云数据库 GaussDB(for MySQL)

内核介绍

文档版本 01

发布日期 2024-08-09

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 GaussDB(for MySQL)内核版本发布记录

本章节介绍云数据库 GaussDB(for MySQL)的内核版本更新说明。

2.0.54.240600

表 1-1 2.0.54.240600 内核版本说明

日期	特性描述
2024-07-19	● 新增功能和性能优化： – 热点行更新优化：热点行出现的场景包括秒杀抢购、演唱会门票预订、热门路线火车票预定等等，本版本支持热点行更新优化，您可以通过手动指定或者自动识别的方式开启热点行更新，该功能开启后可以大幅度提升热点行的更新性能。 – 非阻塞DDL：用户在执行DDL操作的时候，如果目标表存在未提交的长事务或大查询，DDL将持续等待获取MDL-X锁，将导致业务连接的堆积和阻塞。本版本支持非阻塞DDL功能，可以保证即使在无法获得MDL-X锁的情况下，依然允许新事务进入目标表，从而保证整个业务系统的稳定。 – 多租户管理：提供多租户管理功能，让数据库能够为其多个租户服务，提高数据库资源利用率。 – 只读节点支持Binlog拉取：支持只读节点拉取Binlog，您可以以GaussDB(for MySQL)只读节点为数据源，建立Binlog复制链路，实时同步Binlog内容，以便减轻GaussDB(for MySQL)主节点的负载。 – 字段压缩（列压缩）：为了减少数据页面存储空间占用，节省成本，GaussDB(for MySQL)推出细粒度的字段压缩，提供ZLIB和ZSTD两种压缩算法，用户可以综合考虑压缩比和压缩解压性能影响，选择合适的压缩算法，对不频繁访问的大字段进行压缩。 – 支持INTERVAL RANGE分区表：现有的RANGE分区表插入数据时，如果插入的数据超出当前已存在分区的范围，将无法插入并且会返回错误。本版本支持INTERVAL RANGE分区表后，当新插入的数据超过现有分区的范围时，允许数据库根据INTERVAL子句提前指定的规则来添加新分区。 – 支持LIST DEFAULT HASH分区表特性：LIST DEFAULT HASH是在同一级别支持两种分区类型：LIST和HASH。前面是普通的LIST分区，不符合LIST分区规则的数据会放在DEFAULT分区里，DEFAULT分区如果有多个分区则根据HASH规则计算。LIST DEFAULT HASH分区类型常用在LIST VALUES分布不均匀以及无法全部枚举的场景。 ● 问题修复： – 优化表级恢复性能。 – 优化大规格实例高并发场景备机的执行性能。

2.0.51.240300

表 1-2 2.0.51.240300 内核版本说明

日期	特性描述
2024-03-30	● 新增功能和性能优化： – 支持高性能全局一致性，在较低的性能损耗下，提供集群维度的强一致性读能力。 – 新增show binary logs no block语法，优化在show binary logs过程中对事务提交的阻塞情况。 – 提供undo truncate能力，优化大量写入场景导致undo空间膨胀的问题。 – 提高全量恢复的并行度，优化备份恢复效率。 ● 问题修复： – 修复一批window function查询结果不准确或异常错误的问题。 – 修复在打开plan cache后反复执行一类prepare statement，数据库节点崩溃的问题。 – 修复在先后执行的存储过程中，由于字符集不一致导致的报错问题。 – 修复一类开启PQ后进行磁盘hash join，查询结果不符合预期的问题。 – 修复一类查询含义group by临时表字段时，报错主键重复的问题。

2.0.48.231200

表 1-3 2.0.48.231200 内核版本说明

日期	特性描述

2024-01-30	● 新增功能和性能优化： – 组合分区能力增强：在社区MySQL的RANGE-HASH、LIST-HASH两类组合分区能力基础上，增加了RANGE-RANGE、RANGE-LIST、LIST-RANGE、LIST-LIST、HASH-HASH、HASH-KEY、HASH-RANGE、HASH-LIST、KEY-HASH、KEY-KEY、KEY-RANGE、KEY-LIST的组合分区能力。 – 向前兼容MySQL 5.7 GROUP BY场景隐式/显式排序。 – 向前兼容MySQL 5.7 max_length_for_sort_data判据，优化特定场景文件排序性能。 – 优化因执行计划选错导致访问information_schema下视图较慢的问题。 – PQ支持EXIST子查询。 – 优化库表或实例按时间点恢复性能。 ● 问题修复： – OPENSSL版本升级。 – 修复time_zone参数默认值SYSTEM会导致部分场景SQL并行执行效率降低的问题。 – 修复一类条件部分下推到物化derived table时，SQL查询结果不准确的问题。 – 修复部分场景磁盘hash join开启PQ后性能劣化的问题。 – 修复控制台赋予用户数据库权限后，通过非控制台的方式删除此数据库，权限页面未更新的问题。

2.0.45.230900

表 1-4 2.0.45.230900 内核版本说明

日期	特性描述

2023-11-24	● 新增功能和性能优化： – 优化datatime/timestamp/time字段行为向前兼容。 – 优化PQ支持并行磁盘hash join场景。 – 启用并行INSERT/REPLACE SELECT的功能优化查询速度。 – 增加连接建立/断开日志打印，提高定位连接相关问题效率。 – 优化慢日志中增加对慢SQL问题定位有用的信息，提升定位慢SQL定位效率。 – 支持动态开启Binlog。 – 优化NDP bloom过滤器。 – 支持使用CAST(... AS INT) 语法。 – 优化Nested Loop Join + Distinct 性能。 – 优化快速识别慢IO对应的slice id。 – 增加sal_init日志，后续出现存储接口超时，时延可定位性增强。 ● 问题修复： – 修复全量SQL中缺少trx_id和cpu_time字段的问题。 – 修复prepare语句中where比较时，字段是int类型、参数是字符串导致转换有误的问题。 – 修复备机上DDL与查询的并发访问时，极小概率导致crash的问题。 – 修复Binlog数量短期暴涨未及时清理的问题。 – 修复多表JOIN SQL语句打开PQ开关后，可能出现执行结果不一致的问题。 – 修复Backwad Index Scan与ICP无法兼容导致查询性能不及预期的问题。 – 修复weight_string函数不支持level子句的问题。 – 修复特殊场景下，相同的SQL语句选用不同的索引得出结果不一致的问题。 – 修复部分场景下，同时开启NDP和PQ特性recycle lsn长时间不推进的问题。

2.0.42.230600

表 1-5 2.0.42.230600 内核版本说明

日期	特性描述

2023-08-31	● 新增功能和性能优化： – 优化全量与增量备份放到备库进行，减少主机内存/CPU占用。 – 优化UNDO损坏场景的快速定位：启动undo损坏时，明确打印出undo损坏和对应表名称。 – 优化备机查询性能劣后于主库问题。 – 优化in-list转临时表。 – NDP特性规模商用。 – 用Statement Outline方法稳定执行计划。 – PQ特性支持Round函数。 ● 问题修复： – 修复快速排序和优先级队列排序算法不稳定导致ORDER BY LIMIT与ORDER LIMIT结果集有重合的问题。 – 修复PQ语句极小概率情况返回错误结果的问题。 – 修复部分场景PREPARE语句执行报错的问题。 – 修复部分场景UNION查询上的PQ断言错误的问题。 – 修复实例主节点INSERT大数据量的时候只读升主，升主成功后用全文索引查询的结果不准确的问题。 – 修复备机使用general_log和slow_log表打印warning日志的问题。 – 修复部分场景设置锁等待时间参数innodb_lock_wait_timeout后，实际超时等待时间不一致的问题。 – 修复只读升主过程中，小概率出现Failed to find page in slice manager导致升主失败的问题。 – 修复salsql日志pwal扫描进度percentage值大于100%的问题。 – 修复执行sqlsmith工具, 查询语句在explain阶段偶现mysqld coredump。 – 修复SELECT DISTINCT + CAST函数转换datetime类型为float类型时，结果不正确的问题。

2.0.39.230300

表 1-6 2.0.39.230300 版本说明

日期	特性描述
2023-05-11	● 新特性及优化： – 支持小规格实例。 – 备机DDL失效方案优化。 – SALSQL使用空间容量计算优化。 – 支持对单个SQL语句使用资源进行限制。 – 支持admin port和local socket使用per thread。 – pwalScanner内存优化。 – 支持修改default_collation_for_utf8mb4参数。 – 支持大事务检测能力。 – 支持Kill idle transactions。 – 优化增量恢复速度。 – 新增数据库描述和账号描述。 – 支持buffer pool resize加速。 ● 问题修复： – 修复Ptrc可能会导致Nestedloop join的结果不一致问题。 – 修复使用windows函数进行排序的子查询可能会导致crash问题。 – 修复使用rewrites view时，如果评估可能会把left joins转化为inner joins问题。 – 修复指定过滤条件的decimal类型的数据不返回结果问题。 – 修复内存非对齐问题。 – 修复全量日志中记录scan_row不准确问题。

2.0.28.18

表 1-7 2.0.28.18 版本说明

日期	特性描述
2023-05-17	修复对含有大的JSON列的排序报超过排序内存问题。

2.0.28.17

表 1-8 2.0.28.17 版本说明

日期	特性描述
2023-04-02	修复prepare statement中字符集混合使用问题。

2.0.28.16

表 1-9 2.0.28.16 版本说明

日期	特性描述
2023-03-14	● 新特性： 优化主备时延。 ● 修复问题： – 修复prepare statement中使用json相关函数处理错误问题。 – 修复指定过滤条件查询结果不返回的问题; – 修复WINDOWS函数生成磁盘临时表后，出现空指针异常问题。 – 修复windows functions空指针使用导致的crash问题。 – 修复prepared statements执行失败的问题。

2.0.28.15

表 1-10 2.0.28.15 版本说明

日期	特性描述
2023-01-11	● 新特性 – 支持SQL限流。 – 读流控优化。 – 主备执行计划一致优化。 – slice异步预创建。 ● 问题修复 – 修复系统变量INNODB_VALIDATE_TABLESPACE_PATHS关闭情况下undo space truncate的时候出现的crash问题。 – 修复查询information_schema.innodb_trx较慢问题。 – 修复查询结果不一致的问题：left joins没有转化为inner joins。 – 修复优化子查询的过程中导致的crash问题。 – 修复并发instantDDL和DML场景下未按实际获取instant字段值的问题。 – 修复当load有FTS索引的两个INNODB表时可能导致OOM的问题。 – 修复更新百万级别的表的数据字典可能导致OOM的问题。

2.0.28.12

表 1-11 2.0.28.12 版本说明

日期	特性描述
2022-12-07	修复更新有虚拟列的表之后，使用skip scan触发了扫描错误的问题。

2.0.28.10

表 1-12 2.0.28.10 版本说明

日期	特性描述
2022-11-16	修复主备倒换过程中，连接备机超时触发数据库崩溃问题。

2.0.28.9

表 1-13 2.0.28.9 版本说明

日期	特性描述
2022-09-23	● 修复在Condition_pushdown::replace_columns_in_cond中使用不正确的条件判断的问题。 ● 修复递归调用存储函数之后导致数据库崩溃的问题。 ● 修改多表删除和full-text搜索的时候导致数据库崩溃的问题； ● 修复运行多个窗口函数的SQL查询语句之后导致数据库崩溃的问题； ● 修复具有全局级别权限的用户，执行SHOW CREATE DATABASE失败的问题。

2.0.28.7

表 1-14 2.0.28.7 版本说明

日期	特性描述
2022-08-25	修复存储过程中的ptrc crash问题。

2.0.28.4

表 1-15 2.0.28.4 版本说明

日期	特性描述
2022-07-22	● 修复空用户导致数据库崩溃的问题。 ● 修复更新用于聚合的临时表时，BLOB指针指向的数据过期的问题。

2.0.28.1

表 1-16 2.0.28.1 版本说明

日期	特性描述
2022-05-16	● 新特性 – GaussDB(for MySQL)增加orphaned definer check控制开关。 – GaussDB(for MySQL)支持Proxy IP透传。 – GaussDB(for MySQL) Proxy提供会话一致性功能。 ● 问题修复 – 修复主机DDL未提交导致的备机dd（data dictionary）未更新问题。 – 修复故障切换的主机的auto increment回退的问题。 – 修复备机性能异常问题。

2.0.31.220700

表 1-17 2.0.31.220700 版本说明

日期	特性描述
2022-08-12	● 新特性及性能优化 – 支持SQL限流。 – 新增FasterDDL并行数限制。 – 支持Faster DDL的所有ROW格式。 – 扩展全量SQL字段。 – 优化流量控制。 – 支持ALTER TABLE快速超时。 – 支持Query plan cache。 – 备机统计信息优化。 ● 问题修复 – 修复主机rename partition-table之后备机crash的问题。 – 修改sql tracer的默认buffer size。 – 修复备机truncate lsn落后很多情况下备机拉起失败的问题。 – 修复含有多个相同范围的SQL查询导致的执行计划错误的问题。 – 修复空账户导致的crash的问题。 – 修复drop database可能导致的crash的问题。

2 常见内核功能

2.1 并行查询（PQ）

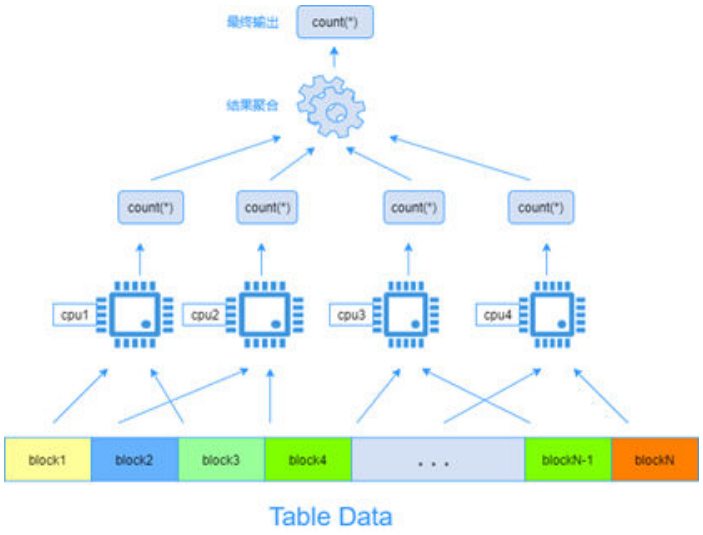
2.1.1 并行查询简介

什么是并行查询

云数据库 GaussDB(for MySQL)支持了并行执行的查询方式，用以降低分析型查询场景的处理时间，满足企业级应用对查询低时延的要求。并行查询的基本实现原理是将查询任务进行切分并分发到多个CPU核上进行计算，充分利用CPU的多核计算资源来缩短查询时间。并行查询的性能提升倍数理论上与CPU的核数正相关，也就是说并行度越高能够使用的CPU核数就越多，性能提升的倍数也就越高。

下图是使用CPU多核资源并行计算一个表的count(*)过程的基本原理：表数据进行切块后分发给多个核进行并行计算，每个核计算部分数据得到一个中间count(*)结果，并在最后阶段将所有中间结果进行聚合得到最终结果。具体如下：

图 2-1 并行查询原理图



应用场景

并行查询适用于大部分SELECT语句，例如大表查询、多表连接查询、计算量较大的查询。对于非常短的查询，效果不太显著。

- 轻分析类业务

  报表查询通常SQL复杂而且比较耗费时间，通过并行查询可以加速单次查询效率。

- 系统资源相对空闲

  并行查询会使用更多的系统资源，只有当系统的CPU较多、IO负载不高、内存够大的时候，才可以充分使用并行查询来提高资源利用率和查询效率。

- 数据频繁查询

  针对数据密集型查询，通过并行查询，可以提高查询处理执行效率，减少网络流量和计算节点的压力。

2.1.2 注意事项

- 并行查询特性当前处于公测阶段，建议在测试环境使用。
- 云数据库GaussDB(for MySQL)的引擎版本需要为MySQL 8.0.22及以上。
- 只读节点和主节点均支持并行查询，由于并行查询对计算资源（CPU、内存等）比较消耗，考虑到实例稳定性，云数据库GaussDB(for MySQL)的内核版本为2.0.42.230600及以上时，并行查询默认在主节点不生效，如需使用可联系客服人员开启。
- **并行查询支持的场景：**
  - 支持全表扫描、索引扫描、索引范围扫描、索引逆序扫描、索引点查询、索引下推等。
  - 支持单表查询、多表JOIN、视图VIEW、子查询，部分CTE查询等。
  - 支持多种JOIN算法，包括：BNL JOIN、BKA JOIN、HASH JOIN、NESTED LOOP JOIN、SEMI JOIN、ANTI JOIN、OUTER JOIN等。
  - 支持多种子查询，包括：条件子查询、SCALAR子查询、部分关联子查询、非关联子查询、DERIVED TABLE等。
  - 支持多种数据类型，包括：整型数据、字符型数据、浮点型数据、时间型数据等。
  - 支持算术表达式计算（+、-、*、%、/、|、&），条件表达式运算（<、<=、>、>=、<>、BETWEEN/AND、IN等），逻辑运算（OR、AND、NOT等），一般函数（字符函数、整型函数、时间函数等），聚合函数（COUNT/SUM/AVG/MIN/MAX）等等。

    说明

    COUNT聚合函数需关闭“innodb_parallel_select_count”才能并行执行。
  - 支持非分区表查询、分区表单分区查询。
  - 支持排序ORDER BY、分组GROUP BY/DISTINCT、分页LIMIT/OFFSET、过滤WHERE/HAVING、列投影等。
  - 支持UNION/UNION ALL查询。
  - 支持EXPLAIN查看并行执行计划多种方式，包括传统EXPLAIN、EXPLAIN FORMAT=TREE、EXPLAIN FORMAT=JSON等。
- **并行查询不支持的场景：**

- 非查询语句
- 窗口函数
- 触发器
- PREPARED STATEMENTS
- 空间索引
- 查询表为系统表/临时表/非INNODB表
- 使用全文索引
- 存储过程
- 不能转换成SEMIJOIN的子查询
- 不满足ONLY_FULL_GROUP_BY
- 使用索引归并INDEX MERGE
- HASH JOIN溢出到磁盘
- 加锁查询，如SERIALIZABLE隔离级别，FOR UPDATE/SHARE LOCK
- 递归查询
- WITH ROLLUP
- 存在HIGH_PRIORITY关键字
- 执行结果返回0行数据（执行计划显示：Zero limit、Impossible WHERE、Impossible HAVING、No matching min/max row、Select tables optimized away、Impossible HAVING noticed after reading const tables、no matching row in const table等）
- 查询中包含zerofill的列，并且这些列能被优化为常量
- generated column、BLOB、TEXT、JSON和GEOMETRY
- Spatial相关函数（如SP_WITHIN_FUNC等）
- aggregation(distinct)，如sum(distinct)、avg(distinct)、count(distinct)
- GROUP_CONCAT
- JSON_ARRAYAGG/JSON_OBJECTAGG
- 用户自定义函数
- STD/STDDEV/STDDEV_POP
- VARIANCE/VAR_POP/VAR_SAMP
- BIT_AND, BIT_OR and BIT_XOR
- set_user_var
- rand(不含参数的除外)
- json_*(如json_length，json_type等)
- st_distance
- get_lock
- is_free_lock，is_used_lock，release_lock， release_all_locks
- sleep
- xml_str
- weight_string
- ref函数(VIEW_REF, OUTER_REF, AGGREGATE_REF)，
- SHA，SHA1，SHA2，MD5

- row_count
- user相关函数（user, current_user, session_user, system_user等）
- 函数extractvalue
- 函数GeomCollection，GeometryCollection，LineString，MultiLineString，MultiPoint，MultiPolygon，Polygon
- 函数MASTER_POS_WAIT
- 空间关系函数（MBRContains，MBRCoveredBy，MBRCovers，MBRDisjoint，MBREquals，MBRIntersects, MBROverlaps, MBRTouches，MBRWithin）
- 函数Point
- 函数PS_CURRENT_THREAD_ID()
- 函数PS_THREAD_ID(CONNECTION_ID())
- 函数WAIT_FOR_EXECUTED_GTID_SET
- 函数WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS
- 函数UNCOMPRESS(COMPRESS())
- 函数STATEMENT_DIGEST_TEXT
- 函数BINARY、函数CONVERT
- ST_开头的函数匀不支持

● **并行执行的执行结果可能存在与串行执行不兼容的情况，主要表现在：**

- 错误或者告警提示次数可能会增多

  对于在串行执行中出现错误/告警提示的查询，在并行执行情况下，每个工作线程可能都会提示错误/告警，导致总体错误/告警提示数会增多。

```
mysql> SELECT dt1 = 99991231235959.999999 AS a, dt2 = 99991231235959.999999 AS b FROM t7;
+------+------+
| a    | b    |
+------+------+
|    0 |    0 |
|    0 |    0 |
|    0 |    1 |
+------+------+
3 rows in set, 2 warnings (0.00 sec)

mysql> show warnings;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1441 | Datetime function: datetime field overflow                            |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
+---------+------+-----------------------------------------------------------------------+
2 rows in set (0.00 sec)

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

mysql> SELECT dt1 = 99991231235959.999999 AS a, dt2 = 99991231235959.999999 AS b FROM t7;
+------+------+
| a    | b    |
+------+------+
|    0 |    0 |
|    0 |    0 |
|    0 |    1 |
+------+------+
3 rows in set, 12 warnings (0.01 sec)

mysql> show warnings;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1441 | Datetime function: datetime field overflow                            |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                            |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                            |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                            |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                            |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                            |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
+---------+------+-----------------------------------------------------------------------+
12 rows in set (0.00 sec)
```


- 精度问题

  并行执行的执行过程中，当select的内容是函数类型时，会比非并行执行多出中间结果的存储过程，可能会导致浮点部分精度差别，导致最终结果有细微的差别。

```
mysql> create table tb(double_col double);
Query OK, 0 rows affected (0.08 sec)

mysql> insert into tb values (-1.7976931348623157e308),(-1.7976931348623157e308);
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select sum(double_col) from tb;
+-----------------+
| sum(double_col) |
+-----------------+
|               0 |
+-----------------+
1 row in set (0.00 sec)

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

mysql> select sum(double_col) from tb;
+-------------------------+
| sum(double_col)         |
+-------------------------+
| -1.7976931348623157e308 |
+-------------------------+
1 row in set (0.02 sec)
```

- 截断问题

  并行执行的执行过程中，当select的内容是函数类型时，会比非并行执行多出中间结果的存储过程。在这个过程中，需要缓存函数的计算结果，可能出现截断(一般是类型转换导致的，例如浮点数类型转为字符串等), 导致最终结果与串行结果有差别。

- 结果集顺序差别

  因为是多个工作线程执行查询，返回的结果集可能与非并行执行顺序不一致。在具有LIMIT查询的情况下，更容易出现与串行结果顺序不同的现象。对于不可见字符，当MySQL判断多个不可见字符相等时，可能会出现结果集顺序不同，或者group by字段不同的现象。

```
mysql> select a,count(*) from t group by a;
+------+----------+
| a    | count(*) |
+------+----------+
|    0 |    32768 |
|    1 |    32768 |
|    2 |    32768 |
|    3 |    32768 |
|    4 |    32768 |
|    5 |    32768 |
|    6 |    32768 |
|    7 |    32768 |
|    8 |    32768 |
|    9 |    32768 |
+------+----------+
10 rows in set (6.37 sec)

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

mysql> select a,count(*) from t group by a;
+------+----------+
| a    | count(*) |
+------+----------+
|    4 |    32768 |
|    5 |    32768 |
|    6 |    32768 |
|    7 |    32768 |
|    8 |    32768 |
|    9 |    32768 |
|    0 |    32768 |
|    1 |    32768 |
|    2 |    32768 |
|    3 |    32768 |
+------+----------+
10 rows in set (4.35 sec)
```

- **union all结果集差别**

  union all会忽略其中的排序算子，并行执行下返回的结果集顺序可能与非并行不一致。在有limit查询的情况下，会出现结果集不同的现象。

2.1.3 开启并行查询

系统参数及状态变量说明

- 支持的系统参数如**表2-1**。

表 2-1 系统参数

参数名称	级别	描述
force_parallel_execute	Global, Session	是否开启并行查询，当设置为“ON”时，表示查询SQL尽可能地使用并行执行。 ● 取值范围：ON, OFF ● 默认值OFF

参数名称	级别	描述
parallel_max_threads	Global	并行执行的最大活跃线程个数。当并行执行的活跃线程超过该值时，新的查询将不允许启用并行执行。 ● 取值范围：0-4294967295 ● 默认值：64
parallel_default_dop	Global, Session	并行执行的默认并行度。当查询语句没有指定并行度时，使用该值。 ● 取值范围：0-1024 ● 默认值：4
parallel_cost_threshold	Global, Session	启用并行执行的代价阈值。只有当查询的执行代价超过该阈值时才有可能进行并行执行。 ● 取值范围：0-4294967295 ● 默认值：1000
parallel_queue_timeout	Global, Session	当不满足并行查询的条件时，请求并行执行的SQL等待超时时间。当等待时间超过该值后，则不再等待，开始进行单线程执行。 ● 取值范围：0-4294967295 ● 默认值：0
parallel_memory_limit	Global	并行执行可用的内存上限。当并行执行使用的内存量超过该值时，新的SQL查询将不会进行并行执行。 ● 取值范围：0-4294967295 ● 默认值：104857600

- 支持的状态变量如表2-2。

表 2-2 状态变量

变量名	级别	描述
PQ_threads_running	Global	当前正在运行的并行执行的总线程数。
PQ_memory_used	Global	当前并行执行使用的总内存量。
PQ_threads_refused	Global	由于总线程数限制，导致未能执行并行执行的查询总数。
PQ_memory_refused	Global	由于总内存限制，导致未能执行并行执行的查询总数。

开启并行查询

支持通过设置系统参数和使用HINT语法两种方式，开启或关闭并行查询。

- **方法一：通过设置系统参数开启或关闭并行查询**

  在管理控制台的参数修改页面，通过设置**系统参数**，开启和关闭并行查询，并设置并行度。

  通过全局参数“force_parallel_execute”来控制是否强制启用并行执行。

  使用全局参数“parallel_default_dop”来控制使用多少线程并行执行。

  使用全局参数“parallel_cost_threshold”来控制当执行代价为多大时，开启并行执行。

  上述参数在使用过程中，随时可以修改，无需重启数据库。

  例如，想要强制开启并行执行，并且并发度为4，最小执行代价为0，可参照如下进行设置：

  ```
  SET force_parallel_execute=ON
  SET parallel_default_dop=4
  SET parallel_cost_threshold=0
  ```

- **方法二：使用HINT开启或关闭并行查询**

  使用HINT语法可以控制单个语句是否进行并行执行。在系统默认关闭并行执行的情况下, 可以使用hint对特定的SQL进行开启。反之，也可以禁止某条SQL进行并行执行。

  开启并行执行：采用下面的HINT语法可以开启并行执行。

  采用默认的参数配置：SELECT /*+ PQ() */ … FROM …

  采用默认的参数配置，同时指定并发度为8：SELECT /*+ PQ(8) */ … FROM …

  采用默认的参数配置，同时指定并行表为t1：SELECT /*+ PQ(t1) */ … FROM …

  采用默认的参数配置，同时指定并行表为t1，并发度为8：SELECT /*+ PQ(t1 8) */ … FROM …

  说明

  PQ HINT紧跟着SELECT关键字才能生效。PQ HINT的并发度参数dop正常取值范围[1, min(parallel_max_threads, 1024)]。

  dop超出正常取值范围时，PQ不生效。

  关闭并行执行：当并行查询开启时，可使用"NO_PQ"的hint语法关闭单条SQL的并行执行。

  SELECT /*+ NO_PQ */ … FROM …

  说明

  NO_PQ hint的优先级高于PQ hint，如果SQL语句出现NO_PQ hint，即使配置PQ hint，该单条SQL也不会并行执行。

查看并行执行的状态

通过如下SQL，查看并行执行的当前状态，显示结果请见**图1**。

show status like "%PQ%"

图 2-2 状态显示

```
mysql> show status like "%PQ%";
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| PQ_memory_refused   | 0     |
| PQ_memory_used      | 0     |
| PQ_threads_refused  | 0     |
| PQ_threads_running  | 0     |
+---------------------+-------+
```

通过EXPLAIN展示查询语句的并行执行计划，显示结果请见**图2**。

图 2-3 并行执行计划结果展示

```
mysql> explain select/*+ PQ(4) */  n_name, sum(l_extendedprice * (1 - l_discount)) as revenue from customer, orders, lineitem, supplier, nation, region where c_custkey = o_custkey and l_orderkey = o_order
key and l_suppkey = s_suppkey and c_nationkey = s_nationkey and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = 'MIDDLE EAST' and o_orderdate >= date '1994-01-01' and o_orderdate < da
te '1994-01-01' + interval '1' year group by n_name order by revenue desc;
+----+-------------+------------+------------+--------+-----------------------+---------------+---------+----------------------------+------+----------+-----------------------------------------------------+
| id | select_type | table      | partitions | type   | possible_keys         | key           | key_len | ref                        | rows | filtered | Extra                                               |
+----+-------------+------------+------------+--------+-----------------------+---------------+---------+----------------------------+------+----------+-----------------------------------------------------+
|  1 | SIMPLE      | <gather2>  | NULL       | ALL    | NULL                  | NULL          | NULL    | NULL                       |   10 |   100.00 | Parallel execute (4 workers, tpch.supplier)         |
|  2 | SIMPLE      | region     | NULL       | ALL    | PRIMARY               | NULL          | NULL    | NULL                       |    5 |    20.00 | Using where; Using temporary; Using filesort        |
|  2 | SIMPLE      | supplier   | NULL       | index  | PRIMARY,SUPPLIER_FK1  | SUPPLIER_FK1  | 4       | NULL                       |   10 |   100.00 | Using index; Using join buffer (Block Nested Loop)  |
|  2 | SIMPLE      | nation     | NULL       | eq_ref | PRIMARY,NATION_FK1    | PRIMARY       | 4       | tpch.supplier.S_NATIONKEY  |    1 |    20.00 | Using where                                         |
|  2 | SIMPLE      | customer   | NULL       | ref    | PRIMARY,CUSTOMER_FK1  | CUSTOMER_FK1  | 4       | tpch.supplier.S_NATIONKEY  |    6 |   100.00 | Using index                                         |
|  2 | SIMPLE      | orders     | NULL       | ref    | PRIMARY,ORDERS_FK1    | ORDERS_FK1    | 4       | tpch.customer.c_custkey    |   15 |    11.11 | Using where                                         |
|  2 | SIMPLE      | lineitem   | NULL       | ref    | PRIMARY               | PRIMARY       | 4       | tpch.orders.o_orderkey     |    4 |    10.00 | Using where                                         |
+----+-------------+------------+------------+--------+-----------------------+---------------+---------+----------------------------+------+----------+-----------------------------------------------------+
7 rows in set, 1 warning (0.02 sec)
```

说明

与传统的执行计划相比，并行执行计划多了一行记录。在查询结果的第一行，展示了并发度、并行表等信息。

2.1.4 验证并行查询效果

本章节使用TPCH测试工具测试并行查询对22条QUERY的性能提升情况。

测试的实例信息如下：

- 实例规格：32 vCPUs | 256 GB
- 内核版本：2.0.26.1
- 并行线程数：16
- 测试数据量：100GB

操作步骤

步骤1 生成测试数据。

1. 请在https://github.com/electrum/tpch-dbgen下载TPCH共用源码。
2. 请在下载的源码文件中，找到makefile.suite文件，并按照如下内容进行修改，修改完成后进行保存。

```
CC      = gcc
# Current values for DATABASE are: INFORMIX, DB2, TDAT (Teradata)
#                                  SQLSERVER, SYBASE, ORACLE
# Current values for MACHINE are:  ATT, DOS, HP, IBM, ICL, MVS,
#                                  SGI, SUN, U2200, VMS, LINUX, WIN32
# Current values for WORKLOAD are:  TPCH
DATABASE= SQLSERVER
MACHINE = LINUX
WORKLOAD = TPCH
```

3. 在源码根目录下，执行下列命令，编译生成TPCH数据工具dbgen。

make -f makefile.suite

4. 使用dbgen执行如下命令，生成TPCH数据100G。

   ./dbgen -s 100

步骤2 登录目标GaussDB(for MySQL)实例，创建目标数据库，并使用如下命令创建TPCH的表。

```
CREATE TABLE nation  ( N_NATIONKEY  INTEGER NOT NULL,
                       N_NAME       CHAR(25) NOT NULL,
                       N_REGIONKEY  INTEGER NOT NULL,
                       N_COMMENT    VARCHAR(152));
CREATE TABLE region  ( R_REGIONKEY  INTEGER NOT NULL,
                       R_NAME       CHAR(25) NOT NULL,
                       R_COMMENT    VARCHAR(152));
CREATE TABLE part    ( P_PARTKEY     INTEGER NOT NULL,
                       P_NAME        VARCHAR(55) NOT NULL,
                       P_MFGR        CHAR(25) NOT NULL,
                       P_BRAND       CHAR(10) NOT NULL,
                       P_TYPE        VARCHAR(25) NOT NULL,
                       P_SIZE        INTEGER NOT NULL,
                       P_CONTAINER   CHAR(10) NOT NULL,
                       P_RETAILPRICE DECIMAL(15,2) NOT NULL,
                       P_COMMENT     VARCHAR(23) NOT NULL );
CREATE TABLE supplier  ( S_SUPPKEY     INTEGER NOT NULL,
                         S_NAME        CHAR(25) NOT NULL,
                         S_ADDRESS     VARCHAR(40) NOT NULL,
                         S_NATIONKEY   INTEGER NOT NULL,
                         S_PHONE       CHAR(15) NOT NULL,
                         S_ACCTBAL     DECIMAL(15,2) NOT NULL,
                         S_COMMENT     VARCHAR(101) NOT NULL);
CREATE TABLE partsupp  ( PS_PARTKEY     INTEGER NOT NULL,
                         PS_SUPPKEY     INTEGER NOT NULL,
                         PS_AVAILQTY    INTEGER NOT NULL,
                         PS_SUPPLYCOST  DECIMAL(15,2)  NOT NULL,
                         PS_COMMENT     VARCHAR(199) NOT NULL );
CREATE TABLE customer  ( C_CUSTKEY     INTEGER NOT NULL,
                         C_NAME        VARCHAR(25) NOT NULL,
                         C_ADDRESS     VARCHAR(40) NOT NULL,
                         C_NATIONKEY   INTEGER NOT NULL,
                         C_PHONE       CHAR(15) NOT NULL,
                         C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
                         C_MKTSEGMENT  CHAR(10) NOT NULL,
                         C_COMMENT     VARCHAR(117) NOT NULL);
CREATE TABLE orders  ( O_ORDERKEY       INTEGER NOT NULL,
                       O_CUSTKEY        INTEGER NOT NULL,
                       O_ORDERSTATUS    CHAR(1) NOT NULL,
                       O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
                       O_ORDERDATE      DATE NOT NULL,
                       O_ORDERPRIORITY  CHAR(15) NOT NULL,
                       O_CLERK          CHAR(15) NOT NULL,
                       O_SHIPPRIORITY   INTEGER NOT NULL,
                       O_COMMENT        VARCHAR(79) NOT NULL);
CREATE TABLE lineitem ( L_ORDERKEY    INTEGER NOT NULL,
                        L_PARTKEY     INTEGER NOT NULL,
                        L_SUPPKEY     INTEGER NOT NULL,
                        L_LINENUMBER  INTEGER NOT NULL,
                        L_QUANTITY    DECIMAL(15,2) NOT NULL,
                        L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
                        L_DISCOUNT    DECIMAL(15,2) NOT NULL,
                        L_TAX         DECIMAL(15,2) NOT NULL,
                        L_RETURNFLAG  CHAR(1) NOT NULL,
                        L_LINESTATUS  CHAR(1) NOT NULL,
                        L_SHIPDATE    DATE NOT NULL,
                        L_COMMITDATE  DATE NOT NULL,
                        L_RECEIPTDATE DATE NOT NULL,
                        L_SHIPINSTRUCT CHAR(25) NOT NULL,
                        L_SHIPMODE     CHAR(10) NOT NULL,
                        L_COMMENT      VARCHAR(44) NOT NULL);
```

步骤3 使用如下命令，将生成的数据导入到TPCH的表中。

```
load data INFILE '/path/customer.tbl' INTO TABLE customer FIELDS TERMINATED BY '|';
load data INFILE '/path/region.tbl' INTO TABLE region FIELDS TERMINATED BY '|';
load data INFILE '/path/nation.tbl' INTO TABLE nation FIELDS TERMINATED BY '|';
load data INFILE '/path/supplier.tbl' INTO TABLE supplier FIELDS TERMINATED BY '|';
load data INFILE '/path/part.tbl' INTO TABLE part FIELDS TERMINATED BY '|';
load data INFILE '/path/partsupp.tbl' INTO TABLE partsupp FIELDS TERMINATED BY '|';
load data INFILE '/path/orders.tbl' INTO TABLE orders FIELDS TERMINATED BY '|';
load data INFILE '/path/lineitem.tbl' INTO TABLE lineitem FIELDS TERMINATED BY '|';
```

步骤4 创建TPCH表的索引。

```
alter table region add primary key (r_regionkey);
alter table nation add primary key (n_nationkey);
alter table part add primary key (p_partkey);
alter table supplier add primary key (s_suppkey);
alter table partsupp add primary key (ps_partkey,ps_suppkey);
alter table customer add primary key (c_custkey);
alter table lineitem add primary key (l_orderkey,l_linenumber);
alter table orders add primary key (o_orderkey);
```

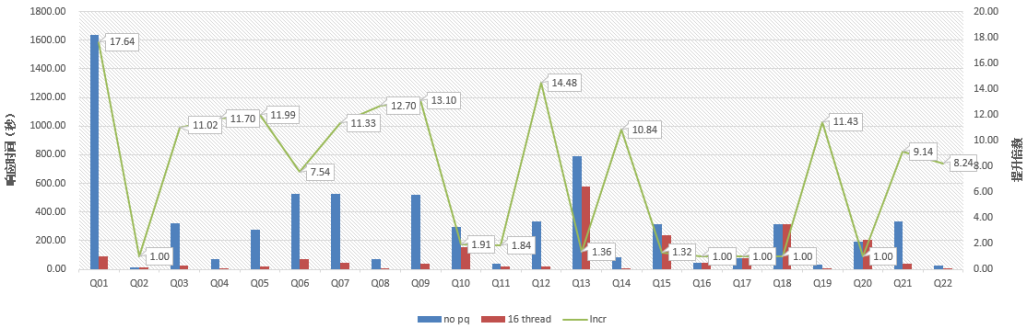
步骤5 请在https://github.com/dragansah/tpch-dbgen/tree/master/tpch-queries获取TPCH 22个查询query语句，并进行相应操作。

----结束

测试结果

通过采用16线程并行查询，其中的17条QUERY的性能得到明显提升，查询速度平均提升10倍以上。TPCH性能测试结果如下图所示。

图 2-4 测试结果



2.2 算子下推（NDP）

2.2.1 功能介绍

什么是算子下推（NDP）

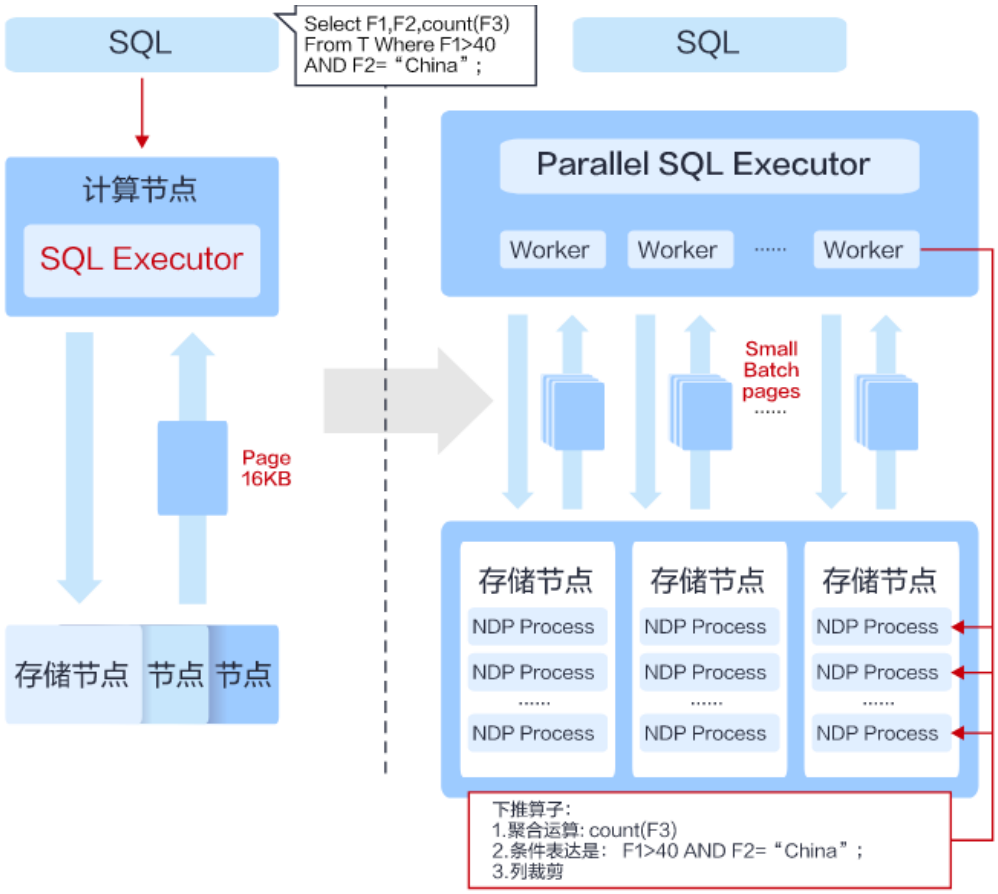
NDP(Near Data Processing)是云数据库GaussDB(for MySQL)发布的旨在提高数据查询效率的计算下推的解决方案。针对数据密集型查询，将提取列、聚合运算、条件过滤等操作从计算节点向下推送给GaussDB(for MySQL)的分布式存储层的多个节点，并行执行。通过计算下推方法，提升了并行处理能力，减少网络流量和计算节点的压力，提高了查询处理执行效率。

工作原理

云数据库GaussDB(for MySQL)采用计算与存储分离的架构，以减少网络流量为主要架构准则，通过NDP设计将该准则应用到查询操作。没有NDP之前，查询处理需要将原始数据从存储节点全部传输到计算节点。通过NDP设计，查询中的I/O密集型和CPU密集型的大部分工作被下推到存储节点完成，仅将所需列及筛选后的行或聚合后的结果值回传给计算节点，使网络流量大幅减少。同时跨存储节点并行处理，使计算节点CPU使用率下降，提升了查询效率性能。

另外，NDP框架同GaussDB(for MySQL)并行查询进行融合，并进行了页面批量预取的设计，达成执行全流程并行，进一步提升查询执行效率。

图 2-5 NDP 工作原理图



使用场景

查询业务能进行下推的场景包主要包括三大类：Projection、Aggregate、Select。

- **Projection**

  列裁剪，只有查询语句所需相关字段内容才会被发送到计算节点。
- **Aggregate**

  典型的聚合操作包括：count、sum、avg、max、min、group by，只发送聚合结果（而不是所有元组）到查询引擎，count (*)是一个最常见的场景。
- **Select - where子句过滤**

  常见的条件表达式：Compare(>=,<=,<,>,==)、Between、In、And/Or，like。

将过滤表达式下推送到存储节点，只有满足条件的行才会发送到计算节点。

支持范围

1. 当前支持对InnoDB表进行计算下推。
2. 当前支持对COMPACT或DYNAMIC行格式的表进行计算下推。
3. 当前支持对Primary Key或BTREE Index进行计算下推，HASH Index或Full-Text Index不支持计算下推。
4. 当前只支持SELECT查询操作进行计算下推，其他DML语句不支持计算下推，INSERT INTO SELECT也不支持计算下推；SELECT 加锁查询(如 SELECT FOR SHARE/UPDATE)不支持计算下推。
5. 表达式下推支持数值类型、日志和时间类型和部分字符串类型(CHAR, VARCHAR)，支持utf8mb4, utf8字符集。
6. 表达式下推谓词支持比较运算(<,>,=,<=,>=,!=), IN, NOT IN, LIKE, NOT LIKE, BETWEEN AND, AND/OR等操作符。

参数说明

表 2-3 参数说明

参数名	级别	描述
ndp_mode	Global **说明** • 如果需要开启Global级别的NDP，请联系技术支持协助解决。 • NDP当前处于邀请测试阶段，共10个名额！	NDP特性开关。 取值范围：off/on 默认取值：off

2.3 DDL 优化

2.3.1 并行 DDL

传统的DDL操作基于单核和传统硬盘设计，导致针对大表的DDL操作耗时较久，延迟过高。以创建二级索引为例，过高延迟的DDL操作会阻塞后续依赖新索引的DML查询操作。

云数据库 GaussDB(for MySQL)支持并行DDL的功能。当数据库硬件资源空闲时，您可以通过并行DDL功能加速DDL执行，避免阻塞后续相关的DML操作，缩短执行DDL操作的窗口期。

约束与限制

- 内核版本为2.0.45.230900及以上版本支持使用该功能。
- 并行创建索引功能，目前支持的索引为Btree二级索引。
- 不支持主键索引、spatial index和fulltext index。如果一个并行创建索引的SQL语句包含主键索引，或者spatial index和fulltext index，客户端将会收到一个告警，

提示该操作不支持并行创建索引，同时该语句会采用单线程创建索引的方式执行完成。假设在修改主键索引时，虽然指定了多线程，但是会收到一个告警，实际上只能通过单线程建索引。

开启并行 DDL

表 2-4 参数说明

参数名称	级别	描述
innodb_rds_parallel_index_creation_threads	Global，Session	• 并行创建索引的线程数。 • 如果值大于1，并行创建；否则，单线程创建。 • 该值范围建议cpu核数的一半，同时不要超过innodb_rds_parallel_index_creation_threads_max，该值默认为8。

使用示例

1. 假设使用sysbench的表，表内有1亿条数据。

   图 2-6 查看表

```
mysql> show create table sbtest1;
+---------+----------------------------------------------------------------
---------------------------------------------------------------------------+
| Table   | Create Table
                                                                           |
+---------+----------------------------------------------------------------
---------------------------------------------------------------------------+
| sbtest1 | CREATE TABLE `sbtest1` (
  `id` int NOT NULL AUTO_INCREMENT,
  `k` int 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=10000001 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

2. 在该表的“k”字段创建索引。

   如图2-7所示，采用社区默认单线程创建索引，耗时146.82s。

   图 2-7 单线程创建创建索引

```
mysql> alter table sbtest1 add index idx_s(k);
Query OK, 0 rows affected (2 min 26.82 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

3. 通过设置innodb_rds_parallel_index_creation_threads= 4，启用4个线程创建索引。

   从图2-8中可以看到创建索引耗时38.72s，与社区单线程相比速度提升了3.79倍。

图 2-8 多线程创建索引

```
mysql> set innodb_rds_parallel_index_creation_threads = 4;
Query OK, 0 rows affected (0.00 sec)

mysql> alter table sbtest1 add index idx_p4(k);
Query OK, 0 rows affected (38.72 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

4. 假设要修改主键索引，虽然指定了多线程，但是会收到一个告警，实际上只能通过单线程建索引。

图 2-9 修改主键索引

```
mysql> set innodb_rds_parallel_index_creation_threads = 4;
Query OK, 0 rows affected (0.00 sec)

mysql> alter table sbtest1 add primary key(id,k);
Query OK, 0 rows affected, 1 warning (10 min 11.36 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> show warnings;
+---------+------+-----------------------------------------------------------+
| Level   | Code | Message                                                   |
+---------+------+-----------------------------------------------------------+
| Warning | 7519 | InnoDB: Creating or rebuilding PK in parallel is disallowed. |
+---------+------+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

2.3.2 DDL 快速超时

对一些特定的DDL操作，实现了单独设置其MDL等锁时间的功能，基于此功能可以实现让这类操作在等待MDL锁时快速超时，避免阻塞后续DML操作。

约束与限制

- 内核版本为2.0.45.230900及以上版本支持使用该功能。
- 目前支持的DDL操作包括：ALTER TABLE、CREATE INDEX、DROP INDEX。

开启 DDL 快速超时

表 2-5 参数说明

参数名称	级别	描述

rds_ddl_lock_wait_timeout	Global，Session	控制当前会话或者全局的DDL超时时间。 • 时间单位为秒，范围为1秒到31536000，默认值为31536000，相当于不开启。 • 对于DDL的等锁超时，其真实超时时间是lock_wait_timeout和rds_ddl_lock_wait_timeout的最小值。 • 对于DDL过程中InnoDB层的加表锁超时（行锁不在该考虑范围），其真实超时时间是innodb_lock_wait_timeout和rds_ddl_lock_wait_timeout）的最小值。

使用示例

1. 首先开启一个客户端，执行加锁操作，示例如下。

   图 2-10 加锁操作

   ```
   mysql>
   mysql>
   mysql> flush tables with read lock;
   Query OK, 0 rows affected (0.01 sec)

   mysql>
   ```

2. 通过如下命令，查看DDL快速超时功能的状态。

   show variables like "%rds_ddl_lock_wait_timeout%";

   图 2-11 查看状态

   ```
   mysql> show variables like "rds_ddl_lock_wait_timeout";
   +----------------------------+----------+
   | Variable_name              | Value    |
   +----------------------------+----------+
   | rds_ddl_lock_wait_timeout | 31536000 |
   +----------------------------+----------+
   1 row in set (0.02 sec)
   ```

   如上图所示，查询到“rds_ddl_lock_wait_timeout”的值是“31536000”，此时是默认值，相当于不开启DDL快速超时功能。如果此时等锁，就会卡在这里。

   ```
   mysql> set rds_ddl_lock_wait_timeout=31536000;
   Query OK, 0 rows affected (0.00 sec)

   mysql>
   mysql>
   mysql> alter table lzk.t_lzk drop index indexa;
   ```

如果需要开启DDL快速超时功能，可以将这个值设置为预期值，操作请参考3。

3. 设置参数。
   执行如下命令，设置“rds_ddl_lock_wait_timeout”参数值。
   set rds_ddl_lock_wait_timeout=1;

   图 2-12 设置参数

```
mysql> show variables like "%rds_ddl_lock_wait_timeout%";
+---------------------------+----------+
| Variable_name             | Value    |
+---------------------------+----------+
| rds_ddl_lock_wait_timeout | 31536000 |
+---------------------------+----------+
1 row in set (0.01 sec)

mysql> set rds_ddl_lock_wait_timeout=1;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like "%rds_ddl_lock_wait_timeout%";
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| rds_ddl_lock_wait_timeout | 1     |
+---------------------------+-------+
1 row in set (0.01 sec)
```

4. 然后执行如下创建索引命令，发现DDL操作会快速超时失败，符合预期。
   alter table lzk.t_lzk drop index indexa;

   图 2-13 执行创建索引命令

```
mysql> alter table lzk.t_lzk drop index indexa;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

2.3.3 非阻塞 DDL

用户在执行DDL操作的时候，如果目标表存在未提交的长事务或大查询，DDL将持续等待获取MDL-X锁。在GaussDB(for MySQL)中，由于MDL-X锁具有最高优先级，DDL在等待MDL-X锁的过程中，将阻塞目标表上所有的新事务，这将导致业务连接的堆积和阻塞，可能会造成整个业务系统崩溃的严重后果。GaussDB(for MySQL)提供的非阻塞DDL功能，可以保证即使在无法获得MDL-X锁的情况下，依然允许新事务进入目标表，从而保证整个业务系统的稳定。

前提条件

内核版本为2.0.54.240600及以上版本支持使用该功能。

使用须知

- 开启非阻塞DDL功能会导致DDL的优先级降低，同时因MDL锁获取失败从而导致执行DDL的失败概率也会相应增大。
- ALTER TABLE、RENAME TABLE、CREATE INDEX、DROP INDEX和OPTIMIZE TABLE语句支持非阻塞DDL功能。

参数说明

您可以先使用“rds_nonblock_ddl_enable”参数开启非阻塞DDL功能，然后通过“rds_nonblock_ddl_retry_times”参数设置获取MDL-X锁超时重试的次数，“rds_nonblock_ddl_retry_interval”参数设置获取MDL-X锁超时重试的时间间隔，“rds_nonblock_ddl_lock_wait_timeout”参数设置获取MDL-X锁超时的时间。

表 2-6 参数说明

参数名称	级别	描述
rds_nonblock_ddl_enable	Global，Session	非阻塞DDL功能开关。取值范围: ● ON：开启非阻塞 DDL功能 ● OFF：关闭非阻塞DDL功能 默认值：OFF
rds_nonblock_ddl_lock_wait_timeout	Global，Session	设置获取MDL-X锁超时时间，取值范围1~31536000，单位为秒，默认值为1。
rds_nonblock_ddl_retry_interval	Global，Session	设置获取MDL-X锁重试的时间间隔，取值范围1~31536000，单位为秒，默认值为6。
rds_nonblock_ddl_retry_times	Global，Session	设置获取MDL-X锁的重试次数，取值范围0~31536000，默认值为0。 当值为0时，由参数lock_wait_timeout和rds_ddl_lock_wait_timeout的较小值计算得到，对于不支持参数rds_ddl_lock_wait_timeout的语句，由lock_wait_timeout计算得到。

使用方法

1. 使用SysBench创建1个测试表sbtest1, 并插入1000000行数据。
   ```
   ./oltp_read_write.lua --mysql-host="集群地址" --mysql-port="端口号" --mysql-user="用户名" --mysql-password="用户密码" --mysql-db="sbtest" --tables=1 --table-size=1000000 --report-interval=1 --percentile=99 --threads=8 --time=6000 prepare
   ```
2. 通过SysBench中的oltp_read_write.lua模拟用户业务。
   ```
   ./oltp_read_write.lua --mysql-host="集群地址" --mysql-port="端口号" --mysql-user="用户名" --mysql-password="用户密码" --mysql-db="sbtest" --tables=1 --table-size=1000000 --report-interval=1 --percentile=99 --threads=8 --time=6000 run
   ```
3. 在目标表sbtest1 上开启一个新事务但不提交，该事务持有目标表sbtest1的MDL锁。
   ```
   begin;
   select * from sbtest1;
   ```
4. 开启一个新会话，在开启和关闭非阻塞 DDL功能的条件下，分别对表sbtest1进行加列操作，观察TPS的变化情况。
   ```
   alter table sbtest1 add column d int;
   ```
5. 测试结果。
   - 关闭非阻塞DDL， TPS持续跌零。默认超时时间为31536000秒，严重影响用户业务。

```
[ 282s ] thds: 8 tps: 1243.98 qps: 24886.58 (r/w/o: 17423.71/4974.92/2487.96) lat (ms,99%): 28.16 err/s: 0.00 reconn/s: 0.00
[ 283s ] thds: 8 tps: 1245.88 qps: 24920.63 (r/w/o: 17444.34/4984.53/2491.76) lat (ms,99%): 25.74 err/s: 0.00 reconn/s: 0.00
[ 284s ] thds: 8 tps: 1219.50 qps: 24404.96 (r/w/o: 17083.97/4881.99/2439.01) lat (ms,99%): 30.26 err/s: 0.00 reconn/s: 0.00
[ 285s ] thds: 8 tps: 1213.57 qps: 24218.09 (r/w/o: 16948.74/4842.21/2427.14) lat (ms,99%): 23.95 err/s: 0.00 reconn/s: 0.00
[ 286s ] thds: 8 tps: 1165.99 qps: 23339.74 (r/w/o: 16338.82/4668.95/2331.97) lat (ms,99%): 26.20 err/s: 0.00 reconn/s: 0.00
[ 287s ] thds: 8 tps: 1238.99 qps: 24818.53 (r/w/o: 17377.64/4962.91/2477.98) lat (ms,99%): 23.95 err/s: 0.00 reconn/s: 0.00
[ 288s ] thds: 8 tps: 1271.04 qps: 25381.55 (r/w/o: 17763.37/5076.11/2542.07) lat (ms,99%): 23.10 err/s: 0.00 reconn/s: 0.00
[ 289s ] thds: 8 tps: 1243.10 qps: 24891.17 (r/w/o: 17427.53/4977.43/2486.21) lat (ms,99%): 24.38 err/s: 0.00 reconn/s: 0.00
[ 290s ] thds: 8 tps: 1277.05 qps: 25503.99 (r/w/o: 17847.69/5103.20/2553.10) lat (ms,99%): 21.89 err/s: 0.00 reconn/s: 0.00
[ 291s ] thds: 8 tps: 1309.02 qps: 26199.49 (r/w/o: 18342.34/5238.10/2619.05) lat (ms,99%): 19.65 err/s: 0.00 reconn/s: 0.00
[ 292s ] thds: 8 tps: 1142.00 qps: 22830.94 (r/w/o: 15979.96/4561.99/2288.99) lat (ms,99%): 21.89 err/s: 5.00 reconn/s: 0.00
[ 293s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 294s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 295s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 296s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 297s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 298s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 299s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 300s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 301s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 302s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 303s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 304s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 305s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 306s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 307s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 308s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 309s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 310s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
[ 311s ] thds: 8 tps: 0.00 qps: 0.00 (r/w/o: 0.00/0.00/0.00) lat (ms,99%): 0.00 err/s: 0.00 reconn/s: 0.00
```

– 开启非阻塞DDL， TPS周期性下降，但未跌零，对用户业务影响较小。

```
[ 446s ] thds: 8 tps: 1382.19 qps: 27619.78 (r/w/o: 19328.65/5526.76/2764.38) lat (ms,99%): 20.37 err/s: 0.00 reconn/s: 0.00
[ 447s ] thds: 8 tps: 1474.77 qps: 29473.85 (r/w/o: 20625.22/5899.09/2949.54) lat (ms,99%): 16.71 err/s: 0.00 reconn/s: 0.00
[ 448s ] thds: 8 tps: 1472.88 qps: 29487.17 (r/w/o: 20644.79/5895.60/2946.78) lat (ms,99%): 17.32 err/s: 0.00 reconn/s: 0.00
[ 449s ] thds: 8 tps: 1374.07 qps: 27468.49 (r/w/o: 19230.04/5490.29/2748.15) lat (ms,99%): 21.11 err/s: 0.00 reconn/s: 0.00
[ 450s ] thds: 8 tps: 1500.13 qps: 30034.56 (r/w/o: 21024.79/6009.51/3000.26) lat (ms,99%): 16.12 err/s: 0.00 reconn/s: 0.00
[ 451s ] thds: 8 tps: 715.01 qps: 14305.30 (r/w/o: 10021.21/2847.06/1437.03) lat (ms,99%): 21.50 err/s: 7.00 reconn/s: 0.00
[ 452s ] thds: 8 tps: 725.83 qps: 14619.61 (r/w/o: 10257.62/2910.32/1451.66) lat (ms,99%): 1013.60 err/s: 0.00 reconn/s: 0.00
[ 453s ] thds: 8 tps: 1370.32 qps: 27368.38 (r/w/o: 19152.46/5476.28/2739.64) lat (ms,99%): 20.37 err/s: 0.00 reconn/s: 0.00
[ 454s ] thds: 8 tps: 1325.85 qps: 26544.90 (r/w/o: 18582.83/5309.38/2652.69) lat (ms,99%): 20.37 err/s: 0.00 reconn/s: 0.00
[ 455s ] thds: 8 tps: 1262.04 qps: 25234.86 (r/w/o: 17663.60/5048.17/2523.09) lat (ms,99%): 19.65 err/s: 0.00 reconn/s: 0.00
[ 456s ] thds: 8 tps: 1383.16 qps: 27693.36 (r/w/o: 19381.34/5544.69/2767.33) lat (ms,99%): 19.29 err/s: 0.00 reconn/s: 0.00
[ 457s ] thds: 8 tps: 1527.61 qps: 30491.10 (r/w/o: 21341.47/6094.42/3055.21) lat (ms,99%): 14.46 err/s: 0.00 reconn/s: 0.00
[ 458s ] thds: 8 tps: 705.90 qps: 14121.98 (r/w/o: 9886.58/2819.60/1415.80) lat (ms,99%): 18.95 err/s: 4.00 reconn/s: 0.00
[ 459s ] thds: 8 tps: 753.10 qps: 15166.09 (r/w/o: 10634.46/3025.42/1506.21) lat (ms,99%): 24.83 err/s: 0.00 reconn/s: 0.00
[ 460s ] thds: 8 tps: 1428.37 qps: 28534.57 (r/w/o: 19970.31/5709.51/2854.75) lat (ms,99%): 17.63 err/s: 0.00 reconn/s: 0.00
[ 461s ] thds: 8 tps: 1387.34 qps: 27740.77 (r/w/o: 19423.75/5540.34/2776.68) lat (ms,99%): 22.28 err/s: 0.00 reconn/s: 0.00
[ 462s ] thds: 8 tps: 1429.64 qps: 28642.76 (r/w/o: 20044.93/5738.55/2859.28) lat (ms,99%): 19.29 err/s: 0.00 reconn/s: 0.00
[ 463s ] thds: 8 tps: 1547.19 qps: 30931.86 (r/w/o: 21656.70/6180.77/3094.39) lat (ms,99%): 14.73 err/s: 0.00 reconn/s: 0.00
[ 464s ] thds: 8 tps: 1484.16 qps: 29654.08 (r/w/o: 20756.15/5929.62/2968.31) lat (ms,99%): 18.28 err/s: 0.00 reconn/s: 0.00
[ 465s ] thds: 8 tps: 721.01 qps: 14451.30 (r/w/o: 10123.21/2879.06/1449.03) lat (ms,99%): 20.00 err/s: 7.00 reconn/s: 0.00
[ 466s ] thds: 8 tps: 716.98 qps: 14446.66 (r/w/o: 10128.76/2883.93/1433.97) lat (ms,99%): 995.51 err/s: 0.00 reconn/s: 0.00
[ 467s ] thds: 8 tps: 1381.13 qps: 27611.68 (r/w/o: 19330.88/5518.54/2762.27) lat (ms,99%): 17.95 err/s: 0.00 reconn/s: 0.00
[ 468s ] thds: 8 tps: 1391.96 qps: 27836.19 (r/w/o: 19482.43/5569.84/2783.92) lat (ms,99%): 18.61 err/s: 0.00 reconn/s: 0.00
[ 469s ] thds: 8 tps: 1372.91 qps: 27476.13 (r/w/o: 19237.69/5492.63/2745.81) lat (ms,99%): 17.95 err/s: 0.00 reconn/s: 0.00
[ 470s ] thds: 8 tps: 1271.28 qps: 25417.51 (r/w/o: 17793.86/5081.10/2542.55) lat (ms,99%): 25.74 err/s: 0.00 reconn/s: 0.00
[ 471s ] thds: 8 tps: 1416.82 qps: 28335.37 (r/w/o: 19833.46/5668.27/2833.64) lat (ms,99%): 15.00 err/s: 0.00 reconn/s: 0.00
```

2.3.4 创建二级索引进度查询

在pfs关闭的情况下，当用户在生产环境中执行建索引的操作时，比较耗时，为支持用户查询ddl进度，本特性用于在用户关闭performance schema 之后仍可以显示高耗时创建index操作的进度。

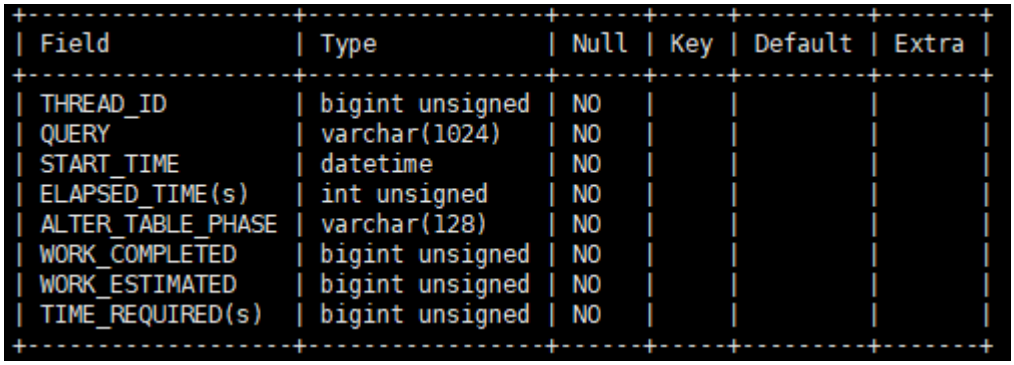
约束与限制

- GaussDB(for MySQL)实例内核版本为2.0.51.240300及以上版本支持使用该功能。
- 仅支持创建二级索引查询进度信息，不支持空间索引，全文索引以及其他DDL进度查询。

功能介绍

该特性默认打开，当表在创建索引的时候，通过查询INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS这个表的信息可以获取当前进度，表结构如下：

图 2-14 表结构



Field	Type	Null	Key	Default	Extra
THREAD_ID	bigint unsigned	NO			
QUERY	varchar(1024)	NO			
START_TIME	datetime	NO			
ELAPSED_TIME(s)	int unsigned	NO			
ALTER_TABLE_PHASE	varchar(128)	NO			
WORK_COMPLETED	bigint unsigned	NO			
WORK_ESTIMATED	bigint unsigned	NO			
TIME_REQUIRED(s)	bigint unsigned	NO			

- THREAD_ID为线程ID。
- QUERY是指客户端下发的创建index语句。
- START_TIME为创建index命令下发时间。
- ELAPSED_TIME是指已经用了多少时间。
- ALTER_TABLE_PHASE是指当前到哪个阶段了。
- WORK_COMPLETED是指当前已经完成的工作量。
- WORK_ESTIMATED是指整个创建index流程一共多少工作量的估计值。
- TIME_REQUIRED是预计还需要多长时间。
- WORK_ESTIMATED和TIME_REQUIRED会随着index创建的进行，一直调整，所以不是并非线性变化。

使用示例

步骤1 执行如下SQL查询目标表结构。

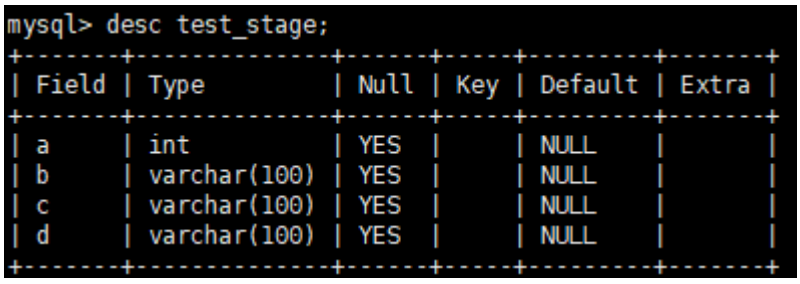
desc *table_name*;

例如：

查询表test_stage的结构。

desc test_stage;

图 2-15 查看表结构



```
mysql> desc test_stage;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int         | YES  |     | NULL    |       |
| b     | varchar(100) | YES  |     | NULL    |       |
| c     | varchar(100) | YES  |     | NULL    |       |
| d     | varchar(100) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

从上述表结构中可以看出，表test_stage中不存在二级索引。

步骤2 执行如下SQL，为目标表的某一列增加索引。

ALTER TABLE *table_name* **ADD INDEX idxa(***field_name***);**

例如：

对test_stage表的a列增加索引。

ALTER TABLE test_stage ADD INDEX idxa(a);

步骤3 执行如下SQL查询创建索引的进度。

SELECT QUERY, ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;

图 2-16 查询创建索引的进度

```
mysql> SELECT QUERY,ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;
+-----------------------------------------------------------+---------------------------------------+
| QUERY                                                     | ALTER_TABLE_PHASE                     |
+-----------------------------------------------------------+---------------------------------------+
| alter table test_stage add index indexa(a), ALGORITHM=INPLACE | alter table (read PK and internal sort) |
+-----------------------------------------------------------+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT QUERY,ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;
+-----------------------------------------------------------+--------------------------+
| QUERY                                                     | ALTER_TABLE_PHASE        |
+-----------------------------------------------------------+--------------------------+
| alter table test_stage add index indexa(a), ALGORITHM=INPLACE | alter table (merge sort) |
+-----------------------------------------------------------+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT QUERY,ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;
+-----------------------------------------------------------+----------------------+
| QUERY                                                     | ALTER_TABLE_PHASE    |
+-----------------------------------------------------------+----------------------+
| alter table test_stage add index indexa(a), ALGORITHM=INPLACE | alter table (insert) |
+-----------------------------------------------------------+----------------------+
1 row in set (0.00 sec)
```

----结束

2.4 Backward Index Scan

Backward Index Scan为索引反向索引扫描，可以通过反向扫描索引的方式消除排序，由于反向扫描与其他一些特性（例如: Index Condition Pushdown（ICP））不兼容，导致优化器选择Backward Index Scan后出现性能劣化的情况。

为了解决上面的问题，GaussDB(for MySQL)对Backward Index Scan特性增加开关，支持动态开启和关闭，帮助用户解决上述问题。

使用须知

内核版本大于等于2.0.48.231200可使用该功能。

开启 BackwardIndexScan

表 2-7 参数说明

参数名称	级别	描述
optimizer_switch	Global，Session	查询优化的总控制开关。 其中，BackwardIndexScan子控制开关为backward_index_scan，控制是否能使用BackwardIndexScan特性，默认值为ON。 ● ON：优化器能选用BackwardIndexScan。 ● OFF：优化器不能选用BackwardIndexScan。

除了使用上述开关来控制BackwardIndexScan特性，还可以使用HINT来实现，语法如下。

- 在SQL语句执行期间开启Backward Index Scan特性
  /*+ set_var(optimizer_switch='backward_index_scan=on') */ :
- 在SQL语句执行期间关闭Backward Index Scan特性
  /*+ set_var(optimizer_switch='backward_index_scan=off') */ :

使用示例

1. 开启BackwardIndexScan。
   - 在optimizer_switch参数里设置此开关值。
     ```
     mysql> set optimizer_switch='backward_index_scan=on';
     Query OK, 0 rows affected (0.00 sec)

     mysql> set optimizer_switch='backward_index_scan=off';
     Query OK, 0 rows affected (0.00 sec)
     ```
   - 通过HINT方式在SQL语句中设置开关值。
     ```
     mysql> explain select /*+ set_var(optimizer_switch='backward_index_scan=on') */
     c13,c16 from tt where c10=10 and c7=7 and c12=12 and to_days(c13)=547864 and
     c16 is not null order by c13 desc;

     mysql> explain select /*+ set_var(optimizer_switch='backward_index_scan=off') */
     c13,c16 from tt where c10=10 and c7=7 and c12=12 and to_days(c13)=547864 and
     c16 is not null order by c13 desc;
     ```
2. 查看控制效果。

   通过执行explai 语句查看执行计划中是否包含"Backward index scan"来确认控制效果。

   a. 准备数据。
      ```
      create table tt(
        id int not null primary key,
      ```

```
  a int,
  b int,
  c int,
  key idx_a_b(a, b));
insert into tt values(1,1,1,1),(2,1,2,1),(3,2,3,2),(4,2,4,3),(5,2,4,4);
```

b. 当Backward Index Scan特性开启时，优化器会选择该特性来消除排序，可以通过查询optimizer_switch参数来判断backward_index_scan特性是否开启。

```
mysql> select @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch:
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=o
n,engine_condition_pushdown=on,index_condition_pushdown=on,mrr=on,mrr_cost_based=on,blo
ck_nested_loop=on,batched_key_access=off,materialization=on,semijoin=on,loosescan=on,firstma
tch=on,duplicateweedout=on,subquery_materialization_cost_based=on,use_index_extensions=on,
condition_fanout_filter=on,derived_merge=on,use_invisible_indexes=off,skip_scan=on,hash_join=o
n,subquery_to_derived=off,prefer_ordering_index=on,hypergraph_optimizer=off,derived_condition
_pushdown=on,derived_merge_no_subquery_check=off,gen_col_partition_prune=off,partial_result
_cache=off,offset_pushdown=off,backward_index_scan=on
1 row in set (0.00 sec)

mysql> explain select * from tt where a = 2 order by b desc;
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+---------------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered |
Extra               |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+---------------------+
|  1 | SIMPLE      | tt    | NULL       | ref  | idx_a_b       | idx_a_b | 5       | const |    3 |   100.00 |
Backward index scan |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+---------------------+
1 row in set, 1 warning (0.00 sec)
```

c. 当Backward Index Scan特性关闭时，优化器会增加Sort算子进行排序，查看以下执行计划。

```
mysql> set optimizer_switch='backward_index_scan=off';
Query OK, 0 rows affected (0.00 sec)

mysql> explain select * from tt where a = 2 order by b desc;
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered |
Extra          |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+----------------+
|  1 | SIMPLE      | tt    | NULL       | ref  | idx_a_b       | idx_a_b | 5       | const |    3 |   100.00 |
Using filesort |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+----------------+
1 row in set, 1 warning (0.00 sec)
```

性能测试

执行某条SQL语句时优化器选用上了Backward Index Scan特性，查询耗时约为4.54s。

```
mysql> explain analyze select detail_record_id, record_id, business_id, business_detail_id, unique_code,
create_time, creator, last_updater, last_update_time, tenant_id, is_usable, operation_time,
detail_operation_type, work_time, operator_id from detail_record d where d.tenant_id=554008 and d.creator
= 585764 and operation_type = 3 and to_days(operation_time) = to_days(now()) and detail_operation_type
is not null order by operation_time desc limit 1\G
*************************** 1. row ***************************
EXPLAIN: -> Limit: 1 row(s)  (cost=151707.54 rows=1) (actual time=4539.137..4539.137 rows=0 loops=1)
    -> Filter: ((d.creator = 585764) and (to_days(d.operation_time) = <cache>(to_days(now())))) and
(d.detail_operation_type is not null))  (cost=151707.54 rows=263565) (actual time=4539.135..4539.135
rows=0 loops=1)
        -> Index lookup on d using idx_time (tenant_id=554008, operation_type=3; iterate backwards)
(cost=151707.54 rows=2928502) (actual time=0.089..4449.445 rows=1562755 loops=1)
1 row in set (4.54 sec)
```

通过HINT的方式控制优化器不能选用BackwardIndexScan特性后，规避此场景下BackwardIndexScan特性与索引条件下推不兼容的问题，再次查询耗时约为0.37s，执行效率明显提升。

```
mysql> explain analyze select /*+ set_var(optimizer_switch='backward_index_scan=off') */ detail_record_id, record_id, business_id, business_detail_id, unique_code, create_time, creator, last_updater, last_update_time, tenant_id, is_usable, operation_time, detail_operation_type, work_time, operator_id from detail_record d where d.tenant_id=554008 and d.creator = 585764 and operation_type = 3 and to_days(operation_time) = to_days(now()) and detail_operation_type is not null order by operation_time desc limit 1\G
*************************** 1. row ***************************
EXPLAIN: -> Limit: 1 row(s)  (cost=209431.59 rows=1) (actual time=370.208..370.208 rows=0 loops=1)
    -> Sort: d.operation_time DESC, limit input to 1 row(s) per chunk  (cost=209431.59 rows=2928502) (actual time=370.207..370.207 rows=0 loops=1)
        -> Filter: ((d.creator = 585764) and (d.detail_operation_type is not null))  (actual time=370.189..370.189 rows=0 loops=1)
            -> Index lookup on d using idx_time (tenant_id=554008, operation_type=3), with index condition: (to_days(d.operation_time) = <cache>(to_days(now())))  (actual time=370.188..370.188 rows=0 loops=1)
1 row in set (0.37 sec)
```

2.5 Statement Outline

MySQL数据库实例运行过程中，SQL语句的执行计划经常会发生改变，导致数据库不稳定。GaussDB(for MySQL) 设计了一套利用 MySQL Optimizer/Index hint 来稳定执行计划的方法，称为 Statement outline，并提供了一组管理接口方便使用（ dbms_outln package ）。

前提条件

GaussDB(for MySQL)的内核版本大于等于2.0.42.230600，支持Statement Outline功能。

注意事项

1. Statement Outline默认关闭，如果您需要使用，则需要联系客服人员开启。
2. Statement Outline不开启情况下对性能没有影响，但是特性开启后在规则较多的情况下对性能会产生影响，会导致性能下降。

功能描述

Statement Outline支持MySQL8.0的Optimizer Hints和Index Hints场景：

- Optimizer Hints

  根据作用域（ query block ）和Hint对象，分为Global-Level Hint, Table-Level Hint, Index-Level Hint和Join-Order Hint等，详情请参见**Optimizer Hints**。
- Index Hints

  Index Hint是向优化器提供有关在查询处理期间如何选择索引，不更改优化器策略。合理的索引可以加快数据索引操作，常用的索引Hint方式有三种，USE（ 参考使用 ），IGNORE（ 忽略 ），FORCE（ 强制 ），详情请参见**Index Hints**。

Statement Outline 表介绍

GaussDB(for MySQL)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。创建表的SQL语句如下：

```
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`)
) ENGINE=InnoDB
 DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0 COMMENT='Statement outline'
```

各字段解释如下：

表 2-8 字段解释

参数	说明
Id	outline ID。
Schema_name	数据库名称。
Digest	Digest_text进行哈希计算得到的64字节的哈希字符串。
Digest_text	SQL语句的特征。
Type	Optimizer Hints中，Hint类型的取值为OPTIMIZER。 Index Hints中，Hint类型的取值为USE INDEX、FORCE INDEX或IGNORE INDEX。
Scope	仅Index Hints需要提供该参数，取值如下： ● FOR GROUP BY ● FOR ORDER BY ● FOR JOIN ● 空字符串 **说明** 如果设置为空字符串，表示所有类型的Index Hints。
State	本规则是否启用，取值范围： ● N ● Y（默认）
Position	● Optimizer Hints Position表示Query Block，因为所有的Optimizer Hints必须作用到Query Block上，Position从1开始，Hint作用在语句的第几个关键字上，Position取值即为对应的值。 ● Index Hints Position表示表的位置， 也是从1开始，Hint作用在第几张表上，Position取值即为对应的值。

Hint	• Optimizer Hints中，Hint表示完整的Hint字符串，例如/*+ MAX_EXECUTION_TIME(1000) */。 • Index Hints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，定义了六个本地存储规则。

- **add_optimizer_outline**

  增加Optimizer Hint。

  - **语法**

    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，建议直接设置Query。

  - **参数说明**

参数名称	是否必选	类型	含义
Schema_name	是	VARCHAR	语句所属的数据库名称。 可设置为空/NULL, 设置为空/NULL后，语句不能匹配。
Digest	否	VARCHAR	语句特征哈希值。 和 *Query*参数可以选择其一设置， 不设置的话需要设置为空字符串。
Query_block	是	INT	Hint作用对象在语句中的位置。 取值范围： 大于等于1。
Hint	是	VARCHAR	Hint名称。
Query	否	VARCHAR	SQL语句。 • 和 *Digest参数*可以选择其一，不设置的话需要设置为空字符串''。 • 如果两个都设置的话，确认*Digest*和*Query*是匹配的，否则参数校验不通过，执行失败。

– **示例**

```
call dbms_outln.add_optimizer_outline("outline_db", '', 1, '/*+ MAX_EXECUTION_TIME(1000) */', "select * from t1 where id = 1");
call dbms_outln.add_optimizer_outline("outline_db", '', 1, '/*+ SET_VAR(foreign_key_checks=OFF) */', "select * from t1 where id = 1");

mysql> explain select * from t1 where id = 1;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+--------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+--------------------------------+
|  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | no matching row in const table |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+--------------------------------+
1 row in set, 1 warning (0.01 sec)

mysql> show warnings;
+-------+------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                                                                                                                 |
+-------+------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select /*+ MAX_EXECUTION_TIME(1000) SET_VAR(foreign_key_checks='OFF') */ NULL AS `id`,NULL AS `col1`,NULL AS `col2` from `outline_db`.`t1` where multiple equal(1, NULL) |
+-------+------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

- **add_index_outline**

  增加Index Hint。

  – **语法**

  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，建议直接设置Query。

  – **参数说明**

参数名称	是否必选	类型	含义
Schema_name	是	VARCHAR	语句所属的db name。 可设置为空/NULL, 设置为空后，语句不能匹配
Digest	否	VARCHAR	语句特征哈希值。 和 *Query*可以选择其一，不设置的话需要设置为''
Position	是	INT	Hint作用对象在语句中的位置，Index Hint作用对象为table, 即为表在语句中的位置。 取值范围：大于等于1。
Type	是	ENUM	Hint类型。取值如下： ● OPTIMIZER ● USE INDEX ● FORCE INDEX ● IGNORE INDEX
Hint	是	VARCHAR	Hint名称，即索引名称集合，多个索引名之间用英文逗号分隔。

Scope	是	ENUM	Hint作用域。取值如下： ● FOR GROUP BY ● FOR ORDER BY ● FOR JOIN ● 空字符串
Query	否	VARCHAR	SQL语句。 ● 和 *Digest*可以选择其一，不设置的话需要设置为''。 ● 如果两个都设置的话，确认*Digest*和*Query*是匹配的，否则参数校验不通过，执行失败。

– **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。

  – **语法**

  dbms_outln.preview_outline(<*Schema_name*>,<*Query*>);

  – **参数说明**

参数名称	是否必选	参数类型	含义
Schema_name	是	VARCHAR	数据库名称。
Query	是	VARCHAR	SQL语句。

  – **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 | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | 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。

  – **语法**

  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 | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 | OPTIMIZER |       |    1 | /*+ SET_VAR(foreign_key_checks=OFF) */                 |    1 |        0 | SELECT * FROM `t1` WHERE `id` = ?                                                     |
|   32 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 | OPTIMIZER |       |    1 | /*+ MAX_EXECUTION_TIME(1000) */                        |    2 |        0 | SELECT * FROM `t1` WHERE `id` = ?                                                     |
|   34 | outline_db | d4dcef634a4a664518e5fb8a21c6ce9b79fccb44b773e86431eb67840975b649 | OPTIMIZER |       |    1 | /*+ BNL(t1,t2) */                                      |    1 |        0 | SELECT `t1` . `id` , `t2` . `id` FROM `t1` , `t2`                                     |
|   35 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f | OPTIMIZER |       |    2 |  /*+ QB_NAME(subq1) */                                 |    2 |        0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )                 |
|   36 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f | OPTIMIZER |       |    1 | /*+ SEMIJOIN(@subq1 MATERIALIZATION, DUPSWEEDOUT) */   |    2 |        0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )                 |
|   30 | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | USE INDEX |       |    1 | ind_1                                                  |    3 |        0 | SELECT * FROM `t1` WHERE `t1` . `col1` = ? AND `t1` . `col2` = ?                      |
|   31 | outline_db | 33c71541754093f78a1f2108795cfb45f8b15ec5d6bff76884f4461fb7f33419 | 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：

i. HIT为Statement Outline命中的次数。
ii. OVERFLOW为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。

  - **语法**

    dbms_outln.del_outline(<*id*>);

  - **参数说明**

参数名称	是否必选	类型	含义
id	是	INT	outline规则的序号ID，为mysql.outline表中的id列的值。不能为空。

  - **示例**

```
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)
```

    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。

- **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。

  - **语法**

    dbms_outln.flush_outline();

  - **示例**

```
update mysql.outline set Position = 1 where Id = 18;
call dbms_outln.flush_outline();
```

功能验证

验证Statement Outline是否有效果，有如下方法：

- 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 | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE      | t1         |     1 | USE INDEX (`ind_1`) |
+------------+----------------------------------------------------------------------+------------+------------+-------+--------------------+
1 row in set (0.00 sec)
```

- 直接使用EXPLAIN查看

2.6 主动终止空闲事务

2.6.1 功能介绍

当一个事务长时间空闲且不提交，执行回滚操作时，会对数据库的资源和性能造成损耗。如果有大量的空闲事务长期不执行也不提交，在业务高峰期的时候发生回滚，性能损耗会较为严重。GaussDB(for MySQL)支持主动终止空闲事务，可针对不同类型的事务用不同的参数进行控制，当空闲事务超时后会被自动的回滚并断开连接。

说明

内核版本为2.0.39.230300及以后的版本支持该功能。

2.6.2 参数介绍

```
mysql> show variables like '%idle%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| idle_readonly_transaction_timeout | 0     |
| idle_transaction_timeout          | 0     |
| idle_write_transaction_timeout    | 0     |
+-----------------------------------+-------+
```

表 2-9 参数说明

参数名	级别	参数描述
idle_readonly_transaction_timeout	global、session	控制只读事务连接的超时时间，单位为秒。 参数设置为0时不生效，即表示只读事务连接的超时时间没有限制。
idle_transaction_timeout	global、session	控制一般空闲事务连接的超时时间，单位为秒。 参数设置为0时不生效，即表示一般空闲事务的连接超时时间没有限制。
idle_write_transaction_timeout	global、session	控制读写事务连接的超时时间，单位为秒。 参数设置为0时不生效，即表示读写事务连接的超时时间没有限制。

上述三个变量的关系：idle_readonly_transaction_timeout和idle_write_transaction_timeout的优先级高于idle_transaction_timeout。

图 2-17 只读事务

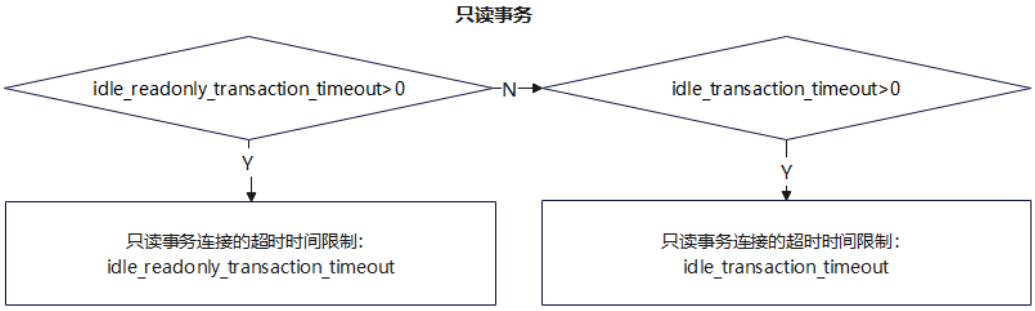
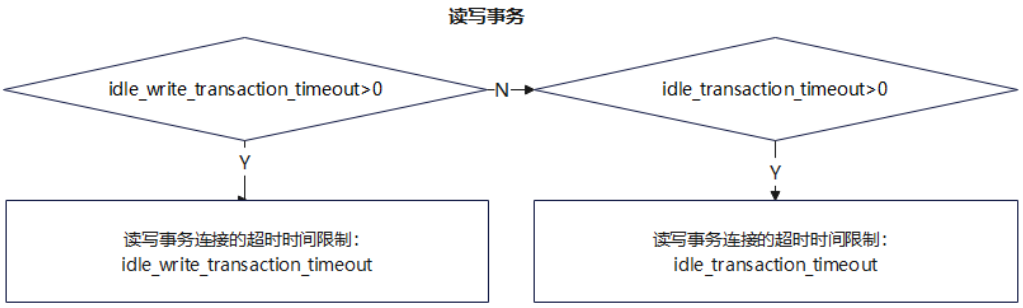


图 2-18 读写事务



2.6.3 使用示例

1. 设置参数：idle_transaction_timeout=10，idle_readonly_transaction_timeout=0，idle_write_transaction_timeout=0
   - 只读事务

     ```
     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)
     ```

     在“idle_transaction_timeout”设置的10s范围以外执行一次查询操作，结果如下：

     ```
     mysql> select * from t1;
     ERROR 2013 (HY000): Lost connection to MySQL server during query
     ```

   - 读写事务

     使用begin开启事务之前，执行查询语句，查询结果如下：

     ```
     mysql> select * from t1;
     +---------+
     | col_int |
     +---------+
     |       1 |
     +---------+
     1 row in set (0.00 sec)
     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)

     mysql> insert into t1 values(2);
     Query OK, 1 row affected (0.00 sec)
     ```

     在“idle_transaction_timeout”设置的10s范围以外执行一次查询操作，结果如下：

     ```
     mysql> select * from t1;
     ERROR 2013 (HY000): Lost connection to MySQL server during query
     ```

新建一个连接，执行查询语句，结果如下，表示此时事务已经回滚。

```
mysql> select * from t1;
+---------+
| col_int |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)
```

2. 设置参数：idle_write_transaction_timeout=15
   - 读写事务

     使用begin开启事务之前，执行查询语句，查询结果如下：

     ```
     mysql> select * from t1;
     +---------+
     | col_int |
     +---------+
     |       2 |
     +---------+
     1 row in set (0.00 sec)

     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)

     mysql> insert into t1 values(3);
     Query OK, 1 row affected (0.00 sec)
     ```

     在“idle_write_transaction_timeout”设置的15s范围以外执行一次查询操作，结果如下：

     ```
     mysql> select * from t1;
     ERROR 2013 (HY000): Lost connection to MySQL server during query
     ```

     新建一个连接，执行查询语句，结果如下，表示此时事务已经回滚。

     ```
     mysql> select * from t1;
     +---------+
     | col_int |
     +---------+
     |       2 |
     +---------+
     1 row in set (0.01 sec)
     ```

3. 设置参数：idle_readonly_transaction_timeout=15
   - 只读事务

     ```
     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)
     ```

     在“idle_readonly_transaction_timeout”设置的15s范围以外执行一次查询操作，结果如下：

     ```
     mysql> select * from t1;
     ERROR 2013 (HY000): Lost connection to MySQL server during query
     ```

2.7 LIMIT OFFSET 下推

2.7.1 功能介绍

社区MySQL的的LIMIT(N)/OFFSET(P)的SELECT语句，引擎层返回所有满足WHERE条件的行给SQL层处理，SQL丢弃OFFSET对应的P行，返回N行数据。当查询二级索引需要访问主表列的时候，引擎层还会先返回表获取所有需要的列信息。对于OFFSET的P远大于LIMIT的N的时候，将会导致引擎层反馈大量的数据到SQL层处理。

GaussDB(for MySQL)提供的LIMIT OFFSET下推功能是把LIMIT OFFSET下推到引擎层处理，提升查询效率。

2.7.2 使用方法

表 2-10 参数说明

参数名称	级别	描述
optimizer_switch	Global，Session	查询优化的总控制开关。 其中，计算下推的子控制开关如下： offset_pushdown：LIMIT OFFSET下推优化开关，默认值为OFF。 ● ON：开启LIMIT OFFSET下推优化开关。 ● OFF：关闭LIMIT OFFSET下推优化开关。

除了使用上述特性开关来控制OFFSET下推功能生效或者不生效，还可以使用HINT来实现。

- OFFSET_PUSHDOWN(table_name)：生效OFFSET下推优化。
- NO_OFFSET_PUSHDOWN(table_name)：不生效OFFSET下推优化。

示例：

基于TPCH的Schema进行举例，通过特性开关打开或者使用HINT方式可以生效，执行EXPLAIN SQL查看执行计划时，Extra列会展示为Using limit-offset pushdown。

- 打开特性开关。

```
mysql> EXPLAIN SELECT * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| id | select_type | table    | partitions | type | possible_keys | key | key_len | ref | rows     | filtered |
Extra                 |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL         | NULL | NULL    | NULL | 59281262 |   100.00 |
Using offset pushdown |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

- 使用HINT。

```
mysql> EXPLAIN SELECT /*+ OFFSET_PUSHDOWN() */ * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| id | select_type | table    | partitions | type | possible_keys | key | key_len | ref | rows     | filtered |
Extra                 |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL         | NULL | NULL    | NULL | 59281262 |   100.00 |
Using offset pushdown |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT /*+ NO_OFFSET_PUSHDOWN() */ * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-------+
```

```
| id | select_type | table   | partitions | type | possible_keys | key | key_len | ref | rows     | filtered | Extra |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL          | NULL | NULL    | NULL | 59281262 |   100.00 |
NULL |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-------+
1 row in set, 1 warning (0.00 sec)
```

2.7.3 性能测试

- 如下SQL语句为Q1，访问主表且无谓词条件。

```
mysql> EXPLAIN SELECT * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| id | select_type | table   | partitions | type | possible_keys | key | key_len | ref | rows     | filtered |
Extra                 |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL          | NULL | NULL    | NULL | 59281262 |   100.00 |
Using offset pushdown |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

- 带有谓词条件的查询，如下SQL语句为Q2。访问二级索引，且包含二级索引范围条件，同时需要回表获取其他列的信息。

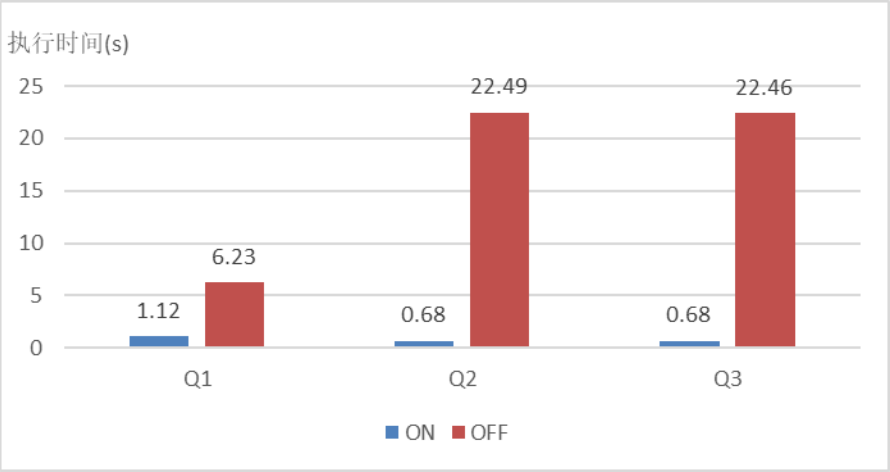
```
mysql> EXPLAIN SELECT * FROM lineitem WHERE l_partkey > 10 AND l_partkey < 200000 LIMIT
5000000, 10;
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-------------------------------------------------+
| id | select_type | table   | partitions | type | possible_keys                 | key         | key_len | ref |
rows     | filtered | Extra                                           |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-------------------------------------------------+
| 1 | SIMPLE     | lineitem | NULL       | range | i_l_partkey_suppkey,i_l_partkey | i_l_partkey | 4       |
NULL | 10949662 |   100.00 | Using offset pushdown; Using index condition |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-------------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

- 带有谓词条件的查询，如下SQL语句为Q3, 带有Order by且可以利用索引排序。

```
mysql> EXPLAIN SELECT * FROM lineitem WHERE l_partkey > 10 AND l_partkey < 200000 ORDER BY
l_partkey LIMIT 5000000, 10;
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-------------------------------------------------+
| id | select_type | table   | partitions | type | possible_keys                 | key         | key_len | ref |
rows     | filtered | Extra                                           |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-------------------------------------------------+
| 1 | SIMPLE     | lineitem | NULL       | range | i_l_partkey_suppkey,i_l_partkey | i_l_partkey | 4       |
NULL | 10949662 |   100.00 | Using offset pushdown; Using index condition |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-------------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

基于TPCH 10 scale的数据，针对上述的查询示例Q1、Q2、Q3。开启与关闭LIMIT OFFSET下推功能的性能对比如下。

图 2-19 性能对比



2.8 IN 谓词转子查询

2.8.1 功能介绍

GaussDB(for MySQL)支持IN谓词转子查询功能。对于满足如下条件的复杂查询，通过该功能优化器可以将某些大的IN谓词转换为IN子查询，从而提升复杂查询的执行性能。

- GaussDB(for MySQL)内核版本为2.0.42.230600或以上的版本。
- IN列表中的元素个数超过rds_in_predicate_conversion_threshold参数设置的个数。

概述

社区MySQL在处理column IN (const1, const2, ...)时，如果column上面有索引，那么通常优化器会选择Range scan进行扫描。但是在进行Range scan分析的时候，range_optimizer_max_mem_size定义了分析过程中需要的最大内存。如果IN后面的list非常大，使用的内存会超过定义的最大内存，会使得Range scan失效，从而引发查询的性能下降。如果想解决该问题，可以调大允许使用的最大内存，但是该内存是session级别的，也就是说每个session进行该查询都会占用同样的内存，所以容易引发用户实例发生OOM。另外即使可以做range optimizer，如果in value过多超过eq_range_index_dive_limit限制，导致无法走index dive而只能走索引统计信息，而大量value所对应的统计信息比较简单，很有可能出现估算不准确的情况，导致性能回退。将IN子句转换为子查询，转换为subquery之后，后续优化器会继续考虑是否转换为semijoin，提升性能。具体变换过程如下：

```
select ... from lineitem where l_partkey in (...)
```

====>

```
select ... from lineitem where l_partkey in
 (select tb._col_1 from (values (9628136),(19958441),...) tb)
```

2.8.2 注意事项

支持的查询语句

- SELECT
- INSERT ... SELECT
- REPLACE ... SELECT
- 支持视图，PREPARED STMT

约束与限制

- 只支持常量IN LIST（包括NOW(), ? 等不涉及表查询的语句）。
- 不支持STORED PROCEDURE/FUNCTION/TRIGGER。
- 不支持NOT IN，以及无法使用索引的场景。

2.8.3 使用方法

您可以通过“rds_in_predicate_conversion_threshold”参数设置IN谓词转子查询功能。

说明

该参数值默认为0, 表示关闭该功能。如果您需要使用，则需要联系客服人员开启。

表 2-11 参数说明

参数名称	级别	描述
rds_in_predicate_conversion_threshold	Global	IN谓词转子查询功能控制开关。当SQL语句的IN列表中的元素个数超过该参数的取值时，则SQL语句进行转换，将IN谓词转换为子查询。

示例：

- 原查询：

```
mysql> explain select * from t where a in (1,2,3,4,5);
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------
+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered |
Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------
+-------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | idx1          | NULL | NULL    | NULL |    5 |   100.00 | Using
where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain format=tree select * from t where a in (1,2,3,4,5);
+-------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                         |
+-------------------------------------------------------------------------------------------------+
| -> Filter: (t.a in (1,2,3,4,5))  (cost=0.75 rows=5)
    -> Table scan on t  (cost=0.75 rows=5)
```

```
 |
+-----------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

- 转化后：

```
mysql> set rds_in_predicate_conversion_threshold=3;
Query OK, 0 rows affected (0.00 sec)

mysql> explain select * from t where a in (1,2,3,4,5);
+----+-------------+------------------+------------+--------+---------------------+---------------------+---------
+----------+------+----------+-------------------+
| id | select_type | table            | partitions | type   | possible_keys       | key                 | key_len | ref
| rows | filtered | Extra             |
+----+-------------+------------------+------------+--------+---------------------+---------------------+---------
+----------+------+----------+-------------------+
|  1 | SIMPLE      | t                | NULL       | ALL    | idx1                | NULL                | NULL    | NULL
|    5 |   100.00 | Using where       |
|  1 | SIMPLE      | <in_predicate_2> | NULL       | eq_ref | <auto_distinct_key> | <auto_distinct_key> |
5       | test.t.a |    1 |   100.00 | IN-list converted |
+----+-------------+------------------+------------+--------+---------------------+---------------------+---------
+----------+------+----------+-------------------+
2 rows in set, 1 warning (0.00 sec)

mysql> explain format=tree select * from t where a in (1,2,3,4,5);
+------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------
-----------------+
|
EXPLAIN
                                                                                                          |
+------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------
-----------------+
| -> Nested loop inner join  (cost=2.50 rows=5)
    -> Filter: (t.a is not null)  (cost=0.75 rows=5)
        -> Table scan on t  (cost=0.75 rows=5)
    -> Single-row index lookup on <in_predicate_2> using <auto_distinct_key> (a=t.a)  (cost=0.27
rows=1)
 |
+------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------
-----------------+
```

使用explain，在执行计划中的table列存在<in_predicate_*>（*为数字），该表即为构造的临时表，其中存储了IN查询中的所有数据。

也可以通过查看optimize trace，trace中存在in_to_subquery_conversion相关信息。

```
| explain format=tree select * from t where a in (1,2,3,4,5) | {
  "steps": [
    {
      "join_preparation": {
        "select#": 1,
        "steps": [
          {
            "IN_uses_bisection": true
          },
          {
            "in_to_subquery_conversion": {
              "item": "(`t`.`a` in (1,2,3,4,5))",
              "steps": [
                {
                  "creating_tmp_table": {
                    "tmp_table_info": {
                      "table": "intermediate_tmp_table",
                      "columns": 1,
                      "row_length": 5,
                      "key_length": 5,
                      "unique_constraint": false,
```

```
      "makes_grouped_rows": false,
      "cannot_insert_duplicates": true,
      "location": "TempTable"
    }
  }
},
```

2.8.4 性能测试

使用sysbench模型测试。

1. 准备1000w数据。
   ```
   sysbench /usr/share/sysbench/oltp_read_only.lua --tables=1 --report-interval=10 --table-size=10000000  --mysql-user=root --mysql-password=123456 --mysql-host=127.0.0.1 --mysql-port=3306 --mysql-db=sbtest --time=300 --max-requests=0  --threads=200   prepare
   ```
2. 查询带1万个IN。
   ```
   select count(*) from sbtest1 where id/k in (... ...);
   ```

性能对比如下表所示：

表 2-12 性能数据

测试方法	开启转换	关闭转换（不适用range_opt）	性能对比
带索引	0.09	2.48	提升26.5倍

2.9 多表连接场景下 DISTINCT 优化

对于多表连接+DISTINCT场景，MySQL 8.0需要扫描表连接后的结果，当表连接数量多或基表数据量大时，需要扫描的数据量很大，导致执行效率很低。

为了提升DISTINCT，尤其多表连接下DISTINCT的查询效率，GaussDB(for MySQL)在执行优化器中加入了剪枝功能，可以去除不必要的扫描分支，提升查询性能。

适用场景

- Nested Loop Inner Join + Distinct
- Nested Loop Outer Join + Distinct

使用须知

内核版本大于等于2.0.51.240300时可使用该功能。

开启多表连接 DISTINCT 优化

表 2-13 参数说明

参数名称	级别	描述
rds_nlj_distinct_optimize	Global，Session	DISTINCT优化特性开关，默认值为OFF。 ● ON：开启DISTINCT优化特性。 ● OFF：关闭DISTINCT优化特性。

除了使用上述开关来控制优化特性生效或者不生效，还可以使用HINT来实现，语法如下。

- 开启DISTINCT优化特性
  /*+ SET_VAR(rds_nlj_distinct_optimize=ON) */
- 关闭 DISTINCT 优化特性
  /*+ SET_VAR(rds_nlj_distinct_optimize=OFF) */

使用示例

1. 使用如下任意方式开启DISTINCT优化特性。
   - 通过SET命令设置此开关值。
     ```
     mysql> SET rds_nlj_distinct_optimize=ON;
     Query OK, 0 rows affected (0.00 sec)

     mysql> SET rds_nlj_distinct_optimize=OFF;
     Query OK, 0 rows affected (0.00 sec)
     ```
   - 通过HINT方式在SQL语句中设置开关值。
     ```
     mysql> EXPLAIN ANALYZE SELECT/*+ SET_VAR(rds_nlj_distinct_optimize=ON) */
     DISTINCT tt1.a FROM t1 AS tt1 JOIN t1 AS tt2 JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a;

     mysql> EXPLAIN ANALYZE SELECT/*+ SET_VAR(rds_nlj_distinct_optimize=OFF) */
     DISTINCT tt1.a FROM t1 AS tt1 JOIN t1 AS tt2 JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a;
     ```
2. 确认多表连接场景下DISTINCT优化效果。

   通过执行Explain Analyze/Explain Format=tree语句可以确认优化是否生效，执行计划出现'with distinct optimization'关键字时，说明优化生效。

   具体步骤如下：

   a. 准备数据。
      ```
      CREATE TABLE t1(a INT, KEY(a));
      INSERT INTO t1 VALUES(1),(2),(5),(6),(7),(8),(9),(11);
      ANALYZE TABLE t1;
      ```
   b. 关闭特性，执行以下SQL语句，优化器选择默认的执行计划。
      ```
      mysql> SET rds_nlj_distinct_optimize=OFF;
      Query OK, 0 rows affected (0.00 sec)

      mysql> EXPLAIN FORMAT=TREE SELECT DISTINCT tt1.a FROM t1 AS tt1 LEFT JOIN t1
      AS tt2 ON TRUE LEFT JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a\G
      ```

```
*************************** 1. row ***************************
EXPLAIN: -> Table scan on <temporary>
    -> Temporary table with deduplication  (cost=29.18 rows=64)
        -> Nested loop left join  (cost=29.18 rows=64)
            -> Left hash join (no condition)  (cost=6.78 rows=64)
                -> Index scan on tt1 using a  (cost=1.05 rows=8)
                -> Hash
                    -> Index scan on tt2 using a  (cost=0.13 rows=8)
            -> Filter: ((tt2.a + 3) = tt3.a)  (cost=0.25 rows=1)
                -> Index lookup on tt3 using a (a=(tt2.a + 3))  (cost=0.25 rows=1)
```

c. 开启特性，执行以下SQL语句，执行计划中可以看到有"with distinct optimization"关键字，说明此优化生效。

```
mysql> SET rds_nlj_distinct_optimize=ON;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN FORMAT=TREE SELECT DISTINCT tt1.a FROM t1 AS tt1 LEFT JOIN t1
AS tt2 ON TRUE LEFT JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a\G
*************************** 1. row ***************************
EXPLAIN: -> Table scan on <temporary>
    -> Temporary table with deduplication  (cost=29.18 rows=64)
        -> Nested loop left join with distinct optimization  (cost=29.18 rows=64)
            -> Left hash join (no condition)  (cost=6.78 rows=64)
                -> Index scan on tt1 using a  (cost=1.05 rows=8)
                -> Hash
                    -> Index scan on tt2 using a  (cost=0.13 rows=8)
            -> Filter: ((tt2.a + 3) = tt3.a)  (cost=0.25 rows=1)
                -> Index lookup on tt3 using a (a=(tt2.a + 3))  (cost=0.25 rows=1)
```

性能测试

GaussDB(for MySQL)执行耗时2.7秒完成，只需要扫描约61万行数据。相比MySQL 8.0 社区版本执行耗时约186秒，扫描数据量4400万，执行效率大大提升。

如下示例中，对7个表连接后的结果做DISTINCT，使用MySQL 8.0.30社区版本，执行耗时186秒，扫描了约4400万行数据。

GaussDB(for MySQL)执行耗时2.7秒，扫描约61万行数据。

查询语句：

```
select distinct ed.code,et.*
from ele_template et
left join ele_template_tenant ett on ett.template_id = et.id
left join ele_relation tm on tm.tom_id = et.id and tm.jerry_type = 'chapter'
left join ele_relation mv on mv.tom_id = tm.jerry_id and mv.jerry_type = 'variable'
left join ele_relation cv on cv.jerry_id = mv.jerry_id and cv.tom_type = 'column'
left join ele_doc_column edc on edc.id = cv.tom_id
left join ele_doc ed on ed.id = edc.doc_id
where ett.uctenantid = 'mmo0l3f8'
and ed.code = 'contract'
and et.billtype = 'contract'
order by ifnull(et.utime,et.ctime)
desc limit 0,10;
```

执行计划：

```
+----+-------------+-------+------------+--------+-------------------------+-----------------+---------
+----------------------+------+----------+--------------------------------------------+
| id | select_type | table | partitions | type   | possible_keys           | key             | key_len | ref               |
rows | filtered | Extra                                      |
+----+-------------+-------+------------+--------+-------------------------+-----------------+---------
+----------------------+------+----------+--------------------------------------------+
```

```
|  1 | SIMPLE      | ed    | NULL       | ref    | PRIMARY,idx_code         | idx_code         | 203     | const
|    1 |   100.00 | Using index; Using temporary; Using filesort |
|  1 | SIMPLE      | ett   | NULL       | ref    | PRIMARY,idx_uctenanatid | idx_uctenanatid | 203     |
const               |  352 |   100.00 | Using index                                  |
|  1 | SIMPLE      | et    | NULL       | eq_ref | PRIMARY,idx_billtype     | PRIMARY          | 8       |
test.ett.template_id |    1 |    94.57 | Using where                                  |
|  1 | SIMPLE      | tm    | NULL       | ref    | idx_tom_id,idx_jerry_id | idx_tom_id       | 9       |
test.ett.template_id |   59 |    10.00 | Using index condition; Using where; Distinct |
|  1 | SIMPLE      | mv    | NULL       | ref    | idx_tom_id,idx_jerry_id | idx_tom_id       | 9       |
test.tm.jerry_id     |   59 |    10.00 | Using where; Distinct                        |
|  1 | SIMPLE      | cv    | NULL       | ref    | idx_tom_id,idx_jerry_id | idx_jerry_id     | 9       | test.mv.jerry_id
|   47 |    10.00 | Using where; Distinct                        |
|  1 | SIMPLE      | edc   | NULL       | eq_ref | PRIMARY,idx_doc_id       | PRIMARY          | 8       |
test.cv.tom_id       |    1 |    50.00 | Using where; Distinct                        |
+----+-------------+-------+------------+--------+--------------------------+------------------+---------
+----------------------+------+----------+----------------------------------------------+
```

图 2-20 执行耗时对比

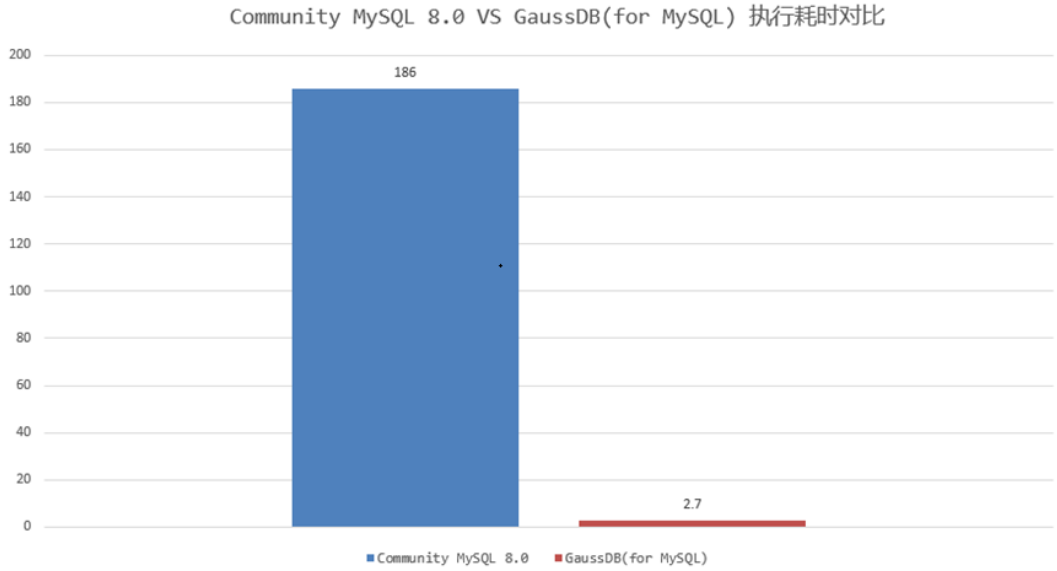
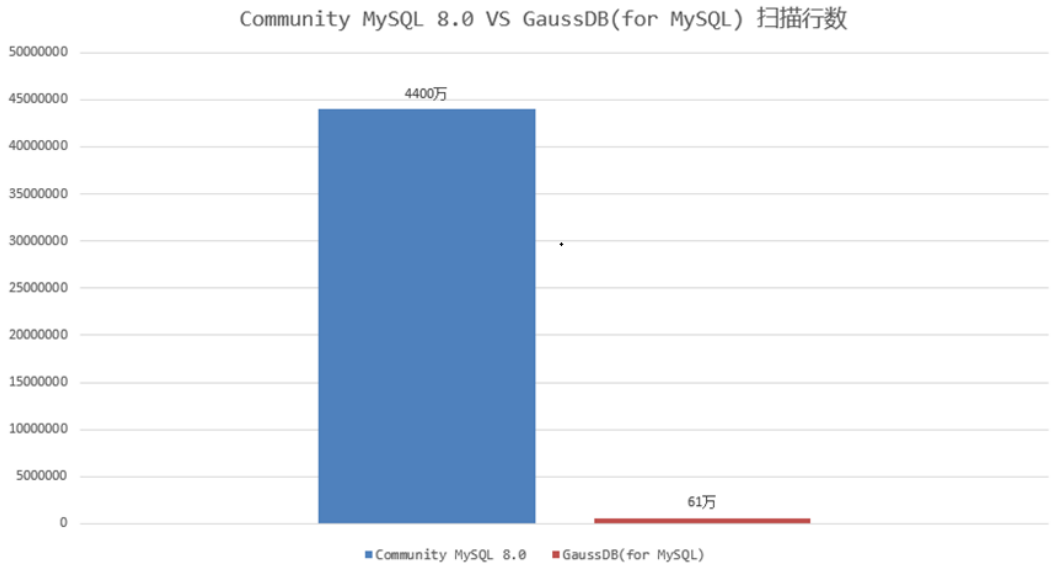


图 2-21 扫描行数



2.10 大事务检测能力

众所周知，大事务的存在对实例的健康平稳运行有一些影响，典型场景如大事务的回滚时间很长，会导致升级、规格变更时间变长。GaussDB(for MySQL)提供了大事务检测的能力，当出现大事务，可以通过告警通知客户及时提交。

前提条件

- 内核2.0.39.230300及以上版本支持该功能。
- 需要根据以下条件设置对应的参数值为“ON”。
  - 内核版本小于2.0.45.230900时，需要确认“log-bin”参数值为“ON”。
  - 内核版本大于或等于2.0.45.230900时，需要确认“rds_global_sql_log_bin”参数值为“ON”。

使用方法

1. 增加系统变量“rds_warn_max_binlog_cache_size”检测大事务。

表 2-14 参数说明

参数名称	级别	描述
rds_warn_max_binlog_cache_size	global	当大事务产生的Binlog超过该值时，则报WARNING提示客户。 默认值为：18446744073709547520 取值范围为：[4096，18446744073709547520]

事务产生的Binlog大小超过rds_warn_max_binlog_cache_size时，为了提醒客户及时提交事务，系统会报WARNING到客户端。为了避免给客户端发送多个WARNING，限制事务中每个statement发送一次这个WARNING给客户端。

当配置rds_warn_max_binlog_cache_size为40KB(40960)：

```
mysql> CREATE TABLE t1 (
    -> a longtext
    -> ) DEFAULT CHARSET=latin1;
Query OK, 0 rows affected (0.12 sec)

mysql> show variables like 'rds_warn_max_binlog_cache_size';
+---------------------------------+-------+
| Variable_name                   | Value |
+---------------------------------+-------+
| rds_warn_max_binlog_cache_size | 40960 |
+---------------------------------+-------+
1 row in set (0.01 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> show warnings;
+---------+------
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
----+
| Level   | Code | Message                                |
+---------+------
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
----+
| Warning | 4008 | Recommend you to INSERT/UPDATE/DELETE rows in batches by multiple
transactions. The current transaction required more than 'rds_warn_max_binlog_cache_size' (40960)
bytes of storage. Which shall cause replication latency. Please commit it. |
+---------+------
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
----+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',50000));
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> show warnings;
+---------+------
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------+
| Level   | Code | Message                                |
+---------+------
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------+
| Warning | 4008 | Recommend you to INSERT/UPDATE/DELETE rows in batches by multiple
transactions. The current transaction required more than 'rds_warn_max_binlog_cache_size' (40960)
bytes of storage. Which shall cause replication latency. |
+---------+------
+-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

2. 查看当前连接的事务Binlog cache大小。

```
mysql> CREATE TABLE t1 (
->   a longtext
-> ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
Query OK, 0 rows affected (0.10 sec)

mysql> SHOW STATUS LIKE 'Rds_binlog_trx_cache_size';
+---------------------------+-------+
| Variable_name             | Value |
```

```
+----------------------------+-------+
| Rds_binlog_trx_cache_size | 0     |
+----------------------------+-------+
1 row in set (0.04 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected (0.01 sec)

mysql> SHOW STATUS LIKE 'Rds_binlog_trx_cache_size';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Rds_binlog_trx_cache_size | 20150 |
+----------------------------+-------+
1 row in set (0.05 sec)
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW STATUS LIKE 'Rds_binlog_trx_cache_size';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Rds_binlog_trx_cache_size | 0     |
+----------------------------+-------+
1 row in set (0.09 sec)
```

3. 查看当前所有连接的事务Binlog cache大小。

```
mysql> SHOW GLOBAL STATUS LIKE 'rds_binlog_trx_cache_size';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Rds_binlog_trx_cache_size | 40300 |
+----------------------------+-------+
1 row in set (0.05 sec)
```

2.11 分区表增强

2.11.1 二级分区

2.11.1.1 功能介绍

GaussDB(for MySQL)分区表完全兼容社区MySQL的语法和功能。同时，GaussDB(for MySQL)分区表相对于社区MySQL进行了功能增强，支持丰富的分区表类型及组合，使您可以更加便携、简单和高效的使用分区表。

GaussDB(for MySQL)兼容的社区MySQL分区表类型如下：

- HASH
- KEY
- RANGE
- LIST
- RANGE-HASH
- RANGE-KEY
- LIST-HASH

- LIST-KEY

组合分区由一级分区（主分区）和二级分区（子分区）组成。

GaussDB(for MySQL)组合分区功能支持的分区表类型如下：

- RANGE-RANGE
- RANGE-LIST
- LIST-RANGE
- LIST-LIST
- HASH-HASH
- HASH-KEY
- HASH-RANGE
- HASH-LIST
- KEY-KEY
- KEY-HASH
- KEY-RANGE
- KEY-LIST

2.11.1.2 注意事项

- GaussDB(for MySQL)的内核版本需要大于等于2.0.48.231200。
- 如需使用分区表扩展类型功能，请在管理控制台右上角，选择“**工单 > 新建工单**”，提交申请。

2.11.1.3 RANGE-RANGE

约束与限制

- RANGE类型要求每个分区定义的分区键值value或value_list必须是单调递增的。
- MAXVALUE只能位于最后位置。
- 对于NULL的处理，认为NULL值是无限小的值，因此NULL值总是会插入第一个分区定义里。
- 每个主分区下的子分区可以看成是一个新的一级的RANGE分区表，因此所有的规则和约束都同一级的RANGE是一样的。

语法

创建一个或多个RANGE-RANGE分区表，其中每个分区可能有一个或一个以上的RANGE类型的子分区。

```
CREATE TABLE ... PARTITION BY RANGE {(expr) | COLUMNS(column_list)}
  SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
[(partition_definition [, partition_definition] ...)];
```

其中，partition_definition为：

```
PARTITION partition_name
      VALUES LESS THAN {(value | MAXVALUE | value_list) | MAXVALUE}
[(subpartition_definition [, subpartition_definition] ...)]
```

subpartition_definition为：

```
SUBPARTITION subpartition_name
    VALUES LESS THAN {value | value_list | MAXVALUE}
```

表 2-15 参数说明

参数名称	描述
expr	分区字段表达式。目前只支持INT类型，不支持字符类型。
column_list	RANGE COLUMNS的情况下使用。分区字段列表，不支持表达式，可以支持多列。
value	分区边界值。
value_list	RANGE COLUMNS的情况下使用，多个字段的边界值。
MAXVALUE	最大值。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称， 同一个表中不可重复。

使用示例

- **RANGE-RANGE类型的使用示例**

```
CREATE TABLE tbl_range_range (col1 INT, col2 INT, col3 varchar(20))
PARTITION BY RANGE(col1)
SUBPARTITION BY RANGE(col2)
(
 PARTITION p0 VALUES LESS THAN (1000) (
   SUBPARTITION s0 VALUES LESS THAN(100),
   SUBPARTITION s1 VALUES LESS THAN(MAXVALUE)
 ),
 PARTITION p1 VALUES LESS THAN (2000)
 (
   SUBPARTITION s2 VALUES LESS THAN(100),
   SUBPARTITION s3 VALUES LESS THAN(200)
 ),
 PARTITION p2 VALUES LESS THAN (MAXVALUE)
 (
   SUBPARTITION s4 VALUES LESS THAN(200),
   SUBPARTITION s5 VALUES LESS THAN(400)
 )
);
```

- **RANGE COLUMNS-RANGE类型的使用示例**

```
CREATE TABLE tbl_range_col_range (col1 INT, col2 INT, col3 INT)
PARTITION BY RANGE COLUMNS(col1, col2)
SUBPARTITION BY RANGE(col3)
(
 PARTITION p1 VALUES LESS THAN(1000, MAXVALUE)(
   SUBPARTITION s0 VALUES LESS THAN(100),
   SUBPARTITION s1 VALUES LESS THAN(MAXVALUE)
 ),
 PARTITION p2 VALUES LESS THAN(2000, MAXVALUE)(
   SUBPARTITION s2 VALUES LESS THAN(100),
```

```
  SUBPARTITION s3 VALUES LESS THAN(200)
 ),
 PARTITION p3 VALUES LESS THAN(MAXVALUE, MAXVALUE)(
  SUBPARTITION s4 VALUES LESS THAN(200),
  SUBPARTITION s5 VALUES LESS THAN(400)
 )
);
```

2.11.1.4 RANGE-LIST

约束与限制

- LIST分区类型要求分区定义中的value或value_list都是唯一的，不同分区定义里也需要不能重复引入。
- 对于NULL的处理，只有value中包含了NULL值，才能插入或查询出NULL值，否则是不符合定义的，不允许插入。
- 每个主分区下的子分区可以看成是一个新的一级的LIST分区表，因此所有的规则和约束都同一级的LIST是一样的, 不同主分区下的子分区定义可以不一样。

语法

创建一个或多个RANGE-LIST分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE ... PARTITION BY RANGE {(expr) | COLUMNS(column_list)}
  SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
[(partition_definition [, partition_definition] ...)];
```

其中，partition_definition为：

```
PARTITION partition_name
    VALUES LESS THAN {(value | value_list) | MAXVALUE}
[(subpartition_definition [, subpartition_definition] ...)]
```

subpartition_definition为：

```
SUBPARTITION subpartition_name
    VALUES IN {(value | value_list)}
```

表 2-16 参数说明

参数名称	描述
expr	分区字段表达式。目前只支持INT类型，不支持字符类型。
column_list	RANGE COLUMNS的情况下使用。分区字段列表，不支持表达式，可以支持多列。
value	分区边界值。
value_list	RANGE COLUMNS的情况下使用，多个字段的边界值。
MAXVALUE	最大值。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称， 同一个表中不可重复。

使用示例

- **RANGE-LIST类型的使用示例**

```
CREATE TABLE tbl_range_list (col1 INT, col2 INT, col3 varchar(20))
PARTITION BY RANGE(col1)
   SUBPARTITION BY LIST(col2)
(
 PARTITION m1 VALUES LESS THAN(1000) (
   SUBPARTITION p0 VALUES in (1, 2),
   SUBPARTITION p1 VALUES in (3, 4),
   SUBPARTITION p2 VALUES in (5, 6)
 ),
 PARTITION m2 VALUES LESS THAN(2000) (
   SUBPARTITION p3 VALUES in (1, 2),
   SUBPARTITION p4 VALUES in (3, 4),
   SUBPARTITION p5 VALUES in (5, 6)
 ),
 PARTITION m3 VALUES LESS THAN(MAXVALUE) (
   SUBPARTITION p6 VALUES in (1, 2),
   SUBPARTITION p7 VALUES in (3, 4),
   SUBPARTITION p8 VALUES in (5, 6)
 )
);
```

- **RANGE COLUMNS-LIST类型的使用示例**

```
CREATE TABLE tbl_range_columns_list
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY RANGE COLUMNS(col4)
   SUBPARTITION BY LIST(col1)
(
 PARTITION dp1 VALUES LESS THAN('2023-01-01')(
   SUBPARTITION p0 VALUES in (1, 2),
   SUBPARTITION p1 VALUES in (3, 4),
   SUBPARTITION p2 VALUES in (5, 6)
 ),
 PARTITION dp2 VALUES LESS THAN('2024-01-01')(
   SUBPARTITION p3 VALUES in (1, 2),
   SUBPARTITION p4 VALUES in (3, 4),
   SUBPARTITION p5 VALUES in (5, 6)
 ),
 PARTITION dp3 VALUES LESS THAN('2025-01-01')(
   SUBPARTITION p6 VALUES in (1, 2),
   SUBPARTITION p7 VALUES in (3, 4),
   SUBPARTITION p8 VALUES in (5, 6)
 )
);
```

2.11.1.5 LIST-RANGE

约束与限制

- LIST分区类型要求分区定义中的value或value_list都是唯一的，不同分区定义里也需要不能重复引入。
- 对于NULL的处理，只有value中包含了NULL值，才能插入或查询出NULL值，否则是不符合定义的，不允许插入。

- 每个主分区下的子分区可以看成是一个新的一级的LIST分区表，因此所有的规则和约束都同一级的LIST是一样的, 不同主分区下的子分区定义可以不一样。

语法

创建一个或多个LIST-RANGE分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY LIST {(expr) | COLUMNS(column_list)}
   SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
   (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name VALUES IN (value_list)
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name VALUES LESS THAN {value | value_list | MAXVALUE}
```

表 2-17 参数说明

参数名称	描述
expr	分区字段表达式。目前只支持INT类型，不支持字符类型。
column_list	LIST COLUMNS的情况下使用，分区字段列表，不支持表达式。
value	分区边界值。
value_list	LIST COLUMNS的情况下使用，多个字段的边界值。
MAXVALUE	最大值。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称， 同一个表中不可重复。

使用示例

- **LIST-RANGE类型的使用示例**

```
CREATE TABLE tbl_list_range
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY LIST (col1)
SUBPARTITION BY RANGE(col2)
(
  PARTITION p0 VALUES in (1, 2)(
    SUBPARTITION s0 VALUES LESS THAN(1000),
    SUBPARTITION s1 VALUES LESS THAN(2000)
  ),
```

```
 PARTITION p1 VALUES in (3, 4)(
   SUBPARTITION s2 VALUES LESS THAN(1000),
   SUBPARTITION s3 VALUES LESS THAN(MAXVALUE)
 ),
 PARTITION p2 VALUES in (5, 6)(
   SUBPARTITION s4 VALUES LESS THAN(3000),
   SUBPARTITION s5 VALUES LESS THAN(MAXVALUE)
 )
);
```

- **LIST COLUMNS-RANGE类型的使用示例**

```
CREATE TABLE tbl_list_columns_range
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY LIST COLUMNS(col3)
SUBPARTITION BY RANGE(month(col4))
(
 PARTITION europe VALUES in ('FRANCE', 'ITALY')(
   SUBPARTITION q1_2012 VALUES LESS THAN(4),
   SUBPARTITION q2_2012 VALUES LESS THAN(7)
 ),
 PARTITION asia VALUES in ('INDIA', 'PAKISTAN')(
   SUBPARTITION q1_2013 VALUES LESS THAN(4),
   SUBPARTITION q2_2013 VALUES LESS THAN(7)
 ),
 PARTITION americas VALUES in ('US', 'CANADA')(
   SUBPARTITION q1_2014 VALUES LESS THAN(4),
   SUBPARTITION q2_2014 VALUES LESS THAN(7)
 )
);
```

2.11.1.6 LIST-LIST

语法

创建一个或多个LIST-LIST分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY LIST {(expr) | COLUMNS(column_list)}
   SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
   (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name VALUES IN (value_list)
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name VALUES IN (value_list)
```

表 2-18 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
column_list	LIST COLUMNS的情况下使用，分区字段列表，不支持表达式。
value_list	LIST COLUMNS的情况下使用，多个字段的边界值。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称， 同一个表中不可重复。

使用示例

- **LIST-LIST类型的使用示例**

```
CREATE TABLE tbl_list_list
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY LIST (col1)
SUBPARTITION BY LIST (col2)
(
 PARTITION p0 VALUES in (1, 2)(
   SUBPARTITION partno0 VALUES in (1, 2),
   SUBPARTITION partno1 VALUES in (3, 4),
   SUBPARTITION partno2 VALUES in (5, 6)
 ),
 PARTITION p1 VALUES in (3, 4)(
   SUBPARTITION partno3 VALUES in (1, 2),
   SUBPARTITION partno4 VALUES in (3, 4),
   SUBPARTITION partno5 VALUES in (5, 6)
 ),
 PARTITION p2 VALUES in (5, 6)(
   SUBPARTITION partno6 VALUES in (1, 2),
   SUBPARTITION partno7 VALUES in (3, 4),
   SUBPARTITION partno8 VALUES in (5, 6)
 )
);
```

- **LIST COLUMNS-LIST类型的使用示例**

```
CREATE TABLE tbl_list_columns_list
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY LIST COLUMNS(col3)
SUBPARTITION BY LIST (col1)
(
  PARTITION europe VALUES in ('FRANCE', 'ITALY')(
```

```
  SUBPARTITION p0 VALUES in (1, 2),
      SUBPARTITION p1 VALUES in (3, 4),
      SUBPARTITION p2 VALUES in (5, 6)
  ),
  PARTITION asia VALUES in ('INDIA', 'PAKISTAN')(
   SUBPARTITION p3 VALUES in (1, 2),
      SUBPARTITION p4 VALUES in (3, 4),
      SUBPARTITION p5 VALUES in (5, 6)
  ),
  PARTITION americas VALUES in ('US', 'CANADA')(
   SUBPARTITION p6 VALUES in (1, 2),
      SUBPARTITION p7 VALUES in (3, 4),
      SUBPARTITION p8 VALUES in (5, 6)
  )
);
```

2.11.1.7 HASH-HASH

约束与限制

- HASH类型的分区表的定义可以省略，如果指定了PARTITIONS num，默认创建num个分区定义，否则一般情况默认会创建1个分区定义。
- 对于二级分区，如果要省略分区定义，需要所有的子分区都不能给出定义，否则都要指定分区定义。

语法

创建一个或多个HASH-HASH分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] HASH(expr) [PARTITIONS num]
   SUBPARTITION BY [LINEAR] HASH(expr) [SUBPARTITIONS sub_num]
   [partition_definition [, partition_definition] ...];
```

其中，partition_definition为：

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name
```

表 2-19 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
num	用于分区是HASH或者KEY类型的分区表，来指定分区个数。
sub_num	用于二级分区是HASH或者KEY类型的分区表，来指定单个分区的子分区的个数。

参数名称	描述
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称， 同一个表中不可重复。

使用示例

HASH-HASH类型的使用示例

```
CREATE TABLE tbl_hash_hash
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY HASH(col1) PARTITIONS 9
  SUBPARTITION BY HASH(col2) SUBPARTITIONS 3;
```

2.11.1.8 HASH-KEY

约束与限制

- KEY类型的分区表的定义可以省略，如果指定了PARTITIONS num，默认创建num个分区定义，否则一般情况默认会创建1个分区定义。
- 对于二级分区，如果要省略分区定义，需要所有的子分区都不能给出定义，否则都要指定分区定义。

语法

创建一个或多个HASH-KEY分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] HASH(expr) [PARTITIONS num]
   SUBPARTITION BY [LINEAR] KEY(expr) [SUBPARTITIONS sub_num]
   (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name
```

表 2-20 参数说明

参数名称	描述
table_name	要创建的表名称。

参数名称	描述
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称，同一个表中不可重复。

使用示例

HASH-KEY类型的使用示例

```
CREATE TABLE tbl_hash_key
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY HASH(col1) PARTITIONS 3
  SUBPARTITION BY KEY(col3) SUBPARTITIONS 2;
```

2.11.1.9 HASH-RANGE

语法

创建一个或多个HASH-RANGE分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
  PARTITION BY [LINEAR] HASH(expr)
  SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
  (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

其中，subpartition_definition为：

```
SUBPARTITION subpartition_name VALUES LESS THAN {value | valuse_list | MAXVALUE}
```

表 2-21 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
column_list	LIST COLUMNS的情况下使用，分区字段列表，不支持表达式。

参数名称	描述
value	分区边界值。
value_list	LIST COLUMNS的情况下使用，多个字段的边界值。
MAXVALUE	最大值。
partition_name subpartition_name	分区名称，同一个表中不可重复。 子分区名称，同一个表中不可重复。

使用示例

HASH-RANGE类型的使用示例

```
CREATE TABLE tbl_hash_range
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY HASH(col1)
SUBPARTITION BY RANGE(col2)
(
  PARTITION p0 (
    SUBPARTITION s0 VALUES LESS THAN(4),
    SUBPARTITION s1 VALUES LESS THAN(7),
    SUBPARTITION s2 VALUES LESS THAN(10),
    SUBPARTITION s3 VALUES LESS THAN(13)
  ),
  PARTITION p1
  (
    SUBPARTITION s4 VALUES LESS THAN(4),
    SUBPARTITION s5 VALUES LESS THAN(7),
    SUBPARTITION s6 VALUES LESS THAN(10),
    SUBPARTITION s7 VALUES LESS THAN(13)
  ),
  PARTITION p2
  (
    SUBPARTITION s8 VALUES LESS THAN(4),
    SUBPARTITION s9 VALUES LESS THAN(7),
    SUBPARTITION s10 VALUES LESS THAN(10),
    SUBPARTITION s11 VALUES LESS THAN(13)
  )
);
```

2.11.1.10 HASH-LIST

语法

创建一个或多个HASH-LIST分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
```

```
PARTITION BY [LINEAR] HASH(expr)
  SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
  (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name VALUES IN (value_list)
```

表 2-22 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
column_list	LIST COLUMNS的情况下使用，分区字段列表，不支持表达式。
value_list	LIST COLUMNS的情况下使用，多个字段的边界值。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称，同一个表中不可重复。

使用示例

HASH-LIST类型的使用示例

```
CREATE TABLE tbl_hash_list
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY HASH(col1)
SUBPARTITION BY LIST(col2)
(
  PARTITION dp0 (
    SUBPARTITION p0 VALUES in (1, 2),
    SUBPARTITION p1 VALUES in (3, 4),
    SUBPARTITION p2 VALUES in (5, 6)
  ),
  PARTITION dp1
  (
    SUBPARTITION p3 VALUES in (1, 2),
    SUBPARTITION p4 VALUES in (3, 4),
    SUBPARTITION p5 VALUES in (5, 6)
  ),
  PARTITION dp2
  (
    SUBPARTITION p6 VALUES in (1, 2),
    SUBPARTITION p7 VALUES in (3, 4),
```

```
    SUBPARTITION p8 VALUES in (5, 6)
  )
);
```

2.11.1.11 KEY-HASH

语法

创建一个或多个KEY-HASH分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] KEY(expr) [PARTITIONS num]
   SUBPARTITION BY [LINEAR] HASH(expr) [SUBPARTITIONS sub_num]
   (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name
```

表 2-23 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称，同一个表中不可重复。

使用示例

KEY-HASH类型的使用示例

```
CREATE TABLE tbl_key_hash
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY KEY(col1) PARTITIONS 3
   SUBPARTITION BY HASH(col2) SUBPARTITIONS 2;
```

2.11.1.12 KEY-KEY

语法

创建一个或多个KEY-KEY分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] KEY(expr) [PARTITIONS num]
   SUBPARTITION BY [LINEAR] KEY(expr) [SUBPARTITIONS sub_num]
   (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name
```

表 2-24 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称，同一个表中不可重复。

使用示例

KEY-KEY类型的使用示例

```
CREATE TABLE tbl_key_key
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY KEY(col1) PARTITIONS 3
   SUBPARTITION BY KEY(col2) SUBPARTITIONS 2;
```

2.11.1.13 KEY-RANGE

语法

创建一个或多个KEY-RANGE分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
```

```
  PARTITION BY [LINEAR] KEY (column_list)
  SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
  (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name
  VALUES LESS THAN {value | value_list | MAXVALUE}
```

表 2-25 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
column_list	RANGE COLUMNS的情况下使用，分区字段列表，不支持表达式。
value	分区边界值。
value_list	LIST COLUMNS的情况下使用，多个字段的边界值。
MAXVALUE	最大值。
partition_name subpartition_name	分区名称，同一个表中不可重复。 子分区名称，同一个表中不可重复。

使用示例

KEY-RANGE类型的使用示例

```
CREATE TABLE tbl_key_range
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY KEY(col1)
   SUBPARTITION BY RANGE COLUMNS(col4)
(
 PARTITION  p0(
   SUBPARTITION p0_q1_2023 VALUES LESS THAN('2023-04-01'),
   SUBPARTITION p0_q2_2023 VALUES LESS THAN('2023-07-01'),
   SUBPARTITION p0_q3_2023 VALUES LESS THAN('2023-10-01'),
   SUBPARTITION p0_q4_2023 VALUES LESS THAN('2024-01-01')
 ),
 PARTITION  p1(
   SUBPARTITION p1_q1_2023 VALUES LESS THAN('2023-04-01'),
   SUBPARTITION p1_q2_2023 VALUES LESS THAN('2023-07-01'),
   SUBPARTITION p1_q3_2023 VALUES LESS THAN('2023-10-01'),
```

```
  SUBPARTITION p1_q4_2023 VALUES LESS THAN('2024-01-01')
  ),
 PARTITION  p2(
  SUBPARTITION p2_q1_2023 VALUES LESS THAN('2023-04-01'),
  SUBPARTITION p2_q2_2023 VALUES LESS THAN('2023-07-01'),
  SUBPARTITION p2_q3_2023 VALUES LESS THAN('2023-10-01'),
  SUBPARTITION p2_q4_2023 VALUES LESS THAN('2024-01-01')
 )
);
```

2.11.1.14 KEY-LIST

语法

创建一个或多个KEY-LIST分区表，其中每个分区可能有一个或一个以上的子分区。

```
CREATE TABLE [ schema. ]table_name
 table_definition
 PARTITION BY [LINEAR] KEY(expr)
  SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
  (partition_definition [, partition_definition] ...);
```

其中，partition_definition为：

```
PARTITION partition_name
   (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition为：

```
SUBPARTITION subpartition_name VALUES IN (value_list)
```

表 2-26 参数说明

参数名称	描述
table_name	要创建的表名称。
expr	分区字段表达式，目前只支持INT类型，不支持字符类型。
column_list	LIST COLUMNS的情况下使用，分区字段列表，不支持表达式。
value_list	字段的值。
partition_name	分区名称，同一个表中不可重复。
subpartition_name	子分区名称，同一个表中不可重复。

使用示例

KEY-LIST类型的使用示例

```
CREATE TABLE tbl_key_list
(
   col1 INT,
   col2 INT,
```

```
  col3 varchar(20),
  col4 DATE
)
PARTITION BY KEY(col1)
SUBPARTITION BY LIST(col2)
(
 PARTITION dp0 (
  SUBPARTITION p0 VALUES in (1, 2),
  SUBPARTITION p1 VALUES in (3, 4),
  SUBPARTITION p2 VALUES in (5, 6)
 ),
 PARTITION dp1
 (
  SUBPARTITION p3 VALUES in (1, 2),
  SUBPARTITION p4 VALUES in (3, 4),
  SUBPARTITION p5 VALUES in (5, 6)
 ),
 PARTITION dp2
 (
  SUBPARTITION p6 VALUES in (1, 2),
  SUBPARTITION p7 VALUES in (3, 4),
  SUBPARTITION p8 VALUES in (5, 6)
 )
);
```

2.11.2 LIST DEFAULT HASH

LIST DEFAULT HASH是在同一级别支持两种分区类型：LIST和HASH。前面是普通的LIST分区，不符合LIST分区规则的数据会放在DEFAULT分区里，DEFAULT分区如果有多个分区则根据HASH规则计算。LIST DEFAULT HASH分区类型常用在LIST VALUES分布不均匀以及无法全部枚举的场景。

前提条件

- GaussDB(for MySQL)的内核版本大于等于2.0.54.240600。
- 设置特性开关rds_list_default_partition_enabled的值为ON。

使用限制

- 一级DEFAULT PARTITION支持一个或多个DEFAULT分区。
- 支持二级分区LIST+DEFAULT，但每个PARTITION仅支持一个二级DEFAULT分区。
- 一级分区一个DEFAULT分区的情况下，支持所有类型的二级分区。
- 一级分区多个DEFAULT分区的情况下，仅支持HASH或KEY二级分区。

参数说明

在参数配置页面通过设置参数rds_list_default_partition_enabled的值来启用或禁用LIST DEFAULT HASH功能。

表 2-27 参数说明

参数名称	级别	说明
rds_list_default_partition_enabled	Global	LIST DEFAULT HASH功能控制开关。 取值范围如下： • ON: 启用LIST DEFAULT HASH功能。 • OFF: 关闭LIST DEFAULT HASH功能。

创建 LIST DEFAULT HASH 分区表

- 语法

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY LIST [COLUMNS] (expr)
   SUBPARTITION BY ...
   (list_partition_definition[, ..., list_partition_definition],
    default_partition_definition
   )
```

其中，default_partition_definition为：

```
PARTITION partition_name DEFAULT [PARTITIONS number]
```

每个分区的定义也可以包含二级分区， 二级分区也支持使用LIST DEFAULT分区，定义如下：

```
SUBPARTITION subpartition_name DEFAULT
```

表 2-28 参数说明

参数名称	参数说明
table_name	要创建的表名称。
partition_name	• 只有一个DEFAULT分区时，表示分区名称。不可与其他分区表重复。 • 当有多个DEFAULT分区时，表示分区名称前缀。“partition_name+序号”表示分区名称。
subpartition_name	子分区名称。同一个表中不可重复，子分区最多只支持一个DEFAULT分区。
number	DEFAULT分区按照哈希规则分成number个分区，通过number指定分区个数。PARTITIONS number是可选项，不指定时，则默认为一个DEFAULT分区。

- 示例

创建单个DEFAULT分区示例如下：

```
CREATE TABLE list_default_tbl (
 a INT,
 b INT
```

```
)
PARTITION BY LIST (a)
(PARTITION p0 VALUES IN (1,2,3,4,5),
 PARTITION p1 VALUES IN (6,7,8,9,10),
 PARTITION pd DEFAULT);
```

创建多个DEFAULT分区示例如下：

```
CREATE TABLE list_default_hash (
  a INT,
  b INT
)
PARTITION BY LIST (a)
(PARTITION p0 VALUES IN (1,2,3,4,5),
 PARTITION p1 VALUES IN (6,7,8,9,10),
 PARTITION pd DEFAULT PARTITIONS 3);
```

使用LIST COLUMNS示例如下：

```
CREATE TABLE t_goods
(
  country   VARCHAR(30),
  year      VARCHAR(60),
  goods     TEXT
) PARTITION BY LIST COLUMNS(country)
(
  PARTITION p1 VALUES IN ('China'),
  PARTITION p2 VALUES IN ('USA'),
  PARTITION p3 VALUES IN ('Asia'),
  PARTITION p3 VALUES IN ('India'),
  PARTITION p_deft DEFAULT PARTITIONS 5
);
```

通过explain查看分区：

```
EXPLAIN SELECT * FROM list_default_hash;
```

显示结果如下：

```
+----+-------------+-------------------+-------------------+------+---------------+------+---------+------+------
+----------+-------+
| id | select_type | table             | partitions        | type | possible_keys | key  | key_len | ref  | rows |
filtered | Extra |
+----+-------------+-------------------+-------------------+------+---------------+------+---------+------+------
+----------+-------+
|  1 | SIMPLE      | list_default_hash | p0,p1,pd0,pd1,pd2 | ALL  | NULL          | NULL | NULL    | NULL |
1 |   100.00 | NULL  |
+----+-------------+-------------------+-------------------+------+---------------+------+---------+------+------
+----------+-------+
1 row in set (0.04 sec)
```

二级分区支持LIST DEFAULT类型，示例如下：

```
CREATE TABLE test (a int, b int)
PARTITION BY RANGE(a)
SUBPARTITION BY LIST(b) (
PARTITION part0 VALUES LESS THAN (10)
( SUBPARTITION sub0 VALUES IN (1,2,3,4,5),
  SUBPARTITION sub1 DEFAULT),
PARTITION part1 VALUES LESS THAN (20)
( SUBPARTITION sub2 VALUES IN (1,2,3,4,5),
  SUBPARTITION sub3 DEFAULT),
PARTITION part2 VALUES LESS THAN (30)
( SUBPARTITION sub4 VALUES IN (1,2,3,4,5),
  SUBPARTITION sub5 DEFAULT));
```

一级分区存在多个LIST DEFAULT HASH分区的情况下，仅支持HASH或KEY二级分区：


```
CREATE TABLE list_default_hash_sub (
  a INT,
  b INT
)
PARTITION BY LIST (a)
SUBPARTITION BY HASH (b) SUBPARTITIONS 20
(PARTITION p0 VALUES IN (1,2,3,4,5),
 PARTITION p1 VALUES IN (6,7,8,9,10),
 PARTITION pd DEFAULT PARTITIONS 3);
```

修改 LIST DEFAULT HASH 分区表

LIST DEFAULT HASH分区支持所有修改分区表的语法，包括ALTER TABLE ADD PARTITION、ALTER TABLE DROP PARTITION、ALTER TABLE REORGANIZE PARTITION、ALTER TABLE TRUNCATE PARTITION、ALTER TABLE EXCHANGE PARTITION、ALTER TABLE OPTIMIZE PARTITIONALTER TABLE REBUILD PARTITION、ALTER TABLE REPAIR PARTITION、ALTER TABLE ANALYZE PARTITION、ALTER TABLE CHECK PARTITION操作。除了ALTER TABLE ADD PARTITION，ALTER TABLE DROP PARTITION，ALTER TABLE REORGANIZE PARTITION有特殊的使用方法和限制，其他的语法同其他类型的分区表使用方法一样。

- ALTER TABLE ADD PARTITION
  - ADD DEFAULT PARTITION

    对于没有DEFAULT分区的普通LIST分区表，通过ADD PARTITION增加DEFAULT分区，使之变成LIST DEFAULT HASH分区表。

    ```
    ALTER TABLE table_name ADD PARTITION(default_partition_definition)
    ```

    增加一个DEFAULT分区示例如下：

    ```
    CREATE TABLE list_tab (
      a INT,
      b INT
    )
    PARTITION BY LIST (a)
    (PARTITION p0 VALUES IN (1,2,3,4,5),
     PARTITION p1 VALUES IN (6,7,8,9,10)
    );
    ALTER TABLE list_tab ADD PARTITION(PARTITION pd DEFAULT);
    ```

    增加两个DEFAULT分区示例如下：

    ```
    CREATE TABLE list_tab (
      a INT,
      b INT
    )
    PARTITION BY LIST (a)
    (PARTITION p0 VALUES IN (1,2,3,4,5),
     PARTITION p1 VALUES IN (6,7,8,9,10)
    );
    ALTER TABLE list_tab ADD PARTITION(PARTITION pd DEFAULT PARTITIONS 2);
    ```

  - ADD LIST PARTITION

    LIST DEFAULT HASH分区表ALTER TABLE ADD PARTITION语法支持使用WITHOUT VALIDATION选项添加LIST分区。

    ```
    ALTER TABLE table_name ADD PARTITION(
      list_partition_definition[, ..., list_partition_definition])
    WITHOUT VALIDATION
    ```

    新增一个LIST分区的示例如下：

```
CREATE TABLE list_default_hash (
  a INT,
  b INT
)
PARTITION BY LIST (a)
(PARTITION p0 VALUES IN (1,2,3,4,5),
 PARTITION p1 VALUES IN (6,7,8,9,10),
 PARTITION pd DEFAULT PARTITIONS 3);

ALTER TABLE list_default_hash ADD PARTITION(
  PARTITION p2 VALUES IN (11,12,13)
)WITHOUT VALIDATION;
```

执行后，list_default_hash表会增加一个LIST分区p2，p2中没有数据。

说明

一旦使用了without validation添加list分区，需要您手动执行`ALTER TABLE .. REBUILD ALL`命令，重新分配数据。否则数据不会重新分配，满足新添加的分区定义的数据仍存放在DEFAULT分区中，在查询时候会把default分区全部标记，不会裁剪掉，会导致查询性能下降。建议您使用ALTER TABLE REORGANIZE PARTITION语法，从DEFAULT分区中分离部分数据，建立新的LIST分区。

- ALTER TABLE DROP PARTITION

  DROP PARTITION操作时，只能一次性删除全部DEFAULT分区，不支持只删除部分DEFAULT分区。

  执行DROP PARTITION操作，删除所有分区的示例如下：

  ```
  ALTER TABLE list_default_hash DROP PARTITION pd0,pd1,pd2;
  Query OK, 0 rows affected (0.33 sec)
  Records: 0  Duplicates: 0  Warnings: 0
  ```

  说明

  单独删除部分DEFAULT分区时会报错。

  ```
  ALTER TABLE list_default_hash DROP PARTITION pd0;
  ```

  报错信息如下：

  ```
  ERROR 8078 (HY000): DROP PARTITION cannot be used on default partitions of LIST DEFAULT, except once dropping all default partitions
  ```

- ALTER TABLE REORGANIZE PARTITION

  REORGANIZE PARTITION操作时，只能一次性修改全部DEFAULT分区，不支持只修改部分DEFAULT分区。

  - 使用REORGANIZE PARTITION操作可以改变DEFAULT分区的个数：

    ```
    ALTER TABLE list_default_hash
    REORGANIZE PARTITION
      pd0,pd1
    INTO(
      PARTITION pd DEFAULT PARTITIONS 3);
    ```

    执行后，DEFAULT分区的个数会由2个变成3个。

  - 使用REORGANIZE PARTITION可以从DEFAULT分区中分离出一个LIST分区：

    ```
    ALTER TABLE list_default_hash
    REORGANIZE PARTITION
      pd0,pd1
    INTO (
     PARTITION p2 VALUES IN (20,21),
     PARTITION pd DEFAULT PARTITIONS 2);
    ```

    执行后，list_default_hash分区表会增加一个LIST分区p2， p2中会有从DEFAULT分区中分离出来的符合VALUES IN (20,21)的数据。

- 使用REORGANIZE PARTITION可以合并一个LIST分区到DEFAULT分区

```
ALTER TABLE list_default_hash
REORGANIZE PARTITION
  p2, pd0, pd1
INTO (
 PARTITION pd DEFAULT PARTITIONS 2);
```

执行后，LIST分区p2会合并到DEFAULT分区里。

- 使用REORGANIZE PARTITION可以从DEFAULT分区中分离部分values放到LIST分区：

```
ALTER TABLE list_default
REORGANIZE partition
  p2, pd0, pd1
INTO (
  PARTITION p2 VALUES IN (20,21,22,23,24),
  PARTITION pd DEFAULT PARTITIONS 4);
```

执行后，p2的定义由PARTITION p2 VALUES IN (20,21)变为PARTITION p2 VALUES IN (20,21,22,23,24)， 相应的数据也会从DEFAULT分区挪到p2。

2.11.3 INTERVAL RANGE

INTERVAL RANGE分区表是RANGE分区表的扩展，向RANGE分区表插入数据时，如果插入的数据超出当前已存在分区的范围，将无法插入并且会返回错误；

而对于INTERVAL RANGE分区表，当新插入的数据超过现有分区的范围时，允许数据库根据INTERVAL子句提前指定的规则来添加新分区。

前提条件

- GaussDB(for MySQL)的内核版本大于等于2.0.54.240600。
- 设置特性开关rds_interval_range_enabled的值为ON。

使用限制

- INTERVAL RANGE表只支持HASH/KEY二级分区。
- 采取RANGE COLUMNS(column_list) INTERVAL([type], value)形式描述INTERVAL RANGHE规则时：
  - column_list（分区键）中列个数只能为1，并且只能是是整数类型或者DATE/TIME/DATETIME类型。
  - 如果分区键是整型，INTERVAL的间隔类型type不需要填写。
  - 如果分区键为DATE类型，INTERVAL的间隔类型type只能取YEAR、QUARTER、MONTH、WEEK、DAY。
  - 如果分区键为TIME类型，INTERVAL的间隔类型type只能取HOUR、MINUTE、SECOND。
  - 如果分区间为DATETIME类型，INTERVAL的间隔类型type可以取YEAR、QUARTER、MONTH、WEEK、DAY、HOUR、MINUTE、SECOND。
  - 间隔值value只能为正整数。
  - 如果INTERVAL的间隔类型是SECOND，间隔不能小于60。
- 采取RANGE(expr) INTERVAL(value)形式描述INTERVAL RANGHE规则时，expr表达式的结果应为整数，间隔值value只能为正整数。
- 不支持使用INSERT ...SELECT语句、INSERT ...ON DUPLICATE KEY UPDATE语句、UPDATE语句触发分区新增。

- LOAD DATA时不会触发分区新增（如果分区覆盖了所有插入数据的范围，能使用load data导入数据，如果分区没有覆盖插入数据的范围，load data无法触发自增分区，导入数据失败）。
- 事务中如果触发分区自增，一旦新分区创建成功，不支持回滚。
- 自增的分区使用'_p'作为分区名的前缀，因此客户设置的以此开头的分区名可能导致分区自增失败。
- SET INTERVAL([type], value)子句只适用于INTERVAL RANGE表和RANGE表，如果这两种表有二级分区，只支持二级分区为HASH或KEY类型。
- SET INTERVAL([type], value)子句的type和value取值要受原表的分区表达式expr或分区键column_list的约束。

参数说明

表 2-29 参数说明

参数名称	级别	参数说明
rds_interval_range_enabled	Global	INTERVAL RANGE功能控制开关。 取值范围如下： ● ON：启用INTERVAL RANGE功能。 ● OFF：关闭INTERVAL RANGE功能。

创建 INTERVAL RANGE 分区表

INTERVAL RANGE分区表定义格式类似于RANGE分区表，但多了INTERVAL子句。

语法：

```
CREATE TABLE [IF NOT EXISTS] [schema.]table_name
table_definition
partition_options;
```

其中，partition_options为：

```
PARTITION BY
    RANGE {(expr) | COLUMNS(column_list)}
    {INTERVAL(value) | INTERVAL(type, expr)}
    (partition_definition [, partition_definition] ...)
```

partition_definition为：

```
PARTITION partition_name
    [VALUES LESS THAN {expr | MAXVALUE}]
    [[STORAGE] ENGINE [=] engine_name]
    [COMMENT [=] 'string' ]
    [DATA DIRECTORY [=] 'data_dir']
    [INDEX DIRECTORY [=] 'index_dir']
    [MAX_ROWS [=] max_number_of_rows]
    [MIN_ROWS [=] min_number_of_rows]
    [TABLESPACE [=] tablespace_name]
```

其中，INTERVAL子句仅支持设置间隔数值（value）和间隔类型（type）。

INTERVAL子句关联参数说明：

表 2-30 参数说明

参数名称	参数说明
INTERVAL(value)	使用RANGE(expr) 或者 RANGE COLUMNS(column_list)且column是整型字段时，INTERVAL子句的格式，其中value代表间隔数值，必须是正整数。
expr	RANGE(expr)中的分区表达式，目前只支持整数类型。
column_list	RANGE COLUMNS(column_list)的分区字段列表，在INTERVAL RANGE分区表中，column_list只能是单列。
INTERVAL(type, value)	使用RANGE COLUMNS(column_list)且column_list是DATE/TIME/DATETIME类型时，INTERVAL子句的格式，其中type代表间隔类型，value代表间隔数值。 type目前支持8种时间类型（YEAR、QUARTER、MONTH、WEEK、DAY、HOUR、MINUTE、SECOND）。 value代表间隔数值，必须是正整数；当type为SECOND类型时，间隔不能小于60。

间隔数值（value）和间隔类型（type）的进一步说明：

- 间隔数值（expr）

  新增记录对应的分区表达式值expr或者整型分区字段column_list，连续的1000个数字进入同一个分区。

  示例如下：

  ```
  INTERVAL(1000)
  ```

- 时间类型
  - 年（YEAR）

    以年为单位设置自动分区的间隔，每一年的数据进入同一个分区。

    示例如下：

    ```
    INTERVAL(YEAR, 1)
    ```

  - 季度（QUARTER）

    以季度为单位设置自动分区的间隔，每一季度的数据进入同一个分区。

    示例如下：

    ```
    INTERVAL(QUARTER, 1)
    ```

  - 月（MONTH）

    以月为单位设置自动分区的间隔，每一月的数据进入同一个分区。

    示例如下：

    ```
    INTERVAL(MONTH, 1)
    ```

  - 周（WEEK）

    以周为单位设置自动分区的间隔，每一周的数据进入同一个分区，示例如下：

```
INTERVAL(WEEK, 1)
```

- 日（DAY）

  以日为单位设置自动分区的间隔，每一日的数据进入同一个分区。

  示例如下：

  ```
  INTERVAL(DAY, 1)
  ```

- 时（HOUR）以小时为单位设置自动分区的间隔，每一小时的数据进入同一个分区。

  示例如下：

  ```
  INTERVAL(HOUR, 1)
  ```

- 分（MINUTE）以分钟为单位设置自动分区的间隔，每一分钟的数据进入同一个分区。

  示例如下：

  ```
  INTERVAL(MINUTE, 1)
  ```

- 秒（SECOND）以秒为单位设置自动分区的间隔，每60秒的数据进入同一个分区。

  示例如下：

  ```
  INTERVAL(SECOND, 60)
  ```

以下示例将order_time作为分区键，按间隔划分sales表。

在数据库中创建一个新的INTERVAL RANGE分区表，并向表中插入数据，示例如下：

```
CREATE TABLE sales
(
  id BIGINT,
  uid BIGINT,
  order_time DATETIME
)
PARTITION BY RANGE COLUMNS(order_time) INTERVAL(MONTH, 1)
(
  PARTITION p0 VALUES LESS THAN('2021-9-1')
);
```

向INTERVAL RANGE分区表中插入数据，示例如下：

```
INSERT INTO sales VALUES(1, 1010101010, '2021-11-11');
```

插入数据后，SHOW CREATE TABLE查询sales表定义。新的表定义如下：

```
CREATE TABLE `sales` (
  `id` bigint DEFAULT NULL,
  `uid` bigint DEFAULT NULL,
  `order_time` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
/*!50500 PARTITION BY RANGE  COLUMNS(order_time) */ /*!99990 800220201
INTERVAL(MONTH, 1) */
/*!50500 (PARTITION p0 VALUES LESS THAN ('2021-9-1') ENGINE = InnoDB,
 PARTITION _p20211001000000 VALUES LESS THAN ('2021-10-01 00:00:00') ENGINE = InnoDB,
 PARTITION _p20211101000000 VALUES LESS THAN ('2021-11-01 00:00:00') ENGINE = InnoDB,
 PARTITION _p20211201000000 VALUES LESS THAN ('2021-12-01 00:00:00') ENGINE = InnoDB)
*/
```

通过上述示例发现INTERVAL RANGE分区自动新增加了_p20211001000000、_p20211101000000、_p20211201000000三个分区，这里要**注意‘_p’作为前缀的分区名将会保留为系统命名规则，手动管理分区（创建新分区或者重命名分区的操作）时，不建议使用‘_p’作为前缀的分区名**。

INTERVAL RANGE表支持HASH或KEY类型类型的二级分区，例如：

```
CREATE TABLE sales_ir_key
(
  dept_no     INT,
  part_no     INT,
  country     varchar(20),
  date        DATE,
  amount      INT
)
PARTITION BY RANGE(month(date)) INTERVAL(1)
SUBPARTITION BY KEY(date) SUBPARTITIONS 2
(
  PARTITION q1_2012 VALUES LESS THAN(4)
    (SUBPARTITION sp_001,
     SUBPARTITION sp_002),
  PARTITION q2_2012 VALUES LESS THAN(7)
    (SUBPARTITION sp_003,
     SUBPARTITION sp_004)
);

CREATE TABLE sales_ir_hash
(
  dept_no     INT,
  part_no     INT,
  country     varchar(20),
  date        DATE,
  amount      INT
)
PARTITION BY RANGE COLUMNS(date) INTERVAL(YEAR, 1)
SUBPARTITION BY HASH(TO_DAYS(date)) SUBPARTITIONS 2
(
  PARTITION q1_2012 VALUES LESS THAN('2021-01-01')
    (SUBPARTITION sp_001,
     SUBPARTITION sp_002),
  PARTITION q2_2012 VALUES LESS THAN('2022-01-01')
    (SUBPARTITION sp_003,
     SUBPARTITION sp_004)
);
```

INTERVAL RANGE 分区表与任意类型表的相互转换

语法：

其他类型表转化为INTERVAL RANGE分区表。

```
ALTER TABLE table_name table_definition
partition_options;

partition_options:
    PARTITION BY
    { RANGE{(expr) | COLUMNS(column_list)} }
    { INTERVAL(type, value) | INTERVAL(value) }
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
        [VALUES LESS THAN {expr | MAXVALUE}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'string' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
```



```
    [MIN_ROWS [=] min_number_of_rows]
    [TABLESPACE [=] tablespace_name]
```

INTERVAL子句关联参数说明：

表 2-31 参数说明

参数名称	参数说明
INTERVAL(value)	使用RANGE(expr) 或者 RANGE COLUMNS(column_list)且column是整型字段时，INTERVAL子句的格式，其中value代表间隔数值，必须是正整数。
expr	RANGE(expr)中的分区表达式，目前只支持整数类型。
column_list	RANGE COLUMNS(column_list)的分区字段列表，在INTERVAL RANGE分区表中，column_list只能是单列。
INTERVAL(type, value)	使用RANGE COLUMNS(column_list)且column_list是DATE/TIME/DATETIME类型时，INTERVAL子句的格式，其中type代表间隔类型，value代表间隔数值。 type目前支持8种时间类型（YEAR、QUARTER、MONTH、WEEK、DAY、HOUR、MINUTE、SECOND）。 value代表间隔数值，必须是正整数；当type为SECOND类型时，间隔不能小于60。

INTERVAL RANGE分区表转化为其他任意类型的表，这里partition_options是可选的，具体定义取决于要转化的目标表类型。

```
ALTER TABLE table_name table_definition
[partition_options];
```

partition_options选项取决于要转化为表的分区类型信息。

示例：

将其他类型表转为INTERVAL RANGE表：

```
CREATE TABLE orders(
  orderkey BIGINT NOT NULL,
  custkey BIGINT NOT NULL,
  orderdate DATE NOT NULL
);

ALTER TABLE orders
PARTITION BY RANGE COLUMNS(orderdate) INTERVAL(MONTH, 1) (
  PARTITION p0 VALUES LESS THAN('2021-10-01')
);
```

将INTERVAL RANGE表转化为其他类型表：

```
CREATE TABLE orders (a INT, b DATETIME)
PARTITION BY RANGE (a) INTERVAL(10)
(
    PARTITION p0 VALUES LESS THAN(10),
    PARTITION p2 VALUES LESS THAN(20)
);
```

```
ALTER TABLE orders PARTITION BY LIST COLUMNS (a)
(
   PARTITION p0 VALUES IN (1, 11, 25)
);
```

修改INTERVAL RANGE表的INTERVAL子句信息：

```
CREATE TABLE orders (a INT, b DATETIME)
PARTITION BY RANGE (a) INTERVAL(10)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);

ALTER TABLE orders PARTITION BY RANGE (a) INTERVAL(20)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);

# 消除INTERVAL子句
ALTER TABLE orders PARTITION BY RANGE (a)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);

# 添加INTERVAL子句
ALTER TABLE orders PARTITION BY RANGE (a) INTERVAL(100)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);
```

SET INTERVAL 子句支持

支持使用SET INTERVAL子句修改INTERVAL RANGE表定义的INTERVAL子句间隔类型和间隔值，也可实现消除或添加INTERVAL子句。

语法：

```
ALTER TABLE table_name SET INTERVAL {() | (type, value) | (value)};
```

表 2-32 参数说明

参数名称	参数说明
type	目前支持8种时间类型（YEAR、QUARTER、MONTH、WEEK、DAY、HOUR、MINUTE、SECOND），不显式指定默认是数字类型的间隔。
value	指定间隔的数值大小。当type为SECOND类型时，间隔不能小于60。

示例：

修改INTERVAL RANGE表的INTERVAL类型和值。

```
CREATE TABLE orders(
  orderkey BIGINT NOT NULL,
  custkey BIGINT NOT NULL,
  orderdate DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(orderdate) INTERVAL(MONTH, 1) (
  PARTITION p0 VALUES LESS THAN('2021-10-01')
);

ALTER TABLE orders SET INTERVAL(YEAR, 1);
```

实现RANGE表和INTERVAL RANGE表之间的转化。

```
CREATE TABLE orders(
  orderkey BIGINT NOT NULL,
  custkey BIGINT NOT NULL,
  orderdate DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(orderdate) INTERVAL(MONTH, 1) (
  PARTITION p0 VALUES LESS THAN('2021-10-01')
);

# 删除INTERVAL子句
ALTER TABLE sales SET INTERVAL();

# 添加INTERVAL子句
ALTER TABLE sales SET INTERVAL(DAY, 60);
```

⚠ 注意

即使当前功能开关rds_interval_range_enabled关闭，"ALTER TABLE table_name SET INTERVAL()"语句也可使用（用于消除INTERVAL RANGE表的INTERVAL子句定义信息，转化为普通RANGE表）。

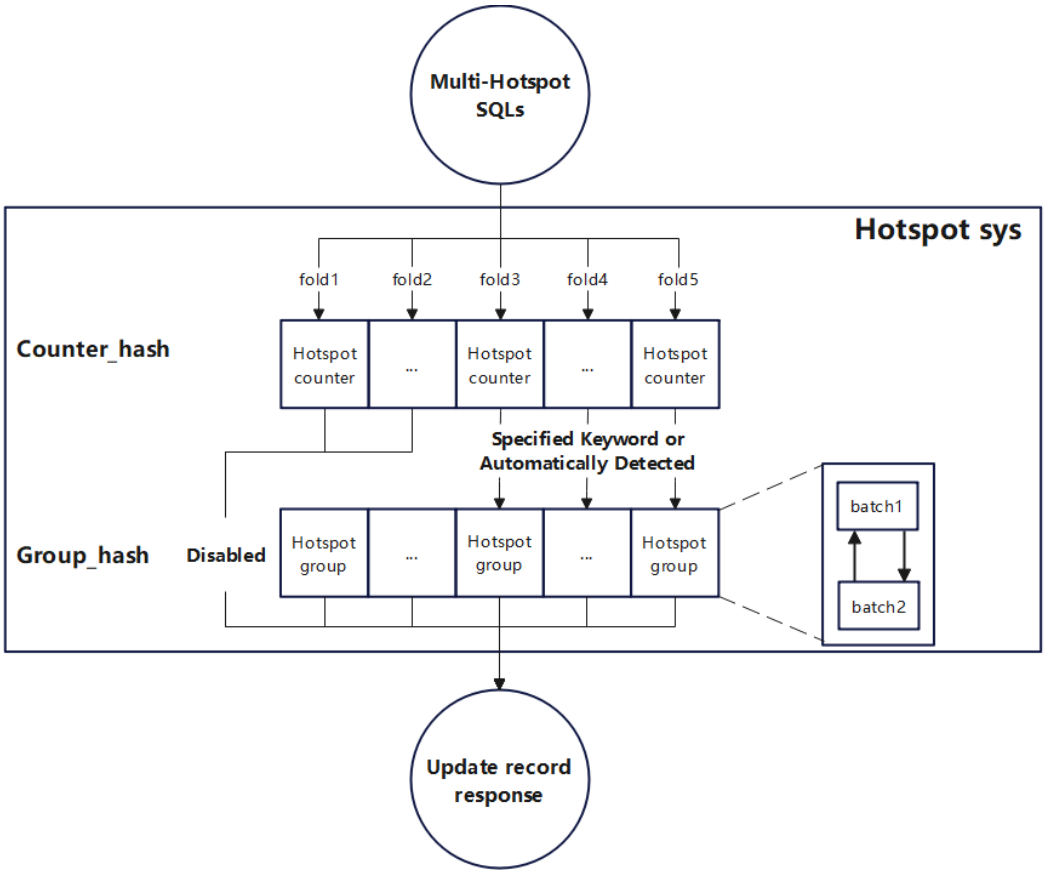
2.12 热点行更新

数据库中频繁被增删改查的记录被称为热点行。热点行出现的场景包括秒杀抢购、演唱会门票预订、热门路线火车票预定等等。由于事务对数据更新时，需要持有行锁，同一时刻内只能有一个事务更新热点行，其他事务只能等待行锁释放后继续执行，热点行更新的性能存在瓶颈，并且传统的分库分表策略在性能提升方面并不会有太大帮助。

云数据库 GaussDB(for MySQL)支持热点行更新优化，您可以通过手动指定或者自动识别的方式开启热点行更新。热点行更新开启后可以大幅度提升热点行的更新性能。

原理介绍

GaussDB(for MySQL)热点行更新的架构如下图所示：分为Counter_hash，Group_hash两部分。其中Counter_hash主要用于实现热点行的自动判断。Group_hash是热点行的实际实现部分，由多个Hotspot group组成。每个Hotspot group对应一个热点行，每个Hotspot group由多个batch组成，交替为热点行提供批量提交服务。



约束与限制

- GaussDB(for MySQL)实例的内核版本为2.0.54.240600及以上时支持使用该功能。
- 功能使用约束如下：
  - where条件中只能使用主键或唯一索引的等值匹配，并且只能更新单条记录。否则将绕过优化正常更新。
  - 不允许修改索引列，否则将绕过优化正常更新。
  - 只对修改列为整数的更新生效，否则将绕过优化正常更新。
  - 只允许对热点行记录进行两个元素的加减操作，且第一个元素与等号左侧相等并满足唯一索引等约束，不允许赋值操作。假设c列为待修改列，d为记录的普通列，那么只允许进行类似c=c+1，或者c=c-1的操作，不允许进行c=d+1，c=1+c，c=c+1+1，c=1+c+1等操作。否则将绕过优化正常更新。
  - 只允许对隐式事务生效。即要求AUTOCOMMIT为ON，并且不在BEGIN，COMMIT显示开启的事务中使用。否则将绕过优化正常更新。
  - 需要使用HOTSPOT显式标记热点行更新事务，或者将rds_hotspot_auto_detection_threshold设置为非0，开启热点行更新自动识别功能。否则将绕过优化正常更新。rds_hotspot_auto_detection_threshold的详细用法请见参数说明。
  - 只对RC级别生效。数据库处于其他隔离级别时将绕过优化正常更新。
  - 无法在stored function, trigger以及event中使用，否则将对客户端报如下错误：

    ```
    HOTSPOT hints can not be used in stored function, trigger or event
    ```

- 行为变更： 一个hotspot事务组内，除了执行失败或者在更新阶段killed的事务外，其他事务被按批次集中提交，集中记录redo log和undo log，只能集中提交或者回滚，无法单独回滚。每个批次提交的事务个数为几十到几百个不等。

参数说明

表 2-33 参数说明

参数名称	参数说明
rds_hotspot	热点行更新优化开关。将其设置为ON将启用热点行更新优化。
rds_hotspot_follower_wait_commit_interval	热点行更新follower事务等待leader事务日志持久化时，进入阻塞前的睡眠时间。单位：微秒。对日志持久化速度慢的实例，建议调大。对于持久化快速的实例，建议设置为0，不休眠直接阻塞。
rds_hotspot_leader_wait_follower_interval	热点行更新leader事务等待follower更新记录的时间单位。单位：微秒。低并发适当调低可以避免性能下降。高并发适当调高可以提升性能。当QPS超过20w时，建议将该值设置为100或者更大。
rds_hotspot_auto_detection_threshold	热点行更新的自动识别功能开关。设置为0表示不启用自动识别功能。设置为非0表示热点行更新的识别阈值。当某个符合热点行要求的行每秒更新次数超过阈值时将启动热点行更新功能。
rds_hotspot_batch_size_lower_limit	每批热点事务大小的建议最小值。每个batch尽可能达到该大小。但是，这并不是严格保证的。当leader发现所有需要等待的follower都已经到达时，batch就进入提交状态。
rds_hotspot_max_memory_size	热点行更新中group和counter占用的内存上限。当group占用的内存超过限制时，将清空group所占用的内存。当counter占用的内存超过限制时，将清空counter所占用的内存。申请新的内存时才会尝试清空旧内存。
rds_hotspot_enable_time_statistics	是否开启热点行更新的时间相关的状态统计功能。将其设置为ON以启用该功能。

状态说明

表 2-34 状态说明

状态	说明
Hotspot_total_trx	使用hotspot总事务数

状态	说明
Hotspot_update_errors	更新阶段出错的热点行更新事务，这些出错的事务只会自己更新失败，不会影响其他热点行更新事务的提交。
Hotspot_trx_rollbacked	更新成功，但是由于最终回滚的热点行更新事务数量。当队长（leader）决定回滚时，所有组员（follower）跟着一起回滚。
Hotspot_trx_committed	提交成功的热点行更新事务数量。
Hotspot_batch_size	热点行更新事务分批次提交。该值表示当前批次热点行更新事务的数量。
Hotspot_batch_wait_time	热点行更新按批次持有锁和提交事务。此时间为当前热点行更新批次等待上一批次释放锁的时间，单位微秒。
Hotspot_leader_wait_follower_time	在一个批次中leader需要等待follower完成记录更新，此时间为当前批次leader等待follower的时间，单位微秒。
Hotspot_leader_total_time	当前批次leader的热点行更新事务总时间，单位微秒。
Hotspot_follower_total_time	当前批次某一个follower的热点行更新事务总时间，单位微秒。
Hotspot_follower_wait_commit_time	在一个批次中follower需要等待leader持久化日志，此时间为当前批次某一个follower等待leader持久化日志的时间，单位微秒。
Hotspot_group_counts	每个热点行更新对应一个组，组内事务分批次提交。该值为使用热点行更新的组数。
Hotspot_counter_counts	counter用于自动判断热点行更新。当counter中的统计值满足要求时，将会创建group使用热点行更新。该值为counter的总数。

新增关键字

新增标记语句的关键字如下：

表 2-35 新增关键字

关键字	描述
HOTSPOT	表示开启热点更新功能。
NOT_MORE_THAN	可选项。表示目标值不大于某值。

关键字	描述
NOT_LESS_THAN	可选项。表示目标值不小于某值。

上述关键字放置在SQL语句末尾。HOTSPOT必须在最前面，NOT_MORE_THAN和NOT_LESS_THAN没有位置前后的要求。

例如：假设id是主键列，c是int类型列，那么支持以下语法：

```
UPDATE c=c+1 where id=10 HOTSPOT;
UPDATE c=c+1 where id=10 HOTSPOT NOT_MORE_THAN 100; // c值不大于100
UPDATE c=c-1 where id=10 HOTSPOT NOT_LESS_THAN 0;  // c值不小于0
UPDATE c=c+1 where id=10 HOTSPOT NOT_MORE_THAN 100 NOT_LESS_THAN 0; // c值不大于100，不小于0
UPDATE c=c+1 where id=10 HOTSPOT NOT_LESS_THAN 0 NOT_MORE_THAN 100; // c值不大于100，不小于0
```

当超过NOT_MORE_THAN或者NOT_LESS_THAN的限制时，会向客户端报如下错误：

```
HOTSPOT field value exceeds limit
```

使用示例

1. 创建表，准备数据。
   ```
   CREATE TABLE test.hotspot1 (
     `id` int NOT NULL primary key,
     `c` int NOT NULL DEFAULT '0'
   ) ENGINE=InnoDB;
   INSERT INTO test.hotspot1 VALUES (1, 1);
   ```
2. 打开热点行更新开关。
   ```
   SET GLOBAL rds_hotspot = ON;
   ```
3. 修改隔离级别，AUTOCOMMIT。
   ```
   SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
   SET SESSION AUTOCOMMIT = ON;
   ```
4. 发起带HOTSPOT关键字的更新。
   ```
   UPDATE test.hotspot1 SET c=c+1 WHERE id=1 HOTSPOT;
   ```
5. 检查热点行更新状态。
   ```
   SHOW STATUS like "%hotspot%";
   ```
   性能测试
   - 测试环境

     实例规格：8U32GB、32U128GB

     ECS规格：32U64GB

     测试环境：华北-北京四

     测试工具：sysbench-1.0.18

     数据模型：
     - 1张表，1条数据。
     - 8张表，每张表1条数据。
   - 参数配置：

     rds_hotspot=ON

transaction_isolation=READ-COMMITTED

max_prepared_stmt_count=1048576

rds_global_sql_log_bin=OFF

- 测试方法

  测试所需数据表定义：

  CREATE TABLE sbtest (id int NOT NULL AUTO_INCREMENT,k int NOT NULL DEFAULT '0',PRIMARY KEY (id));

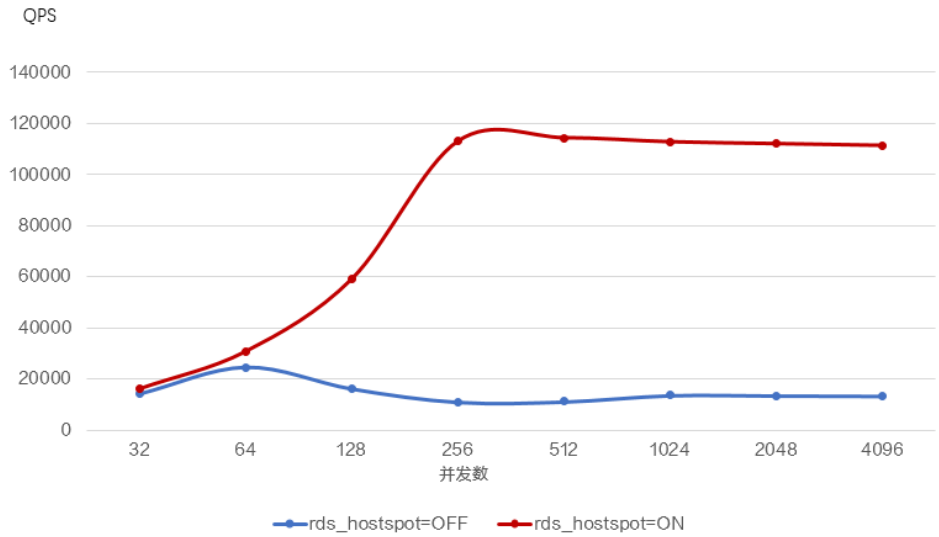
  测试语句：

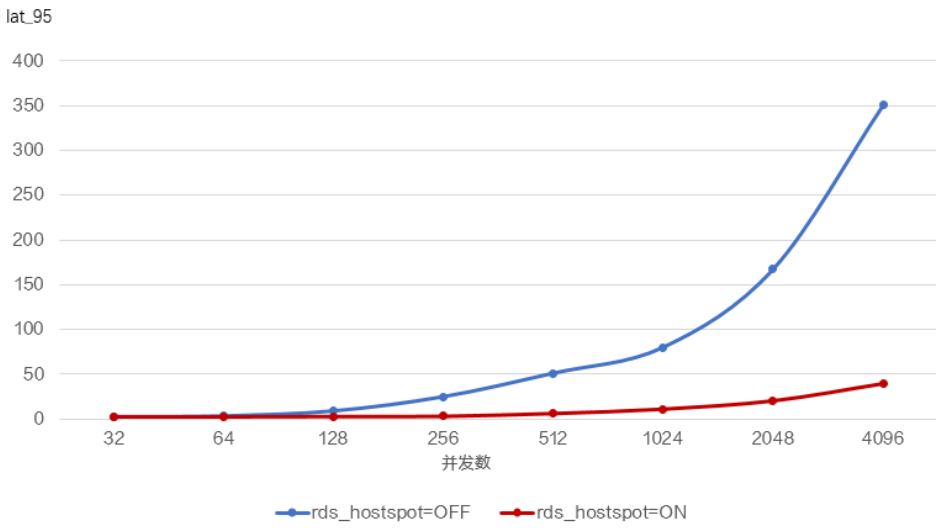
  UPDATE sbtest%u SET k=k+1 WHERE id=1 hotspot;

- 测试场景和测试结果

  测试场景1：8U32GB实例单个热点行更新

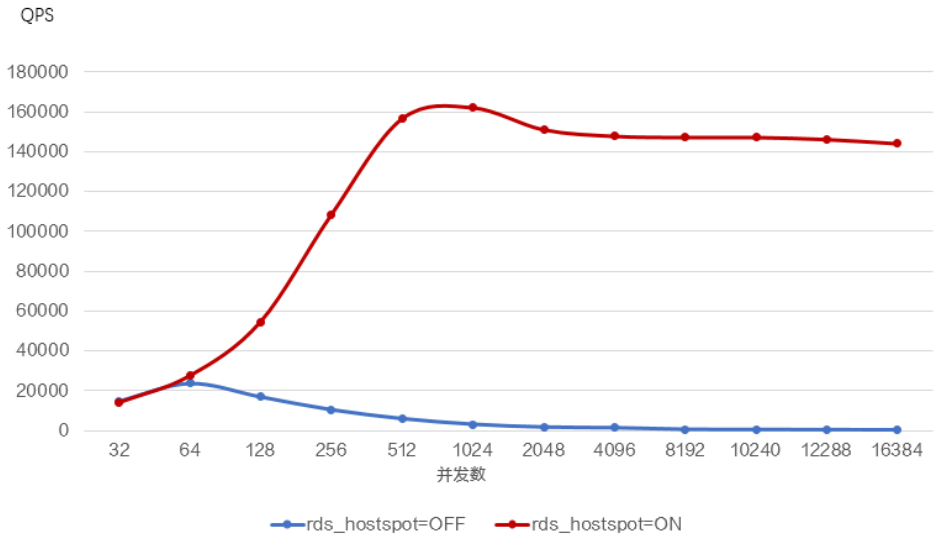
  测试结果：所有并发均有不同程度提升，64并发及以下并发提升不明显，128并发及以上并发提升明显，最高提升9.26倍。

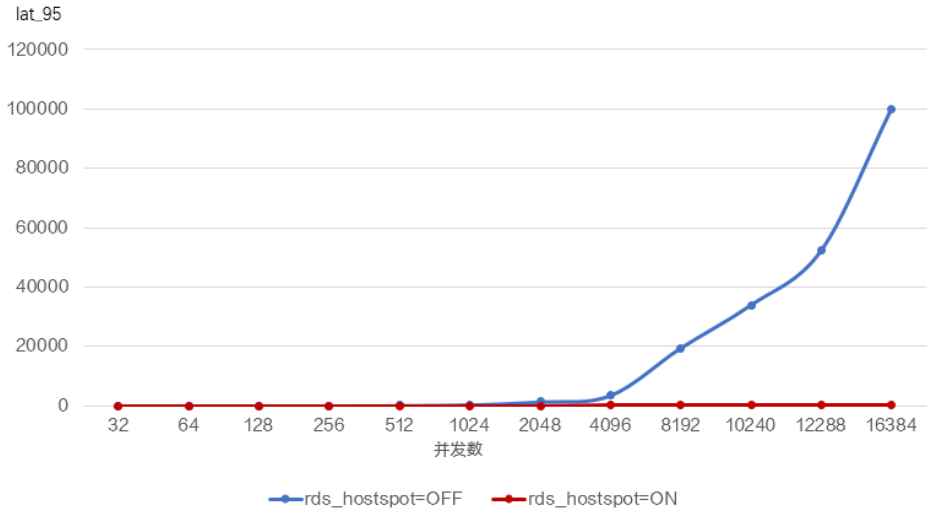




测试场景2：32U128GB实例单个热点行更新

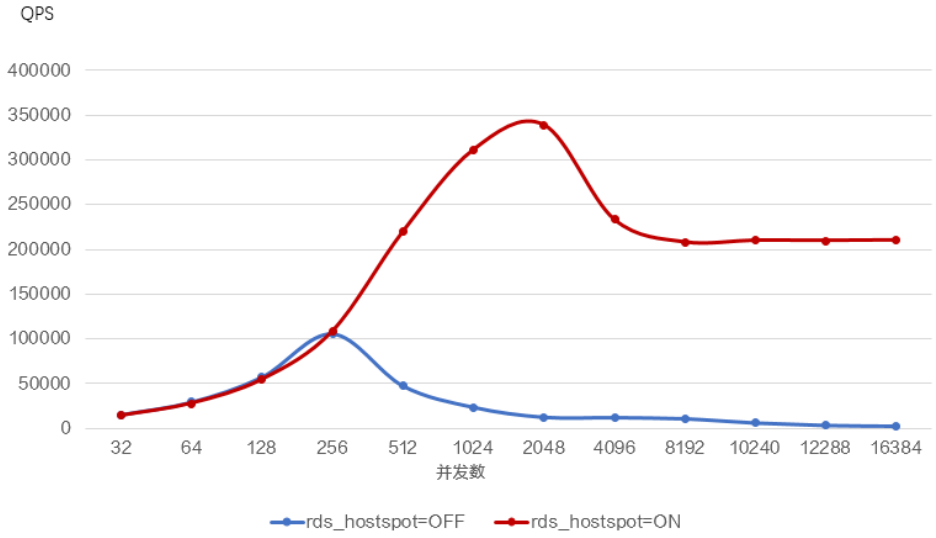
测试结果：128并发及以上并发提升明显，最高提升639倍。

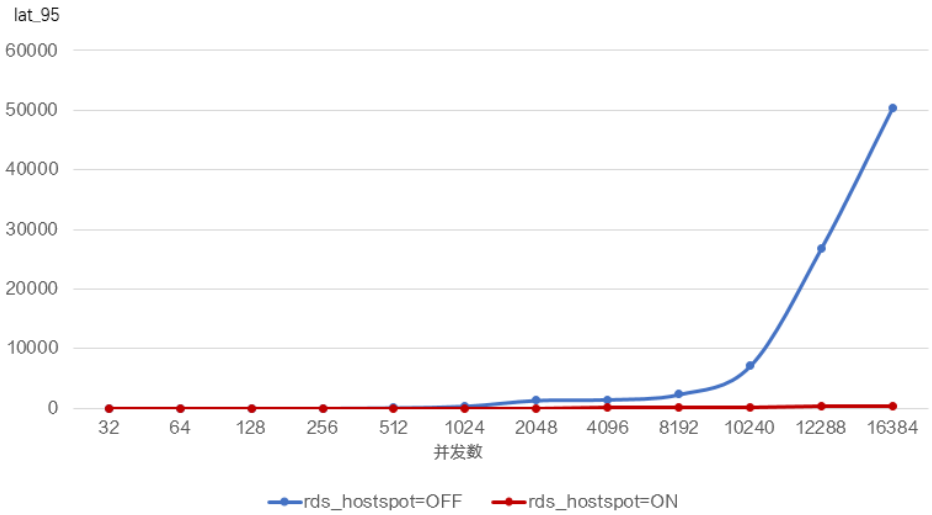




测试场景3：32U 128GB实例8个热点行更新

测试结果：256及以下并发无提升，512及以上并发提升效果明显，最高提升78倍。





2.13 多租户管理与资源隔离

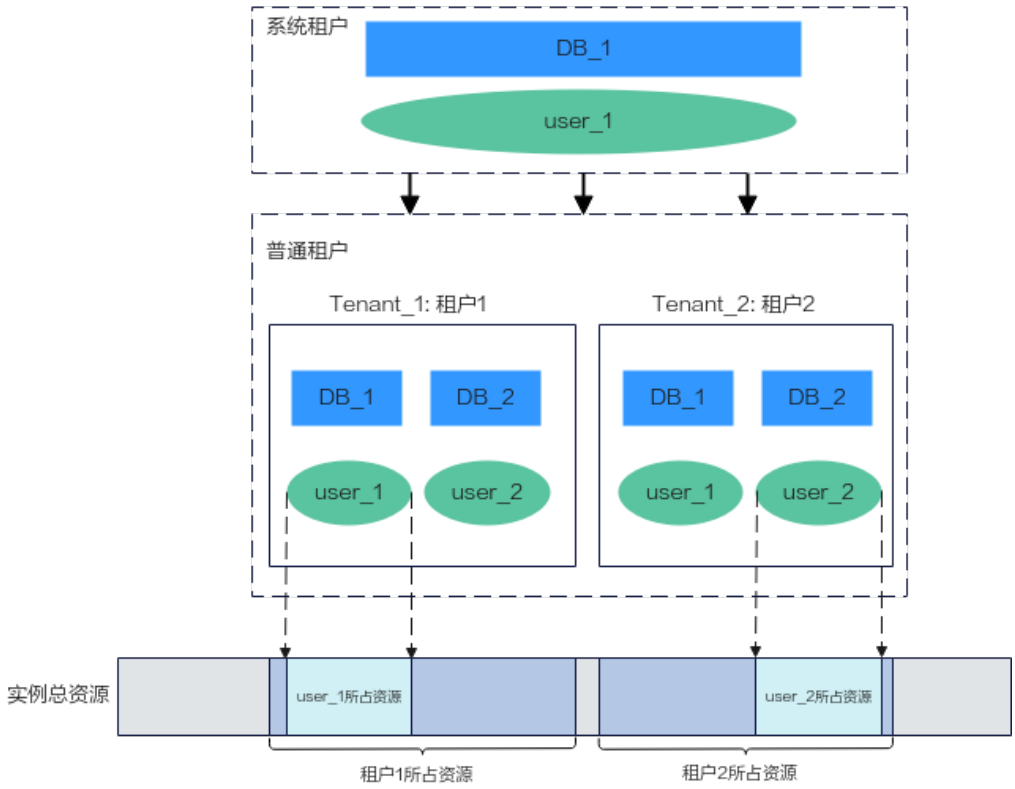
本文介绍了GaussDB(for MySQL)提供的多租户数据隔离以及资源隔离的相关语法和使用说明。

功能简介

GaussDB(for MySQL)提供的多租户管理功能，让数据库能够为其多个租户服务，提高数据库资源利用率。租户间实现数据隔离，不同租户能访问自己的数据。支持租户级资源隔离和用户级资源隔离，资源隔离能够避免浪费和性能扰邻。支持资源动态调整，能够及时应对不同租户或用户的业务高峰和波谷。

多租户管理功能的原理图如下：

图 2-22 多租户管理原理图

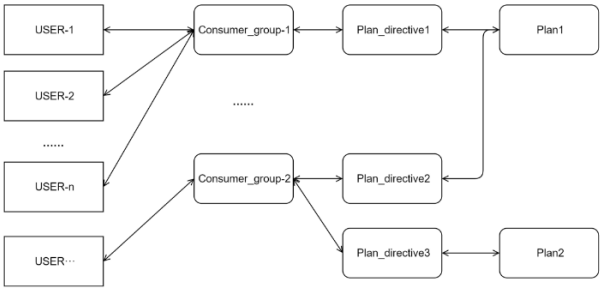


常用概念

如表2-36所示为租户级和用户级的相关术语。

表 2-36 租户级和用户级相关术语

所属层级	相关术语
租户级	**租户（Tenant）** 租户的层级结构在数据库实例之下，在数据库与用户之上。租户是为了数据隔离、资源隔离提出的概念，实际访问数据库还是需要以用户的身份去访问。 租户分为系统租户和普通租户。 ● **系统租户（sys_tenant）** 系统租户是为了适配原有模式下用户的使用，原有数据库实例中的用户默认属于系统租户，也可以称为系统用户。当通过系统租户下的用户连接数据库时，若该用户拥有对应的数据库实例访问权限，即可访问所有租户下的数据库实例。 ● **普通租户（user_tenant）** 普通租户需要在系统租户下进行创建，普通租户下的数据库实例与用户完全隔离，无法互相访问，并且普通租户无法访问系统租户下的数据库实例。在进行CPU资源调度时，根据min_cpu是否大于0将普通租户分为独占型租户和共享型租户。 – 独占型租户：min_cpu>0，需要保证任何时刻的CPU资源不小于min_cpu。 – 共享型租户：min_cpu=0。系统优先保证独占租户的资源请求，然后将剩余资源配给共享租户。同时，系统保留一部分CPU份额（对应参数mt_shared_cpu_reserved）给共享租户，保证共享租户不会饿死。独占型租户和共享型租户可以通过调整min_cpu的值进行相互转换。 **资源配置（resource_config）** 为了实现租户级资源隔离所提出的概念，一个resource_config描述了对应租户下能够使用的资源。当前仅支持对CPU资源进行隔离以及调度。

所属层级	相关术语
用户级	用户的层级结构在数据库实例和租户之下，同一个租户可以拥有多个用户。 **图 2-23** 用户级资源配置关系图  **资源消费组（consumer_group）** 同一个资源配置下的用户集合。多个用户可以归属到同1个资源消费组，同一个资源消费组下的用户共享资源配置。 **资源计划指令（plan_directive）** 具体的资源配置。1个资源计划指令绑定1个资源消费组，描述当前消费组的具体资源配置。 **资源计划（plan）** 资源配置的集合。1个资源计划绑定1个或多个资源计划指令。启用/禁用资源计划可使对应的资源计划指令生效/失效。

使用须知

- 多租户管理与资源隔离功能支持租户级数据隔离、租户级CPU资源隔离、用户级CPU资源隔离。
- 多租户管理与资源隔离功能需要GaussDB(for MySQL)实例内核版本为2.0.54.240600及以上。
- 开启多租户管理与资源隔离前必须开启thread pool功能。
- 开启多租户管理与资源隔离功能前，要求实例中的数据库名称和用户名称不能包含@字符。
- Serverless不支持多租户管理与资源隔离功能。
- 租户跨实例迁移：

  开启多租户管理与资源隔离功能后，DRS迁移可以迁移所有租户下的数据，但DRS不会同步多租元数据，故租户信息不会被同步到目标端。如果需要迁移某个租户到另一个实例，需要遵循以下步骤：

  a. 目标端需要使用具备多租特性的实例，首先在目标端手动创建租户。
  b. 使用DRS数据同步功能创建库级同步任务（如果源端和目标端租户名有改动，则这里需要改动目标端库名）。

c. 进行同步。

- Binlog：

  目前Binlog没有实现租户隔离，如果允许普通租户拉取Binlog会破坏租户间数据隔离，因此当前禁止普通租户下的用户拉取Binlog。

  本特性引入的租户和资源隔离相关接口，记录的Binlog为row格式。在binlog_format不为row的情况下，通过Binlog进行同步，将无法同步租户及资源配置信息。

- Proxy：
  - 支持带Proxy使用，带Proxy使用时show processlist显示结果是节点级别。
  - 因HTAP标准版和轻量版不支持带@字符的数据库，普通租户下的数据库迁移到该类HTAP引擎时会更改目标端库名，但Proxy的自动行存/列存引流功能要求源端和目标端的库名一致，因此普通租户创建的数据库将无法使用Proxy的自动行存/列存引流功能。
- 备份恢复：

  恢复已打开多租开关的多租实例数据到新实例，未开启多租开关下，新实例不支持创建带@的用户和DB。

- 兼容性：
  - 先开启后关闭多租开关，创建DB和用户时，名称中不能包含@字符。
  - 普通租户下，数据库名称最大长度由64降低为50，用户名称最大长度由32降低为20。
  - 当前仅支持普通租户访问information_schema，其他系统库暂不开放给普通租户。
  - 普通租户不支持tablespace相关语法。

开启多租户模式

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角的，选择区域和项目。

步骤3 在页面左上角单击，选择“数据库 > 云数据库 GaussDB(for MySQL)”。

步骤4 在实例管理页面，单击目标实例名称，进入基本信息页面。

步骤5 在实例信息区域，单击多租户隔离右侧，在弹框中单击“确认”，开启多租户隔离功能。

开启多租户隔离前要求存量的数据库名称和用户名称不能包含@字符，否则会开启失败。

----结束

资源管理

资源配置（resource_config）和租户是一对多的关系，当某一租户绑定此资源配置时便可限制此租户下用户使用的CPU资源。

- 创建资源配置
  ```
  CREATE resource_config config_name MAX_CPU [=] {max_cpu_value} [MIN_CPU [=] {min_cpu_value}];
  ```
- 更新资源配置
  ```
  ALTER resource_config config_name MAX_CPU [=] {max_cpu_value} [MIN_CPU [=] {min_cpu_value}];
  ```
- 删除资源配置
  ```
  DROP resource_config config_name;
  ```
- 查看资源配置
  ```
  SELECT * FROM information_schema.DBA_RSRC_TENANT_RESOURCE_CONFIGS;
  ```

说明

- 仅在高权限root用户下可使用。
- 参数说明：
  - config_name： 资源配置名称。最大长度为64，仅支持包含大写字母、小写字母、数字或下划线。
  - MAX_CPU：该资源配置能够使用的CPU的最大数量，最小值为0.1，最大值为mt_flavor_cpu。粒度0.1U。
  - MIN_CPU：该资源配置能够使用的CPU的最小数量，为可选项。默认等于MAX_CPU，最小值为0，最大值不超过MAX_CPU。粒度为0.1U。如果设置为0，表示该租户为共享租户，如果大于0则为独占租户。
- 更新resource_config时，如果该resource_config已经绑定租户，且更新后MIN_CPU值大于原有的MIN_CPU值，则需要校验修改后的值是否满足资源约束，否则不做资源约束校验。
- 删除resource_config时，若租户正在使用该资源配置，则无法进行删除操作。
- 实际使用中，共享租户的资源分配不依赖MAX_CPU，是随机争夺资源的。
- CPU争抢时，租户间资源分配会尽量按照租户的MIN_CPU分配资源，但存在一定的误差，误差通常在1U以内。
- 租户性能比同等规格实例性能低，实际使用中多个租户抢占资源性能将更低。

租户管理

创建租户时，需要绑定已经创建的资源配置（resource_config），以此限制该租户下用户使用的CPU资源。

- 创建租户
  ```
  CREATE TENANT tenant_name RESOURCE_CONFIG config_name [COMMENT [=] 'comment_string'];
  ```
- 更改租户
  ```
  ALTER TENANT tenant_name RESOURCE_CONFIG config_name [COMMENT [=] 'comment_string'];
  ```
- 删除租户
  ```
  DROP TENANT tenant_name;
  ```
- 查看租户
  ```
  SELECT * FROM information_schema.DBA_RSRC_TENANT;
  ```

说明

- 仅在高权限root用户下可使用。
- 创建租户
  - tenant_name长度不超过10个字符，仅支持包含小写字母、数字或下划线_。
  - 创建租户会进行资源约束检查，需要保证所有租户的资源配置中MIN_CPU之和满足资源约束。
- 更改租户resource_config
  - 如果新绑定的resource_config的MIN_CPU值大于等于原有resource_config的MIN_CPU值时，会进行资源约束检查，需要保证所有租户的资源配置中MIN_CPU之和满足资源约束。
  - 独占租户如果新绑定的resource_config的MIN_CPU值为0，会导致租户变为共享租户，将同时删除租户关联的用户级资源隔离相关的元数据。
- 删除租户
  - 需要保证租户下的DB和用户已经被删除，否则，无法删除租户。
  - 同时将删除租户关联的用户级资源隔离相关的配置。

用户管理

开启多租模式后，用户分为系统租户下的用户和普通租户下的用户。存量的用户均属于系统租户，新创的用户根据接口语义属于系统租户或者普通租户。

- 在系统租户下管理用户

  创建用户

  创建系统租户下的用户：

  ```
  CREATE user [IF NOT EXISTS] user_name@host;
  ```

  创建普通租户下的用户：

  ```
  CREATE user [IF NOT EXISTS] 'user_name@tenant_name'@host;
  ```

  重命名用户

  重命名系统租户下的用户：

  ```
  RENAME USER user_from@host1 TO user_to@host2;
  ```

  重命名普通租户下的用户：

  ```
  RENAME USER 'user_from@tenant_name'@host1 TO 'user_to@tenant_name'@host2;
  ```

  删除用户

  删除系统租户下的用户：

  ```
  DROP USER [IF EXISTS] user_name@host;
  ```

  删除普通租户下的用户：

  ```
  DROP USER [IF EXISTS] 'user_name@tenant_name'@host;
  ```

  授权用户

  为用户user1@tenant_1授予租户tenant_1下的priv_type权限：

  ```
  GRANT priv_type ON *.* to 'user_1@tenant_1'@'%' with grant option;
  ```

  查看权限：

  ```
  SHOW grants for 'user_1@tenant_1';
  ```

- 在普通租户下管理用户

  创建用户

  创建当前租户下的用户：

```
CREATE user [IF NOT EXISTS] user_name@host;
```

重命名用户

```
RENAME USER user_from@host1 TO user_to@host2;
```

删除用户

```
DROP USER [IF EXISTS] user_name@host;
```

授权用户

为用户user1授予当前租户下的priv_type权限：

```
GRANT priv_type ON *.* to 'user_1'@'%' with grant option;
```

查看权限：

```
SHOW grants for 'user_1';
```

说明

- 在系统租户下创建或删除普通租户下的用户时，需要以user_name@tenant_name的方式对用户进行操作。
- 普通租户下的用户名称的长度被限制为不超过20个字符。
- 租户内不可以创建部分特殊用户：mysql.sys, mysql.session, mysql.infoschema以及参数rds_reserved_users中保留的用户。
- 普通租户下的用户重命名时，需要保证user_from和user_to中的tenane_name相同，如不相同，则接口返错。
- 特性开关关闭时，无法重命名普通租户下的用户。

数据库管理

数据库分为系统租户下的数据库和普通租户下的数据库。系统租户可以访问所有数据库，普通租户只能访问属于自己的数据库。

- 在系统租户下管理数据库

  创建数据库

  创建系统租户的数据库：

  ```
  CREATE DATABASE [IF NOT EXISTS] `db_name`;
  ```

  创建普通租户的数据库：

  ```
  CREATE DATABASE [IF NOT EXISTS] `db_name@tanent_name`;
  ```

  删除数据库

  删除系统租户的数据库：

  ```
  DROP DATABASE [IF EXISTS] `db_name`;
  ```

  删除普通租户的数据库：

  ```
  DROP DATABASE [IF EXISTS] `db_name@tanent_name`;
  ```

- 在普通租户下管理数据库

  创建当前租户的DB：

  ```
  CREATE DATABASE [IF NOT EXISTS] 'db_name';
  ```

  删除当前租户的DB：

  ```
  DROP DATABASE [IF EXISTS] 'db_name';
  ```

说明

- 在系统租户下创建或删除普通租户下的DB时，需要以db_name@tenant_name的方式对DB进行操作。
- 普通租户当前只支持INFORMATION_SCHEMA系统库的访问，PERFORMANCE_SCHEMA、SYS、MYSQL等系统库暂不允许普通租户访问。
- 租户内不可以创建部分特殊DB：INFORMATION_SCHEMA, PERFORMANCE_SCHEMA, MYSQL, SYS, __recyclebin。

- 存量数据库分配租户

  为保证兼容性，升级或者迁移到多租实例后，存量的DB默认属于系统租户，可以通过如下语法将存量的DB分配给租户。此外，对于已经开启多租特性后，系统租户创建的且未分配给普通租户的DB，也可通过如下语法分配数据库给对应租户。

  数据库分配

  将DB分配给租户名为tenant_name的普通租户

  ```
  ALTER DATABASE db_name TENANT = `tenant_name`;
  ```

  将DB回收到系统租户下

  ```
  ALTER DATABASE db_name TENANT = ``;
  ```

  查看映射关系

  ```
  SELECT * FROM information_schema.DBA_RSRC_TENANT_DB;
  ```

  说明

  - 仅在高权限root用户下可使用。
  - 如果DB是开启多租后创建的，以db_name@tenant_name格式命名的DB不允许分配调用此接口，接口会返错。
  - 如果tenant不存在，且不为空，接口返错。

- 通过租户下的用户连接数据库

  在系统租户下，原有连接方式不变。

  在普通租户下，指定用户时，需要以user_name@tenant_name的形式；指定DB时，支持db_name和db_name@tenant_name两种形式。

  ```
  mysql --host=**** -u user1@tenant_1 -D db1 -p pwssword;
  mysql --host=**** -u user1@tenant_1 -D db1@tenant_1 -p pwssword;
  ```

  连接成功后，该用户将受到对应租户下的资源限制。

用户级资源配置

租户下的用户默认共享当前租户的资源，若要进行用户级资源限制，请使用本节提供的用户级资源配置接口。

- 资源消费组（consumer_group）管理

  通过租户下的用户连接数据库，进行资源消费组管理。

  创建消费组

  ```
  dbms_resource_manager.create_consumer_group (
    consumer_group     CHAR(128),
    comment            CHAR(2000));
  ```

  绑定消费组

  consumer_group参数不为'' ，此时将用户和consumer_group绑定。

  ```
  dbms_resource_manager.set_consumer_group_mapping (
    attribute        CHAR(128),
  ```

```
  value            varbinary(128),
  consumer_group   CHAR(128));
```

解绑消费组

consumer_group参数为'' ，此时将用户和consumer_group解绑。

```
dbms_resource_manager.set_consumer_group_mapping (
  attribute        CHAR(128),
  value            varbinary(128),
  '');
```

删除消费组

```
dbms_resource_manager.delete_consumer_group (
   consumer_group   CHAR(128));
```

查看消费组

视图DBA_RSRC_CONSUMER_GROUPS记录了所属租户（tenant）和消费组（consumer_group）的关联关系。

```
select * from information_schema.DBA_RSRC_CONSUMER_GROUPS;
```

查看消费组映射

视图DBA_RSRC_GROUP_MAPPINGS记录所属租户（tenant），用户（user）和消费组（consumer_group）的关联关系。

```
select * from information_schema.DBA_RSRC_GROUP_MAPPINGS;
```

说明

- 对于共享租户，禁止使用。
- 参数说明：

  consumer_group：资源消费组名称，仅支持包含大写字母、小写字母、数字或下划线_，在解绑消费组时可设置为''。

  comment：资源消费组说明，可为''。

  attribute：要添加或修改的映射属性，当前版本仅支持USER。

  value：要添加或修改的映射属性，当前版本仅支持用户名。
- 删除consumer_group的时候会同步删除该资源消费组对应的plan_directive以及consumer_group_mapping条目。
- 如果特性开关打开，删除用户时，也会删除此用户关联的消费组条目。
- 如果特性开关打开，重命名用户时，也会同步更新此用户关联的消费组条目。
- 重新打开特性，如果发现对应的用户已经无效，则自动删除这些无效的条目。
- 如果特性开关关闭后，删除用户，再创建同名用户，后续打开特性，该用户对应的条目会被保留。

资源计划（plan）管理

通过租户下的用户连接数据库，进行资源计划管理。

- 创建资源计划

```
dbms_resource_manager.create_plan (
  plan_name        VARCHAR(128),
  comment          VARCHAR(2000));
```

- 启用资源计划

```
dbms_resource_manager.set_resource_manager_plan(
   plan_name     VARCHAR(128));
```

- 禁用资源计划

```
dbms_resource_manager.set_resource_manager_plan ('');
```

- 删除资源计划

```
dbms_resource_manager.delete_plan (
   plan_name    VARCHAR(128));
```

- 查看资源计划

  视图DBA_RSRC_PLANS记录当前计划（plan）的详细信息。

```
SELECT * FROM information_schema.DBA_RSRC_PLANS;
```

  说明

  - 对于共享租户，禁止使用。
  - 参数说明：

    plan_name：资源计划名称，仅支持包含大写字母、小写字母、数字或下划线。

    comment：资源计划描述信息，可为''。
  - 若删除当前启用的资源计划，则会将当前启用的资源计划设置为空，同时删除对应的资源计划指令（plan_directive）。
  - plan默认可配置的最大数量为128，由参数mt_resource_plan_num控制。

资源计划指令(plan_directive)管理

通过租户下的用户连接数据库，进行资源计划指令管理。

- **创建资源计划指令**

```
dbms_resource_manager.create_plan_directive (
  plan                  CHAR(128),
  group_or_subplan      CHAR(128),
  comment               VARCHAR(2000),
  mgmt_p1               bigint(20),
  utilization_limit     bigint(20));
```

- **更新资源计划指令**

```
dbms_resource_manager.update_plan_directive (
  plan                  CHAR(128),
  group_or_subplan      CHAR(128),
  new_comment           VARCHAR(2000),
  new_mgmt_p1           bigint(20),
  new_utilization_limit bigint(20));
```

- **删除资源计划指令**

```
dbms_resource_manager.delete_plan_directive (
   plan                 CHAR(128),
   group_or_subplan     VARCHAR(128));
```

- **查看资源计划指令**

  DBA_RSRC_PLAN_DIRECTIVES记录计划（plan）、消费组（consumer_group）的关联关系，以及对应消费组（consumer_group）的资源配置。

```
SELECT * FROM information_schema.DBA_RSRC_PLAN_DIRECTIVES;
```

说明

- 对于共享租户，禁止使用；
- 参数说明：

  plan：plan的名称。

  group_or_subplan：consumer_group的名称。

  comment：资源计划指令的描述信息，可为''。

  mgmt_p1：在系统满负载情况下，承诺分配给本消费组的CPU占比，取值范围[0, 100]，100表示使用租户100%的CPU。

  utilization_limit：指定消费组使用的CPU资源的上限，取值范围为 [1, 100]。100表示最大可使用租户全部CPU资源，如果取值为70则表示最大可使用租户 70%的CPU资源。
- 删除正在启用的plan_directive, 将导致对应用户的资源配置失效。
- 同一消费组下的用户使用的资源总和不超过当前消费组的资源限制。例如：同一租户下有用户user1和user2，user1和user2属于consumer_group1，consumer_group1的UTILIZATION_LIMIT为70；则user1和user2实际使用的CPU资源总和最大为当前租户70%的CPU资源。

清空用户级配置

通过租户下的用户连接数据库，清空当前租户下相关的资源配置数据，包括DBA_RSRC_CONSUMER_GROUPS、DBA_RSRC_GROUP_MAPPINGS、DBA_RSRC_PLAN_DIRECTIVES、DBA_RSRC_PLANS表中的数据。

```
dbms_resource_manager.clear_all_configs();
```

CPU 使用率统计

- **用户级CPU使用率**

  新增information_schema.cpu_summary_by_user表显示当前租户下各用户的CPU使用信息，具体使用如下：

  ```
  SELECT * FROM information_schema.cpu_summary_by_user;
  ```

  说明

  - 需要提前为当前用户（user）配置消费组（consumer_group）。
  - 查询结果中的列名说明：

    TENANT_NAME：用户所属的租户名称。

    USER_NAME：用户名称。

    CPU_USAGE：用户CPU使用率，为用户实际使用的CPU占所在租户配置的MAX_CPU的比例。例如，当前租户的max_cpu为4U, 而用户实际用了2U， 那么CPU_USAGE为50%。

- **租户级CPU使用率**

  新增information_schema.cpu_summary_by_tenant表显示各租户的CPU使用信息，具体使用如下：

  ```
  SELECT * FROM information_schema.cpu_summary_by_tenant;
  ```

  说明

  查询结果中的列名说明：

  TENANT_NAME：租户名称。

  CPU_USAGE：租户CPU使用率，为租户实际使用的CPU占配置的MAX_CPU的占比。例如，当前租户的max_cpu为4U, 而租户实际用了2U， 那么CPU_USAGE为50%。

2.14 字段压缩

为了减少数据页面存储空间占用，节省成本，GaussDB(for MySQL)推出细粒度的字段压缩，提供ZLIB和ZSTD两种压缩算法，用户可以综合考虑压缩比和压缩解压性能影响，选择合适的压缩算法，对不频繁访问的大字段进行压缩。同时，字段压缩特性提供自动压缩的能力，帮助用户更方便地使用此特性。

使用场景：降本，表中包含不频繁访问的大字段。

使用须知

- GaussDB(for MySQL)实例内核版本大于等于2.0.54.240600可使用该功能。
- 不支持分区表、临时表、非InnoDB引擎表。
- 压缩字段上不能包含索引(主键、唯一索引、二级索引、外键、全文索引)。
- 仅支持的数据类型：BLOB（包含TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB），TEXT（包含TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT），VARCHAR，VARBINARY。
- 不支持在生成列上使用此特性。
- 不支持在分区表和带有压缩字段的表之间执行EXCHANGE PARTITION式语句。
- 不支持IMPORT TABLESPACE。
- 仅支持在CREATE TABLE/ALTER TABLE ADD/ALTER TABLE CHANGE/ALTER TABLE MODIFY场景使用此特性。
- ALTER TABLE ADD COLUMN不支持INSTANT算法，ALTER TABLE {CHANGE|MODIFY} 语法涉及数据变化时不支持使用INSTANT算法。
- 自动压缩场景(rds_column_compression=2)，定义字段的最大长度需大于等于字段压缩阈值(rds_column_compression_threshold)时才可以被添加压缩属性；显式压缩场景(rds_column_compression=1)，若定义压缩字段的最大长度小于字段压缩阈值，字段仍可被添加压缩属性，同时收到warning信息。
- 若当前表中包含压缩字段，暂不支持NDP计算下推。
- 客户手动拉取BINLOG同步过程中ALTER语句会出现不兼容的问题，推荐客户侧使用HINT的方式来解决。
- 利用DRS迁移至无该特性的实例，压缩属性被消除。全量迁移任务可以进行，增量迁移时ALTER语句中存在压缩字段会导致迁移任务失败。
- 物理备份方面，利用备份去做数据恢复的版本也必须是带有字段压缩特性的。
- 升级至新版本之后，如果已经使用字段压缩功能，不支持回退至无该特性的版本。

语法

扩展column_definition定义，支持在CREATE TABLE/ALTER TABLE ADD/ALTER TABLE CHANGE/ALTER TABLE MODIFY场景定义列属性时使用压缩特性。

```
create_definition: {
    col_name column_definition
  | {INDEX | KEY} [index_name] [index_type] (key_part,...)
      [index_option] ...
  | {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name] (key_part,...)
      [index_option] ...
```

```
  | [CONSTRAINT [symbol]] PRIMARY KEY
      [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
      [index_name] [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (col_name,...)
      reference_definition
  | check_constraint_definition
}

alter_option: {
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST | AFTER col_name]
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  ...
```

其中column_definition为：

```
column_definition: {
    data_type [NOT NULL | NULL] [DEFAULT {literal | (expr)} ]
      [VISIBLE | INVISIBLE]
      [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
      [COMMENT 'string']
      [COLLATE collation_name]
      [COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
      [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}|COMPRESSED[={ZLIB|ZSTD}**]]
      [ENGINE_ATTRIBUTE [=] 'string']
      [SECONDARY_ENGINE_ATTRIBUTE [=] 'string']
      [STORAGE {DISK | MEMORY}]
      [reference_definition]
      [check_constraint_definition]
  | data_type
      [COLLATE collation_name]
      [GENERATED ALWAYS] AS (expr)
      [VIRTUAL | STORED] [NOT NULL | NULL]
      [VISIBLE | INVISIBLE]
      [UNIQUE [KEY]] [[PRIMARY] KEY]
      [COMMENT 'string']
      [reference_definition]
      [check_constraint_definition]
}
```


特性参数说明

表 2-37 参数说明

参数名称	描述	取值范围	默认值	级别	是否动态生效
rds_column_compression	• 当参数取值为0时，表示字段压缩特性关闭，不再支持显式或者自动创建压缩列。 • 当参数取值为1时，表示仅支持显式创建压缩列。 • 当参数取值为2时，表示同时支持显式或自动创建压缩列。	[0,2]	0	GLOBAL	是
rds_default_column_compression_algorithm	设置字段压缩特性默认的压缩算法，适用于如下场景： • 显式创建压缩字段而不指定具体的压缩算法。 • 自动压缩场景。	ZLIB或ZSTD	ZLIB	GLOBAL	是
rds_column_compression_threshold	设置字段压缩特性的阈值。 • 当列定义最大长度小于此阈值时，可以显式创建压缩列，但会收到提示信息，不能自动地创建压缩列。 • 当列定义最大长度大于等于此阈值时，支持显式或者自动创建压缩列。	[20-4294967295]	100	GLOBAL	是

参数名称	描述	取值范围	默认值	级别	是否动态生效
rds_zlib_column_compression_level	控制字段压缩特性中zlib压缩算法的压缩级别。 ● 当参数取值为0时代表不压缩。 ● 取值为除0外范围内的其他参数值时，取值越小，压缩速度越快但压缩效果越差；取值越大，压缩速度越慢但压缩效果越好。	[0,9]	6	GLOBAL	是
rds_zstd_column_compression_level	控制字段压缩特性中zstd压缩算法的压缩级别。 取值越小，压缩速度越快但压缩效果越差；取值越大，压缩速度越慢但压缩效果越好。	[1,22]	3	GLOBAL	是

使用示例

1. 显式创建压缩字段。

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

mysql> show variables like 'rds_default_column_compression_algorithm';
+-----------------------------------------+-------+
| Variable_name                           | Value |
+-----------------------------------------+-------+
| rds_default_column_compression_algorithm | ZLIB  |
+-----------------------------------------+-------+
1 row in set (0.00 sec)

mysql> create table t1(c1 varchar(100) compressed, c2 varchar(100) compressed=zlib, c3
varchar(100) compressed=zstd) default charset=latin1;
Query OK, 0 rows affected (0.06 sec)

mysql> show create table t1\G
*************************** 1. row ***************************
Table: t1
Create Table: CREATE TABLE `t1` (
`c1` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
`c2` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
`c3` varchar(100) /*!99990 800220201 COMPRESSED=ZSTD */ DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

2. 自动创建压缩字段。

```
mysql> set global rds_column_compression = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'rds_column_compression';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| rds_column_compression | 2     |
+------------------------+-------+
1 row in set (0.01 sec)

mysql> show variables like 'rds_column_compression_threshold';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| rds_column_compression_threshold | 100   |
+-----------------------------------+-------+
1 row in set (0.01 sec)

mysql> show variables like 'rds_default_column_compression_algorithm';
+-------------------------------------------+-------+
| Variable_name                             | Value |
+-------------------------------------------+-------+
| rds_default_column_compression_algorithm | ZLIB  |
+-------------------------------------------+-------+
1 row in set (0.01 sec)

mysql> create table t2(c1 varchar(99), c2 varchar(100)) default charset=latin1;
Query OK, 0 rows affected (0.05 sec)

mysql> show create table t2\G
*************************** 1. row ***************************
Table: t2
Create Table: CREATE TABLE `t2` (
`c1` varchar(99) DEFAULT NULL,
`c2` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

3. 关闭特性。

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

mysql> show variables like 'rds_column_compression';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| rds_column_compression | 0     |
+------------------------+-------+
1 row in set (0.01 sec)

mysql> show variables like 'rds_column_compression_threshold';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| rds_column_compression_threshold | 100   |
+-----------------------------------+-------+
1 row in set (0.01 sec)

mysql> create table t3(c1 varchar(100) compressed, c2 varchar(100) compressed=zlib, c3
varchar(100) compressed=zstd) default charset=latin1;
Query OK, 0 rows affected, 3 warnings (0.04 sec)

mysql> show create table t3\G
*************************** 1. row ***************************
Table: t3
Create Table: CREATE TABLE `t3` (
`c1` varchar(100) DEFAULT NULL,
`c2` varchar(100) DEFAULT NULL,
```

```
`c3` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

确认效果

1. 通过执行show create table语句展示表结构信息，发现其中包含/*!99990 800220201 COMPRESSED=xxxx */的内容，可认为已使用字段压缩特性。

   例如：

   ```
   mysql> show create table t1\G
   *************************** 1. row ***************************
          Table: t1
   Create Table: CREATE TABLE `t1` (
     `c1` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
     `c2` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
     `c3` varchar(100) /*!99990 800220201 COMPRESSED=ZSTD */ DEFAULT NULL
   ) ENGINE=InnoDB DEFAULT CHARSET=latin1
   1 row in set (0.00 sec)
   ```

2. 利用系统视图information_schema.columns查询压缩字段。

   例如：

   ```
   mysql> select TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, EXTRA from
   information_schema.columns where extra like '%compressed%';
   +--------------+------------+-------------+-----------------+
   | TABLE_SCHEMA | TABLE_NAME | COLUMN_NAME | EXTRA           |
   +--------------+------------+-------------+-----------------+
   | test         | t1         | c1          | COMPRESSED=ZLIB |
   | test         | t1         | c2          | COMPRESSED=ZLIB |
   | test         | t1         | c3          | COMPRESSED=ZSTD |
   | test         | t2         | c2          | COMPRESSED=ZLIB |
   +--------------+------------+-------------+-----------------+
   4 rows in set (0.50 sec)
   ```

3. 通过查询status信息，以确认字段压缩或解压缩接口的实际调用次数。

   ```
   mysql> show global status like '%column%compress%';
   +---------------------------------+-------+
   | Variable_name                   | Value |
   +---------------------------------+-------+
   | Innodb_column_compress_count    | 243   |
   | Innodb_column_uncompress_count | 34    |
   +---------------------------------+-------+
   ```

4. 通过执行如下语句或者查看监控界面信息，对比压缩前后表占用大小，确认压缩效果。

   ```
   SELECT table_name AS Table, round(((data_length + index_length) / 1024 / 1024), 2) AS Size in MB
   FROM information_schema.TABLES WHERE table_schema = "***" and table_name='***'
   ```

压缩比和性能影响验证

1. 插入随机数据场景，表中有1万行数据，每行是由400个MD5函数返回的32位字符串构成。

   ```
   CREATE TABLE `random_data` (
   `id` int(11) NOT NULL AUTO_INCREMENT,
   `data` longtext,
   PRIMARY KEY (`id`)
   ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

   DELIMITER $$
   CREATE PROCEDURE `generate_random_data`()
   BEGIN
     DECLARE i INT DEFAULT 1;
     DECLARE j INT DEFAULT 1;
     DECLARE str longtext;
     WHILE i <= 10000 DO
       SET j = 1;
   ```

```
    SET str = '';
    WHILE j <= 400 DO
      SET str = CONCAT(str, MD5(RAND()));
      SET j = j + 1;
    END WHILE;
    INSERT INTO `random_data` (`data`) VALUES (str);
    SET i = i + 1;
  END WHILE;
END$$
DELIMITER ;
```

分别设置rds_column_compression=0和rds_column_compression=2，其他参数默认，导入如上表结构并调用存储过程插入数据，利用zlib/ztsd算法压缩，计算压缩前大小或压缩后大小约为1.8。

2. 通过sysbench导入64张表，每张表1000万行数据，其中c和pad字段类型更改为varchar，修改后的表结构如下所示：

```
CREATE TABLE `sbtest1` (
  `id` int NOT NULL AUTO_INCREMENT,
  `k` int NOT NULL DEFAULT '0',
  `c` varchar(120) COLLATE utf8mb4_0900_bin NOT NULL DEFAULT '',
  `pad` varchar(60) COLLATE utf8mb4_0900_bin NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB AUTO_INCREMENT=10000001 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_bin
```

   - 分别设置rds_column_compression=0和rds_column_compression=2，其他参数默认，导入表结构和数据，计算收益后实际只对c列进行zlib/zstd压缩，计算压缩前大小/压缩后大小约为1.2。
   - 性能测试方面，理论上压缩级别越高对性能的影响越大，压缩后性能损耗在10%左右。