，使用 LZ4 压缩算法时需要注意，LZ4 压缩的最大原始长度为$2^{31}-1$，而 LONGBLOB 的最大长度为$2^{32}-1$，当压缩的原始数据长度大于等于2^{31}时，将会隐式地使用 ZLIB 进行压缩。
3. ZSTD：ZSTD 即 ZStandard 有三种压缩方式。此处列压缩支持的是不含字典的非 streaming 的普通压缩，提供了多压缩级别，参数为 innodb_zstd_column_compression_level，值为1-22，1表示最快的压缩，22表示压缩程度最大的压缩，默认为3。

压缩属性显示

这里展示默认压缩算法：ALGORITHM = ZLIB。

```
CREATE TABLE t2 (a VARCHAR(100) COMPRESSED) ENGINE=InnoDB;

SHOW CREATE TABLE t2;
```

注意事项

- 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 TDSQL-C MySQL 版内部是支持的。其他 MySQL 分支以及官方版本：
  - 官方版本号小于8.0.22，可以直接导入。
  - 官方版本号大于或等于8.0.22，需要在逻辑导出之后，去掉压缩关键字。
- DTS 导出其他云或者用户时，在 binlog 同步过程中可能会出现不兼容的问题，此时可以跳过带压缩关键字的 DDL 语句。
- 物理备份方面，由于备份时其字段在 innodb 内部是已经压缩的状态，因此使用该备份的版本也必须是带有列压缩的版本。

exchange subpartition template

最近更新时间：2024-09-05 21:47:01

功能介绍

exchange subpartition template 功能允许用户将一个分区的子分区模板交换到另一个分区中，同时将另一个分区的子分区模板交换到原来的分区中。这样，就可以快速更改分区表的分区策略，而无需重新创建表或者进行数据迁移。

支持版本

内核版本 TXSQL 8.0 3.1.15及以上。

适用场景

适用于需要快速交换表中数据的场景。

使用说明

使用语法

```
ALTER TABLE pt EXCHANGE SUBPARTITION TEMPLATE p WITH TABLE nt WITH
VALIDATION / WITHOUT VALIDATION
```

使用限制

- pt 和 nt 表不能有全局索引。
- pt 和 nt 表不能有触发器和 check 约束。
- pt 和 nt 表的分区类型以及子分区类型只支持 range 和 list。
- pt 和 nt 表的一级分区定义必须保持一致。

示例

```
-- 将 s20240111中的数据从 hash_range_1转到 hash_range_2

CREATE TABLE hash_range_1(
id INT NOT NULL,
fname VARCHAR(30),
hired date NOT NULL DEFAULT '9999-12-31',
primary key (id, hired))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY LIST(murmurHashCodeAndMod(id,2))
```

```
SUBPARTITION BY RANGE(tdsql_day(hired))
SUBPARTITION TEMPLATE
(SUBPARTITION s20240111 VALUES LESS THAN (20240111),
 SUBPARTITION s20240112 VALUES LESS THAN (20240112),
 SUBPARTITION s20240113 VALUES LESS THAN (20240113))
(PARTITION p1 VALUES IN (0),
 PARTITION p2 VALUES IN (1));

-- (hash_range_2与 hash_range_1完全相同)
CREATE TABLE hash_range_2(
id INT NOT NULL,
fname VARCHAR(30),
hired date NOT NULL DEFAULT '9999-12-31',
primary key (id, hired))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY LIST(murmurHashCodeAndMod(id,2))
SUBPARTITION BY RANGE(tdsql_day(hired))
SUBPARTITION TEMPLATE
(SUBPARTITION s20230111 VALUES LESS THAN (20240111),
 SUBPARTITION s20230112 VALUES LESS THAN (20240112),
 SUBPARTITION s20230113 VALUES LESS THAN (20240113))
(PARTITION p1 VALUES IN (0),
 PARTITION p2 VALUES IN (1));

-- 新建一个临时 temp 表，一级分区和表结构和 hash_range_1一致
CREATE TABLE temp(
id INT NOT NULL,
fname VARCHAR(30),
hired date NOT NULL DEFAULT '9999-12-31',
primary key (id, hired))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
PARTITION BY LIST(murmurHashCodeAndMod(id,2))
(PARTITION p1 VALUES IN (0),
 PARTITION p2 VALUES IN (1));

-- 语法演示
-- 第一步: hash_range_1--> temp
ALTER TABLE  hash_range_1
EXCHANGE subpartition template s20240111
with table temp with validation;
-- 第二步: temp --> hash_range_2
ALTER TABLE  hash_range_2
EXCHANGE subpartition template s20240111
```

```
with table temp with validation;
```

数据校验说明

同官方一样，显式指定 WITH VALIDATION 来进行数据校验或者 WITHOUT VALIDATION 不进行数据校验。默认进行数据校验。
数据校验主要是验证 nt 表中一级分区的数据是否满足在 pt 表中对应一级分区下，二级分区的定义。在上面的示例中，如果第一步 temp 表 p1 分区有 hired 列存在数据满足 tdsql_day(hired)>=20240111，那么将不满足二级分区定义。SUBPARTITION s20240111 VALUES LESS THAN (20240111) 数据校验将会失败。

buffer pool 隔离

最近更新时间：2024-10-18 16:34:52

功能介绍

Innodb 的 buffer pool（BP）通过在 midpoint 插入新页的方式尽可能避免了全表扫描等操作对 BP 的污染，但效果有限。为了进一步提高 BP 的性能，该功能支持在 BP 中设置独立的一块空间，专门用来执行类似全表扫描的操作。当没有全表扫描操作时，这个空间可以完全被正常的 LRU list 使用。而当存在全表扫描操作时，使用 CLOCK 淘汰算法管理这个空间，通过数据页面的物理隔离，避免了全表扫描操作对 BP 的污染。

支持版本

内核版本 TXSQL 8.0 3.1.15及以上。

适用场景

适用于需要频繁对数据库进行全表扫描和进行大规模数据操作的场景。

使用说明

参数说明

参数名	动态	类型	默认	参数值范围	说明
innodb_txsql_independent_buffer_pool_evict_interval	yes	ulong	50	0-50	后台线程主动淘汰 CLOCK list 的时间单位，越小 CLOCK list 页面在内存中驻留越久，设置为0可以快速清空 CLOCK list。
innodb_txsql_independent_buffer_pool_list_move_action	yes	ulong	0	0-2	CLOCK list 页面被正常 query 读到后的行为： • 0代表不做任何变化。 • 1代表同步移动到 LRU list。 • 2代表异步移动到 LRU list，移动操作交给后台线程。
innodb_txsql_independent_buffer_pool_size_pct	yes	ulong	5	1-100	隔离 BP 能使用的 BP 最大比例。比例越高，对正常 query 的影响

					越大，但是全表扫描的操作效果越高。
innodb_txsql_independent_buffer_pool_users	yes	string	nullptr		通过指定用户的方式使用 BP 隔离，具体配置方式为"user1@ip1;user2@ip2"。
innodb_txsql_independent_buffer_pool_enabled	yes	bool	ON	ON/OFF	buffer pool 隔离的开关，关闭后不会再有新的页面进入隔离空间，旧的隔离页面将尽快淘汰。
innodb_txsql_independent_buffer_pool_max_expire_minutes	yes	bool	120	1-1440	CLOCK list 页面在 BP 隔离中最大淘汰时间。控制逻辑：use_times 代表页面的活跃度，是 CLOCK 算法的一个重要指标。页面每次被读取则将其加1，后台线程每隔一个时间单位将所有页面的 use_times 减1。innodb_txsql_independent_buffer_pool_max_expire_minutes 用于控制 use_times 的上限。

新增状态如下：

参数	类型	说明
txsql_independent_buffer_pool_usage_counts	longlong	使用 BP 隔离的 SQL 数量。

BP 隔离的使用方法1

使用名为 independent 的 hint 手动触发对 BP 隔离的使用，具体使用方法为：在 DML 的 INSERT、DELETE、UPDATE 等关键词后立刻添加 `/*+ independent /` ，例如：`select /*+ independent */ id from t;` 需要注意的是将 `/*+ independent */` 加在非前述的任意位置皆无法触发对 BP 隔离的使用，例如：`select id /*+ independent */ from t;` 。

BP 隔离的使用方法2

通过 innodb_txsql_independent_buffer_pool_users 设置默认使用 BP 隔离的用户，此类用户简称为 BP 隔离用户。在 innodb_txsql_independent_buffer_pool_users 中新增、修改、删除 BP 隔离用户后，该用户是否默认使用 BP 隔离的行为只在新连接生效，已建立连接的行为不受影响。为方便排障，在 `show detail processlist;` 中新增 Independent_buffer_pool_session 状态来表示连接是否默认使用 BP 隔离。

BP 隔离与 Prepare statement、Stored Procedure 的关系

- Prepare statement、Stored Procedure 都支持通过 方法1 使用 BP 隔离。
- Prepare statement、Stored Procedure 是否使用 BP 隔离与创建的用户无关，只与执行的用户有关。

BP 隔离的观察

BP 隔离功能在 show engine innodb status 中添加了对 BP 隔离空间（CLOCK list）的监控。在 BUFFER POOL AND MEMORY 模块中添加 CLOCK list 长度和分布的情况。具体而言，use_times 代表页面的活跃度，是 CLOCK 算法的一个重要指标。页面每次被读取则将其加1，后台线程每隔一个时间单位将所有页面的 use_times 减1，当 use_times 为0时将页面淘汰或刷盘。在 BUFFER POOL AND MEMORY 模块中对 use_times 相同的页面进行了聚类，分为 [0,10)，[10, 100)，[100, 1000)，[1000, 10000)，[10000, 100000)，[100000, unlimited) 6个范围。

Nonblocking DDL

最近更新时间：2025-03-18 17:38:12

功能介绍

当一条 DML 语句执行时，如果持有 metadata lock，就会阻塞对应表的 DDL，导致 DDL 被阻塞后，后续的所有 DML 语句都会被阻塞，这可能会消耗大量的线程资源。为了避免这种情况，TDSQL-C MySQL 版引入了 Nonblocking DDL 机制，保护系统不会被等待 MDL 锁的 DDL 操作卡住。当 DDL 操作被 DML 操作阻塞时，后续的 DML 不会被等待的 MDL 锁进一步阻塞。

支持版本

内核版本 TXSQL 8.0 3.1.15.001及以上。

适用场景

适用于避免 DML 语句执行时，由于持有 metadata lock，导致阻塞对应表的 DDL 以及后续的所有 DML 语句的场景。

使用说明

参数名	动态	类型	默认	参数值范围	说明
txsql_nonblock_ddl	yes	bool	OFF	OFF/ON	是否开启 Nonblocking DDL 功能。
txsql_nonblock_ddl_retry_times	yes	ulong	1	0 - 18446744073709551615	DDL 操作被 MDL 锁阻塞后的重试次数，默认值为 1。
txsql_nonblock_ddl_retry_interval	yes	ulong	2	0 - 18446744073709551615	DDL 操作被 MDL 锁阻塞后的重试间隔，默认值为 2s。

注意：
部分非常见 DDL 操作可能覆盖不到，这些场景下，如果 DDL 遇到 DML 阻塞，会直接返回 Lock wait timeout。

二级分区

最近更新时间：2024-10-18 16:31:52

功能介绍

在数据库中，分区是一种可以将表或索引数据分成多个逻辑部分的技术，可以提高查询效率、减少维护成本等。而二级分区则可以更加细粒度地对数据进行划分，如在一个分区内再次划分出多个子分区，可以提高数据的管理和查询效率。TDSQL-C MySQL 版支持创建 range 或 list 类型的二级分区。

支持版本

内核版本 TXSQL 8.0 3.1.15及以上。

适用场景

如需更加细粒度地对数据进行管理和查询，可以使用二级分区来提高查询的效率，如对大表进行分区。

注意事项

- 新增模板语法 SUBPARTITION TEMPLATE，因此每个子分区的定义必定是一致的。
- 每个子分区的名字是：一级分区名 $$ 模板二级分区名。
- 二级分区不支持 column_list。
- truncate ··· with global index 暂时不支持 truncate 所有的分区，如果要 truncate 所有的分区，可以执行 truncate table。
- 包含全局索引的分区表，在 truncate partition 时跟官方保持一致，均不是 online ddl，但是包含全局索引的分区在 truncate partition 时需要维护全局索引，运行时间更长。建议使用 delete 的方式替代 truncate。
- 在模板二级分区名中不能存在 $$ 字符。
- drop partition 需要重建全局索引，推荐多个 drop partition 合并成一个语句执行，减少维护全局索引的代价。
- 不推荐使用 truncate partition 操作，会阻塞 DML。

使用说明

1. 创建包含二级分区的表。

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]

partition_options:
```

```
    PARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1 | 2}] (column_list)
        | RANGE{(expr) | COLUMNS(column_list)}
        | LIST{(expr) | COLUMNS(column_list)} }
    [PARTITIONS num]
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1 | 2}] (column_list) }
      [SUBPARTITIONS num]
    ]
    [SUBPARTITION BY
      {   RANGE{(expr)}     -- range二级分区模板
        | LIST{(expr)}        -- list二级分区模板
        SUBPARTITION TEMPLATE [(subpartition_definition [,
subpartition_definition] ...)]} -- 子分区模板
    ]
    [(partition_definition [, partition_definition] ...)]

subpartition_definition:
  SUBPARTITION logical_name
        [VALUES
           { LESS THAN {(expr | value_list) | MAXVALUE} --Range子分区范
围
           | IN (value_list)}]  --List子分区范围
      [[STORAGE] ENGINE [=] engine_name]
      [COMMENT [=] 'string' ]
      [DATA DIRECTORY [=] 'data_dir']
      [INDEX DIRECTORY [=] 'index_dir']
      [MAX_ROWS [=] max_number_of_rows]
      [MIN_ROWS [=] min_number_of_rows]
      [TABLESPACE [=] tablespace_name]
```

例如：

```
CREATE TABLE `t1` (
  `id` int DEFAULT NULL,
  `purchased` int DEFAULT NULL,
  KEY `idx` (`id`,`purchased`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
/*!50100 PARTITION BY RANGE (`id`)
SUBPARTITION BY RANGE (`purchased`)
SUBPARTITION TEMPLATE
```

```
(SUBPARTITION s0 VALUES LESS THAN (10) ENGINE = InnoDB,
 SUBPARTITION s1 VALUES LESS THAN (20) ENGINE = InnoDB)
(PARTITION p0 VALUES LESS THAN (10) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (20) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (30) ENGINE = InnoDB) */;
```

2. ALTER TABLE 语法。

```
-- 二级分区的支持
ALTER TABLE tbl_name
    [alter_option [, alter_option] ...]
    [partition_options] [subpartition_options] -- 新增
subpartition_options

subpartition_options:
    subpartition_option [subpartition_option] ...

subpartition_options: {
    MODIFY PARTITION partition_name TRUNCATE SUBPARTITION TEMPLATE
subpartition_template_name; -- truncate 分区partition_name中模板名为
subpartition_template_name的分区
  | TRUNCATE SUBPARTITION TEMPLATE subpartition_template_name --
truncate subpartition_template_name的二级分区
  | ADD SUBPARTITION TEMPLATE subpartition_definitions -- 新增
subpartition_definitions模板
  | DROP SUBPARTITION TEMPLATE subpartition_template_name -- drop
subpartition_template_name的二级分区
}

-- 一级分区DDL
ALTER TABLE t1 TRUNCATE p0 WITH GLOBAL INDEX; -- truncate p1分区（若存在
子分区，则truncate下面所有子分区）
```

例如：

```
-- 二级分区 DDL
ALTER TABLE t1 MODIFY PARTITION p0 TRUNCATE SUBPARTITION TEMPLATE s1;
-- truncate 二级分区p0_s1（一级分区p0，二级分区模板名s1）
ALTER TABLE t1 TRUNCATE SUBPARTITION TEMPLATE s1; --  所有分区后缀为_s1
的子分区都将被truncate
ALTER TABLE t1 ADD SUBPARTITION TEMPLATE （SUBPARTITION s2 values in
(1,2,3,4)); -- 所有分区将添加后缀为_s2的的子分区
```

```
ALTER TABLE t1 DROP SUBPARTITION TEMPLATE s2;  -- 所有分区将删除后缀为_s2
的子分区;

-- 一级分区 DDL
ALTER TABLE t1 ADD PARTITION (PARTITION p2 VALUES LESS THAN (20));  --
添加分区，若t1是二级分区表，这里将默认使用模板来生成子分区
ALTER TABLE t1 DROP PARTITION p1;
ALTER TABLE t1 TRUNCATE p0 WITH GLOBAL INDEX; -- truncate p1分区（若存在
子分区，则truncate下面所有子分区）
```

3. 支持新的时间函数。
   - tdsql_year，tdsql_month，tdsql_day，将时间转为 YYYY，YYYYMM，YYYYMMDD 的格式，支持的类型包含 DATE/DATETIME/TINYINT/SMALLINT/MEDIUMINT/BIGINT/CHAR/VARCHAR/VARBINARY/timestamp/binary。
   - 当 tdsql_year，tdsql_month，tdsql_day 作为分区键中的函数时，不支持 binary、timestamp 类型。

自动 kill 空闲事务

最近更新时间：2024-10-14 10:31:11

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 8.0 3.1.10 及以上。

适用场景

对于处于开启事务状态的连接（显式使用 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]	单位：秒，参数值设置为0代表关闭该功能；设置为非0代表启用该功能，并会 kill 掉所设置的参数值时间的空闲事务。

动态线程池

最近更新时间：2024-11-28 17:54:32

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

- 参数 thread_handling 的设置可以切换服务器用于连接线程的线程处理模型，此参数值由 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。
- 参数 thread_handling 的设置可以切换服务器用于连接线程的线程处理模型，此参数值由 one-thread-per-connection 切换为 pool-of-threads 的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	说明
thread_pool_idle_timeout	Yes	uint	60	[1, UINT_MAX]	worker 线程在没有需要处理的网络事件时，最多等待此时间（单位秒）后销毁
thread_pool_oversubscribe	Yes	uint	3	[1,1000]	在一个工作组中最多允许多少个 worker
thread_pool_size	Yes	uint	当前机器 CPU 个数	[1,1000]	线程组个数
thread_pool_stall_limit	Yes	uint	500	[10, UINT_MAX]	每间隔此时间（单位毫秒）timer 线程负责遍历检查一次所有线程组。 当线程组没有 listener、高低优先级队列非空并且没有新增的 IO 网络事件时，认为线程组处于 stall 状态，timer 线程

					负责唤醒或创建新的 worker 线程来缓解该线程组的压力
thread_pool_max_threads	Yes	uint	100000	[1,100000]	线程池中所有 worker 线程的总数
thread_pool_high_prio_mode	Yes, session	enum	transactions	transactions\statement\none	高优先级队列工作模式，包括三种： • transactions：只有一个已经开启了事务的 SQL，并且 thread_pool_high_prio_tickets 不为0，才会进入到高优先级队列中，每个连接在 thread_pool_high_prio_tickets 池被放到优先队列中后，会移到普通队列中 • statement：所有连接都被放入高优先级队列中 • none：与 statement 相反，所有连接都被放入低优先级队列中
thread_pool_high_prio_tickets	Yes, session	uint	UINT_MAX	[0, UINT_MAX]	transactions 工作模式下，给每个连接授予的 tickets 大小
threadpool_workaround_epoll_bug	Yes	bool	false	true/false	是否绕过 linux2.x 中的 epoll bug，该 bug 在 linux3 中修复

`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：2024-10-14 10:31:11

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。

支持 returning

最近更新时间：2024-10-14 10:31:11

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 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)
```

闪回查询

最近更新时间：2024-11-13 10:37:42

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 5.7 2.1.13.001及以上。
- 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';
```

闪回查询还支持了**持久化**，支持将闪回查询内存中的 readview 信息定期持久化到物理表中，重启后仍然可以执行重启前的闪回查询。此能力需要开启闪回查询，并打开闪回查询持久化开关（打开持久化开关：设置 innodb_backquery_persistent = ON），否则无效。

> **说明：**
> 要使用闪回查询持久化能力，需满足内核版本在 TDSQL-C MySQL 版 8.0 3.1.15 及以上。

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	说明
innodb_backquery_enable	全局参数	Boolean	OFF	ON/OFF	否	闪回查询功能的开关。
innodb_backquery_window	全局参数	Integer	900	1 - 86400	否	支持闪回查询的时间范围，单位：秒，此参数的值越大，闪回查询支持的历史数据查询时间越长，同时 undo 表空间占用的存储空间也会上升。
innodb_backquery_history_limit	全局参数	Integer	8000000	1 - 9223372036854775807	否	undo 的历史链表长度限制，超过设定值会忽略 Innodb_backquery_window 触发 purge，直到历史链表长度低于设定值。

				4476000		
innodb_backquery_persistent	非全局参数	Boolean	OFF	ON/OFF	否	闪回查询持久化的开关。

性能类特性
columns 语法

最近更新时间：2024-09-05 21:47:01

功能介绍

columns 语法用于查询表的列信息。

支持版本

内核版本 TXSQL 8.0 3.1.15及以上。

适用场景

适用于查询表的列信息较多，需提高查询性能的场景。

使用说明

List columns 的语法

- columns 为单列

```
CREATE TABLE `t1` (
  `id` int DEFAULT NULL,
  `purchased` varchar(12) DEFAULT NULL,
  KEY `idx` (`id`,`purchased`) GLOBAL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
PARTITION BY RANGE (`id`)
SUBPARTITION BY LIST COLUMNS (`purchased`)
SUBPARTITION TEMPLATE
(SUBPARTITION s0 VALUES IN ('0', '1', '2') ENGINE = InnoDB,
 SUBPARTITION s1 VALUES IN ('5', '6', '8') ENGINE = InnoDB)
(PARTITION p0 VALUES LESS THAN (1990) ,
 PARTITION p1 VALUES LESS THAN (1999));
```

- columns 为多列

```
CREATE TABLE `t2` (
  `id` int DEFAULT NULL,
  `purchased` varchar(12) DEFAULT NULL,
  KEY `idx` (`id`,`purchased`) GLOBAL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

```
    PARTITION BY RANGE (`id`)
    SUBPARTITION BY LIST COLUMNS (`purchased`, `id`)
    SUBPARTITION TEMPLATE
    (SUBPARTITION s0 VALUES IN (('0', 1), ('1', 1), ('2', 1995))
ENGINE = InnoDB,
     SUBPARTITION s1 VALUES IN (('5' ,5), ('6', 6)) ENGINE = InnoDB)
    (PARTITION p0 VALUES LESS THAN (1990) ,
     PARTITION p1 VALUES LESS THAN (1999));
```

Range columns 的语法

- columns 为单列

```
    CREATE TABLE `t3` (
      `id` int DEFAULT NULL,
      `purchased` varchar(12) DEFAULT NULL,
      KEY `idx` (`id`,`purchased`) GLOBAL
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
    PARTITION BY RANGE (`id`)
    SUBPARTITION BY RANGE COLUMNS (`purchased`)
    SUBPARTITION TEMPLATE
    (SUBPARTITION s0 VALUES LESS THAN ('5') ENGINE = InnoDB,
     SUBPARTITION s1 VALUES LESS THAN ('8') ENGINE = InnoDB)
    (PARTITION p0 VALUES LESS THAN (1990) ,
     PARTITION p1 VALUES LESS THAN (1999));
```

- columns 为多列

```
    CREATE TABLE `t4` (
      `id` int DEFAULT NULL,
      `purchased` varchar(12) DEFAULT NULL,
      KEY `idx` (`id`,`purchased`) GLOBAL
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
    PARTITION BY RANGE (`id`)
    SUBPARTITION BY RANGE COLUMNS (`purchased`, `id`)
    SUBPARTITION TEMPLATE
    (SUBPARTITION s0 VALUES LESS THAN ('5', 55) ENGINE = InnoDB,
     SUBPARTITION s1 VALUES LESS THAN ('8', 88) ENGINE = InnoDB)
    (PARTITION p0 VALUES LESS THAN (1990) ,
     PARTITION p1 VALUES LESS THAN (1999));
```

支持的 columns 数据类型

支持的 columns 数据类型，请参见 官方文档 。

裁剪说明

二级分区的裁剪方式和一级分区裁剪是一致，并且是独立的。即一级裁剪过滤 where 条件中的 partition key part 获取裁剪后 partition 集合 s1，二级裁剪过滤 where 条件中的 subpartition key part 获取裁剪后 subpartition template 集合 s2，最终的结果是每个 s1 中的 s2 被选中。

限制说明

新增参数 txsql_subpartition_support_multiple_columns 限制多列的使用，关闭时只允许使用单列语法，打开时允许多列和单列语法。

参数名	动态	类型	默认	参数值范围	说明
txsql_subpartition_support_multiple_columns	yes	bool	OFF	OFF/ON	限制多列的使用，关闭时只允许使用单列语法，打开时允许多列和单列语法。

关联子查询结果缓存

最近更新时间：2024-09-05 21:47:01

功能介绍

鉴于每次执行关联子查询时都需要重新计算子查询的结果，这会导致性能较低。TXSQL 当前支持了关联子查询结果缓存功能（subquery cache），可以缓存子查询的结果，从而避免重复计算，提高查询性能。

支持版本

内核版本 TXSQL 8.0 3.1.15及以上。

适用场景

适用于执行关联子查询较多，需提高查询性能的场景。

使用说明

参数名	动态	类型	默认	参数值范围	说明
txsql_subquery_cache_enabled	yes	Boolean	OFF	ON/OFF	控制是否打开 subquery cache 功能。
txsql_subquery_cache_cost_threshold	yes	Integer	4000	0-DBL_MAX	subquery cache 功能执行代价阈值，只有估计执行代价高于阈值的关联子查询才会使用 subquery cache 功能。

新增状态

新增状态，统计子查询缓存的 cache 命中情况：Txsql_subquery_cache_hit，Txsql_subquery_cache_miss。
使用 `explain format=tree` 可以观察到 Partial result cache 计划。

全局索引

最近更新时间：2024-10-18 16:29:32

功能介绍

支持对分区表创建全局二级索引。

支持版本

内核版本 TXSQL 8.0 3.1.15及以上。

适用场景

在分区表中通过创建全局二级索引，来提高分区表查询的效率和性能。

注意事项

- DDL 使用限制（以下是 官网文档 的 partition_option 中不支持的 DDL 操作）。
  - DISCARD PARTITION 和 IMPORT PARTITION：因为全局索引和 partition 不在一个表空间中。
  - COALESCE PARTITION：当表为无主键表时，执行此操作全局索引会出现重复记录。
  - REORGANIZE PARTITION：可能导致全局索引出现重复数据。
  - EXCHANGE PARTITION：全局索引和 partition 不在一个表空间中，全局索引无法交换数据。
  - REBUILD PARTITION：无主键分区表可能出现重复记录。
  - REPAIR PARTITION：可能导致全局索引丢失数据。
- 全局索引不能作为主键索引，包括隐式转为主键的情况（不包含主键时，unique 的全局索引无法创建）。
- unique 的全局索引必须包含所有的分区字段。
- 全局索引暂不支持需要 copying data 的分区级别 DDL。
- 全局索引不支持压缩表（修改 KEY_BLOCK_SIZE）及透明页压缩。
- 全局索引不支持非分区表，非分区表创建全局索引时会自动转为普通索引。
- 包含全局索引的 hash 分区不支持 add、coalesce 操作。
- 包含全局索引的分区表在 truncate 分区时必须使用 `alter table truncate partition with global index` 的语法。
- 全局索引中不能包含 Generated Columns 。
- 全局索引的 AHI 被默认禁止。
- 推荐使用 simple select 方式来访问全局索引，不建议使用特殊用法。

使用说明

建立全局索引表

```
create table t1 (
  a bigint unsigned not null  PRIMARY KEY,
  b varchar(16) not null,
  pad varchar(128) not null,
  key key_b(b) global   -- 增加 global 关键字，在建表中指定全局索引
)
PARTITION BY RANGE(a) (
  PARTITION p0 VALUES LESS THAN (10000000),
  PARTITION p1 VALUES LESS THAN (20000000),
  PARTITION p2 VALUES LESS THAN (30000000),
  PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

在已经存在的表中建立全局索引

```
create index key_b on t1(b) global; -- 增加 global 关键字，在建索引中指定全局索引
alter table t1 add index key_b(b) global;
```

binlog 写入优化

最近更新时间：2024-10-12 17:06:51

功能介绍

在 row 模式下，单个语句更新多行的大事务每行会生成1个 event，会产生大量 binlog，导致数据库性能下降。腾讯云内核团队通过对大事务场景的分析和优化，开发了该能力，binlog 写入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，并支持使用正则表达式设置生效的库表，从而减少 binlog 并提升数据库的使用性能。

支持版本

内核版本 TDSQL-C MySQL 版8.0 3.1.12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。

使用说明

binlog 写入优化功能是基于 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。

- txsql_optimize_large_trans_binlog_mode 为该功能的开关。
- 当 txsql_optimize_large_trans_binlog_mode 设置为 PART 时，只对满足 txsql_optimize_large_trans_binlog_tables 设置的正则表达式的表生效。例如：

```
set global txsql_optimize_large_trans_binlog_tables =
"test.t3,test.t4";
set global txsql_optimize_large_trans_binlog_tables = "test.t*";
set global txsql_optimize_large_trans_binlog_tables = "somedb*.t1";
```

参数说明

名称	是否全局	类型	默认	取值范围	说明
txsql_optimize_large_trans_binlog_mode	全局	enum	OFF	OFF/ALL/PART	binlog 写入优化的开关，ALL 表示对所有表生效大事务 binlog 优化，PART 表示对正则生效，OFF 表示关闭功能。

txsql_optimize_large_trans_binlog_tables	全局	char	*.*	–	• *.* 指 PART 模式下指定 dbname.tablename。 • dbname 和 tablename 均支持正则，如"somedb.t1"，表示库 somedb 和 表 t1均支持正则。 • 若设置为 test.*，则表示 test 库下的所有表均支持正则，包括设置后新增到 test 库的表。

大事务提交 binlog 优化

最近更新时间：2024-10-12 17:18:51

背景

在数据库中执　　事务时，其他事务的执　会变慢。由于事务执　成功的标志是其提交成功，　事务提交需要等待其 binlog 落盘，binlog 是按顺序写　　件的，且要求逐个完整的排队写　，即同　时间只能写　个事务，当某个事务特别　时，例如100G，其 binlog 落盘的时间就会很　，此时需要提交的其他事务就需要等待该大事务写完，才能接着写入，因此其他事务的执　会很慢。

功能介绍

在大事务提交的时候，会写入大量 binlog 日志，导致其他事务长时间无法提交。腾讯云内核团队针对此场景进行了优化，将大事务产生的 binlog 日志重定向到新的临时文件，避免因大事务提交阻塞其他事务而产生额外的处理时间，当该大事务提交后，会更新临时文件为正式的 binlog 日志序列，并更新原数据映射关系。

支持版本

内核版本 TDSQL-C MySQL 版8.0 3.1.12及以上。

适用场景

在数据库中执行大事务以及其他事务场景下，用于缩短大事务提交阻塞其他事务的时间。

使用说明

通过 txsql_non_blocking_binlog_threshold 参数使用该功能，将该参数值设置为非默认值，即可使用该功能，详细参数说明如下：

名称	是否全局	类型	默认	取值范围	说明
txsql_non_blocking_binlog_threshold	全局	ulong	UINT64_MAX	134217728-UINT64_MAX	当事务的日志量大于等于这个值（单位：字节），则事务将使用大事务提交 binlog 优化的方式进行提交。

purge binlog 性能优化

最近更新时间：2024-10-12 10:30:32

功能介绍

在数据库中，binlog 是用于记录数据库操作的日志文件，包括数据库的增删改操作等。随着时间的推移，binlog 文件会越来越多，占用大量磁盘空间，因此需要定期进行清理，也就是 purge binlog 操作。然而，purge binlog 和正常的 rotate 以及写事务会发生冲突，尤其是当 binlog 文件数量很多时，会导致在 purge binlog 操作时容易引起性能抖动。腾讯云内核团队通过对清理 binlog 文件场景的分析，对 purge binlog 的性能进行了优化。

- **优化 purge binlog 与 rotate binlog 的 LOCK_index 冲突，避免 rotate binlog 读取不到锁而无法继续 binlog 写入。**

  当前在 purge binlog 持有 index lock 期间，binlog 达到了256M后需要 rotate binlog 时读取不到锁，导致写入停止，直到获取到 index lock 后才能继续。针对上述问题，主要优化为，如果 rotate binlog 读取不到 index lock，将不会 rotate binlog，将会继续写 binlog 到当前 binlog 文件。之后不断尝试获取 index lock，直到读取到 index lock 才会 rotate binlog。
- **减少了需要 purge 的 binlog 文件读取量，从而减少了锁。**

  purge binlog 过程需要读取所有需要 purge 的 binlog 里的 Gtid 信息，然后写入到 mysql.gtid executed 表，所以 purge 期间的会一直持有 gtid 的排他锁，导致正常事务不能获取到 gtid 锁。针对上述问题，主要优化为，只读取 binlog 列表里的最后一个 binlog 文件里的 gtid，从而减少了 gtid 锁的持有时间，减少对正常事务的影响。
- **binlog 单个文件上限提高到10G，从而大量　少 rotate binlog 的次数。**

  默认 binlog 大小为256MB，5000QPS压力下，大约每1.4s rotate 一次，每次 rotate 耗时10~20ms。当前 max_binlog_size 支持从256MB提升到10G（开源 MySQL 最大支持1G），同样压力下，56s rotate 一次，从而减少了 index 锁的获取频次。当前策略是设置为1G，可以根据业务压力的需要，进一步调整 max_binlog_size 的参数值。

支持版本

- 内核版本 TDSQL-C MySQL 版8.0 3.1.12及以上。
- 内核版本 TDSQL-C MySQL 版5.7 2.1.12及以上。

适用场景

在 binlog 文件数量很多时，执行 purge binlog 操作产生锁争用导致业务 binlog 写入阻塞的场景。

使用说明

通过参数使用该功能，将参数 txsql_binlog_rotate_try_lock_index，txsql_binlog_rotate_try_lock_log 设置为 ON，并设置参数 txsql_binlog_purge_check_file_count 的值为10，即可使用该能力，详细参数说明如下：

名称	是否全局	类型	默认	取值范围	说明
txsql_binlog_rotate_try_lock_index	全局	bool	ON	ON/OFF	设置为 ON 后，执行 binlog rotate 时如果读取不到 index 文件的锁，则放弃本次 rotate。
txsql_binlog_rotate_try_lock_log	全局	bool	ON	ON/OFF	设置为 ON 后，执行 binlog rotate 时如果读取不到 binlog 的锁，则放弃本次 rotate。
txsql_binlog_purge_check_file_count	全局	ulonglong	10	0-UINT64_MAX	执行 purge binlog 时会从 binlog index 文件获取 binlog 文件名，当 binlog 数量很多时，这个操作会很耗时，而正常情况下，只需要获取一个 binlog 文件名称就能达到目的。该参数控制了获取的文件名数量，0表示不限制，一般设置成10即可。

计划缓存点查优化

最近更新时间：2024-10-14 10:25:41

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。

`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：2024-10-12 14:27:11

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：2024-10-14 10:25:41

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：2024-10-12 14:27:12

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：2024-10-12 15:00:43

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%

安全类特性
数据库审计

最近更新时间：2025-05-28 16:16:32

功能介绍

腾讯云为 TDSQL-C MySQL 版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 SQL 命令详情、客户端 IP、用户账号、数据库名、错误码、SQL 类型、风险等级、执行时间（微秒）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

数据库审计当前支持同步审计模式。同步审计同步记录所有审计日志，平均性能影响小于6%。

使用说明

为 TDSQL-C MySQL 版实例开通数据库审计，请参见 开通数据库审计 。

稳定性特性
statement outline

最近更新时间：2024-10-12 16:08:51

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 * 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 `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#qb_name(qb2)";
outline "select * from t1 where t1.a in (select b from t2)" set
outline_info "OPT: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#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：2024-10-14 10:23:21

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

SQL 限流

最近更新时间：2024-10-14 10:23:21

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 MySQL5.7 2.1.9以上版本
- 内核版本 MySQL8.0 3.1.5及以上版本

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

分析引擎特性
LibraDB 引擎功能特性

最近更新时间：2025-02-07 16:25:52

功能介绍

LibraDB 引擎主要服务于高效的分析类查询。是一个为客户提供实时且高性能的复杂 SQL 处理的扩展只读分析组件。利用 LibraDB 引擎的列式存储能力、向量化并行执行引擎以及分布式并行执行而扩展的优化器，可以让客户能够很简单的在数据库中原地体验到高效地分析能力，另外 LibraDB 的列式存储为高 QPS 的变更、事务的 ACID，进行了针对性的优化，保证了查询数据的实时性以及一致性。

原理

LibraDB 引擎内核能够自动从主节点中将数据转换为列式存储，并实时同步 binlog，使得数据与主节点实时保持一致。然后通过并行计算能力与向量化的执行引擎，处理用户复杂的 SQL，加速查询执行。

支持的功能

LibraDB 引擎内核功能支持多种优异特性，下文为您简单介绍一下产品支持的功能。

一、并行计算能力

在 LibraDB 引擎中，可以充分利用节点的计算资源去执行用户的 SQL。当 SQL 在 LibraDB 引擎中运行时，可以将一个 SQL 查询任务拆分为多个子任务，并允许这些子任务在多个处理器中同时运行。这种多线程的运行方式可以有效的提升 CPU 利用率以及 IO 资源，从而达到加速处理的目的。

二、向量化执行引擎

针对 LibraDB 引擎而言，数据不仅仅按列存储，还会基于列进行计算。在 TXSQL 这种传统的 OLTP 引擎中，通常会基于行存储进行计算，主要原因是事务以点查、点读、点写为主。但是在分析作为主要场景的 LibraDB 引擎中，单 SQL 的计算量可能极大。所以在 LibraDB 引擎中，我们实现了向量化的执行模式，在对内存中的列式数据，一个 batch 调用一次 SIMD 指令，减少了函数的调用次数，降低了 cache miss。同时还可以充分利用 SIMD 指令的并行能力，缩短计算耗时。

三、支持高速变更场景下的列式存储

在 TDSQL-C MySQL 版的读写实例中，可以支撑超百万级别的数据在线操作 QPS。而作为一个可以支撑实时数据分析的 LibraDB 引擎，必须要能够满足在如此之高的数据变更场景下的数据一致性。传统的列式存储在数据的大批量写入具有一定的优势，但是在面对大规模数据的 delete 和 update 就显得性能不足。在传统的实时数仓场景下，好的实践就是将 update 变更为 delete 和 insert，同时在数据同步层去实现数据的批量执行能力。但是列式存储在 delete 场景下依然存在一些性能劣势。综合以上的情况，传统列式的存储必定会存在较高的数据延时，无法达到实时数据分析的效果。

LibraDB 引擎通过在存储层的优化和支持，可以满足用户在高并发场景下的数据变更的数据一致性，避免因为读写实例的数据频繁变更带来的数据延时而错过分析时间。

四、指定数据加载能力

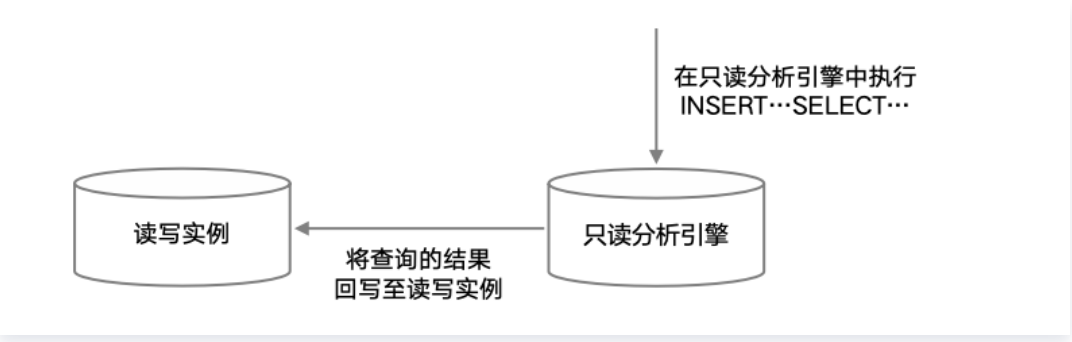
针对 TDSQL-C MySQL 版中的数据而言，并非所有的数据都具备数据分析价值，所以不需要所有的对象都加载为列存。故 libraDB 引擎支持指定对象加载的能力。可在数据加载的控制台设置或者通过命令行 SQL 指定需要加载的 LibraDB 的对象。

加速 ETL 回写

最近更新时间：2025-01-14 14:36:13

在数据库中常常会有执行 INSERT INTO <Table> SELECT ……的场景，如果 SELECT 的执行时间过长会导致整体执行效率过低。只读分析引擎作为承载在线 SQL 中的复杂查询。可以通过使用此功能将 INSERT ……SELECT……语句中的 SELECT 部分放到只读分析引擎中执行，加速 SELECT 的查询效率。然后将数据自动回写至读写节点中，从而大幅度提升业务的执行效率。

技术原理



如上图所示，您可以在只读分析引擎中执行 `INSERT…SELECT…` 语句。其中 SELECT 语句将会通过只读分析引擎加速执行，并且将查询的结果直接写入读写实例中，从而提升整体执行效率。

使用场景

> **注意：**
> 此功能仅适用于对数据延迟不敏感的场景。因为只读分析引擎为异步复制模式，所以当只读分析引擎存在延迟时，查询的结果与读写实例中存在一定的时差。

推荐在以下场景使用此功能：当查询条件复杂、SQL 语句执行时间较长但查询结果集的数据量较小时。在这种场景下，使用此功能可以显著提高性能，因为只读分析引擎会加速 SELECT 查询的执行效率。
使用此功能并不是在所有场景下都能带来性能收益，在某些场景下性能可能会下降。例如：

- `INSERT…SELECT…` 语句中的 SELECT 查询比较简单时，从只读分析引擎读取数据并回写到读写实例会带来额外的网络开销。对比直接从读写实例读取数据的优势并不明显。
- `INSERT…SELECT…` 语句中的 SELECT 查询结果集数据量大的情况下，此时主要的性能瓶颈在于结果集通过网络传输并写入读写实例的过程。此场景无法使用此功能来优化性能。

分页保序能力

最近更新时间：2025-04-15 15:35:22

在 MySQL 中，当查询数据时使用 order by 子句，且 order by 字段存在重复值时，若再结合 limit 子句进行分页查询，可能会出现第二页数据与第一页数据重复的问题。这是因为 MySQL 在排序时使用了堆排序的方法，而堆排序是一种不稳定的排序方法，当排序的字段存在重复值时，每一次排序输出的结果可能会不一致。特别是在并行执行 SQL 时，多线程排序会导致排序结果更加不稳定。因此，在使用 order by ... limit ... 场景的推荐做法是要让 order by 的字段存在索引，或者添加其他字段到 order by 字段中，让数据保持唯一性。
而只读分析引擎为了避免此问题的出现，支持了分页保序功能。分页保序可以让用户可以无需关心数据库的实现逻辑，即使排序字段存在重复，order by ... limit ... 输出的结果也不会出现重复，确保了序列的一致性。

实现原理

只读分析引擎会在确保业务排序逻辑后，默认对全量数据进行隐式排序。这样就避免了使用 limit 操作时带来的数据重复问题。

保序场景

- 无 order by 操作符，仅仅有 limit 的场景。
- order by 字段存在重复值。
- 子查询中包含排序，但是外层查询中未进行排序。

性能影响

在打开分页保序功能后，会引入额外的排序操作，故部分查询可能会出现性能回退。请根据业务的实际情况选择是否使用分页排序能力。

使用说明

分页保序的开启和使用介绍详情请参考 分页保序功能 。