阿里云
云数据库RDS

AliSQL内核

文档版本：20200303

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务时间约十分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意:** 权重设置为0，该服务器不会再接受新请求。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	单击设置 > 网络 > 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击**确定**。
`Courier字体`	命令。	执行`cd /d C:/window`命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{}或者{a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1 AliSQL功能概览

本文介绍AliSQL和其他版本的功能对比。

AliSQL介绍

AliSQL是阿里云深度定制的独立MySQL分支，除了社区版的所有功能外，AliSQL提供了类似于MySQL企业版的诸多功能，如企业级备份恢复、线程池、并行查询等，并且AliSQL还提供兼容Oracle的能力，如sequence引擎等。RDS MySQL使用AliSQL内核，为用户提供了MySQL所有的功能，同时提供了企业级的安全、备份、恢复、监控、性能优化、只读实例等高级特性。

功能列表

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
企业增值服务	*24*7 支持*	未提供	√	√	√
	紧急故障救援	未提供	√	√	√
	专家服务顾问支持	未提供	√	√	√
MySQL Features	*MySQL Database Server*	√	√	√	√
	MySQL Document Store	√	√	MySQL 8.0支持	MySQL 8.0支持
	MySQL Connectors	√	√	支持公开发行版	支持公开发行版
	MySQL Replication	√	√	√	√
	MySQL Router	√	√	MaxScale（MySQL 8.0支持）	数据库单租户代理
	MySQL Partitioning	√	√	√	√

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
	Storage Engine	InnoDB MyISAM NDB	InnoDB MyISAM NDB	InnoDB X-Engine	InnoDB X-Engine
Oracle Compatibility	*Sequence Engine*	未提供	未提供	MySQL 8.0支持	MySQL 8.0支持
MySQL Enterprise Monitor	*Enterprise Dashboard*	未提供	√	开发中	Enhanced Monitor
	Enterprise Advisors	未提供	√	开发中	CloudDBA
	Query Analyzer	未提供	√	开发中	Performance Insight
	Replication Monitor	未提供	√	开发中	√
	Enhanced OS Metrics	未提供	未提供	未提供	Enhanced Monitor
MySQL Enterprise Backup	*Hot backup for InnoDB*	未提供	√	√	√
	Full, Incremental, Partial, Optimistic Backups	未提供	√	√	库表级备份
	Full, Partial, Selective, Hot Selective restore	未提供	√	√	库表级恢复
	Point-In-Time-Recovery	未提供	√	√	√
	Cross-Region Backup	未提供	未提供	未提供	跨地域备份
	Recycle bin	未提供	未提供	MySQL 8.0支持	MySQL 8.0支持

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
	Flashback	未提供	未提供	√	√
MySQL Enterprise Security	*Enterprise TDE*	本地密钥替换	√	BYOK TDE，Key Rotating	BYOK TDE，Key Rotating
	Enterprise Disk Data Encryption at Rest	未提供	未提供	未提供	BYOK 落盘加密
	Enterprise Encryption	SSL	√	SSL	SSL
	Enterprise Audit	未提供	√	SQL洞察	SQL洞察
	安全加密算法 SM4	未提供	未提供	√	√
MySQL Enterprise Scalability	*Thread Pool*	未提供	√	MySQL 8.0支持	MySQL 8.0支持
	Enterprise Readonly Request Extention	未提供	未提供	√	只读实例
MySQL Enterprise Reliability	*Zero Data Loss*	未提供	未提供	√	三节点企业版
	SQL Outline	未提供	未提供	√	√
	Hot Massive Update	未提供	未提供	√	√
	Hot SQL Limit	未提供	未提供	√	√
	Hot SQL Firewall	未提供	未提供	√	√
MySQL Enterprise High-Availability	*Enterprise Automatic Failover Switch*	未提供	未提供	需要第三方HA机制	高可用版
	InnoDB Cluster	√	√	√	三节点企业版
	Multi-Source Replication	√	√	√	只读实例高可用

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
	Cross-Region Standby	未提供	未提供	未提供	灾备实例

版本支持情况

详情请参见*#unique_27*。

2 AliSQL Release Notes

本文介绍AliSQL的内核版本更新说明。

MySQL 8.0

20200229

- 新特性
  - *Performance Agent*：更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。
  - 在半同步模式下添加网络往返时间，并记录到性能数据。
- 性能优化
  - 允许在只读实例上进行语句级并发控制（CCL）操作。
  - 备实例支持Outline。
  - Proxy短连接优化。
  - 优化不同CPU架构下的pause指令执行时间。
  - 添加内存表查看线程池运行情况。
- Bug修复
  - 在低于4.9的Linux Kenerls中禁用ppoll，使用poll代替。
  - 修复wrap_sm4_encrypt函数调用错误问题。
  - 修复在滚动审核日志时持有全局变量锁的问题。
  - 修复恢复不一致性检查的问题。
  - 修复io_statistics表出现错误time值的问题。
  - 修复无效压缩算法导致崩溃的问题。
  - 修复用户列与5.6不兼容的问题。

20200110

- 新特性

  Inventory Hint：新增了三个hint， 支持SELECT、UPDATE、INSERT、DELETE 语句，快速提交/回滚事务，提高业务吞吐能力。

- 性能优化
  - 启动实例时，先初始化Concurrency Control队列结构，再初始化Concurrency Control规则。
  - 异步清除文件时继续取消小文件的链接。
  - 优化*Thread Pool*性能。
  - 默认情况下禁用恢复不一致性检查。
  - 更改设置变量所需的权限：
    - 设置以下变量所需的权限已更改为普通用户权限：
      - auto_increment_increment
      - auto_increment_offset
      - bulk_insert_buffer_size
      - binlog_rows_query_log_events
    - 设置以下变量所需的权限已更改为超级用户或系统变量管理用户权限：
      - binlog_format
      - binlog_row_image
      - binlog_direct
      - sql_log_off
      - sql_log_bin

20191225

- 新特性

  Recycle Bin：临时将删除的表转移到回收站，还可以设置保留的时间，方便您找回数据。
- 性能优化
  - 提高短连接处理性能。
  - 使用专用线程为maintain user服务，避免HA失败。
  - 通过Redo刷新Binlog时出现错误会显式释放文件同步锁。
  - 删除不必要的TCP错误日志。
  - 默认情况下启用线程池。
- Bug修复
  - 修复慢日志刷新的问题。
  - 修复锁定范围不正确的问题。
  - 修复TDE的Select函数导致的核心转储问题。

20191115

新特性

Statement Queue：针对语句的排队机制，将语句进行分桶排队，尽量把可能具有相同冲突的语句放在一个桶内排队，减少冲突的开销。

20191101

- 新特性
  - 为*TDE*添加SM4加密算法。
  - 保护备实例信息：拥有SUPER或REPLICATION_SLAVE_ADMIN权限的用户才能插入/删除/修改表slave_master_info、slave_relay_log_info、slave_worker_info。
  - 提高自动递增键的优先级：如果表中没有主键或非空唯一键，具有自动增量的非空键将是第一候选项。
  - 对系统表和处于初始化状态线程用到的表，不进行Memory引擎到MyISAM引擎的自动转换。
  - Redo Log刷新到磁盘之前先将Binlog文件刷新到磁盘。
  - 实例被锁定时也会影响临时表。
  - 添加新的基于LSM树的事务存储引擎X-Engine。
- 性能优化
  - *Thread Pool*：互斥优化。
  - *Performance Insight*：性能点支持线程池。
  - 参数调整：
    - `primary_fast_lookup`：会话参数，默认值为true。
    - `thread_pool_enabled`：全局参数，默认值为true。

20191015

- 新特性
  - *TDE*：支持透明数据加密TDE（Transparent Data Encryption）功能，可对数据文件执行实时I/O加密和解密，数据在写入磁盘之前进行加密，从磁盘读入内存时进行解密。
  - *Returning*：Returning功能支持DML语句返回Resultset，同时提供了工具包（DBMS_TRANS）便于您快捷使用。
  - 强制将引擎从MyISAM/MEMORY转换为InnoDB：如果全局变量force_memory/mysiam_to_innodb为ON，则创建/修改表时会将表引擎从MyISAM/MEMORY转换为InnoDB。
  - 禁止非高权限账号切换主备实例。
  - 性能代理插件：收集性能数据并保存到本地格式化文本文件，采用文件轮循方式，保留最近的秒级性能数据。
  - Innodb mutex timeout cofigurable：可配置全局变量innodb_fatal_semaphore_wait_threshold，默认值：600。
  - 忽略索引提示错误：可配置全局变量ignore_index_hint_error，默认值：false。
  - 可关闭*SSL*加密功能。
  - TCP错误信息：返回TCP方向（读取、读取等待、写入等待）错误及错误代码到end_connection事件，并且输出错误信息到错误日志。
- Bug修复
  - 支持本地AIO的Linux系统内，在触发线性预读之前会合并AIO请求。
  - 优化表/索引统计信息。
  - 如果指定了主键，则直接访问主索引。

20190915

Bug修复

修复Cmd_set_current_connection内存泄露问题。

20190816

- 新特性
  - *Thread Pool*：将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。
  - *Statement Concurrency Control*：通过控制并发数应对突发的数据库请求流量、资源消耗过高的语句访问以及SQL访问模型的变化，保证MySQL实例持续稳定运行。
  - *Statement Outline*：利用Optimizer Hint和Index Hint让MySQL稳定执行计划。
  - *Sequence Engine*：简化获取序列值的复杂度。
  - *Purge Large File Asynchronously*：删除单个表空间时，会将表空间文件重命名为临时文件，等待异步清除进程清理临时文件。
  - *Performance Insight*：专注于实例负载监控、关联分析、性能调优的利器，帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。
  - 优化实例锁状态：实例锁定状态下，可以drop或truncate表。
- Bug修复
  - 修复文件大小计算错误的问题。
  - 修复偶尔出现的内存空闲后再次使用的问题。
  - 修复主机缓存大小为0时的崩溃问题。
  - 修复隐式主键与CTS语句的冲突问题。
  - 修复慢查询导致的slog出错问题。

20190601

- 性能优化
  - 缩短日志表MDL范围，减少MDL阻塞的可能性。
  - 重构终止选项的代码。
- Bug修复
  - 修复审计日志中没有记录预编译语句的问题。
  - 屏蔽无效表名的错误日志。

MySQL 5.7基础版/高可用版

20200229

- 新特性
  - *Performance Agent*：更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。
  - 在半同步模式下添加网络往返时间，并记录到性能数据。

- 性能优化
  - 优化不同CPU架构下的pause指令执行时间。
  - Proxy短连接优化。
  - 添加内存表查看线程池运行情况。
- Bug修复
  - 修复DDL重做日志不安全的问题。
  - 修复io_statistics表出现错误time值的问题。
  - 修复更改表导致服务器崩溃的问题。
  - 修复MySQL测试用例。

20200110

性能优化

- 异步清除文件时继续取消小文件的链接。
- 优化*Thread Pool*性能。
- thread_pool_enabled参数的默认值调整为OFF。

20191225

- 新特性

  内部账户管理与防范：调整用户权限保护数据安全。
- 性能优化
  - 提高短连接处理性能。
  - 使用专用线程为maintain user服务，避免HA失败。
  - 删除不必要的TCP错误日志。
  - 优化线程池。
- Bug修复
  - 修复读写分离时mysqld进程崩溃问题。
  - 修复密钥环引起的核心转储问题。

20191115

Bug修复

修复主备切换后审计日志显示变量的问题。

20191101

- 新特性
  - 为*TDE*添加SM4加密算法。
  - 如果指定了主键，则直接访问主索引。
  - 对系统表和处于初始化状态线程用到的表，不进行Memory引擎到MyISAM引擎的自动转换。
- 性能优化
  - *Thread Pool*：互斥优化。
  - 引入审计日志缓冲机制，提高审计日志的性能。
  - *Performance Insight*：性能点支持线程池。
  - 默认开启*Thread Pool*。
- Bug修复
  - 在处理维护用户列表时释放锁。
  - 补充更多TCP错误信息。

20191015

- 新特性
  - 轮换慢日志：为了在收集慢查询日志时保证零数据丢失，轮换日志表会将慢日志表的csv数据文件重命名为唯一名称并创建新文件。您可以使用`show variables like '%rotate_log_table%'`;查看是否开启轮换慢日志。
  - 性能代理插件：收集性能数据并保存到本地格式化文本文件，采用文件轮轮循方式，保留最近的秒级性能数据。
  - 强制将引擎从MEMORY转换为InnoDB：如果全局变量rds_force_memory_to_innodb为ON，则创建/修改表时会将表引擎从MEMORY转换为InnoDB。
  - TDE机制优化：添加keyring-rds插件与管控系统/密钥管理服务进行交互。
  - TCP错误信息：返回TCP方向（读取、读取等待、写入等待）错误及错误代码到end_connection事件，并且输出错误信息到错误日志。
- Bug修复

  修复DDL中的意外错误Error 1290。

20190925

参数修改

- 将系统变量auto_generate_certs的默认值由true改为false。

- 增加全局只读变量auto_detact_certs，默认值为false，有效值为[true | false]。 该系统变量在Server端使用OpenSSL编译时可用，用于控制Server端在启动时是否在数据目录下自动查找SSL加密证书和密钥文件，即控制是否开启Server端的证书和密钥的自动查找功能。

20190915

新特性

Thread Pool：将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。

20190815

- 新特性
  - *Purge Large File Asynchronously*：删除单个表空间时，会将表空间文件重命名为临时文件，等待异步清除进程清理临时文件。
  - *Performance Insight*：专注于实例负载监控、关联分析、性能调优的利器，帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。
  - 优化实例锁状态：实例锁定状态下，可以drop或truncate表。
- Bug修复
  - 禁止在`set rds_current_connection`命令中设置rds_prepare_begin_id。
  - 允许更改已锁定用户的信息。
  - 禁止用关键字actual作为表名。
  - 修复慢日志导致时间字段溢出的问题。

20190510版本

新特性：允许在事务内创建临时表。

20190319版本

新特性：支持在handshake报文内代理设置threadID。

20190131版本

- 升级到官方5.7.25版本。
- 关闭内存管理功能jemalloc。
- 修复内部变量net_lenth_size计算错误问题。

20181226版本

- 新特性：支持动态修改binlog-row-event-max-size，加速无主键表的复制。
- 修复Proxy实例内存申请异常的问题。

20181010版本

- **支持隐式主键。**
- **加快无主键表的主备复制。**
- **支持Native AIO，提升I/O性能。**

20180431版本

新特性：

- **支持高可用版。**
- **支持*#unique_36*。**
- **增强对处于快照备份状态的实例的保护。**

MySQL 5.7三节点企业版

20191128

- **新特性**

  支持读写分离。
- **Bug修复**
  - **修复部分场景下Follower Second_Behind_Master计算错误问题。**
  - **修复表级并行复制事务重试时死锁问题。**
  - **修复XA相关bug。**

20191016

- **新特性**
  - **支持MySQL 5.7高可用版（本地SSD盘）升级到三节点企业版。**
  - **兼容MySQL官方GTID功能，默认不开启。**
  - **合并AliSQL MySQL 5.7基础版/高可用版 20190915版本及之前的自研功能。**
- **Bug修复**

  修复重置备实例导致binlog被关闭问题。

20190909

- 新特性
  - 优化大事务在三节点强一致状态下的执行效率。
  - 支持从Leader/Follower进行Binlog转储。
  - 支持创建只读实例。
  - 系统表默认使用InnoDB引擎。
- Bug修复
  - 修复Follower日志清理命令失效问题。
  - 修复参数slave_sql_verify_checksum=OFF和binlog_checksum=crc32时Slave线程异常退出问题。

20190709

新特性

- 支持三节点功能。
- 禁用semi-sync插件。
- 支持表级并行复制、Writeset并行复制。
- 支持pk_access主键查询加速。
- 支持线程池。
- 合并AliSQL MySQL 5.7基础版/高可用版 20190510版本及之前的自研功能。

MySQL 5.6

20200229

- 新特性

  支持Proxy读写分离功能。
- 性能优化
  - 优化线程池功能。
  - 优化不同CPU架构下的pause指令执行时间。
- Bug修复

  修复XA事务部分提交的问题。

20200110

- 新特性

  Thread Pool：将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。

- 性能优化

  异步清除文件时继续取消小文件的链接。

- Bug修复
  - 修复页面清理程序的睡眠时间计算不正确问题。
  - 修复`SELECT @@global.gtid_executed`导致的故障转移失败问题。
  - 修复*IF CLIENT KILLED AFTER ROLLBACK TO SAVEPOINT PREVIOUS STMTS COMMITTED*问题。

20191212

性能优化

删除不必要的tcp错误日志

20191115

Bug修复

修复慢日志时间戳溢出问题。

20191101

Bug修复

- 修复刷新日志时切换慢日志的问题，仅在执行刷新慢日志时切换慢日志。
- 修正部分显示错误。

20191015

- 新特性
  - 轮换慢日志：为了在收集慢查询日志时保证零数据丢失，轮换日志表会将慢日志表的csv数据文件重命名为唯一名称并创建新文件。您可以使用`show variables like '%rotate_log_table%';`查看是否开启轮换慢日志。
  - SM4加密算法：添加新的SM4加密算法，取代旧的SM加密算法。
  - *Purge Large File Asynchronously*：删除单个表空间时，会将表空间文件重命名为临时文件，等待异步清除进程清理临时文件。
  - TCP错误信息：返回TCP方向（读取、读取等待、写入等待）错误及错误代码到end_connection事件，并且输出错误信息到错误日志。
  - 引入审计日志缓冲机制，提高审计日志的性能。。

- Bug修复
  - 禁用pstack，避免存在大量连接时可能导致pstack无响应。
  - 修复隐式主键与`create table as select`语句之间的冲突。
  - 自动清除由二进制日志创建的临时文件。

20190815

优化实例锁状态：实例锁定状态下，可以drop或truncate表。

20190130版本

修复部分可能导致系统不稳定的bug。

20181010版本

添加参数rocksdb_ddl_commit_in_the_middle（MyRocks）。如果这个参数被打开，部分DDL在执行过程中将会执行commit操作。

201806** （5.6.16）版本

新特性：slow log精度提升为微秒。

20180426（5.6.16）版本

- 新特性：引入隐藏索引，支持将索引设置为不可见，详情请参见参考文档。
- 修复备库apply线程的bug。
- 修复备库apply分区表更新时性能下降问题。
- 修复TokuDB下alter table comment重建整张表问题，详情请参见参考文档。
- 修复由show slave status/show status可能触发的死锁问题。

20171205（5.6.16）版本

- 修复OPTIMIZE TABLE和ONLINE ALTER TABLE同时执行时会触发死锁的问题。
- 修复SEQUENCE与隐含主键冲突的问题。
- 修复SHOW CREATE SEQUENCE问题。
- 修复TokuDB引擎的表统计信息错误。
- 修复并行OPTIMIZE表引入的死锁问题。
- 修复QUERY_LOG_EVENT中记录的字符集问题。
- 修复信号处理引起的数据库无法停止问题，详情请参见参考文档。
- 修复RESET MASTER引入的问题。
- 修复备库陷入等待的问题。
- 修复SHOW CREATE TABLE可能触发的进程崩溃问题。

20170927（5.6.16）版本

修复TokuDB表查询时使用错误索引问题。

20170901（5.6.16）版本

- **新特性：**
  - **升级SSL加密版本到TLS 1.2，详情请参见参考文档。**
  - **支持Sequence。**
- **修复NOT IN查询在特定场景下返回结果集有误的问题。**

20170530 (5.6.16)版本

新特性：支持高权限账号Kill其他账号下的连接。

20170221（5.6.16）版本

新特性：支持*#unique_37*。

MySQL 5.5

20181212

修复调用系统函数gettimeofday(2) 返回值不准确的问题。该系统函数返回值为时间，常用来计算等待超时，时间不准确时会导致一些操作永不超时。

3 X-Engine引擎

3.1 X-Engine简介

X-Engine是阿里云数据库产品事业部自研的联机事务处理OLTP（On-Line Transaction Processing）数据库存储引擎。作为自研数据库POLARDB的存储引擎之一，已经广泛应用在阿里集团内部诸多业务系统中，包括交易历史库、钉钉历史库等核心应用，大幅缩减了业务成本，同时也作为双十一大促的关键数据库技术，挺过了数百倍平时流量的冲击。

为什么设计一个新的存储引擎

X-Engine的诞生是为了应对阿里内部业务的挑战，早在2010年，阿里内部就大规模部署了MySQL数据库，但是业务量的逐年爆炸式增长，数据库面临着极大的挑战：

- **极高的并发事务处理能力（尤其是双十一的流量突发式暴增）。**
- **超大规模的数据存储。**

这两个问题虽然可以通过扩展数据库节点的分布式方案解决，但是堆机器不是一个高效的手段，我们更想用技术的手段将数据库性价比提升到极致，实现以少量资源换取性能大幅提高的目的。

传统数据库架构的性能已经被仔细的研究过，数据库领域的泰斗，图灵奖得主Michael Stonebreaker就此写过一篇论文 *《OLTP Through the Looking Glass, and What We Found There》* **，指出传统关系型数据库，仅有不到10%的时间是在做真正有效的数据处理工作，剩下的时间都浪费在其它工作上，例如加锁等待、缓冲管理、日志同步等。**

造成这种现象的原因是因为近年来我们所依赖的硬件体系发生了巨大的变化，例如多核（众核）CPU、新的处理器架构（Cache/NUMA）、各种异构计算设备（GPU/FPGA）等，而架构在这些硬件之上的数据库软件却没有太大的改变，例如使用B-Tree索引的固定大小的数据页（Page）、使用ARIES算法的事务处理与数据恢复机制、基于独立锁管理器的并发控制等，这些都是为了慢速磁盘而设计，很难发挥出现有硬件体系应有的性能。

基于以上原因，阿里开发了适合当前硬件体系的存储引擎，即X-Engine。

X-Engine架构

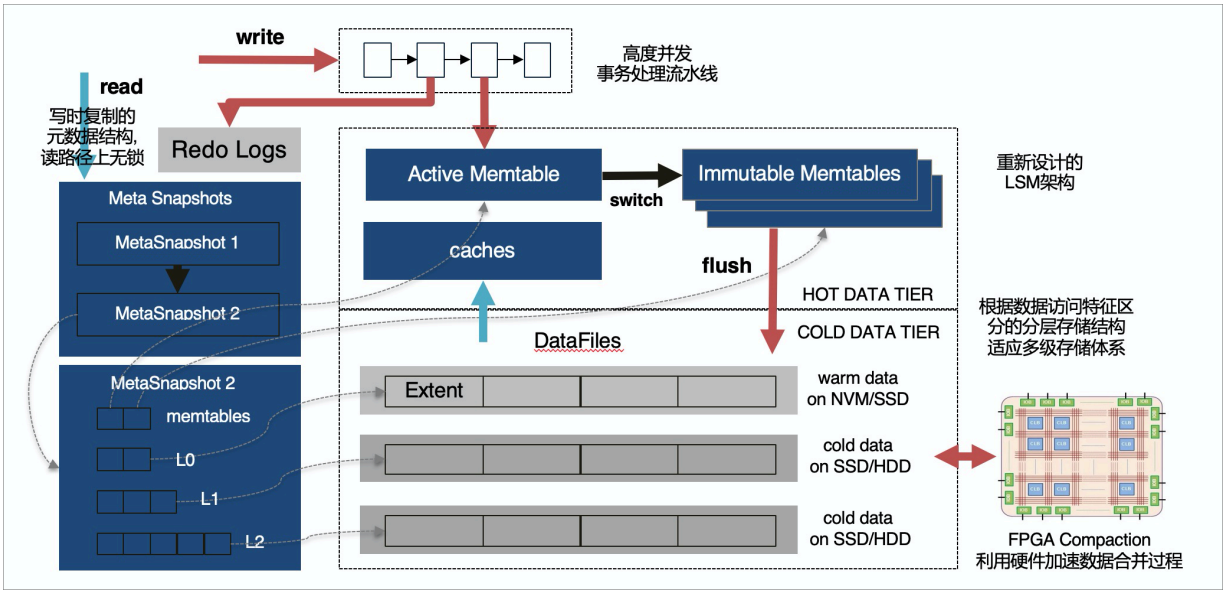
全新架构的X-Engine存储引擎不仅可以无缝对接兼容MySQL（得益于MySQL Pluginable Storage Engine特性），同时X-Engine使用分层存储架构。

因为目标是面向大规模的海量数据存储，提供高并发事务处理能力和降低存储成本，在大部分大数据量场景下，数据被访问的机会是不均等的，访问频繁的热数据实际上占比很少，X-Engine根据

数据访问频度的不同将数据划分为多个层次，针对每个层次数据的访问特点，设计对应的存储结构，写入合适的存储设备。

X-Engine使用了*LSM-Tree*作为分层存储的架构基础，并进行了重新设计：

- 热数据层和数据更新使用内存存储，通过内存数据库技术（Lock-Free index structure/ append only）提高事务处理的性能。
- 流水线事务处理机制，把事务处理的几个阶段并行起来，极大提升了吞吐。
- 访问频度低的数据逐渐淘汰或是合并到持久化的存储层次中，并结合多层次的存储设备（NVM/ SSD/HDD）进行存储。
- 对性能影响比较大的Compaction过程做了大量优化：
  - 拆分数据存储粒度，利用数据更新热点较为集中的特征，尽可能的在合并过程中复用数据。
  - 精细化控制LSM的形状，减少I/O和计算代价，有效缓解了合并过程中的空间增大。
- 同时使用更细粒度的访问控制和缓存机制，优化读的性能。



说明：

X-Engine的架构和优化技术已经被总结成论文 《*X-Engine: An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing*》，**发表在了数据库业界最顶尖的会议SIGMOD'19，这是中国大陆公司首次在国际顶级会议上发表OLTP数据库内核相关的技术成果。**

技术特点

- **利用FPGA硬件加速Compaction过程，使得系统上限进一步提升。这个技术属首次将硬件加速技术应用到在线事务处理数据库存储引擎中，相关论文** 《*FPGA-Accelerated Compactions for LSM-based Key Value Store*》 **已经被2020年的顶级会议*FAST'20*接收。**

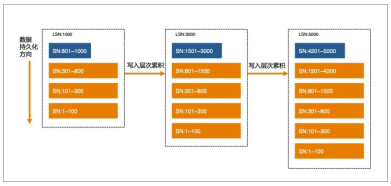
- 通过数据复用技术减少数据合并代价，同时减少缓存淘汰带来的性能抖动。
- 使用多事务处理队列和流水线处理技术，减少线程上下文切换代价，并计算每个阶段任务量配比，使整个流水线充分流转，极大提升事务处理性能。相对于其他类似架构的存储引擎（例如RocksDB），X-Engine的事务处理性能有10倍以上提升。
- X-Engine使用的Copy-on-write技术，避免原地更新数据页，从而对只读数据页面进行编码压缩，相对于传统存储引擎（例如InnoDB），使用X-Engine可以将存储空间降低至10%~50%。
- Bloom Filter快速判定数据是否存在，Surf Filter判断范围数据是否存在，Row Cache缓存热点行，加速读取性能。

LSM基本逻辑

LSM的本质是所有写入操作直接以追加的方式写入内存。每次写到一定程度，即冻结为一层（Level），并写入持久化存储。所有写入的行，都以主键（Key）排序好后存放，无论是在内存中，还是持久化存储中。在内存中即为一个排序的内存数据结构（Skiplist、B-Tree、etc），在持久化存储也作为一个只读的全排序持久化存储结构。

普通的存储系统若要支持事务处理，需要加入一个时间维度，为每个事务构造出一个不受并发干扰的独立视域。例如存储引擎会对每个事务定序并赋予一个全局单调递增的事务版本号（SN），每个事务中的记录会存储这个SN以判断独立事务之间的可见性，从而实现事务的隔离机制。

如果LSM存储结构持续写入，不做其他的动作，那么最终会成为如下结构。



这种结构对于写入是非常友好的，只要追加到最新的内存表中即完成，为实现故障恢复，只需记录Redo Log，因为新数据不会覆盖旧版本，追加记录会形成天然的多版本结构。

但是如此累积，冻结的持久化层次越来越多，会对查询会产生不利的影响。例如对同一个key，不同事务提交产生的多版本记录会散落在各个层次中；不同的key也会散落在不同层次中。读操作需要查找各个层并合并才能得到最终结果。

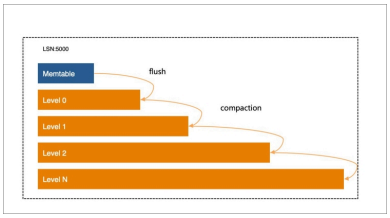
因此LSM引入了Compaction操作解决这个问题，Compaction操作有2种作用：

- 控制LSM层次形状

  一般的LSM形状都是层次越低，数据量越大（倍数关系），目的是为了提升读性能。

  通常存储系统的数据访问都有局部性，大量的访问都集中在少部分数据上，这也是缓存系统能有效工作的基本前提。在LSM存储结构中，如果把访问频率高的数据尽可能放在较高的层

次上，存放在快速存储设备中（例如NVM、DRAM），而把访问频率低的数据放在较低层次中，存放在廉价慢速存储设备中。这就是X-Engine的冷热分层概念。



- 合并数据

  Compaction操作不断的把相邻层次的数据合并，并写入更低层次。合并的过程实际上是把要合并的相邻两层或多层的数据读出来，按key排序，相同的key如果有多个版本，只保留新的版本（比当前正在执行的活跃事务中最小版本号新），丢掉旧版本数据，然后写入新的层，这个操作非常耗费资源。

  合并数据除了考虑冷热分层以外，还需要考虑其他维度，例如数据的更新频率，大量的多版本数据在查询的时候会浪费更多的I/O和CPU，因此需要优先进行合并以减少记录的版本数量。X-Engine综合考虑了各种策略形成自己的Compaction调度机制。

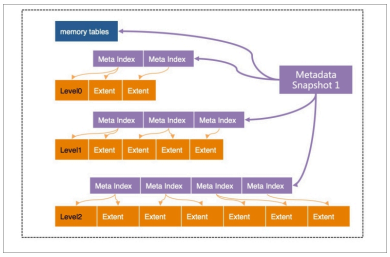
高度优化的LSM

X-Engine的memory tables使用了无锁跳表（Locked-free SkipList），并发读写的性能较高。在持久化层如何实现高效，就需要讨论每层的细微结构。

- 数据组织

  X-Engine的每层都划分成固定大小的Extent，存放每个层次中的数据的一个连续片段（Key Range）。为了快速定位Extent，为每层Extents建立了一套索引（Meta Index），所有

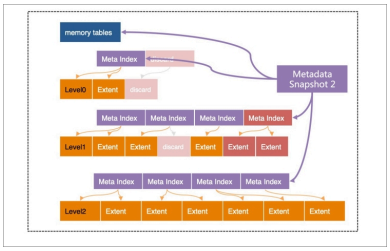
这些索引，加上所有的memory tables（active/immutable）一起组成了一个元数据树（Metadata Tree），root节点为Metadata Snapshot，这个树结构类似于B-Tree。



X-Engine中除了当前的正在写入的active memory tables以外，其他结构都是只读的，不会被修改。给定某个时间点，例如LSN=1000，上图中的Metadata Snapshot 1引用到的结构即包含了LSN=1000时的所有的数据的快照，因此这个结构被称为Snapshot。

即便是Metadata结构本身，也是一旦生成就不会被修改。所有的读请求都是以Snapshot为入口，这是X-Engine实现Snapshot级别隔离的基础。前文说过随着数据写入，累积数据越多，会执行Compaction操作、冻结memory tables等，这些操作都是用Copy-on-write实现，即每次都将修改产生的结果写入新的Extent，然后生成新的Meta Index结构，最终生成新的Metadata Snapshot。

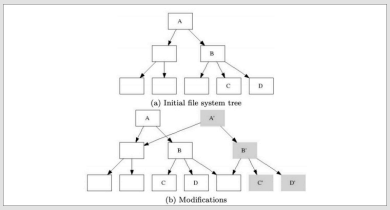
例如执行一次Compaction操作会生成新的Metadata Snapshot，如下图所示。



可以看到Metadata Snapshot 2相对于Metadata Snapshot 1并没有太多的变化，仅仅修改了发生变更的一些叶子节点和索引节点。

说明:

这个技术颇有些类似 *B-trees, Shadowing, and Clones*，如果您阅读那篇论文，会对理解这个过程有所帮助。



- 事务处理

得益于LSM的轻量化写机制，写入操作固然是其明显的优势，但是事务处理不只是把更新的数据写入系统那么简单，还要保证ACID（原子性、一致性、隔离性、持久性），涉及到一整套复杂的流程。X-Engine将整个事务处理过程分为两个阶段：

1. 读写阶段

   校验事务的冲突（写写冲突、读写冲突），判断事务是否可以执行、回滚重试或者等锁。如果事务冲突校验通过，则把修改的所有数据写入Transaction Buffer。

2. 提交阶段

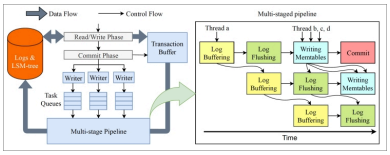
   写WAL、写内存表，以及提交并返回用户结果，这里面既有I/O操作（写日志、返回消息），也有CPU操作（拷贝日志、写内存表）。

为了提高事务处理吞吐，系统内会有大量事务并发执行，单个I/O操作比较昂贵，大部分存储引擎会倾向于聚集一批事务一起提交，称为Group Commit，能够合并I/O操作。但是一组事务提交的过程中，还是有大量等待过程的，例如写入日志到磁盘过程中，除了等待落盘无所事事。

X-Engine为了进一步提升事务处理的吞吐，使用流水线技术，把提交阶段分为4个独立的更精细的阶段：

1. 拷贝日志到缓冲区（Log Buffer）
2. 日志落盘（Log Flush）
3. 写内存表（Write memory table）
4. 提交返回（Commit）

事务到了提交阶段，可以自由选择执行流水线中任意一个阶段，只要流水线任务的大小划分得当，就能充分并行起来，流水线处于接近满载状态。另外这里利用的是事务处理的线程，而非后台线程，每个线程在执行的时候，选择流水线中的一个阶段执行任务，或者空闲后处理其他请求，没有等待，也无需切换，充分利用了每个线程的能力。

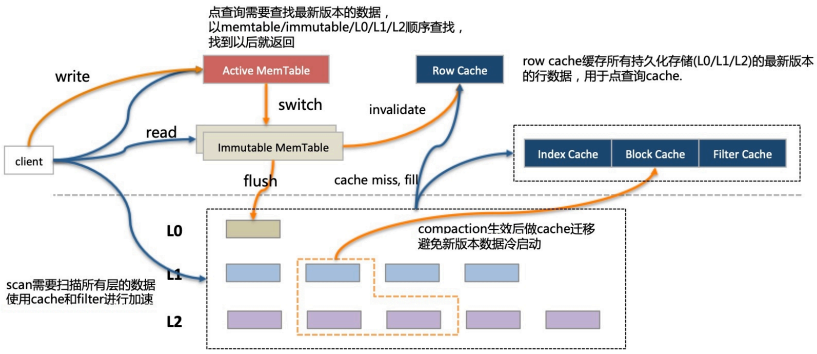


- 读操作

LSM处理多版本数据的方式是新版本数据记录会追加在老版本数据后面，从物理上看，一条记录不同的版本可能存放在不同的层，在查询的时候需要找到合适的版本（根据事务隔离级别定义的可见性规则），一般查询都是查找最新的数据，总是由最高的层次往低层次找。

对于单条记录的查找而言，一旦找到便可以终止，如果记录在比较高的层次，例如memory tables，很快便可以返回；如果记录已经落入了很低的层次，那就得逐层查找，也许Bloom Filter可以跳过某些层次加快这个旅程，但毕竟还是有很多的I/O操作。X-Engine针对单记录查询引入了Row Cache，在所有持久化的层次的数据之上做了一个缓存，在memory tables中没有命中的单行查询，在Row Cache之中也会被捕获。Row Cache需要保证缓存了所有持久化层次中最新版本的记录，而这个记录是可能发生变化的，例如每次flush将只读的memory tables写入持久化层次时，就需要恰当的更新Row Cache中的缓存记录，这个操作比较微妙，需要精心的设计。

对于范围扫描而言，因为没法确定一个范围的key在哪个层次中有数据，只能扫描所有的层次做合并之后才能返回最终的结果。X-Engine采用了一系列的手段，例如SuRF（SIGMOD'18 best paper）提供range scan filter减少扫描层数、异步I/O与预取。

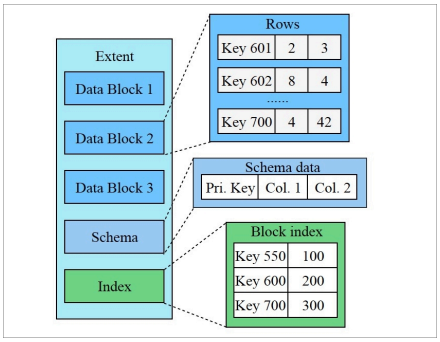


读操作中最核心的是缓存设计，Row Cache负责单行查询，Block Cache负责Row Cache的漏网之鱼，也用来进行范围扫描。由于LSM的Compaction操作会一次更新大量的Data Block，导致Block Cache中大量数据短时间内失效，导致性能的急剧抖动，因此X-Engine做了很多的优化：

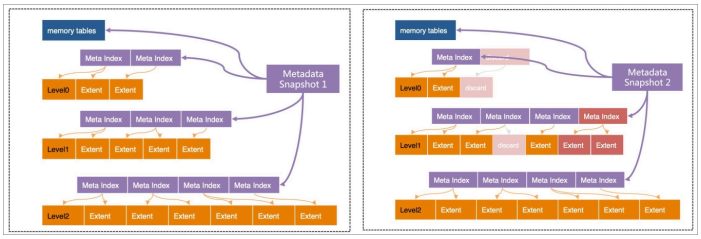
- 减少Compaction的粒度。
- 减少Compaction过程中改动的数据。
- Compaction过程中针对已有的缓存数据做定点更新。

- Compaction

Compaction操作是比较重要的，需要把相邻层次交叉的Key Range数据读取合并，然后写到新的位置。这是为前面简单的写入操作付出的代价。X-Engine为优化这个操作重新设计了存储结构。



如前文所述，X-Engine将每一层的数据划分为固定大小的Extent，一个Extent相当于一个小而完整的排序字符串表（SSTable），存储了一个层次中的一个连续片段，连续片段又进一步划分为一个个连续的更小的片段Data Block，相当于传统数据库中的Page，只不过Data Block是只读而且不定长的。

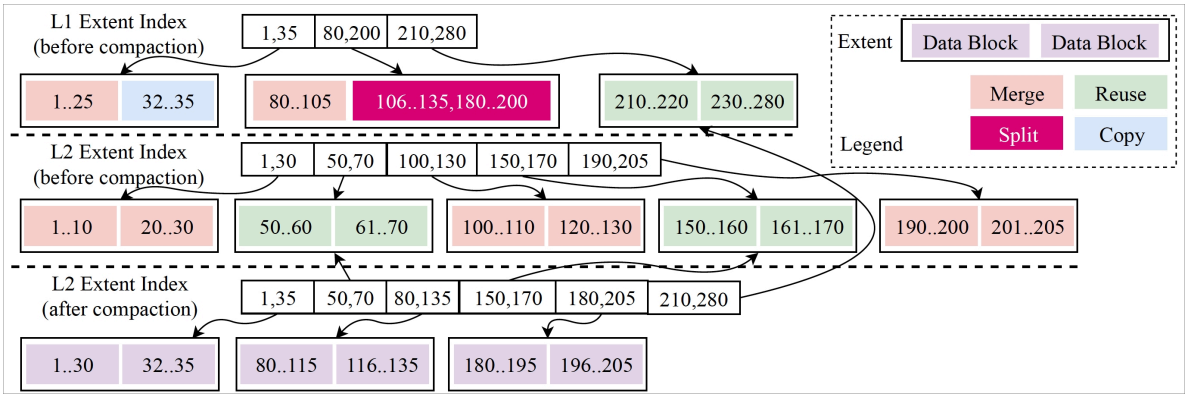


回看并对比Metadata Snapshot 1和Metadata Snapshot 2，可以发现Extent的设计意图。每次修改只需要修改少部分有交叠的数据，以及涉及到的Meta Index节点。两个Metadata Snapshot结构实际上共用了大量的数据结构，这被称为数据复用技术（Data Reuse），而Extent大小正是影响数据复用率的关键，Extent作为一个完整的被复用的物理结构，需要尽可能的小，这样与其他Extent数据交叉点会变少，但又不能非常小，否则需要索引过多，管理成本太大。

X-Engine中Compaction的数据复用是非常彻底的，假设选取两个相邻层次（Level1，Level2）中的交叉的Key Range所涵盖的Extents进行合并，合并算法会逐行进行扫描，只要发现任意的物理结构（包括Data Block和Extent）与其他层中的数据没有交叠，则可以进行复

用。只不过Extent的复用可以修改Meta Index，而Data Block的复用只能拷贝，即便如此也可以节省大量的CPU。

一个典型的数据复用在Compaction中的过程可以参考下图。



可以看出数据复用的过程是在逐行迭代的过程中完成的，不过这种精细的数据复用带来另一个副作用，即数据的碎片化，所以在实际操作的过程中也需要根据实际情况进行分析。

数据复用不仅给Compaction操作本身带来好处，降低操作过程中的I/O与CPU消耗，更对系统的综合性能产生一系列的影响。例如c、Compaction过程中数据不用完全重写，大大降低了写入时空间的增大；大部分数据保持原样，数据缓存不会因为数据更新而失效，减少合并过程中因缓存失效带来的读性能抖动。

实际上，优化Compaction的过程只是X-Engine工作的一部分，更重要的是优化Compaction调度的策略，选什么样的Extent、定义compaction任务的粒度、执行的优先级等，都会对整个系统性能产生影响，可惜并不存在什么完美的策略，X-Engine积累了一些经验，定义了很多规则，而探索更合理的调度策略是未来一个重要方向。

适用场景

请参见*#unique_40*。

如何使用X-Engine

请参见*X-Engine*引擎使用须知。

后续发展

作为MySQL的存储引擎，持续地提升MySQL系统的兼容能力是一个重要目标，后续会根据需求的迫切程度逐步加强原本取消的一些功能，例如外键，以及对一些数据结构、索引类型的支持。

X-Engine作为存储引擎，核心的价值还在于性价比，持续提升性能降低成本，是一个长期的根本目标，X-Engine还在Compaction调度、缓存管理与优化、数据压缩、事务处理等方向上进行深层次的探索。

X-Engine不仅仅局限为一个单机的数据库存储引擎，未来还将作为自研分布式数据库POLARDB分布式版本的核心，提供企业级数据库服务。

3.2 X-Engine引擎使用须知

RDS MySQL提供阿里云自研的X-Engine存储引擎，支持事务并且可以大幅降低磁盘空间占用。

产品介绍

X-Engine是阿里云数据库产品事业部自研的联机事务处理OLTP（On-Line Transaction Processing）数据库存储引擎。作为自研数据库POLARDB的存储引擎之一，已经广泛应用在阿里集团内部诸多业务系统中，包括交易历史库、钉钉历史库等核心应用，大幅缩减了业务成本，同时也作为双十一大促的关键数据库技术，挺过了数百倍平时流量的冲击。

X-Engine是适用于大规模电子商务交易处理的优化存储引擎，X-Engine团队撰写的论文《*X-Engine: An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing*》，详细讲述了X-Engine在数据库存储引擎领域所做的原创性工作，2019年被SIGMOD'19 Industrial Track接收。

与传统的InnoDB引擎不同，X-Engine使用分层存储架构（LSM-Tree）。分层存储有两个比较显著的优点：

- 需要索引的热点数据集更小，写入性能更高。
- 底层持久化的数据页是只读的，数据页采用紧凑存储格式，同时默认进行压缩，存储成本更低。

除了LSM-Tree架构自身的优势之外，X-Engine在工程实现上也进行了大量的创新，主要包含如下几个方面：

- 利用先天性的优势，持续优化写入性能，X-Engine相比同为LSM-tree架构的Rocksdb，有超过10倍的性能提升。
- 在存储层引入数据复用技术等，优化Compaction的性能，降低传统LSM-tree架构中Compaction动作对系统资源的冲击，保持系统性能平稳。
- 支持在同一实例中混合部署SSD/HDD等不同IO能力的存储设备， 利用天然分层结构的特点，结合不同存储硬件的IO读写性能，智能地进行数据的冷热分离存储，在不降低性能的前提下，降低综合成本。
- 引入多个层级Cache，同时结合Cach回填和预取机制，利用精细化访问机制和缓存技术，弥补传统LSM-tree引擎的读性能短板。

通过以上多方面的工程优化，X-Engine成为传统InnoDB引擎的一个替代选项，既支持事务，同时又能够显著的降低业务存储成本（依据数据特征，存储空间可降低至10%~50%），特别适合数据容量巨大，同时又要保证一定事务读写性能的业务。

> **说明:**
> 关于X-Engine引擎适用的业务场景请参见*#unique_40*。

前提条件

实例为RDS MySQL 8.0本地SSD盘。

> **说明:**
> - RDS MySQL 5.5/5.6/5.7的用户如果需要使用X-Engine引擎，请迁移至RDS MySQL 8.0版本实例。详情请参见*RDS*实例间的数据迁移。
> - 转换引擎请参见*InnoDB/TokuDB*引擎转换为*X-Engine*引擎。

使用限制

- 引擎功能限制

  X-Engine在引擎功能上有一些限制，其中部分功能尚在开发中。其他未列出的部分，默认其功能特性与InnoDB引擎相同。

分类	功能	X-Engine引擎	备注
SQL功能	外键	不支持	-
	临时表	不支持	-
	分区表（partition）	不支持（所有partition相关创建及增删改查操作均不支持）	-
	Generated Column	不支持	-
	Handler API	不支持	-
列属性	最大列长度（longblob/longtext/json）	32MB	-
	GIS地理数据类型	所有GIS相关数据类型均不支持（包含geometry、point、linestring、polygon、multipoint、multilinestring、multipolygon、gemometrycollection）	-
索引	哈希索引	不支持	-

分类	功能	X-Engine引擎	备注
	空间索引	不支持（所有fulltext索引相关的创建，使用均不支持）	-
事务	事务隔离级别	2个隔离级别： - 读已提交（RC） - 可重复读（RR）	-
	最大事务	32MB	更大事务的支持在开发中
	Savepoint	不支持	-
	XA事务	不支持	功能开发中
锁	锁粒度	- 支持表级别锁 - 支持行级别锁 - 不支持GAP锁	-
	Skip Locked Lock Nowait	不支持	-
字符集支持	非索引列支持的字符格式	非索引列支持所有的字符集（校对规则）	-
	索引列支持的字符格式	- latin1（latin1_bin） - gbk（gbk_chinese_ci、gbk_bin） - utf8（utf8_general_ci、utf8_bin） - utf8mb4（utf8mb4_0900_ai_ci、utf8mb4_general_ci、utf8mb4_bin）	-
主从复制	Binlog格式	stmt/row/mixed 说明： 默认为row，采用stmt/mixed在特定并发场景可能存在数据安全性问题。	-

- 大事务功能限制

  X-Engine目前不支持大事务。当一个事务修改的行数特别多时，X-Engine会使用commit in middle功能。例如用户在一个事务中修改的行数超过10000行，X-Engine会在内部把该事务

提交，并且重新开启一个事务继续服务当前用户开启的事务。但是commit in middle并不能遵循严格意义上的ACID，您在使用过程中需要注意。以下举例说明：

- 用户开启一个事务插入大量数据，在插入的过程中，由于先提交了一部分数据，其它请求就可以访问到插入的数据。
- 用户开启一个事务修改大量数据，回滚的时候，已经执行了commit in middle的事务无法回滚。

```
drop table t1;
create table t1(c1 int primary key , c2 int)ENGINE=xengine;
begin;
call insert_data(12000); //插入12000行数据，触发commit in middle，前1000行数据已经提交。
rollback;// 回滚只能把最后2000条数据回滚。
select count(*) from t1; // 这里仍然能够查询到10000条数据。
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.00 sec)
```

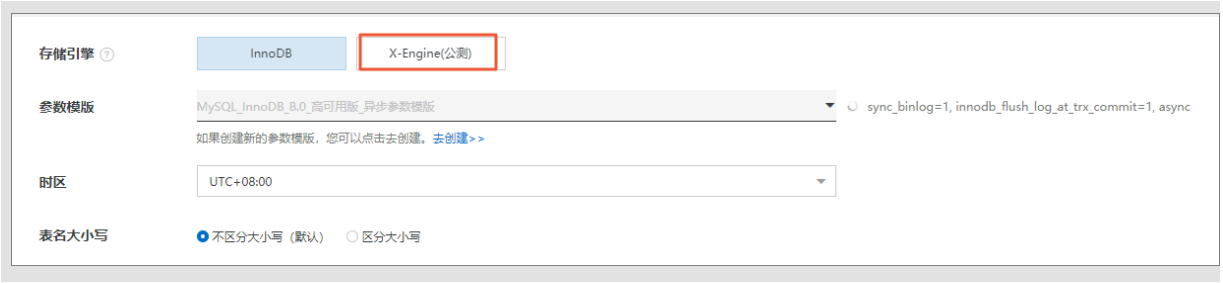
- 用户开启一个事务删除或者修改大量数据时，会遗漏掉本事务修改的行。

```
drop table t1;
create table t1(c1 int primary key , c2 int)ENGINE=xengine;
call insert_data(10000);
begin;
insert into t1 values(10001,10001), (10002,10002);
delete from t1 where c1 >= 0;// delete操作触发commit in middle，导致delete操作没有读到本事务插入的行。
commit;
select * from t1;
+-------+-------+
| c1    | c2    |
+-------+-------+
| 10001 | 10001 |
| 10002 | 10002 |
+-------+-------+
2 rows in set (0.00 sec)
```

参数说明

说明:

在*#unique_4*时，可以选择X-Engine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。



类别	参数	说明	备注
性能	xengine_arena_block_size	memtable向操作系统/jemalloc的外部内存管理系统申请新内存分配的单位	启动后只读
	xengine_batch_group_max_group_size	事务流水线线入队分组数	启动后只读
	xengine_batch_group_max_leader_wait_time_us	事务流水线最大等待时间	启动后只读
	xengine_batch_group_slot_array_size	事务流水线最大batch大小	启动后只读
内存	xengine_block_cache_size	读block缓存的大小	不可修改
	xengine_row_cache_size	行缓存的大小	不可修改
	xengine_write_buffer_size	单Memtable的最大大小	不可修改
	xengine_block_size	磁盘上数据block大小	初始化后只读 启动后只读
	xengine_db_write_buffer_size	所有subtable的Active Memtable的总计大小限制	不可修改
	xengine_db_total_write_buffer_size	所有subtable的Active Memtable/Immutable memtable的总大小限制	不可修改
	xengine_scan_add_blocks_limit	每个请求在范围扫描时，可以加到BlockCache中的Block数目	不可修改
compaction	xengine_flush_delete_percent_trigger	当Memtable中记录数超过此数目时，则xengine_flush_delete_record_trigger参数生效。	-
锁	xengine_max_row_locks	单SQL请求中，最大可以锁定的行数	不可修改

类别	参数	说明	备注
	xengine_lock_wait_timeout	锁等待超时时间	不可修改

运行状态指标

下表为X-Engine的运行状态指标，您可以在监控页面查看。

指标名	含义
xengine_rows_deleted	删除行数
xengine_rows_inserted	写入行数
xengine_rows_read	读取行数
xengine_rows_updated	更新行数
xengine_system_rows_deleted	对引擎为X-Engine的系统表的删除次数
xengine_system_rows_inserted	对引擎为X-Engine的系统表的插入次数
xengine_system_rows_read	对引擎为X-Engine的系统表的读取次数
xengine_system_rows_updated	对引擎为X-Engine的系统表的更新次数
xengine_block_cache_add	向Block Cache添加次数
xengine_block_cache_data_hit	读数据Block命中Cache次数
xengine_block_cache_data_miss	读数据Block时Miss次数
xengine_block_cache_filter_hit	Filter Block的命中次数
xengine_block_cache_filter_miss	Filter Block的miss次数
xengine_block_cache_hit	Block Cache的整体命中次数（data_hit + index_hit）
xengine_block_cache_index_hit	索引Block命中次数
xengine_block_cache_index_miss	索引Block miss次数
xengine_block_cache_miss	Block Cache整体Miss次数（data_miss + index_miss）
xengine_block_cachecompressed_miss	压缩的Block Cache Miss次数
xengine_bytes_read	读物理磁盘的字节数
xengine_bytes_written	加入Block Cache的字节数
xengine_memtable_hit	Memtable命中次数
xengine_memtable_miss	Memtable Miss次数

指标名	含义
xengine_number_block_not_compressed	未压缩的Block数目
xengine_number_keys_read	Key的读取次数
xengine_number_keys_updated	Key的更新次数
xengine_number_keys_written	Key的写入次数
xengine_number_superversion_acquires	Superversion引用的申请次数统计
xengine_number_superversion_cleanups	Superversion的清理次数。当一个Superversion无人再引用时则被清理。
xengine_number_superversion_releases	Superversion的引用释放次数，当一个Superversion的引用次数为0时则被清理。
xengine_snapshot_conflict_errors	在RR隔离级别下，因为Snapshot版本冲突而报错的次数。
xengine_wal_bytes	Redo落盘字节数
xengine_wal_group_syncs	Redo执行GroupCommit的次数
xengine_wal_synced	Redo日志Sync的次数
xengine_write_other	在事务流水线中，作为Follower完成提交的次数。
xengine_write_self	在事务流水线中，作为Leader完成提交的次数。
xengine_write_wal	写Redo日志的次数

3.3 InnoDB/TokuDB引擎转换为X-Engine引擎

RDS MySQL 8.0支持X-Engine引擎，X-Engine可以提供更好的数据压缩能力，降低磁盘空间成本。本文介绍如何将InnoDB/TokuDB引擎转换为X-Engine引擎。

背景信息

X-Engine是阿里云自研的联机事务处理OLTP（On-Line Transaction Processing）数据库存储引擎。作为自研数据库POLARDB的存储引擎之一，X-Engine已经广泛应用在阿里集团内部诸多业务系统中，包括交易历史库、钉钉历史库等，大幅缩减了业务成本；同时也作为双十一大促的关键数据库技术，挺过了数百倍平时流量的冲击。

更多详情请参见：

- *X-Engine*引擎使用须知

- *#unique_40*

注意事项

- **如果原表为InnoDB引擎，转换前请确保实例的剩余磁盘空间是现有数据量的2倍。转换为X-Engine引擎后空间占用会减小至原数据大小的10%~50%。**
- **使用方案二进行全库迁移时，由于需要切换连接地址，请在业务低峰期进行操作。**
- **在转换线上业务引擎之前，请提前进行SQL兼容性测试，确认正常后再转换线上业务引擎。**
- **转换引擎后请修改实例参数default_storage_engine为xengine，这样后续创建的表的引擎即为X-Engine。**

方案建议

- **实例满足如下条件时建议使用方案一，锁表时间短，而且无需配置各种工具。**
  - **实例为RDS MySQL 8.0本地SSD盘。**
  - **已支持X-Engine引擎（可以通过`show engines`确认是否支持X-Engine）。**

    说明:
    如果没有X-Engine，您需要*#unique_22*。
  - **单个表较小（100M以下）。**
  - **业务可接受短时间阻塞。**
- **其他情况时（例如实例为MySQL 5.6/5.7），建议您使用方案二。**

方案一

此方案为直接转换引擎，但有所限制，会全程阻塞DML操作，且大表转换时间较长。

1. *#unique_43*。

2. 在页面上方选择SQL操作 > SQL窗口。

3. 执行如下命令进行转换：

```
alter table <数据库名>.<表名> engine xengine;
```

示例

```
alter table test.sbtest1 engine xengine;
```

方案二

此方案为使用阿里云的数据传输服务DTS（Data Transmission Service）实时同步原表数据到新实例，然后将业务切换到新实例。

1. 执行如下步骤导出原实例的所有表结构脚本。

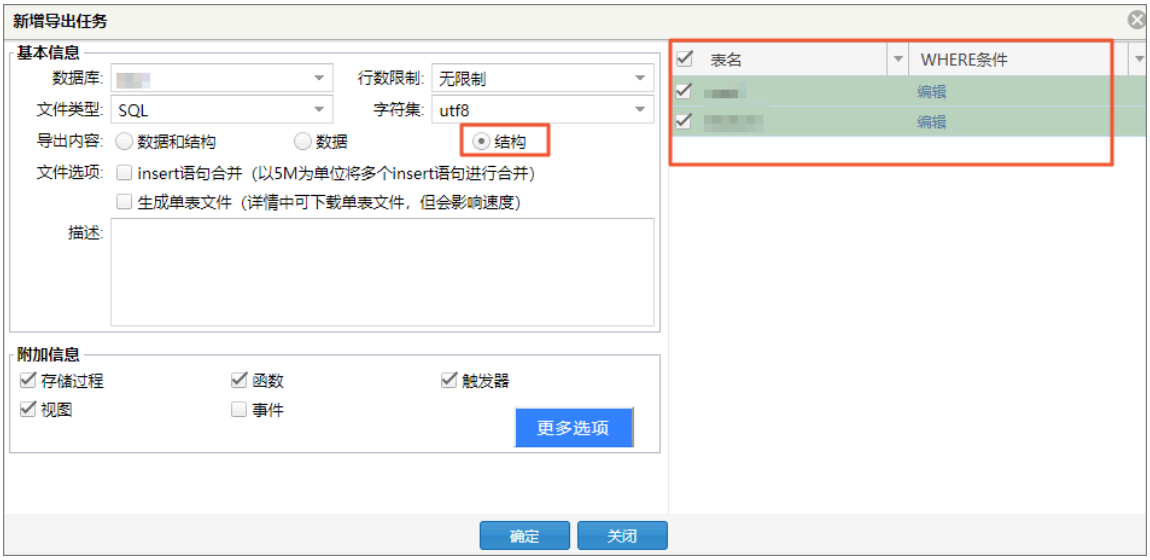
   a) 通过*DMS*登录原实例。

   b) 在页面上方选择数据方案 > 导出。

   c) 选择新增导出任务 > 导出数据库。

   d) 按下图设置导出数据库的所有表结构脚本，单击确定，在弹出的提示框中单击YES。

> **说明:**
> 弹出提示框是因为勾选了更多选项中的SQL脚本扩展选项，忽略即可。



2. 解压缩后修改表结构脚本，将InnoDB或TokuDB修改为xengine。

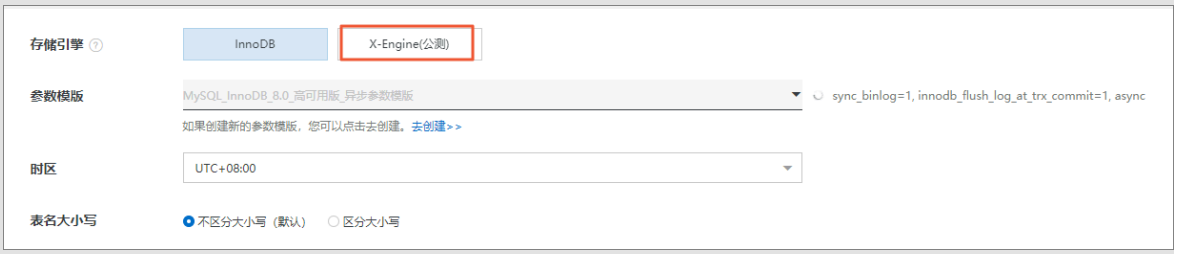
```
SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS `sbtest1`;
CREATE TABLE `sbtest1` (
  `id` int(11) NOT NULL,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `sbtest10`;
CREATE TABLE `sbtest10` (
  `id` int(11) NOT NULL,
  `k` int(11) NOT NULL DEFAULT '0',
```

改为 xengine

3. 新购一个与原实例规格相同的RDS MySQL 8.0实例（购买时选择X-Engine参数模板）。

说明:

在#unique_4时，可以选择应用默认的X-Engine参数模板来设置X-Engine为默认存储引擎。



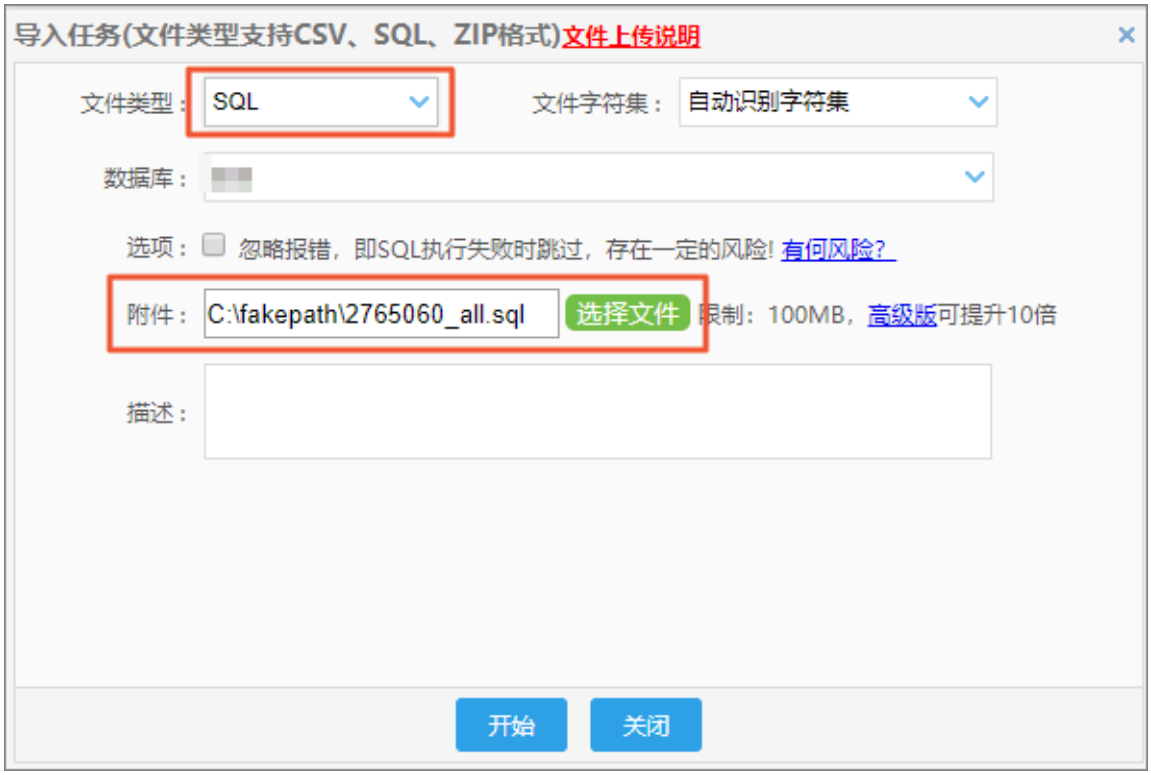
4. 执行如下步骤将表结构脚本导入新实例。

a) 通过*DMS*登录新实例。

b) 在页面上方选择数据方案 > 导入。

c) 单击新增任务。

d) 按下图设置导入表结构脚本，单击开始。



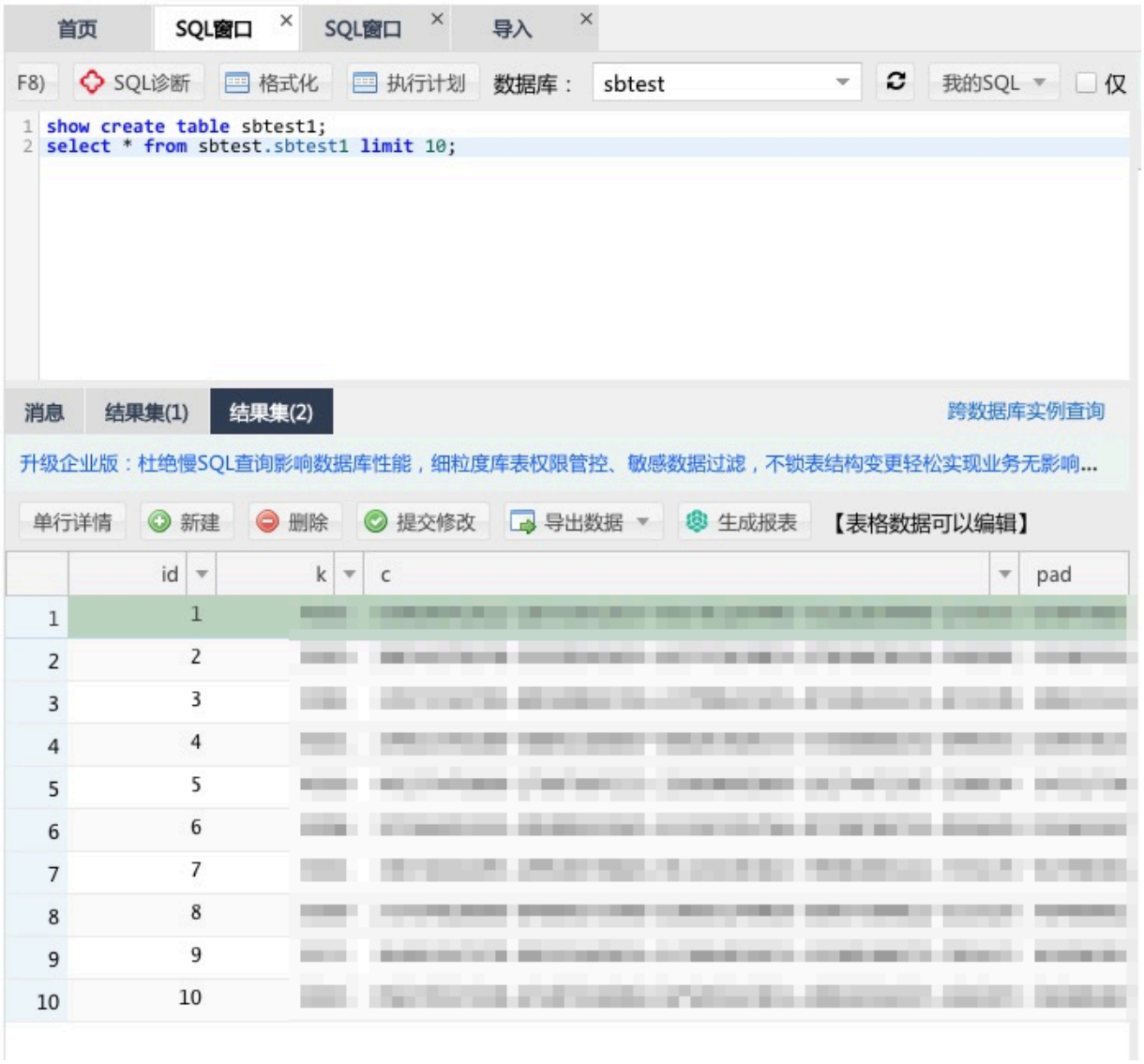
说明:

导入成功后可以通过`show create table <表名>;`命令确认表引擎为X-Engine。

5. 参考*#unique_44*将原实例的数据同步至新实例。

预期结果

同步完成后，您可以查询数据是否同步成功，然后进行SQL兼容性测试，测试正常再转换线上业务引擎。


首页
SQL窗口
SQL窗口
导入
F8)
SQL诊断
格式化
执行计划
数据库：
sbtest
我的SQL
仅
1 show create table sbtest1;
2 select * from sbtest.sbtest1 limit 10;
消息
结果集(1)
结果集(2)
跨数据库实例查询
升级企业版：杜绝慢SQL查询影响数据库性能，细粒度库表权限管控、敏感数据过滤，不锁表结构变更轻松实现业务无影响...
单行详情
新建
删除
提交修改
导出数据
生成报表
【表格数据可以编辑】
id
k
c
pad
1
2
3
4
5
6
7
8
9
10

4 Statement Concurrency Control

为了应对突发的数据库请求流量、资源消耗过高的语句访问以及SQL访问模型的变化，保证MySQL实例持续稳定运行，阿里云提供基于语句规则的并发控制CCL（Concurrency Control），并提供了工具包（DBMS_CCL）便于您快捷使用。

前提条件

实例版本为RDS MySQL 8.0。

注意事项

- CCL的操作不产生Binlog，所以CCL的操作只影响当前实例。例如主实例进行CCL操作，不会同步到备实例、只读实例或灾备实例。
- CCL提供超时机制以应对DML导致事务锁死锁，等待中的线程也会响应事务超时和线程KILL操作以应对死锁。

功能设计

CCL规则定义了如下三个维度的特征：

- SQL command

  SQL命令类型，例如SELECT、UPDATE、INSERT、DELETE等。
- Object

  SQL命令操作的对象，例如TABLE、VIEW等。
- keywords

  SQL命令的关键字。

创建CCL规则表

AliSQL设计了一个系统表（concurrency_control）保存CCL规则，系统启动时会自动创建该表，无需您手动创建。这里提供表的创建语句供您参考：

```
CREATE TABLE `concurrency_control` (
  `Id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Type` enum('SELECT','UPDATE','INSERT','DELETE') NOT NULL DEFAULT '
SELECT',
  `Schema_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Table_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Concurrency_count` bigint(20) DEFAULT NULL,
  `Keywords` text COLLATE utf8_bin,
  `State` enum('N','Y') NOT NULL DEFAULT 'Y',
  `Ordered` enum('N','Y') NOT NULL DEFAULT 'N',
  PRIMARY KEY (`Id`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_bin
```

```
STATS_PERSISTENT=0 COMMENT='Concurrency control'
```

参数	说明
Id	CCL规则ID。
Type	SQL command，即SQL命令类型。
Schema_name	数据库名。
Table_name	数据库内的表名。
Concurrency_count	并发数。
Keywords	关键字，多个关键字用英文分号（;）分隔。
State	本规则是否启用。
Ordered	Keywords中多个关键字是否按顺序匹配。

管理CCL规则

为了便捷地管理CCL规则，AliSQL在DBMS_CCL中定义了四个本地存储规则。详细说明如下：

- add_ccl_rule

  增加规则。命令如下：

  ```
  dbms_ccl.add_ccl_rule('<Type>','<Schema_name>','<Table_name>',<
  Concurrency_count>,'<Keywords>');
  ```

  示例：

  SELECT语句的并发数为10。

  ```
  mysql> call dbms_ccl.add_ccl_rule('SELECT', '', '', 10, '');
  ```

  SELECT语句中出现关键字key1的并发数为20。

  ```
  mysql> call dbms_ccl.add_ccl_rule('SELECT', '', '', 20, 'key1');
  ```

  test.t表的SELECT语句的并发数为10。

  ```
  mysql> call dbms_ccl.add_ccl_rule('SELECT', '', '', 10, '');
  ```

   说明：

  Id越大，规则的优先级越高。

- **del_ccl_rule**

  删除规则。命令如下：

  ```
  dbms_ccl.del_ccl_rule(<Id>);
  ```

  示例：

  删除规则ID为15的CCL规则。

  ```
  mysql> call dbms_ccl.del_ccl_rule(15);
  ```

  说明：

  如果删除的规则不存在，系统会报相应的警告，您可以使用`show warnings;`查看警告内容。

  ```
  mysql> call dbms_ccl.del_ccl_rule(100);
    Query OK, 0 rows affected, 2 warnings (0.00 sec)

  mysql> show warnings;
  +---------+------
  +-----------------------------------------------------+
  | Level   | Code | Message
     |
  +---------+------
  +-----------------------------------------------------+
  | Warning | 7514 | Concurrency control rule 100 is not found in
  table |
  | Warning | 7514 | Concurrency control rule 100 is not found in
  cache |
  +---------+------
  +-----------------------------------------------------+
  ```

- **show_ccl_rule**

  查看内存中已启用规则。命令如下：

  ```
  dbms_ccl.show_ccl_rule();
  ```

  示例：

  ```
  mysql> call dbms_ccl.show_ccl_rule();
  +------+--------+--------+-------+-------+-------
  +-------------------+---------+---------+----------+----------+
  | ID   | TYPE   | SCHEMA | TABLE | STATE | ORDER | CONCURRENCY_COUNT
   | MATCHED | RUNNING | WAITTING | KEYWORDS |
  +------+--------+--------+-------+-------+-------
  +-------------------+---------+---------+----------+----------+
  |   17 | SELECT | test   | t     | Y     | N     |                30
   |       0 |       0 |       0 |          |
  |   16 | SELECT |        |       | Y     | N     |                20
   |       0 |       0 |       0 | key1     |
  |   18 | SELECT |        |       | Y     | N     |                10
   |       0 |       0 |       0 |          |
  ```

```
+------+--------+--------+-------+-------+-------
+-------------------+---------+---------+----------+----------+
```

关于MATCHED、RUNNING和WAITTING的说明如下。

参数	说明
MATCHED	规则匹配成功次数。
RUNNING	此规则下正在并发执行的线程数。
WAITTING	此规则下正在等待执行的线程数。

- flush_ccl_rule

  如果您直接操作了表concurrency_control修改规则，规则不能立即生效，您需要让规则重新生效。命令如下：

  ```
  dbms_ccl.flush_ccl_rule();
  ```

  示例：

  ```
  mysql> update mysql.concurrency_control set CONCURRENCY_COUNT = 15
  where Id = 18;
  Query OK, 1 row affected (0.00 sec)
  Rows matched: 1  Changed: 1  Warnings: 0

  mysql> call dbms_ccl.flush_ccl_rule();
  Query OK, 0 rows affected (0.00 sec)
  ```

功能测试

- 测试规则

  设计如下三条规则对应三个维度：

  ```
  call dbms_ccl.add_ccl_rule('SELECT', 'test', 'sbtest1', 3, '');  //
  SELECT命令操作表sbtest1并发数为3
  call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, 'sbtest2');       //
  SELECT命令关键字sbtest2并发数为2
  call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, '');            //
  SELECT命令并发数为2
  ```

- 测试场景

  使用sysbench进行测试，场景如下：

  - 64 threads
  - 4 tables
  - select.lua

- **测试结果**

  查看规则并发数情况如下：

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+---------+-------+-------
+-------------------+---------+---------+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE   | STATE | ORDER | CONCURRENC
Y_COUNT | MATCHED | RUNNING | WAITTING | KEYWORDS |
+------+--------+--------+---------+-------+-------
+-------------------+---------+---------+----------+----------+
|   20 | SELECT | test   | sbtest1 | Y     | N     |
  3 |     389 |       3 |         9 |          |
|   21 | SELECT |        |         | Y     | N     |
  2 |     375 |       2 |        14 | sbtest2  |
|   22 | SELECT |        |         | Y     | N     |
  2 |     519 |       2 |        34 |          |
+------+--------+--------+---------+-------+-------
+-------------------+---------+---------+----------+----------+
3 rows in set (0.00 sec)
```

  查看RUNNING列，符合预期的并行数量。

5 Statement Outline

生产环境中，SQL语句的执行计划经常会发生改变，导致数据库不稳定。阿里云利用Optimizer Hint和Index Hint让MySQL稳定执行计划，该方法称为Statement Outline，并提供了工具包（DBMS_OUTLN）便于您快捷使用。

前提条件

实例版本为RDS MySQL 8.0。

功能设计

Statement Outline支持官方MySQL 8.0的所有hint类型，分为如下两类：

- **Optimizer Hint**

  根据作用域和hint对象，分为Global level hint、Table/Index level hint、Join order hint等。详情请参见*MySQL*官网。

- **Index Hint**

  根据Index Hint的类型和范围进行分类。详情请参见*MySQL*官网

Statement Outline表介绍

AliSQL内置了一个系统表（outline）保存hint，系统启动时会自动创建该表，无需您手动创建。这里提供表的创建语句供您参考：

```
CREATE TABLE `mysql`.`outline` (
  `Id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Schema_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Digest` varchar(64) COLLATE utf8_bin NOT NULL,
  `Digest_text` longtext COLLATE utf8_bin,
  `Type` enum('IGNORE INDEX','USE INDEX','FORCE INDEX','OPTIMIZER')
CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `Scope` enum('','FOR JOIN','FOR ORDER BY','FOR GROUP BY') CHARACTER
SET utf8 COLLATE utf8_general_ci DEFAULT '',
  `State` enum('N','Y') CHARACTER SET utf8 COLLATE utf8_general_ci NOT
 NULL DEFAULT 'Y',
  `Position` bigint(20) NOT NULL,
  `Hint` text COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`Id`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0 COMMENT='
Statement outline'
```

参数说明如下。

参数	说明
Id	**Outline ID。**

参数	说明
Schema_name	数据库名。
Digest	Digest_text进行hash计算得到的64字节的hash字符串。
Digest_text	SQL语句的特征。
Type	· Optimizer Hint中，hint类型的取值为OPTIMIZER。 · Index Hint中，hint类型的取值为USE INDEX、FORCE INDEX或IGNORE INDEX。
Scope	仅Index Hint需要提供，分为如下三类： · FOR GROUP BY · FOR ORDER BY · FOR JOIN 空串表示所有类型的Index Hint。
State	本规则是否启用。
Position	· Optimizer Hint中，Position表示Query Block，因为所有的Optimizer Hint必须作用到 Query Block上，所以，Position从1开始，hint作用在语句的第几个关键字上，Position就是几。 · Index Hint中，Position表示表的位置，也是从1开始，hint作用在第几个表上，Position就是几。
Hint	· Optimizer Hint中，Hint表示完整的hint字符串，例如/*+ MAX_EXECUTION_TIME(1000) */。 · Index Hint中，Hint表示索引名字的列表，例如ind_1,ind_2。

管理Statement Outline

为了便捷地管理Statement Outline，AliSQL在DBMS_OUTLN中定义了六个本地存储规则。详细说明如下：

· add_optimizer_outline

增加Optimizer Hint。命令如下：

```
dbms_outln.add_optimizer_outline('<Schema_name>','<Digest>','<
query_block>','<hint>','<query>');
```

说明：

Digest和Query（原始SQL语句）可以任选其一。如果填写Query，DBMS_OUTLN会计算Digest和Digest_text。

示例：

```
CALL DBMS_OUTLN.add_optimizer_outline("outline_db", '', 1, '/*+
MAX_EXECUTION_TIME(1000) */',
                                      "select * from t1 where id = 1
");
```

- **add_index_outline**

  增加Index Hint。命令如下：

  ```
  dbms_outln.add_index_outline('<Schema_name>','<Digest>',<Position
  >,'<Type>','<Hint>','<Scope>','<Query>');
  ```

  说明：

  Digest和Query（原始SQL语句）可以任选其一。如果填写Query，DBMS_OUTLN会计算Digest和Digest_text。

  示例：

  ```
  call dbms_outln.add_index_outline('outline_db', '', 1, 'USE INDEX',
   'ind_1', '',
                                    "select * from t1 where t1.col1 =1
  and t1.col2 ='xpchild'");
  ```

- **preview_outline**

  查看匹配Statement Outline的情况，可用于手动验证。命令如下：

  ```
  dbms_outln.preview_outline('<Schema_name>','<Query>');
  ```

  示例：

  ```
  mysql> call dbms_outln.preview_outline('outline_db', "select * from
  t1 where t1.col1 =1 and t1.col2 ='xpchild'");
  +------------
  +-----------------------------------------------------------------
  +------------+------------+-------+--------------------+
  | SCHEMA     | DIGEST
                | BLOCK_TYPE | BLOCK_NAME | BLOCK | HINT
   |
  +------------
  +-----------------------------------------------------------------
  +------------+------------+-------+--------------------+
  | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b
  62d0a140628c4c | TABLE      | t1         |     1 | USE INDEX (`ind_1
  `) |
  +------------
  +-----------------------------------------------------------------
  +------------+------------+-------+--------------------+
  ```

```
1 row in set (0.00 sec)
```

- **show_outline**

  展示Statement Outline在内存中命中的情况。命令如下：

```
dbms_outln.show_outline();
```

  示例：

```
mysql> call dbms_outln.show_outline();
+------+------------
+-----------------------------------------------------------------------
+-----------+-------+------
+-----------------------------------------------------+------
+----------
+-------------------------------------------------------------------------------
+
| ID   | SCHEMA     | DIGEST
                   | TYPE      | SCOPE | POS  | HINT
                                    | HIT  | OVERFLOW | DIGEST_TEXT

  |
+------+------------
+-----------------------------------------------------------------------
+-----------+-------+------
+-----------------------------------------------------+------
+----------
+-------------------------------------------------------------------------------
+
|   33 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d8951
07e2d8d3848d1c60b74bcde6 | OPTIMIZER |       |    1 | /*+ SET_VAR(
foreign_key_checks=OFF) */                |    1 |        0 | SELECT
 * FROM `t1` WHERE `id` = ?
          |
|   32 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d8951
07e2d8d3848d1c60b74bcde6 | OPTIMIZER |       |    1 | /*+ MAX_EXECUT
ION_TIME(1000) */                         |    2 |        0 | SELECT
 * FROM `t1` WHERE `id` = ?
          |
|   34 | outline_db | d4dcef634a4a664518e5fb8a21c6ce9b79fccb44
b773e86431eb67840975b649 | OPTIMIZER |       |    1 | /*+ BNL(t1,t2)
 */                                     |    1 |        0 | SELECT `
t1` . `id` , `t2` . `id` FROM `t1` , `t2`
        |
|   35 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f
015f2ee1f6f2d12dfad72e6f | OPTIMIZER |       |    2 |  /*+ QB_NAME(
subq1) */                                 |    2 |        0 | SELECT
 * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )
          |
|   36 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f
015f2ee1f6f2d12dfad72e6f | OPTIMIZER |       |    1 | /*+ SEMIJOIN
(@subq1 MATERIALIZATION, DUPSWEEDOUT) */  |    2 |        0 | SELECT
 * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )
          |
|   30 | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b
928a1d735b62d0a140628c4c | USE INDEX |       |    1 | ind_1
                                          |    3 |        0 | SELECT
 * FROM `t1` WHERE `t1` . `col1` = ? AND `t1` . `col2` = ?
          |
|   31 | outline_db | 33c71541754093f78a1f2108795cfb45f8b15ec5
d6bff76884f4461fb7f33419 | USE INDEX |       |    2 | ind_2
```

```
                                                |    1 |         0 | SELECT
 * FROM `t1` , `t2` WHERE `t1` . `col1` = `t2` . `col1` AND `t2` . `
col2` = ? |
+------+------------
+-----------------------------------------------------------------------
+-----------+-------+------
+-------------------------------------------------------+------
+----------
+-------------------------------------------------------------------------------
+
7 rows in set (0.00 sec)
```

关于HIT和OVERFLOW的说明如下。

参数	说明
HIT	**此Statement Outline命中的次数。**
OVERFLOW	**此Statement Outline没有找到Query block或相应的表的次数。**

- **del_outline**

  删除内存和表中的某一条Statement Outline。命令如下：

  ```
  dbms_outln.del_outline(<Id>);
  ```

  示例：

  ```
  mysql> call dbms_outln.del_outline(32);
  ```

  说明：

  如果删除的规则不存在，系统会报相应的警告，您可以使用`show warnings;`查看警告内容。

  ```
  mysql> call dbms_outln.del_outline(1000);
  Query OK, 0 rows affected, 2 warnings (0.00 sec)

  mysql> show warnings;
  +---------+------+-----------------------------------------------+
  | Level   | Code | Message                                       |
  +---------+------+-----------------------------------------------+
  | Warning | 7521 | Statement outline 1000 is not found in table |
  | Warning | 7521 | Statement outline 1000 is not found in cache |
  +---------+------+-----------------------------------------------+
  ```

```
 2 rows in set (0.00 sec)
```

- **flush_outline**

  如果您直接操作了表outline修改Statement Outline，您需要让Statement Outline重新生效。命令如下：

  ```
  dbms_outln.flush_outline();
  ```

  示例：

  ```
  mysql> update mysql.outline set Position = 1 where Id = 18;
  Query OK, 1 row affected (0.00 sec)
  Rows matched: 1  Changed: 1  Warnings: 0

  mysql> call dbms_outln.flush_outline();
  Query OK, 0 rows affected (0.01 sec)
  ```

功能测试

验证Statement Outline是否有效果，有如下两种方法：

- **通过preview_outline进行预览。**

  ```
  mysql> call dbms_outln.preview_outline('outline_db', "select * from
  t1 where t1.col1 =1 and t1.col2 ='xpchild'");
  +------------
  +------------------------------------------------------------------
  +------------+------------+-------+--------------------+
  | SCHEMA     | DIGEST
              | BLOCK_TYPE | BLOCK_NAME | BLOCK | HINT
   |
  +------------
  +------------------------------------------------------------------
  +------------+------------+-------+--------------------+
  | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b
  62d0a140628c4c | TABLE      | t1         |     1 | USE INDEX (`ind_1
  `) |
  +------------
  +------------------------------------------------------------------
  +------------+------------+-------+--------------------+
  1 row in set (0.01 sec)
  ```

- **直接使用explain查看。**

  ```
  mysql> explain select * from t1 where t1.col1 =1 and t1.col2 ='
  xpchild';
  +----+-------------+-------+------------+------+---------------
  +-------+---------+-------+------+----------+-------------+
  | id | select_type | table | partitions | type | possible_keys | key
     | key_len | ref   | rows | filtered | Extra       |
  +----+-------------+-------+------------+------+---------------
  +-------+---------+-------+------+----------+-------------+
  |  1 | SIMPLE      | t1    | NULL       | ref  | ind_1         |
  ind_1 | 5       | const |    1 |   100.00 | Using where |
  +----+-------------+-------+------------+------+---------------
  +-------+---------+-------+------+----------+-------------+
  1 row in set, 1 warning (0.00 sec)
  ```

```
mysql> show warnings;
+-------+------
+-----------------------------------------------------------------------------
+
| Level | Code | Message


                                                                      |
+-------+------
+-----------------------------------------------------------------------------
+
| Note  | 1003 | /* select#1 */ select `outline_db`.`t1`.`id` AS `
id`,`outline_db`.`t1`.`col1` AS `col1`,`outline_db`.`t1`.`col2` AS `
col2` from `outline_db`.`t1` USE INDEX (`ind_1`) where ((`outline_db
`.`t1`.`col1` = 1) and (`outline_db`.`t1`.`col2` = 'xpchild')) |
+-------+------
+-----------------------------------------------------------------------------
+
1 row in set (0.00 sec)
```

6 Recycle Bin

由于DDL语句无法回滚，开发或运维人员如果误操作（例如DROP TABLE）可能会导致数据丢失。阿里云支持回收站（Recycle Bin）功能，临时将删除的表转移到回收站，还可以设置保留的时间，方便您找回数据，同时提供了工具包（DBMS_RECYCLE）便于您快捷使用。

Recycle Bin参数

Recycle Bin设计了如下五个参数。

参数	说明
recycle_bin	是否打开回收站功能，包括session级别和global级别。
recycle_bin_retention	回收站保留时间，单位：秒。默认为604800，即一周。
recycle_scheduler	是否打开回收站的异步清理任务线程。
recycle_scheduler_interval	回收站异步清理任务线程的轮询间隔，单位：秒。默认为30。
recycle_scheduler_purge_table_print	是否打印异步清理现场工作的详细日志。

Recycle Bin介绍

- **回收/清理机制**
  - **回收机制**

    执行DROP TABLE/DATABASE语句时，只保留相关的表对象，并移动到专门的recycle bin目录中。其它对象的删除策略如下：

    ■ 如果是与表无关的对象，根据操作语句决定是否保留，不做回收。

    ■ 如果是表的附属对象，可能会修改表数据的，做删除处理，例如Trigger和Foreign key。但Column statistics不做清理，随表进入回收站。
  - **清理机制**

    回收站会启动一个后台线程，来异步清理超过recycle_bin_retention时间的表对象。在清理回收站表的时候，如果遇到大表，会再启动一个后台线程异步删除大表。

- **权限**

  RDS MySQL实例启动时，会初始化一个名为__recycle_bin__的数据库，作为回收站使用的专有数据库。__recycle_bin__是系统级数据库，您无法直接进行修改和删除。

  对于回收站内的表，虽然您无法直接执行drop table语句，但是可以使用call dbms_recycle.purge_table('<TABLE>');进行清理。

  > **说明:**
  > **账号在原表和回收站表都需要具有DROP权限。**

- **回收站表命名规则**

  Recycle Bin会从不同的数据库回收到统一的__recycle_bin__数据库中，所以需要保证目标表表名唯一，所以定义了如下命名格式：

  ```
  "__" + <Storage Engine> + <SE private id>
  ```

  参数说明如下。

参数	说明
Storage Engine	存储引擎名称。
SE private id	存储引擎为每一个表生成的唯一值。例如在InnoDB引擎中就是table id。

- **独立回收**

  回收的设置只会影响该实例本身，不会影响到binlog复制到的节点（备实例、只读实例和灾备实例）上。例如我们可以在主实例上设置回收，保留7天；在备实例上设置回收，保留14天。

  > **说明:**
  > **回收站保留周期不同，将导致实例的空间占用差别比较大。**

注意事项

- **如果回收站数据库和待回收的表跨了文件系统，执行drop table语句将会搬迁表空间文件，耗时较长。**
- **如果Tablespace为General，可能会存在多个表共享同一个表空间的情况，当回收其中一张表的时候，不会搬迁相关的表空间文件。**

前提条件

实例版本为RDS MySQL 8.0。

管理Recycle Bin

AliSQL在DBMS_RECYCLE中定义了两个管理接口。详细说明如下：

- **show_tables**

  展示回收站中所有临时保存的表。命令如下：

  ```
  call dbms_recycle.show_tables();
  ```

  示例：

  ```
  mysql> call dbms_recycle.show_tables();
  +-----------------+--------------+---------------+-------------
  +---------------------+---------------------+
  | SCHEMA          | TABLE         | ORIGIN_SCHEMA | ORIGIN_TABLE |
  RECYCLED_TIME       | PURGE_TIME          |
  +-----------------+--------------+---------------+-------------
  +---------------------+---------------------+
  | __recycle_bin__ | __innodb_1063 | product_db    | t1           |
  2019-08-08 11:01:46 | 2019-08-15 11:01:46 |
  | __recycle_bin__ | __innodb_1064 | product_db    | t2           |
  2019-08-08 11:01:46 | 2019-08-15 11:01:46 |
  | __recycle_bin__ | __innodb_1065 | product_db    | parent       |
  2019-08-08 11:01:46 | 2019-08-15 11:01:46 |
  | __recycle_bin__ | __innodb_1066 | product_db    | child        |
  2019-08-08 11:01:46 | 2019-08-15 11:01:46 |
  +-----------------+--------------+---------------+-------------
  +---------------------+---------------------+
  4 rows in set (0.00 sec)
  ```

参数	说明
SCHEMA	**回收站的数据库名。**
TABLE	**进入回收站后的表名。**
ORIGIN_SCHEMA	**原数据库名。**
ORIGIN_TABLE	**原表名。**
RECYCLED_TIME	**回收时间。**
PURGE_TIME	**预计从回收站删除的时间。**

- **purge_table**

  手动清理回收站中的表。命令如下：

  ```
  call dbms_recycle.purge_table('<TABLE>');
  ```

  说明：

  - TABLE**为进入回收站后的表名。**

- **账号在原表和回收站表都需要具有DROP权限。**

示例：

```
mysql> call dbms_recycle.purge_table('__innodb_1063');
Query OK, 0 rows affected (0.01 sec)
```

7 Thread Pool

为了发挥出RDS的最佳性能，阿里云提供线程池（Thread Pool）功能，将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。

优势

MySQL默认的线程使用模式是会话独占模式，每个会话都会创建一个独占的线程。当有大量的会话存在时，会导致大量的资源竞争，大量的系统线程调度和缓存失效也会导致性能急剧下降。

阿里云RDS的线程池实现了不同类型SQL操作的优先级及并发控制机制，将连接数始终控制在最佳连接数附近，使RDS数据库在高连接大并发情况下始终保持高性能。线程池的优势如下：

- 当大量线程并发工作时，线程池会自动调节并发的线程数量在合理的范围内，从而避免线程调度工作过多和大量缓存失效。
- 大量的事务并发执行时，线程池会将语句和事务分为不同的优先级，分别控制语句和事务的并发数量，从而减少资源竞争。
- 线程池给予管理类的SQL语句更高的优先级，保证这些语句优先执行。这样在系统负载很高时，新建连接、管理、监控等操作也能够稳定执行。
- 线程池给予复杂查询SQL语句相对较低的优先级，并且有最大并发数的限制。这样可以避免过多的复杂SQL语句将系统资源耗尽，导致整个数据库服务不可用。

前提条件

实例版本为RDS MySQL 5.7/8.0。

使用Thread Pool

Thread Pool设计了如下三个参数，您可以在控制台进行修改。详情请参见*#unique_45*。

参数	说明
thread_pool_enabled	是否开启线程池功能。取值： · ON · OFF 默认值：ON。 说明： · 开启/关闭线程池功能使用本参数即可，不再使用参数thread_handling进行控制。 · 开启/关闭线程池功能无需重启实例。

参数	说明
thread_pool_size	分组的数量，默认值：4。线程池中线程的被平均的分到多个组中进行管理。
thread_pool_oversubscribe	每个组中允许的活跃线程的数量，默认值：16。活跃线程是指正在执行SQL语句的线程，但是不包括以下两种情形： · SQL语句在等待磁盘IO； · SQL语句在等待事务提交。

查询Thread Pool状态

您可以通过如下命令查询Thread Pool状态：

```
show status like "thread_pool%";
```

示例：

```
mysql> show status like "thread_pool%";
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| thread_pool_active_threads | 1     |
| thread_pool_big_threads    | 0     |
| thread_pool_dml_threads    | 0     |
| thread_pool_idle_threads   | 19    |
| thread_pool_qry_threads    | 0     |
| thread_pool_total_threads  | 20    |
| thread_pool_trx_threads    | 0     |
| thread_pool_wait_threads   | 0     |
+----------------------------+-------+
8 rows in set (0.00 sec)
```

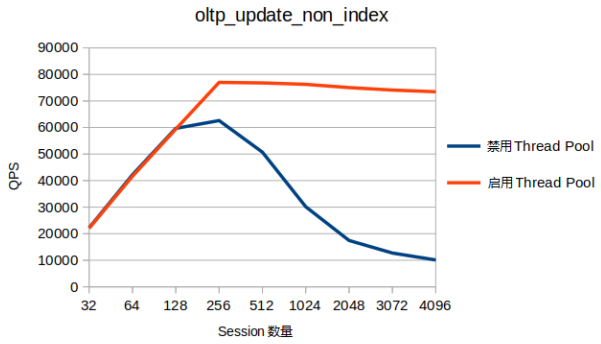
参数说明如下。

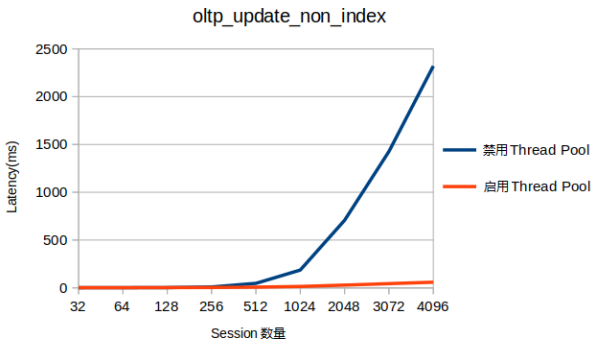
参数	说明
thread_pool_active_threads	线程池中的活跃线程数。
thread_pool_big_threads	线程池中正在执行复杂查询的线程数。复杂查询包括有子查询、聚合函数、group by、limit等的查询语句。
thread_pool_dml_threads	线程池中的在执行DML的线程数。
thread_pool_idle_threads	线程池中的空闲线程数。
thread_pool_qry_threads	线程池中正在执行简单查询的线程数。

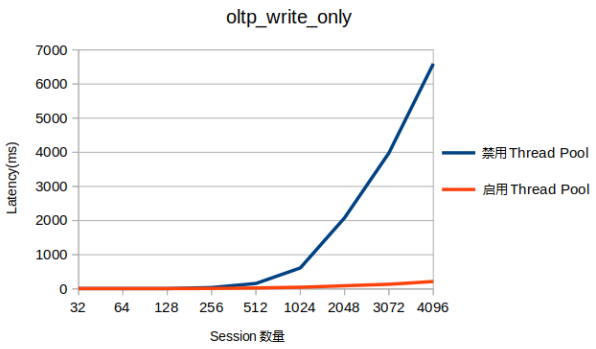
参数	说明
thread_pool_total_threads	线程池中的总线程数。
thread_pool_trx_threads	线程池中正在执行事务的线程数。
thread_pool_wait_threads	线程池中正在等待磁盘IO、事务提交的线程数。

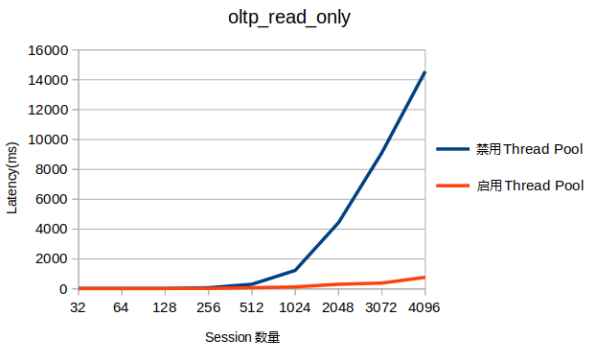
Sysbench测试

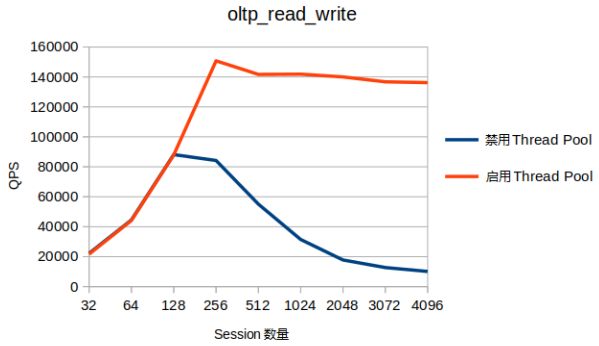
如下是开启线程池和不开启线程池的性能对比。从测试结果可以看出线程池在高并发的情况下有着明显的性能优势。

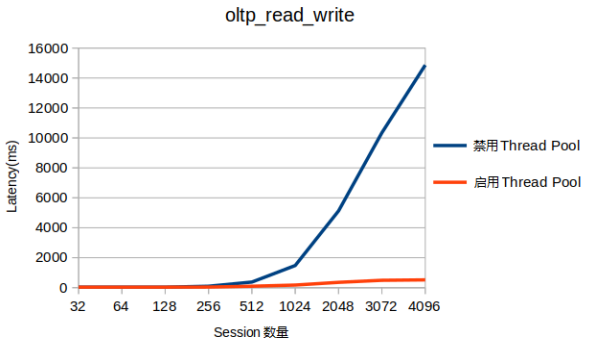


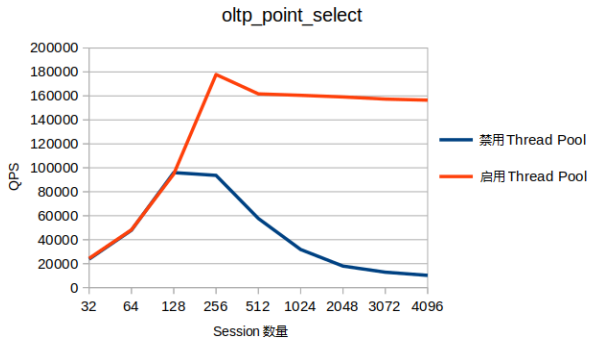


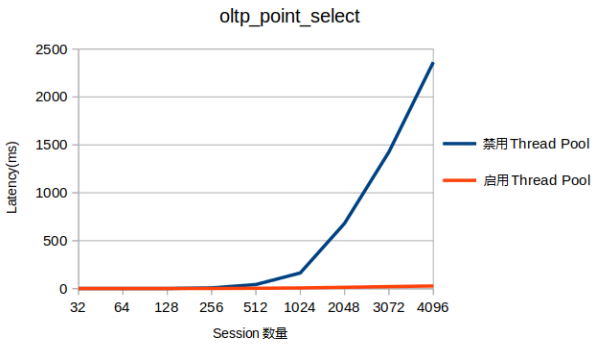





oltp_read_write
QPS
160000
140000
120000
100000
80000
60000
40000
20000
0
32 64 128 256 512 1024 2048 3072 4096
Session 数量
禁用Thread Pool
启用Thread Pool


oltp_read_write
Latency(ms)
16000
14000
12000
10000
8000
6000
4000
2000
0
32 64 128 256 512 1024 2048 3072 4096
Session 数量
禁用Thread Pool
启用Thread Pool


oltp_point_select
QPS
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
32 64 128 256 512 1024 2048 3072 4096
Session 数量
禁用Thread Pool
启用Thread Pool


oltp_point_select
Latency(ms)
2500
2000
1500
1000
500
0
32 64 128 256 512 1024 2048 3072 4096
Session 数量
禁用Thread Pool
启用Thread Pool

8 Sequence Engine

AliSQL提供了Sequence Engine，简化获取序列值的复杂度。

Sequence Engine介绍

在持久化数据库系统中，无论是单节点中的业务主键，还是分布式系统中的全局唯一值，亦或是多系统中的幂等控制，单调递增的唯一值是常见的需求。不同的数据库系统有不同的实现方法，例如MySQL提供的AUTO_INCREMENT，Oracle、SQL Server提供的SEQUENCE。

在MySQL数据库中，如果业务希望封装唯一值，例如增加日期、用户等信息，使用AUTO_INCREMENT的方法会带来很大不便，在实际的系统设计中，也存在不同的折中方法：

- 序列值由Application或者Proxy来生成，不过弊端很明显，状态带到应用端会增加扩容和缩容的复杂度。
- 序列值由数据库通过模拟的表来生成，但需要中间件来封装和简化获取唯一值的逻辑。

AliSQL提供了Sequence Engine，通过引擎的设计方法，尽可能地兼容其他数据库的使用方法，简化获取序列值复杂度。

Sequence Engine实现了MySQL存储引擎的设计接口，但底层的数据仍然使用现有的存储引擎，例如InnoDB或者MyISAM来保存持久化数据，兼容现有的第三方工具（例如Xtrabackup），所以Sequence Engine仅仅是一个逻辑引擎。

Sequence Engine通过Sequence Handler接口访问Sequence对象，实现NEXTVAL的滚动、缓存的管理等，最后透传给底层的基表数据引擎，实现最终的数据访问。

前提条件

实例版本为RDS MySQL 5.7/8.0。

使用限制

- Sequence不支持子查询和join查询。
- 可以使用`SHOW CREATE TABLE`或者`SHOW CREATE SEQUENCE`来访问Sequence结构，但不能使用`SHOW CREATE SEQUENCE`访问普通表。
- 不支持建表的时候指定Sequence引擎，Sequence表只能通过创建*Sequence*的语法来创建。

创建Sequence

创建Sequence语句如下：

```
CREATE SEQUENCE [IF NOT EXISTS] schema.sequence_name
   [START WITH <constant>]
   [MINVALUE <constant>]
```

```
    [MAXVALUE <constant>]
    [INCREMENT BY <constant>]
    [CACHE <constant> | NOCACHE]
    [CYCLE | NOCYCLE]
  ;
```

参数说明如下。

参数	说明
START	**Sequence的起始值。**
MINVALUE	**Sequence的最小值。**
MAXVALUE	**Sequence的最大值。** **说明:** **如果有参数NOCYCLE，到达最大值后会报如下错误：** `ERROR HY000: Sequence 'db.seq' has been run out.`
INCREMENT BY	**Sequence的步长。**
CACHE/NOCACHE	**缓存的大小，为了性能考虑，可以设置较大的缓存，但如果遇到实例重启，缓存内的值会丢失。**
CYCLE/NOCYCLE	**表示Sequence如果用完了后，是否允许从MINVALUE重新开始。取值：** · CYCLE：**允许；** · NOCYCLE：**不允许。**

示例：

```
create sequence s
       start with 1
       minvalue 1
       maxvalue 9999999
       increment by 1
       cache 20
       cycle;
```

为了兼容MySQL Dump的备份方式，您也可以使用另外一种创建Sequence的方法，即创建Sequence表并插入一行初始记录。示例如下：

```
CREATE SEQUENCE schema.sequence_name (
  `currval` bigint(21) NOT NULL COMMENT 'current value',
  `nextval` bigint(21) NOT NULL COMMENT 'next value',
  `minvalue` bigint(21) NOT NULL COMMENT 'min value',
  `maxvalue` bigint(21) NOT NULL COMMENT 'max value',
  `start` bigint(21) NOT NULL COMMENT 'start value',
  `increment` bigint(21) NOT NULL COMMENT 'increment value',
  `cache` bigint(21) NOT NULL COMMENT 'cache size',
  `cycle` bigint(21) NOT NULL COMMENT 'cycle state',
  `round` bigint(21) NOT NULL COMMENT 'already how many round'
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

```
INSERT INTO schema.sequence_name VALUES(0,0,1,9223372036854775807,1,1,
10000,1,0);
COMMIT;
```

Sequence表介绍

由于Sequence是通过真正的引擎表来保存的，所以通过查询创建语句看到仍然是默认的引擎表。

示例如下：

```
SHOW CREATE [TABLE|SEQUENCE] schema.sequence_name;

CREATE SEQUENCE schema.sequence_name (
  `currval` bigint(21) NOT NULL COMMENT 'current value',
  `nextval` bigint(21) NOT NULL COMMENT 'next value',
  `minvalue` bigint(21) NOT NULL COMMENT 'min value',
  `maxvalue` bigint(21) NOT NULL COMMENT 'max value',
  `start` bigint(21) NOT NULL COMMENT 'start value',
  `increment` bigint(21) NOT NULL COMMENT 'increment value',
  `cache` bigint(21) NOT NULL COMMENT 'cache size',
  `cycle` bigint(21) NOT NULL COMMENT 'cycle state',
  `round` bigint(21) NOT NULL COMMENT 'already how many round'
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

查询语法

Sequence支持的查询语法如下：

```
 SELECT [nextval | currval | *] FROM seq;
 SELECT nextval(seq),currval(seq);
 SELECT seq.currval, seq.nextval from dual;
```

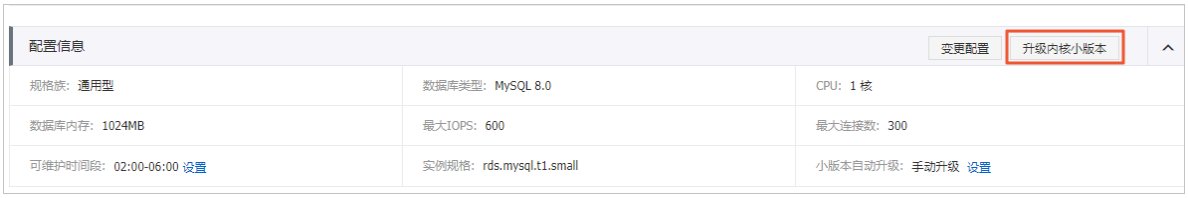
9 Performance Insight

Performance Insight是专注于实例负载监控、关联分析、性能调优的利器，帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。

前提条件

- **实例版本如下：**
  - **MySQL 8.0**
  - **MySQL 5.7**
- **内核小版本需要为20190915或以上。**

> **说明：**
> **您可以在基本信息页面的配置信息区域查看是否有升级内核小版本按钮。如果有按钮，您可以单击按钮查看当前版本；如果没有按钮，表示已经是最新版。详情请参见*#unique_22*。**



Performance Insight介绍

Performance Insight由如下两部分组成：

- **Object statistics**

  Object statistics查询表和索引的统计信息，包括如下两个表：

  - **TABLE_STATISTICS：记录读取和修改的行。**
  - **INDEX_STATISTICS：记录索引的读取行。**
- **Performance point**

  Performance point提供实例的详细性能信息，方便您更快更准确地量化SQL的开销。Performance point包括如下三个维度：

  - **CPU：包括执行任务的总时间（Elapsed time）、CPU执行任务的时间（CPU time）等。**
  - **LOCK：包括服务器MDL锁时间、存储事务锁时间、互斥冲突（仅调试模式）、读写锁冲突等。**
  - **IO：数据文件读写时间、日志文件写入时间、逻辑读取、物理读取、物理异步读取等。**

Object statistics使用方法

1. **确认参数OPT_TABLESTAT和OPT_INDEXSTAT的值为ON。示例如下：**

```
mysql> show variables like "opt_%_stat";
  +---------------+-------+
  | Variable_name | Value |
  +---------------+-------+
  | opt_indexstat | ON    |
  | opt_tablestat | ON    |
  +---------------+-------+
```

说明:

如果参数找不到或参数值不为ON，请确认您的实例版本是否为MySQL 5.7。

2. **在information_schema数据库查询TABLE_STATISTICS表或INDEX_STATISTICS表，查看表和索引的统计信息。示例如下：**

```
mysql> select * from TABLE_STATISTICS limit 10;
  +--------------+--------------+-----------+--------------
+-----------------------+---------------+--------------
+--------------+
  | TABLE_SCHEMA | TABLE_NAME   | ROWS_READ | ROWS_CHANGED |
ROWS_CHANGED_X_INDEXES | ROWS_INSERTED | ROWS_DELETED | ROWS_UPDATED
 |
  +--------------+--------------+-----------+--------------
+-----------------------+---------------+--------------
+--------------+
  | mysql        | db           |         2 |            0 |
             0 |             0 |            0 |            0 |
  | mysql        | engine_cost  |         2 |            0 |
             0 |             0 |            0 |            0 |
  | mysql        | proxies_priv |         1 |            0 |
             0 |             0 |            0 |            0 |
  | mysql        | server_cost  |         6 |            0 |
             0 |             0 |            0 |            0 |
  | mysql        | tables_priv  |         2 |            0 |
             0 |             0 |            0 |            0 |
  | mysql        | user         |         7 |            0 |
             0 |             0 |            0 |            0 |
  | test         | sbtest1      |      1686 |          142 |
           184 |           112 |           12 |           18 |
  | test         | sbtest10     |      1806 |          125 |
           150 |           105 |            5 |           15 |
  | test         | sbtest100    |      1623 |          141 |
           182 |           110 |           10 |           21 |
  | test         | sbtest11     |      1254 |          136 |
           172 |           110 |           10 |           16 |
  +--------------+--------------+-----------+--------------
+-----------------------+---------------+--------------
+--------------+

  mysql> select * from INDEX_STATISTICS limit 10;
  +--------------+--------------+------------+-----------+
  | TABLE_SCHEMA | TABLE_NAME   | INDEX_NAME | ROWS_READ |
  +--------------+--------------+------------+-----------+
  | mysql        | db           | PRIMARY    |         2 |
  | mysql        | engine_cost  | PRIMARY    |         2 |
```

```
| mysql         | proxies_priv | PRIMARY    |          1 |
| mysql         | server_cost  | PRIMARY    |          6 |
| mysql         | tables_priv  | PRIMARY    |          2 |
| mysql         | user         | PRIMARY    |          7 |
| test          | sbtest1      | PRIMARY    |       2500 |
| test          | sbtest10     | PRIMARY    |       3007 |
| test          | sbtest100    | PRIMARY    |       2642 |
| test          | sbtest11     | PRIMARY    |       2091 |
+---------------+--------------+------------+------------+
```

参数说明如下。

参数	说明
TABLE_SCHEMA	**数据库名称。**
TABLE_NAME	**表名称。**
ROWS_READ	**读的行数。**
ROWS_CHANGED	**修改的行数。**
ROWS_CHANGED_X_INDEXES	**索引修改的行数。**
ROWS_INSERTED	**插入的行数。**
ROWS_DELETED	**删除的行数。**
ROWS_UPDATED	**更新的行数。**
INDEX_NAME	**索引名称。**

Performance point使用方法

1. **确认Performance point的相关参数。正常的示例如下：**

```
mysql> show variables like "%performance_point%";
+----------------------------------------+-------+
| Variable_name                          | Value |
+----------------------------------------+-------+
| performance_point_dbug_enabled         | OFF   |
| performance_point_enabled              | ON    |
| performance_point_iostat_interval      | 2     |
| performance_point_iostat_volume_size   | 10000 |
| performance_point_lock_rwlock_enabled  | ON    |
+----------------------------------------+-------+
```

说明:

如果参数找不到，请确认您的实例版本是否为MySQL 5.7。

2. 在performance_schema数据库查询events_statements_summary_by_digest_supplement表，查看排名前10的SQL语句。示例如下：

```
mysql> select * from events_statements_summary_by_digest_supplement
limit 10;
  +--------------------+---------------------------------
+-------------------------------------------+-------------+
  | SCHEMA_NAME        | DIGEST                          |
DIGEST_TEXT                                | ELAPSED_TIME | ......
  +--------------------+---------------------------------
+-------------------------------------------+-------------+
  | NULL               | 6b787dd1f9c6f6c5033120760a1a82de | SELECT
 @@`version_comment` LIMIT ?        |          932 |
  | NULL               | 2fb4341654df6995113d998c52e5abc9 | SHOW
SCHEMAS                                    |         2363 |
  | NULL               | 8a93e76a7846384621567fb4daa1bf95 | SHOW
VARIABLES LIKE ?                           |        17933 |
  | NULL               | dd148234ac7a20cb5aee7720fb44b7ea | SELECT
SCHEMA ( )                                 |         1006 |
  | information_schema | 2fb4341654df6995113d998c52e5abc9 | SHOW
SCHEMAS                                    |         2156 |
  | information_schema | 74af182f3a2bd265678d3dadb53e08da | SHOW
TABLES                                     |         3161 |
  | information_schema | d3a66515192fcb100aaef6f8b6e45603 | SELECT
 * FROM `TABLE_STATISTICS` LIMIT ?  |         2081 |
  | information_schema | b3726b7c4c4db4b309de2dbc45ff52af | SELECT
 * FROM `INDEX_STATISTICS` LIMIT ?  |         2384 |
  | information_schema | dd148234ac7a20cb5aee7720fb44b7ea | SELECT
SCHEMA ( )                                 |          129 |
  | test               | 2fb4341654df6995113d998c52e5abc9 | SHOW
SCHEMAS                                    |          342 |
  +--------------------+---------------------------------
+-------------------------------------------+-------------+
```

参数说明如下。

参数	说明
SCHEMA_NAME	数据库名称。
DIGEST	Digest_text进行hash计算得到的64字节的hash字符串。
DIGEST_TEXT	SQL语句的特征。
ELAPSED_TIME	实际运行时间。单位：μs。
CPU_TIME	CPU运行时间。单位：μs。
SERVER_LOCK_TIME	服务器锁定时间。单位：μs。
TRANSACTIO N_LOCK_TIME	存储事务锁定时间。单位：μs。
MUTEX_SPINS	互斥旋转次数。
MUTEX_WAITS	互斥等待次数。

参数	说明
RWLOCK_SPIN_WAITS	读写闩锁的自旋等待数。
RWLOCK_SPIN_ROUNDS	读写闩锁的旋转循环圈数。
RWLOCK_OS_WAITS	读写闩锁的操作系统等待数。
DATA_READS	数据文件读取次数。
DATA_READ_TIME	数据文件读取时间。单位：μs。
DATA_WRITES	数据文件写入次数。
DATA_WRITE_TIME	数据文件写入时间。单位：μs。
REDO_WRITES	日志文件写入次数。
REDO_WRITE_TIME	日志文件写入时间。单位：μs。
LOGICAL_READS	逻辑页读取次数。
PHYSICAL_READS	物理页读取次数。
PHYSICAL_ASYNC_READS	物理异步页读取次数。

3. 在information_schema数据库查询IO_STATISTICS表，查看最近的数据读写情况。示例如下：

```
mysql> select * from IO_STATISTICS limit 10;
  +---------------------+-----------+----------------+ ......
  | TIME                | DATA_READ | DATA_READ_TIME |
  +---------------------+-----------+----------------+
  | 2019-08-08 09:56:53 |        73 |            983 |
  | 2019-08-08 09:56:57 |         0 |              0 |
  | 2019-08-08 09:59:17 |         0 |              0 |
  | 2019-08-08 10:00:55 |      4072 |          40628 |
  | 2019-08-08 10:00:59 |         0 |              0 |
  | 2019-08-08 10:01:09 |       562 |           5800 |
  | 2019-08-08 10:01:11 |       606 |           6910 |
  | 2019-08-08 10:01:13 |       609 |           6875 |
  | 2019-08-08 10:01:15 |       625 |           7077 |
  | 2019-08-08 10:01:17 |       616 |           5800 |
  +---------------------+-----------+----------------+
```

参数说明如下。

参数	说明
TIME	日期。
DATA_READ	数据读取次数。

参数	说明
DATA_READ_TIME	数据读取总时间。单位：μs。
DATA_READ_MAX_TIME	数据读取最长时间。单位：μs。
DATA_READ_BYTES	数据读取总大小。单位：byte。
DATA_WRITE	数据写入次数。
DATA_WRITE_TIME	数据写入总时间。单位：μs。
DATA_WRITE_MAX_TIME	数据写入最长时间。单位：μs。
DATA_WRITE_BYTES	数据写入总大小。单位：byte。

10 Purge Large File Asynchronously

AliSQL支持通过异步删除大文件的方式保证系统稳定性。

背景信息

使用InnoDB引擎时，直接删除大文件会导致POSIX文件系统出现严重的稳定性问题，因此InnoDB会启动一个后台线程来异步清理数据文件。当删除单个表空间时，会将对应的数据文件先重命名为临时文件，然后清除线程将异步、缓慢地清理文件。

> **说明:**
> **AliSQL提供清除文件日志来保证DDL语句的原子性。**

使用方法

1. 查看实例全局变量设置，示例如下：

```
mysql> SHOW GLOBAL VARIABLES LIKE '%data_file_purge%';
  +-----------------------------------------+-------+
  | Variable_name                           | Value |
  +-----------------------------------------+-------+
  | innodb_data_file_purge                  | ON    |
  | innodb_data_file_purge_all_at_shutdown  | OFF   |
  | innodb_data_file_purge_dir              |       |
  | innodb_data_file_purge_immediate        | OFF   |
  | innodb_data_file_purge_interval         | 100   |
  | innodb_data_file_purge_max_size         | 128   |
  | innodb_print_data_file_purge_process    | OFF   |
  +-----------------------------------------+-------+
```

参数说明如下。

参数	说明
innodb_data_file_purge	**是否启用异步清除策略。**
innodb_data_file_purge_all_at_shutdown	**正常关机时全部清理。**
innodb_data_file_purge_dir	**临时文件目录。**
innodb_data_file_purge_immediate	**取消数据文件的链接但不清理。**

参数	说明
innodb_data_file_purge_interval	清理时间间隔。单位：ms。
innodb_data_file_purge_max_size	每次清理单个文件大小的最大值。单位：MB。
innodb_print_data_file_purge_process	是否打印文件清理工作进程。

 **说明:**

建议使用如下命令进行设置：

```
set global INNODB_DATA_FILE_PURGE = on;
set global INNODB_DATA_FILE_PURGE_INTERVAL = 100;
set global INNODB_DATA_FILE_PURGE_MAX_SIZE = 128;
```

2. **使用如下命令查看清理进度：**

```
select * from information_schema.innodb_purge_files;
```

11 Returning

AliSQL提供returning功能，支持DML语句返回Resultset，同时提供了工具包（DBMS_TRANS）便于您快捷使用。

背景信息

MySQL的语句执行结果报文通常分为两类：Resultset和OK/ERR。针对DML语句返回的是OK/ERR报文，其中包括影响记录、扫描记录等属性。但在很多业务场景下，执行INSERT、UPDATE、DELETE这样的DML语句后，都会跟随SELECT查询当前记录内容，以进行接下来的业务处理，为了减少一次客户端和服务器的交互，returning功能支持使用DML语句后返回Resultset。

前提条件

实例版本为RDS MySQL 8.0。

语法

```
DBMS_TRANS.returning(<Field_list>,<Statement>);
```

参数说明如下。

参数	说明
Field_list	期望的返回字段，多个字段以英文逗号（,）进行分隔，支持表中原生的字段或星号（*），不支持进行计算或者聚合等操作。
Statement	执行的DML语句，支持INSERT、UPDATE、DELETE。

注意事项

`dbms_trans.returning()`不是事务性语句，会根据DML语句来继承事务上下文，结束事务需要显式的提交或者回滚。

INSERT Returning

针对INSERT语句，returning返回插入到表中的记录内容。

示例：

```
CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `col1` int(11) NOT NULL DEFAULT '1',
  `col2` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

mysql> call dbms_trans.returning("*", "insert into t(id) values(NULL),
(NULL)");
```

```
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:39:05 |
|  2 |    1 | 2019-09-03 10:39:05 |
+----+------+---------------------+
2 rows in set (0.01 sec)
```

说明:

- **如果没有填入**`Field_list`**，returning将返回OK或ERR报文。**

```
mysql> call dbms_trans.returning("", "insert into t(id) values(NULL
),(NULL)");
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from t;
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:40:55 |
|  2 |    1 | 2019-09-03 10:40:55 |
|  3 |    1 | 2019-09-03 10:41:06 |
|  4 |    1 | 2019-09-03 10:41:06 |
+----+------+---------------------+
4 rows in set (0.00 sec)
```

- **INSERT Returning只支持**`insert values`**形式的语法，类似**`create as`**、**`insert select`**形式的则不支持。**

```
mysql> call dbms_trans.returning("", "insert into t select * from t
");
ERROR 7527 (HY000): Statement didn't support RETURNING clause
```

UPDATE Returning

针对UPDATE语句，returning返回更新后的记录。

示例:

```
mysql> call dbms_trans.returning("id, col1, col2", "update t set col1
 = 2 where id >2");
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  3 |    2 | 2019-09-03 10:41:06 |
|  4 |    2 | 2019-09-03 10:41:06 |
+----+------+---------------------+
2 rows in set (0.01 sec)
```

说明:

UPDATE Returning不支持多表UPDATE语句。

DELETE Returning

针对DELETE语句，returning返回被删除的记录。

示例：

```
mysql> call dbms_trans.returning("id, col1, col2", "delete from t
where id < 3");
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:40:55 |
|  2 |    1 | 2019-09-03 10:40:55 |
+----+------+---------------------+
2 rows in set (0.00 sec)
```

12 Statement Queue

AliSQL设计了针对语句的排队机制，将语句进行分桶排队，尽量把可能具有相同冲突的语句（例如操作相同行）放在一个桶内排队，减少冲突的开销。

背景信息

MySQL的服务层和引擎层在语句并发执行过程中，有很多串行的点容易导致冲突。例如在DML语句中，事务锁冲突比较常见，InnoDB中事务锁的最细粒度是行级锁，如果语句针对相同行进行并发操作，会导致冲突比较严重，系统吞吐量会随着并发的增加而递减。AliSQL提供Statement Queue机制，能够减少冲突开销、有效提高实例性能。

前提条件

实例版本为RDS MySQL 8.0。

效果

在单行进行并发UPDATE的场景下测试，相比较原生的MySQL，AliSQL有接近4倍的提升。

变量

AliSQL提供了两个变量来定义语句队列的桶数量和大小：

- ccl_queue_bucket_count：表示桶的数量。取值范围：1~64；默认值：4。
- ccl_queue_bucket_size：表示一个桶允许的并发数。取值范围：1~4096；默认值：64。

> **说明:**
> 您可以在RDS控制台修改变量值，详情请参见*#unique_45*。

语法

AliSQL支持两种hint语法：

- ccl_queue_value

  根据值进行hash分桶。

  语法：

  ```
  /*+ ccl_queue_value([int | string)] */
  ```

  示例：

  ```
  update /*+ ccl_queue_value(1) */ t set c=c+1 where id = 1;
  ```

```
update /*+ ccl_queue_value('xpchild') */ t set c=c+1 where name = '
xpchild';
```

- **ccl_queue_field**

  根据where条件中的字段值进行hash分桶。

  语法：

  ```
  /*+ ccl_queue_field(string) */
  ```

  示例：

  ```
  update /*+ ccl_queue_field("id") */ t set c=c+1 where id = 1 and
  name = 'xpchild';
  ```

  说明：

  ccl_queue_field语法中，where条件只支持原始字段（没有在字段上使用任何函数、计算等）的二元运算，并且二元运算的右侧值必须是数字或者字符串。

接口

AliSQL提供两个接口便于您查询Statement Queue状态：

- **dbms_ccl.show_ccl_queue()**

  查询当前Statement Queue状态。

  ```
  mysql> call dbms_ccl.show_ccl_queue();
  +------+-------+-------------------+---------+---------+----------+
  | ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
  +------+-------+-------------------+---------+---------+----------+
  |    1 | QUEUE |                64 |       1 |       0 |        0 |
  |    2 | QUEUE |                64 |   40744 |      65 |        6 |
  |    3 | QUEUE |                64 |       0 |       0 |        0 |
  |    4 | QUEUE |                64 |       0 |       0 |        0 |
  +------+-------+-------------------+---------+---------+----------+
  4 rows in set (0.01 sec)
  ```

  参数说明如下。

参数	说明
CONCURRENCY_COUNT	**最大并发数。**
MATCHED	**命中规则的总数。**
RUNNING	**当前并发的数量。**
WAITTING	**当前等待的数量。**

- **dbms_ccl.flush_ccl_queue()**

  清理内存中的数据。

```
mysql> call dbms_ccl.flush_ccl_queue();
                                        Query OK, 0 rows affected (0.00
sec)

mysql> call dbms_ccl.show_ccl_queue();
+------+-------+--------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+--------------------+---------+---------+----------+
|    1 | QUEUE |                 64 |       0 |       0 |        0 |
|    2 | QUEUE |                 64 |       0 |       0 |        0 |
|    3 | QUEUE |                 64 |       0 |       0 |        0 |
|    4 | QUEUE |                 64 |       0 |       0 |        0 |
+------+-------+--------------------+---------+---------+----------+
4 rows in set (0.00 sec)
```

实践

为避免冗长的应用业务代码的修改，Statement Queue可以配合*Statement Outline*进行在线业务修改，方便快捷。下文使用SysBench的update_non_index为例进行演示。

- **测试环境**
  - **测试表结构**

```
CREATE TABLE `sbtest1` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `k` int(10) unsigned NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 MAX_ROWS=
1000000;
```

  - **测试语句**

```
UPDATE sbtest1 SET c='xpchild' WHERE id=0;
```

  - **测试脚本**

```
./sysbench
--mysql-host= {$ip}
--mysql-port= {$port}
--mysql-db=test
--test=./sysbench/share/sysbench/update_non_index.lua
--oltp-tables-count=1
--oltp_table_size=1
--num-threads=128
```

```
--mysql-user=u0
```

- **测试过程**

  1. **在线增加Statement Outline**。

```
mysql> CALL DBMS_OUTLN.add_optimizer_outline('test', '', 1,
                                             ' /*+ ccl_queue_field
("id") */ ',
                          "UPDATE sbtest1 SET c='xpchild' WHERE id=
0");
Query OK, 0 rows affected (0.01 sec)
```

  2. **查看Statement Outline**。

```
mysql> call dbms_outln.show_outline();
+------+--------
+-------------------------------------------------------------------
+-----------+-------+------+---------------------------------
+------+----------+---------------------------------------------+
| ID   | SCHEMA | DIGEST
                | TYPE      | SCOPE | POS  | HINT
           | HIT  | OVERFLOW | DIGEST_TEXT
         |
+------+--------
+-------------------------------------------------------------------
+-----------+-------+------+---------------------------------
+------+----------+---------------------------------------------+
|    1 | test   | 7b945614749e541e0600753367884acff5df7e7e
e2f5fb0af5ea58897910f023 | OPTIMIZER |       |    1 |  /*+
ccl_queue_field("id") */  |    0 |        0 | UPDATE `sbtest1` SET
 `c` = ? WHERE `id` = ? |
+------+--------
+-------------------------------------------------------------------
+-----------+-------+------+---------------------------------
+------+----------+---------------------------------------------+
1 row in set (0.00 sec)
```

  3. **验证Statement Outline生效**。

```
mysql> explain UPDATE sbtest1 SET c='xpchild' WHERE id=0;
+----+-------------+--------+------------+-------+--------------
+---------+---------+-------+------+----------+-------------+
| id | select_type | table   | partitions | type  | possible_keys
 | key     | key_len | ref   | rows | filtered | Extra       |
+----+-------------+--------+------------+-------+--------------
+---------+---------+-------+------+----------+-------------+
|  1 | UPDATE      | sbtest1 | NULL       | range | PRIMARY
 | PRIMARY | 4       | const |    1 |   100.00 | Using where |
+----+-------------+--------+------------+-------+--------------
+---------+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> show warnings;
+-------+------
+-------------------------------------------------------------------------------
+
| Level | Code | Message

          |
```

```
+-------+------
+-------------------------------------------------------------------------
+
| Note  | 1003 | update /*+ CCL_QUEUE_FIELD('id') */ `test`.`
sbtest1` set `test`.`sbtest1`.`c` = 'xpchild' where (`test`.`
sbtest1`.`id` = 0) |
+-------+------
+-------------------------------------------------------------------------
+
1 row in set (0.00 sec)
```

4. 查询Statement Queue状态。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------
+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING
 |
+------+-------+------------------+---------+---------+----------
+
|    1 | QUEUE |                64 |       0 |       0 |        0
 |
|    2 | QUEUE |                64 |       0 |       0 |        0
 |
|    3 | QUEUE |                64 |       0 |       0 |        0
 |
|    4 | QUEUE |                64 |       0 |       0 |        0
 |
+------+-------+------------------+---------+---------+----------
+
4 rows in set (0.00 sec)
```

5. 开启测试。

```
sysbench
--mysql-host= {$ip}
--mysql-port= {$port}
--mysql-db=test
--test=./sysbench/share/sysbench/update_non_index.lua
--oltp-tables-count=1
--oltp_table_size=1
--num-threads=128
--mysql-user=u0
```

6. 验证测试效果。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------
+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING
 |
+------+-------+------------------+---------+---------+----------
+
|    1 | QUEUE |                64 |   10996 |      63 |        4
 |
|    2 | QUEUE |                64 |       0 |       0 |        0
 |
|    3 | QUEUE |                64 |       0 |       0 |        0
 |
|    4 | QUEUE |                64 |       0 |       0 |        0
 |
```

```
+------+-------+------------------+---------+---------+----------
+
4 rows in set (0.03 sec)

mysql> call dbms_outln.show_outline();
+------+--------+-----------+-----------+-------+------
+---------------------------------+--------+----------
+---------------------------------------------+
| ID   | SCHEMA | DIGEST    | TYPE      | SCOPE | POS  | HINT
                       | HIT    | OVERFLOW | DIGEST_TEXT
                       |
+------+--------+-----------+-----------+-------+------
+---------------------------------+--------+----------
+---------------------------------------------+
|    1 | test   | xxxxxxxxx | OPTIMIZER |       |    1 |  /*+
ccl_queue_field("id") */  | 115795 |        0 | UPDATE `sbtest1`
SET `c` = ? WHERE `id` = ? |
+------+--------+-----------+-----------+-------+------
+---------------------------------+--------+----------
+---------------------------------------------+
1 row in set (0.00 sec)
```

说明:

查询结果显示Statement Outline命中了115,795次规则，Statement Queue状态显示命中了10,996次排队，当前运行并发63个，排队等待4个。

13 Inventory Hint

AliSQL提供Inventory Hint，帮助您快速提交/回滚事务，配合Returning和Statement Queue，能有效提高业务吞吐能力。

背景信息

在秒杀等业务场景中，减少库存是一个常见的需要高并发，同时也需要串行化的任务模型，AliSQL使用排队和事务性hint来控制并发和快速提交/回滚事务，提高业务吞吐能力。

前提条件

实例版本为RDS MySQL 8.0。

语法

新增了三个hint， 支持SELECT、UPDATE、INSERT、DELETE 语句。

- **COMMIT_ON_SUCCESS/ROLLBACK_ON_FAIL**

  两个事务hint为COMMIT_ON_SUCCESS和ROLLBACK_ON_FAIL：

  - **COMMIT_ON_SUCCESS：当前语句执行成功就提交事务上下文。**
  - **ROLLBACK_ON_FAIL：当前语句执行失败就回滚事务上下文。**

  语法：

  ```
  /*+ COMMIT_ON_SUCCESS */
  /*+ ROLLBACK_ON_FAIL */
  ```

  示例：

  ```
  UPDATE /*+ COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL */ T
  SET c = c - 1
  WHERE id = 1;
  ```

- **TARGET_AFFECT_ROW(NUMBER)**

  条件hint为TARGET_AFFECT_ROW(NUMBER)：如果当前语句影响行数是指定的就成功，否则语句失败。

  语法：

  ```
  /*+ TARGET_AFFECT_ROW(NUMBER) */
  ```

  示例：

  ```
  UPDATE /*+ TARGET_AFFECT_ROW(1) */ T
  SET c = c - 1
  ```

```
WHERE id = 1;
```

注意事项

- **事务hint不能运行在autocommit模式下， 例如：**

```
mysql> UPDATE /*+ commit_on_success rollback_on_fail target_aff
ect_row(1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
ERROR 7531 (HY000): Inventory transactinal hints didn't allowed in
autocommit mode
```

- **事务hint不能运行在sub statement下，例如：**

```
mysql> CREATE TRIGGER tri_1
    -> BEFORE INSERT ON t
    -> FOR EACH ROW
    -> BEGIN
    -> INSERT /*+ commit_on_success */ INTO t1 VALUES (1);
    -> end//

mysql> INSERT INTO t VALUES (2, 1);
ERROR HY000: Inventory transactional hints didn't alllowed in stored
 procedure
```

- **条件hint不能运行在SELECT/EXPLAIN statement下， 例如：**

```
mysql> EXPLAIN UPDATE /*+ commit_on_success rollback_on_fail
target_affect_row(1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
ERROR 7532 (HY000): Inventory conditional hints didn't match with
result
```

说明:

您可以指定target_affect_row为一个无效的number进行测试，系统会有告警。

```
mysql> EXPLAIN UPDATE /*+ commit_on_success rollback_on_fail
target_affect_row(-1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
+----+-------------+-------+------------+-------+---------------
+---------+---------+-------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys |
key     | key_len | ref   | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------
+---------+---------+-------+------+----------+-------------+
|  1 | UPDATE      | t     | NULL       | range | PRIMARY       |
PRIMARY | 4       | const |    1 |   100.00 | Using where |
+----+-------------+-------+------------+-------+---------------
+---------+---------+-------+------+----------+-------------+
1 row in set, 2 warnings (0.00 sec)

mysql> show warnings;
+---------+------
+-------------------------------------------------------------------------------
+
```

```
| Level   | Code | Message

                     |
+---------+------
+-------------------------------------------------------------------------------
+
| Warning | 1064 | Optimizer hint syntax error near '-1) */ t set
col1=col1+1 where id =1' at line 1
                      |
| Note    | 1003 | update /*+ COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL */
 `test`.`t` set `test`.`t`.`col1` = (`test`.`t`.`col1` + 1) where
 (`test`.`t`.`id` = 1) |
+---------+------
+-------------------------------------------------------------------------------
+
2 rows in set (0.00 sec)
```

配合Returning使用

Inventory Hint可以配合*Returning*使用，实时返回结果集， 例如：

```
mysql> CALL dbms_trans.returning("*", "update /*+ commit_on_success
rollback_on_fail target_affect_row(1) */ t
                                       set col1=col1+1 where id=1");
+----+------+
| id | col1 |
+----+------+
|  1 |   13 |
+----+------+
1 row in set (0.00 sec)

mysql> CALL dbms_trans.returning("*", "insert /*+ commit_on_success
rollback_on_fail target_affect_row(1) */ into
                                       t values(10,10)");
+----+------+
| id | col1 |
+----+------+
| 10 |   10 |
+----+------+
1 row in set (0.01 sec)
```

配合Statement queue使用

Inventory Hint可以配合*Statement Queue*进行排队，例如：

```
mysql> UPDATE /*+ ccl_queue_field(id) commit_on_success rollback_o
n_fail target_affect_row(1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;

Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE /*+ ccl_queue_value(1) commit_on_success rollback_o
n_fail target_affect_row(1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

14 Performance Agent

Performance Agent是AliSQL提供的一种更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。

背景信息

Performance Agent在information_schema系统库下新增了一张内存表PERF_STATISTICS，用于统计最近一段时间的性能数据，您可以直接查询该表获取相关指标的性能数据。

前提条件

- 实例版本如下：
  - MySQL 8.0
  - MySQL 5.7
- 内核小版本为20200229或以上。

> **说明:**
> 升级内核小版本请参见*#unique_22*。

参数说明

与Performance Agent功能相关的参数说明如下。您可以根据业务情况修改参数。

参数	说明
performance_agent_enabled	是否开启Performance Agent功能。取值：ON | OFF。默认值为ON。
performance_agent_interval	性能数据采集间隔。单位为秒，默认值为1。
performance_agent_perfstat_volume_size	PERF_STATISTICS表的最大数据条数。默认值为3600。即当采样间隔为1秒时，保存最近1小时的性能数据。

表结构说明

PERF_STATISTICS内存表的结构如下：

```
CREATE TEMPORARY TABLE `PERF_STATISTICS` (
  `TIME` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `PROCS_MEM_USAGE` double NOT NULL DEFAULT '0',
  `PROCS_CPU_RATIO` double NOT NULL DEFAULT '0',
  `PROCS_IOPS` double NOT NULL DEFAULT '0',
  `PROCS_IO_READ_BYTES` bigint(21) NOT NULL DEFAULT '0',
  `PROCS_IO_WRITE_BYTES` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_CONN_ABORT` int(11) NOT NULL DEFAULT '0',
```

```
  `MYSQL_CONN_CREATED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_USER_CONN_COUNT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_CONN_RUNNING` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LOCK_IMMEDIATE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LOCK_WAITED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_INSERT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_UPDATE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_DELETE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_SELECT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_COMMIT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_ROLLBACK` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_PREPARE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LONG_QUERY` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TCACHE_GET` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_TCACHE_MISS` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_TMPFILE_CREATED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TMP_TABLES` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TMP_DISKTABLES` int(11) NOT NULL DEFAULT '0',
  `MYSQL_SORT_MERGE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_SORT_ROWS` int(11) NOT NULL DEFAULT '0',
  `MYSQL_BYTES_RECEIVED` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_BYTES_SENT` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_BINLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `MYSQL_IOLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `MYSQL_RELAYLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `EXTRA` json NOT NULL DEFAULT 'null'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

列名	说明
TIME	时间，格式为yyyy-MM-dd HH:mm:ss。
PROCS_MEM_USAGE	物理内存使用量，单位为Byte。
PROCS_CPU_RATIO	CPU使用率。
PROCS_IOPS	系统IO调用次数。
PROCS_IO_READ_BYTES	IO读取数据量，单位为Byte。
PROCS_IO_WRITE_BYTES	IO写入数据量，单位为Byte。
MYSQL_CONN_ABORT	断开连接数。
MYSQL_CONN_CREATED	新建连接数。
MYSQL_USER_CONN_COUNT	当前总的用户连接数。
MYSQL_CONN_RUNNING	当前活跃连接数。
MYSQL_LOCK_IMMEDIATE	当前锁占用数。
MYSQL_LOCK_WAITED	当前锁等待数。
MYSQL_COM_INSERT	插入语句数。
MYSQL_COM_UPDATE	更新语句数。
MYSQL_COM_DELETE	删除语句数。

列名	说明
MYSQL_COM_SELECT	查询语句数。
MYSQL_COM_COMMIT	事务提交数（显式提交）。
MYSQL_COM_ROLLBACK	事务回滚数。
MYSQL_COM_PREPARE	预处理语句数。
MYSQL_LONG_QUERY	慢查询数。
MYSQL_TCACHE_GET	缓存表命中数。
MYSQL_TCACHE_MISS	缓存表未命中数。
MYSQL_TMPFILE_CREATED	临时文件创建数。
MYSQL_TMP_TABLES	临时表创建数。
MYSQL_TMP_DISKTABLES	临时磁盘表创建数。
MYSQL_SORT_MERGE	合并排序次数。
MYSQL_SORT_ROWS	排序行数。
MYSQL_BYTES_RECEIVED	接收数据量，单位为Byte。
MYSQL_BYTES_SENT	发送数据量，单位为Byte。
MYSQL_BINLOG_OFFSET	产生的Binlog文件大小，单位为Byte。
MYSQL_IOLOG_OFFSET	主库发送的Binlog文件大小，单位为Byte。
MYSQL_RELAYLOG_OFFSET	从库应用的Binlog文件大小，单位为Byte。
EXTRA	InnoDB统计信息。EXTRA包含多个字段，为JSON格式。详细字段介绍请参见下方表 14-1: EXTRA字段说明。 说明: InnoDB统计信息的指标项与SHOW STATUS命令显示的值相同。

表 14-1: EXTRA字段说明

字段	说明
INNODB_TRX_CNT	事务数。
INNODB_DATA_READ	读取数据量，单位为Byte。
INNODB_IBUF_SIZE	合并记录页数。
INNODB_LOG_WAITS	Log写入等待次数。

字段	说明
INNODB_MAX_PURGE	清除事务数。
INNODB_N_WAITING	锁等待数。
INNODB_ROWS_READ	读取数据行数。
INNODB_LOG_WRITES	日志写次数。
INNODB_IBUF_MERGES	合并次数。
INNODB_DATA_WRITTEN	写入数据量，单位为Byte。
INNODB_DBLWR_WRITES	双写操作写入次数。
INNODB_IBUF_SEGSIZE	当前插入缓冲大小。
INNODB_ROWS_DELETED	删除数据行数。
INNODB_ROWS_UPDATED	更新数据行数。
INNODB_COMMIT_TRXCNT	提交事务数。
INNODB_IBUF_FREELIST	空闲列表长度。
INNODB_MYSQL_TRX_CNT	MySQL事务数。
INNODB_ROWS_INSERTED	插入数据行数。
INNODB_ACTIVE_TRX_CNT	活跃事务数。
INNODB_OS_LOG_WRITTEN	日志文件写入量，单位为Byte。
INNODB_ACTIVE_VIEW_CNT	活跃视图数。
INNODB_RSEG_HISTORY_LEN	TRX_RSEG_HISTORY表长度。
INNODB_AVG_COMMIT_TRXTIME	平均事务提交时间。
INNODB_MAX_COMMIT_TRXTIME	最长事务提交时间。
INNODB_DBLWR_PAGES_WRITTEN	双写操作完成写入次数。

使用方法

- **直接查询系统表，获取性能数据。 您可以参见如下示例。**
  - **查询最近30秒的CPU和内存使用情况，示例如下：**

```
MySQL> select TIME, PROCS_MEM_USAGE, PROCS_CPU_RATIO from
information_schema.PERF_STATISTICS order by time DESC limit 30;
+---------------------+-----------------+-----------------+
| TIME                | PROCS_MEM_USAGE | PROCS_CPU_RATIO |
+---------------------+-----------------+-----------------+
| 2020-02-27 11:15:36 |       857812992 |           18.55 |
| 2020-02-27 11:15:35 |       857808896 |           18.54 |
| 2020-02-27 11:15:34 |       857268224 |           19.64 |
| 2020-02-27 11:15:33 |       857268224 |           21.06 |
| 2020-02-27 11:15:32 |       857264128 |           20.39 |
| 2020-02-27 11:15:31 |       857272320 |           20.32 |
| 2020-02-27 11:15:30 |       857272320 |           21.35 |
| 2020-02-27 11:15:29 |       857272320 |            28.8 |
| 2020-02-27 11:15:28 |       857268224 |           29.08 |
| 2020-02-27 11:15:27 |       857268224 |           26.92 |
| 2020-02-27 11:15:26 |       857268224 |           23.84 |
| 2020-02-27 11:15:25 |       857264128 |           13.76 |
| 2020-02-27 11:15:24 |       857264128 |           15.12 |
| 2020-02-27 11:15:23 |       857264128 |           14.76 |
| 2020-02-27 11:15:22 |       857264128 |           15.38 |
| 2020-02-27 11:15:21 |       857260032 |           13.23 |
| 2020-02-27 11:15:20 |       857260032 |           12.75 |
| 2020-02-27 11:15:19 |       857260032 |           12.17 |
| 2020-02-27 11:15:18 |       857255936 |           13.22 |
| 2020-02-27 11:15:17 |       857255936 |           20.51 |
| 2020-02-27 11:15:16 |       857255936 |           28.74 |
| 2020-02-27 11:15:15 |       857251840 |           29.85 |
| 2020-02-27 11:15:14 |       857251840 |           29.31 |
| 2020-02-27 11:15:13 |       856981504 |           28.85 |
| 2020-02-27 11:15:12 |       856981504 |           29.19 |
| 2020-02-27 11:15:11 |       856977408 |           29.12 |
| 2020-02-27 11:15:10 |       856977408 |           29.32 |
| 2020-02-27 11:15:09 |       856977408 |            29.2 |
| 2020-02-27 11:15:08 |       856973312 |           29.36 |
| 2020-02-27 11:15:07 |       856973312 |           28.79 |
+---------------------+-----------------+-----------------+
30 rows in set (0.08 sec)
```

  - **查询最近30秒的InnoDB读写的数据行数，示例如下：**

```
MySQL> select TIME, EXTRA->'$.INNODB_ROWS_READ', EXTRA->'$.
INNODB_ROWS_INSERTED' from information_schema.PERF_STATISTICS
order by time DESC limit 30;
+---------------------+----------------------------
+---------------------------------+
| TIME                | EXTRA->'$.INNODB_ROWS_READ' | EXTRA->'$.
INNODB_ROWS_INSERTED' |
+---------------------+----------------------------
+---------------------------------+
| 2020-02-27 11:22:17 | 39209                       | 0
                      |
| 2020-02-27 11:22:16 | 36098                       | 0
                      |
| 2020-02-27 11:22:15 | 38035                       | 0
                      |
```

```
| 2020-02-27 11:22:14 | 37384                          | 0
                      |
| 2020-02-27 11:22:13 | 38336                          | 0
                      |
| 2020-02-27 11:22:12 | 33946                          | 0
                      |
| 2020-02-27 11:22:11 | 36301                          | 0
                      |
| 2020-02-27 11:22:10 | 36835                          | 0
                      |
| 2020-02-27 11:22:09 | 36900                          | 0
                      |
| 2020-02-27 11:22:08 | 36402                          | 0
                      |
| 2020-02-27 11:22:07 | 39672                          | 0
                      |
| 2020-02-27 11:22:06 | 39316                          | 0
                      |
| 2020-02-27 11:22:05 | 37830                          | 0
                      |
| 2020-02-27 11:22:04 | 36396                          | 0
                      |
| 2020-02-27 11:22:03 | 34820                          | 0
                      |
| 2020-02-27 11:22:02 | 37350                          | 0
                      |
| 2020-02-27 11:22:01 | 39463                          | 0
                      |
| 2020-02-27 11:22:00 | 38419                          | 0
                      |
| 2020-02-27 11:21:59 | 37673                          | 0
                      |
| 2020-02-27 11:21:58 | 35117                          | 0
                      |
| 2020-02-27 11:21:57 | 36140                          | 0
                      |
| 2020-02-27 11:21:56 | 37592                          | 0
                      |
| 2020-02-27 11:21:55 | 39765                          | 0
                      |
| 2020-02-27 11:21:54 | 35553                          | 0
                      |
| 2020-02-27 11:21:53 | 35882                          | 0
                      |
| 2020-02-27 11:21:52 | 37061                          | 0
                      |
| 2020-02-27 11:21:51 | 40699                          | 0
                      |
| 2020-02-27 11:21:50 | 39608                          | 0
                      |
| 2020-02-27 11:21:49 | 39317                          | 0
                      |
| 2020-02-27 11:21:48 | 37413                          | 0
                      |
+---------------------+--------------------------------
+---------------------------------+
```

```
30 rows in set (0.08 sec)
```

- **对接性能监控平台，实现实时监控。例如使用*Grafana*。**