分布式数据库 TDSQL

通用参考

腾讯云

【联系我们】

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

文档目录

- META_CLUSTER_REGIONS
- META_CLUSTER_REPLICAS
- META_CLUSTER_RGS
- META_CLUSTER_SCHEDULE_CONFIGS
- PARTITION_POLICIES
- PARTITION_POLICY_AFFINITIES
- PARTITION_POLICY_PARTITIONS
- PARTITIONS
- PARTITIONS_VERBOSE
- PERSIST
- RANGE_CACHE
- TDSTORE_ACTIVE_COMPACTION_STATS
- TDSTORE_BULK_LOAD_TXN_INFO
- TDSTORE_CF_OPTIONS
- TDSTORE_COLUMNAR_COMPACTION_INFO
- TDSTORE_COLUMNAR_FILE_DELETION_INFO
- TDSTORE_COLUMNAR_FILES_INFO
- TDSTORE_COLUMNAR_FILE_VERSION_LIST
- TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO
- TDSTORE_COLUMNAR_REP_GROUP_REPLAY_LATENCY
- TDSTORE_COMMON_EVENT_INFO
- TDSTORE_COMPACTION_HISTORY
- TDSTORE_INSTALL_SNAPSHOT_INFO
- TDSTORE_LOCAL_METADATA_DESTROY_REPLICA
- TDSTORE_LOCAL_METADATA_TSO_META
- TDSTORE_LOCAL_METADATA_WRITE_FENCE
- TDSTORE_PART_CTX
- TDSTORE_PESSIMISTIC_DEADLOCK_DETAIL_INFO
- TDSTORE_PESSIMISTIC_DEADLOCK_INFO
- TDSTORE_REGION_EVENT_INFO
- TDSTORE_REPLICATION_GROUP_EVENT_INFO
- TDSTORE_SST_PROPS

SYS

- LOGSERVICE_DD_PARTITIONS
- LOGSERVICE_DD_TABLES
- META_CLUSTER_TASKS
- META_CLUSTER_JOBS
- RECYCLE_BIN_INFO

通用参考
系统原理
计算引擎：一条 SQL 在 TDSQL Boundless 中的执行全流程
架构简介

最近更新时间：2025-12-24 12:03:44

为什么需要研发 TDSQL Boundless

为解决业务增长中单机数据库的性能瓶颈与分库分表方案的固有不足，腾讯云研发了 TDSQL Boundless。以下是研发背后的核心考量：

目标一：突破单机性能限制

随着业务发展与数据量增长，单机 MySQL 面临硬盘、内存、CPU 等硬件上限。单纯升级硬件不仅成本高，也无法从根本上解决扩展性问题。

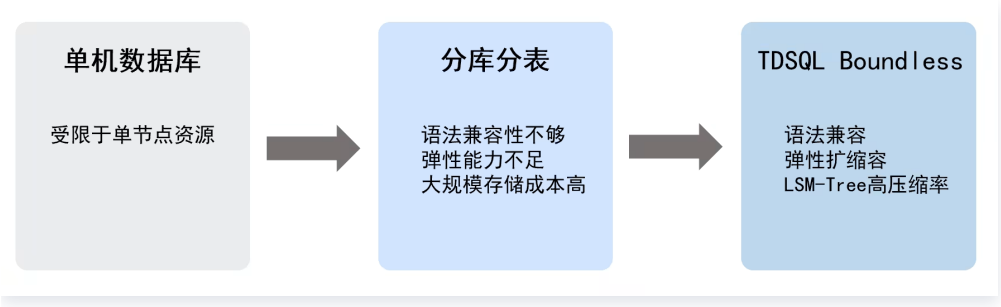
目标二：应对分库分表方案的常见挑战

尽管分库分表架构在扩展性和可用性上有所提升，但依然存在明显短板：

- **兼容性不足，增加业务复杂性**：分库分表（Sharding）方案通常无法完全兼容 MySQL 语法与功能，例如很难支持全局索引、跨分片范围（Range）查询、跨表联合查询与分布式事务一致性等等，业务开发需深度理解数据分布逻辑，业务迁移和改造门槛高。尤其在传统行业，系统复杂、表结构众多，改造代价巨大。
- **弹性扩展能力弱，需要预先设置分片规则**：分库分表（Sharding）方案属于预规划分配模式，需要在业务支出前预先规划分片数量，更适合业务场景稳定或有明确规律的场景。一旦分片数量确定后续调整很困难，扩容缩容往往涉及数据重分布与迁移，流程繁重、难以在线完成，无法适应业务快速变化与敏态需求。例如，电商大促之后，需要将多个 MySQL 实例合并为一个实例时，往往需要通过数据导出导入进行数据搬迁合并，操作复杂且耗时长。
- **海量存储的成本高**：基于 B+Tree 的存储引擎（如 InnoDB）在存储效率上不占优，尤其当数据规模增大、冷数据比例升高时，存储成本显著增加。

TDSQL Boundless 的解决思路

为了解决分库分表（Sharding）方案的这些不足，我们推出了 TDSQL Boundless。它高度兼容 MySQL 语法，支持灵活弹性伸缩，用户无需关心数据如何分布。同时，它采用 LSM-Tree 存储结构，压缩效率更高，更适合存储海量数据，能有效降低存储成本。

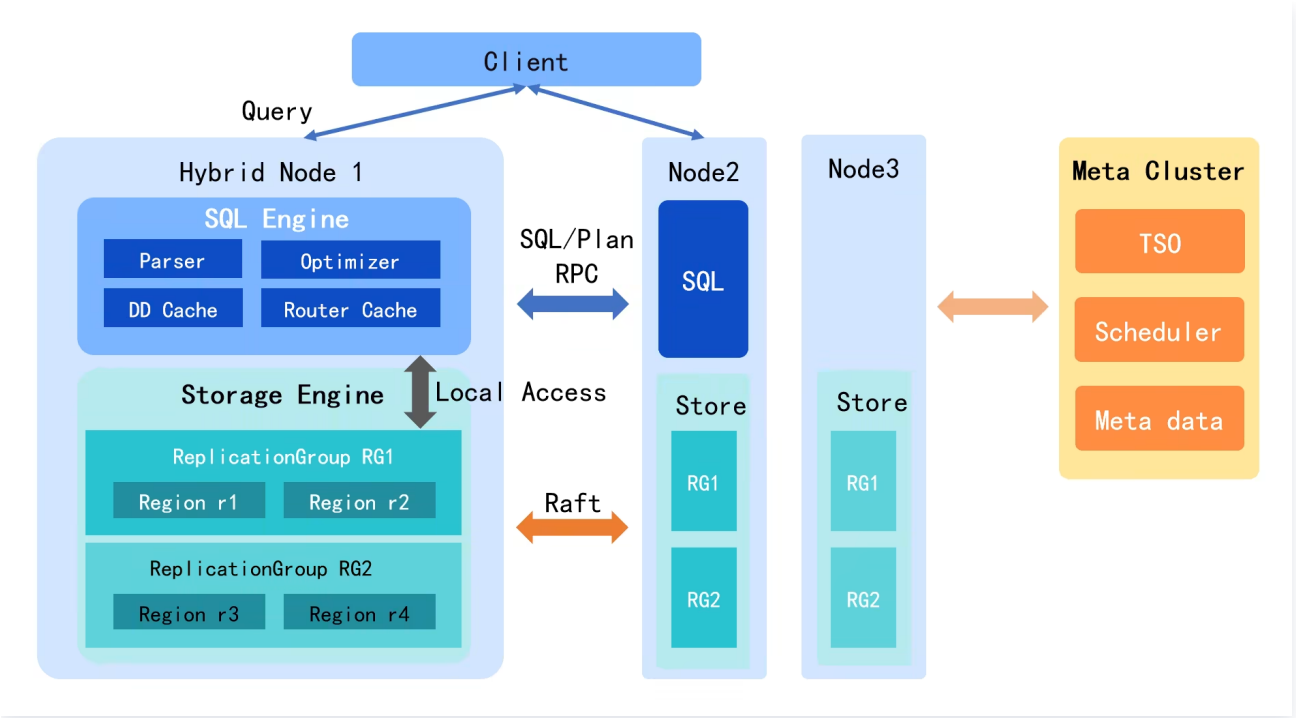


整体架构

TDSQL Boundless 采用存算分离的弹性扩缩容架构，由三个核心组件构成：

组件	角色和功能
SQLEngine（计算层）	• 负责 SQL 解析、查询优化、执行计划生成 • 具备路由管理、MPP 并行计算、Online DDL 等分布式能力 • 无状态设计，支持快速扩缩容
Storage Engine（存储层）	• 负责数据持久化存储 • 通过 Raft 协议实现多副本数据同步 • 根据节点负载自动执行数据拆分、合并与迁移
MetaCluster（元数据管理）	• 分配全局事务时间戳 • 协调存储层数据调度任务 • 确保分布式事务一致性与系统状态同步

工作流程：客户端接入计算层，计算层通过 RPC 访问存储层数据，控制层统一协调时间戳分配与任务调度，实现各层资源的独立弹性伸缩。


Client
Query
Hybrid Node 1
SQL Engine
Parser
Optimizer
DD Cache
Router Cache
Storage Engine
Local Access
ReplicationGroup RG1
Region r1
Region r2
ReplicationGroup RG2
Region r3
Region r4
SQL/Plan
RPC
Raft
Node2
SQL
Store
RG1
RG2
Node3
Store
RG1
RG2
Meta Cluster
TSO
Scheduler
Meta data

如何存储一张表的数据

最近更新时间：2025-12-24 12:03:44

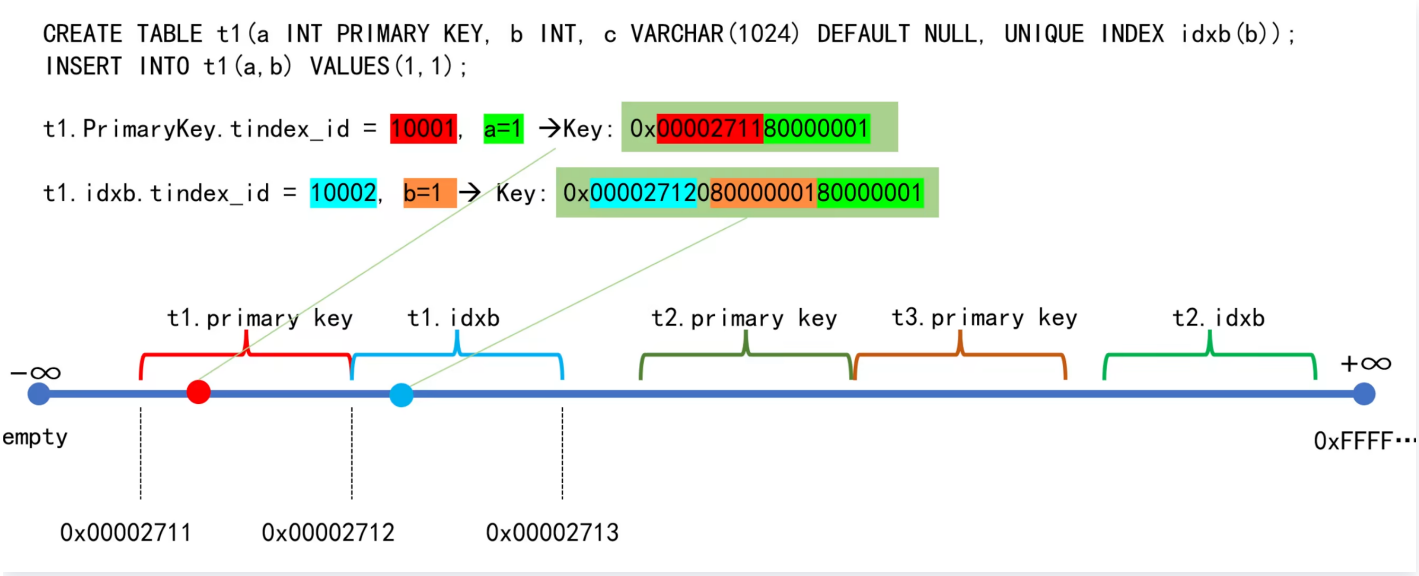
KV 编码和数据空间

在 TDSQL Boundless 中，所有数据都编码为 Key-Value 形式。编码后的 Key 具有 mem-comparable 特性（内存可比较）。

编码规则：系统为每个索引分配一个全局递增的唯一 ID。例如，表 t1 的主键和二级索引各有自己的 ID。同一索引的所有数据，其编码后的 Key 拥有相同的前缀（如 t1 主键的前缀是00002711），因此它们在逻辑上是连续的。

数据空间：数据可以被视为分布在一条无限长的数轴上，每个 Key 占据一个唯一位置。拥有相同前缀的索引数据会集中分布在这条线上的一个连续区间内。这样的一个数据区间称为一个 Region。

因此，同一索引的数据在空间上是连续的，但同一张表的不同索引可能分布在不同的、不连续的 Region 中。



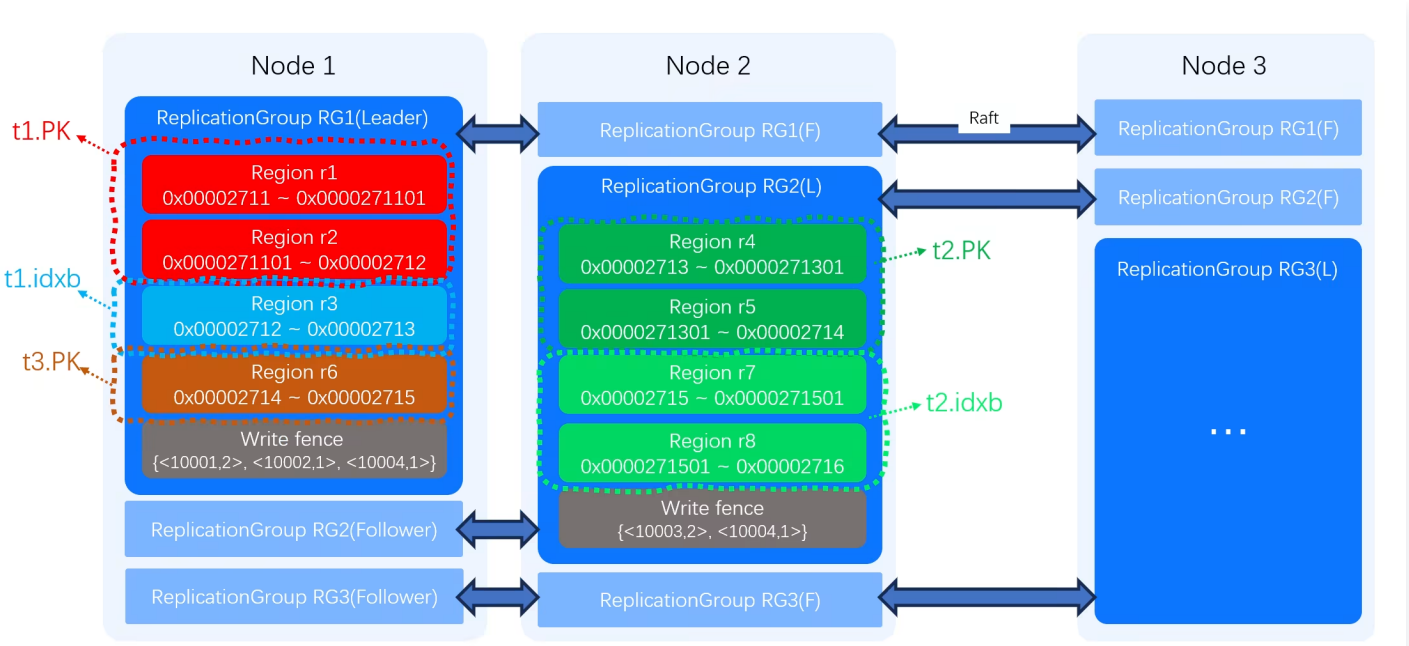
数据分片和复制组

在逻辑数据空间中，每个 Key 对应一个离散的点，但物理上每个 Key-Value 都需要存储空间。当数据量增大时，单个节点无法承载所有数据，因此数据被分割成多个分片，称为 Region。每个 Region 的容量标准为256MB或10万行数据。由于不同索引的数据量不同，Region 的数量会有所差异。

如下所示，表 t1 行数不足10万行，但 Value 字段较多，主键记录占空间更大，因此主键需要2个 Region，二级索引只需1个 Region。

表 t2 行数很多（如20万行），每一行的数据 Key-Value 很短，其主键和二级索引各自需要2个 Region，分别容纳10万行。

为了优化数据调度，在 Region 之上引入了 Replication Group（复制组）。例如，t1.pk 和 t1.idxb 属于不同 Region，但通过复制组可以将这些 Region 调度到相同节点上，从而在 INSERT 操作时避免 2PC 分布式事务，并在查询时避免跨节点回表。一个复制组可包含多个 Region，且复制组的 Leader 节点即为组内所有 Region 的 Leader 节点。

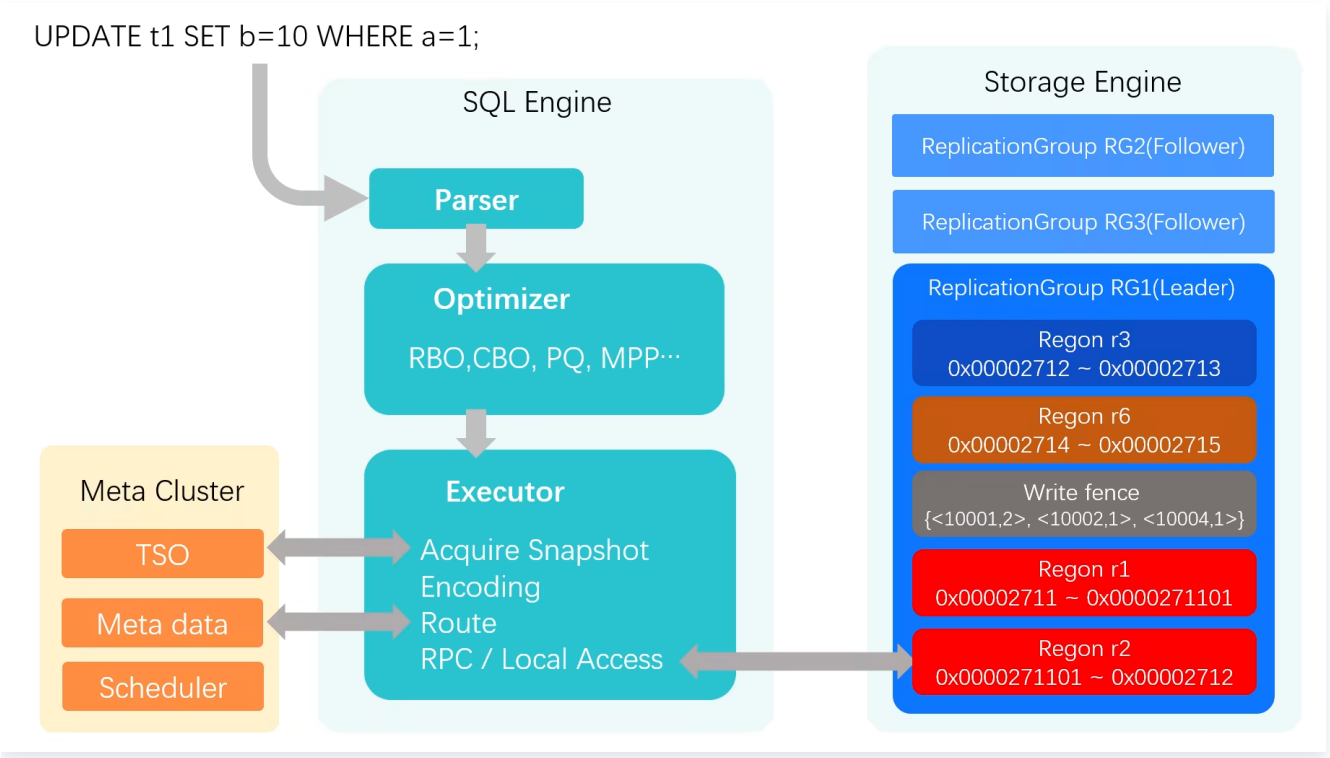


E.g. 每个Region256MB or 10万行

如何查询一张表的数据

最近更新时间：2025-12-24 12:03:44

下面以一条 UPDATE 语句（`UPDATE t1 SET b=10 WHERE a=1;`）为例，讲解其在分布式数据库中的执行关键步骤。该语句经过 SQL 引擎的 Parser 和 Optimizer 处理后，进入执行器阶段。



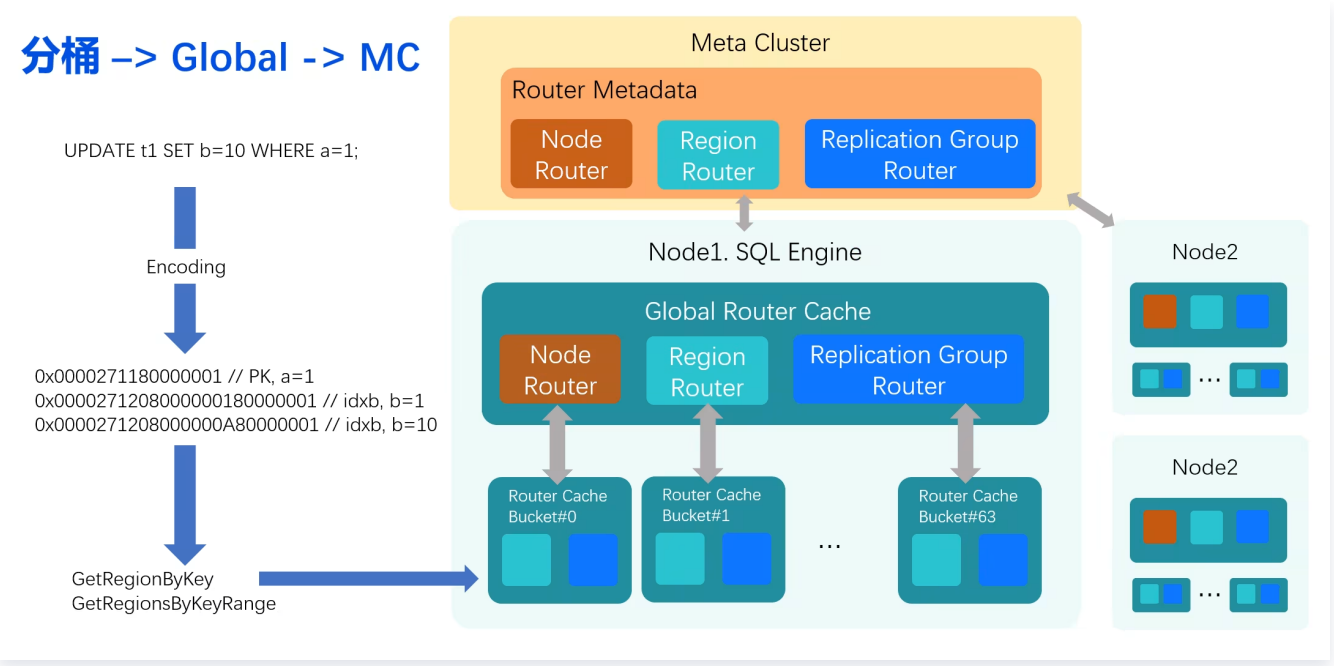
下面详细介绍执行器的关键步骤。

获取全局时间戳

事务开始时，首先需要从中心节点（MC）获取一个**全局时间戳**。此时间戳作为该事务的**快照版本**，在整个事务生命周期中用于多版本控制与一致性判断。MC 提供混合逻辑时间戳（物理时间戳 + 逻辑时间戳）。

数据编码

执行 UPDATE 操作需要定位并修改主键、二级索引的记录（包含旧值和新值）。所有数据均已编码为 Key-Value 形式。以下是此语句涉及的关键 Key 示例：



- `0000271180000001`：对应 a=1 主键的值
- `000027120800000180000001`：对应二级索引 b=1 的旧值（即原记录 a=1, b=1）
- `000027120800000A80000001`：对应二级索引 b=1 的新值（a=1,b=10）

计算引擎路由缓存

生成 Key 后，需要确定每个 Key 的存储节点位置，这依赖于路由功能。在 TDSQL Boundless 中，元数据管理模块 MC 掌握全局路由分布情况。这里的路由本质上是 Key 区间到节点的映射关系。

路由动态特性

相较于分库分表架构（只需确定分片对应的 DB 节点位置），弹性扩缩容的自适应调度架构具有更复杂的路由特性：

- 路由可能动态变化。
- 数据区间动态调整。
- 每个 Key Range 的大小不确定。

两级路由缓存机制

考虑到执行引擎不可能对每条 SQL 的每个 Key 都向 MC 发起 RPC 请求获取最新路由（这将显著增加系统开销），我们在 SQL 引擎内部实现了路由缓存机制。该缓存采用两级设计：

- 分桶缓存：路由缓存被分片存储，每个桶独立缓存所需路由信息。
- 全局缓存：进程内唯一的 global 级别缓存。

两级缓存的工作机制如下：

- 执行 SQL 时首先从所在分桶获取路由
- 当出现缓存未命中或路由过期时，转向 global 缓存查询。

- 若 global 缓存也未命中或路由过期，则通过 RPC 向 MC 获取最新路由。

这种分层设计既减少了大锁竞争的影响，又降低了 RPC 调用开销。

路由层级分析

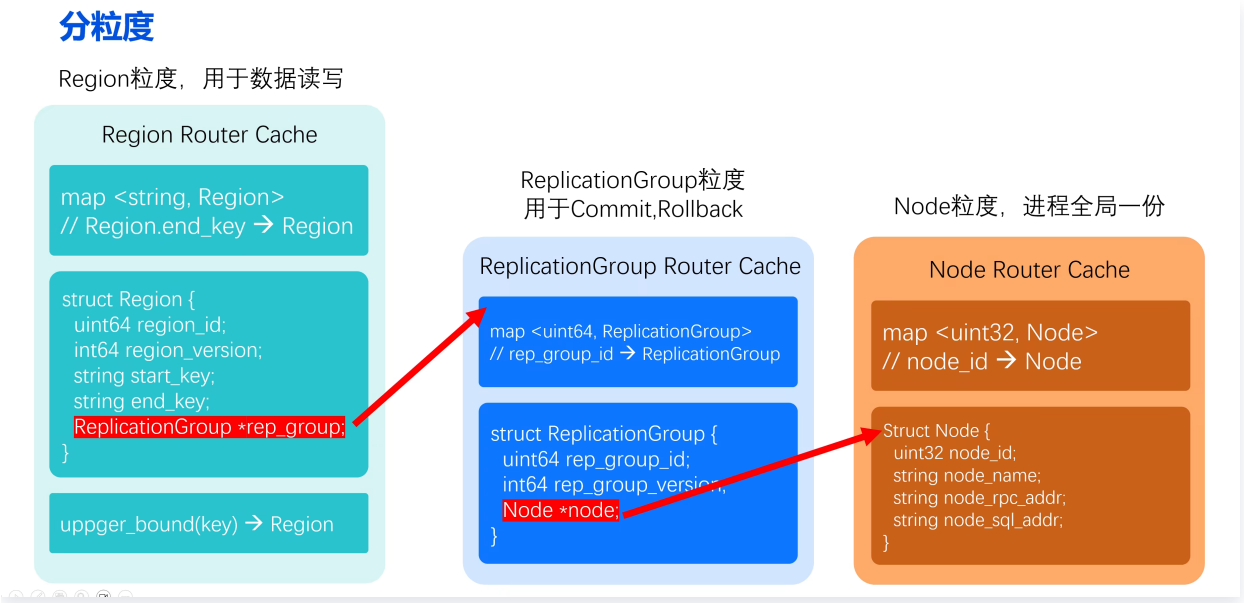
Region 粒度路由

- 是数据读写操作中最常用的路径。当执行 put 或 get 等数据读写操作时，系统通过 key 值定位数据所属的数据分片。
- 核心数据结构是 key 值到 region 的映射关系。
- 采用 upper bound 方法确定 key 所属的 region，进一步定位到具体节点。

Replication Group 粒度路由

- 主要用于事务提交和回滚时的重试操作。
- 当事务提交或回滚时，系统不再关注具体 key 值访问，而是基于已有的事务上下文，通过获取相应的 Replication Group 路由进行重试操作。

Node 粒度路由



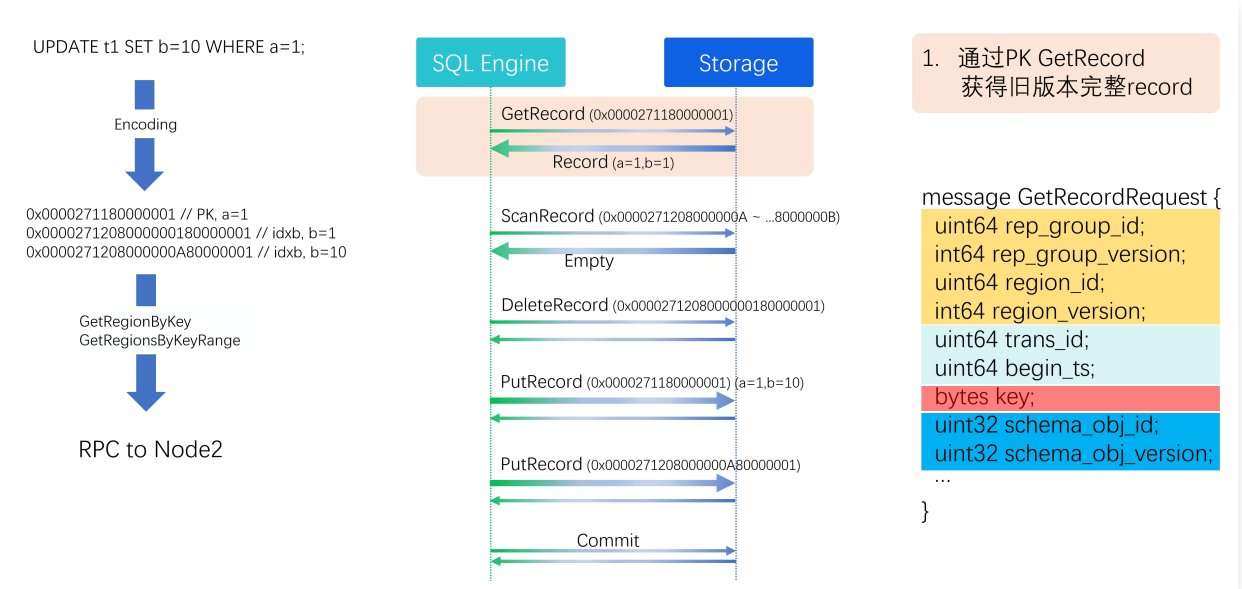
- Node 粒度路由，进程全局一份，基本只读。
- 在系统弹性扩缩容场景下，最常见的调度操作大部分集中在 region 层面（包括数据分片的切分与合并）和 replication group 层面（如主节点切换，即将整个replication group 及其下属 region 从 node1 迁移至 node2）。Node 粒度相对稳定，仅在面临资源瓶颈需要扩容时才会增加节点。

RPC 异常处理

在确定 KV 数据目标节点后，系统通过 RPC 将操作指令发送至存储引擎。以 UPDATE 操作为例，需要执行以下 RPC 流程：

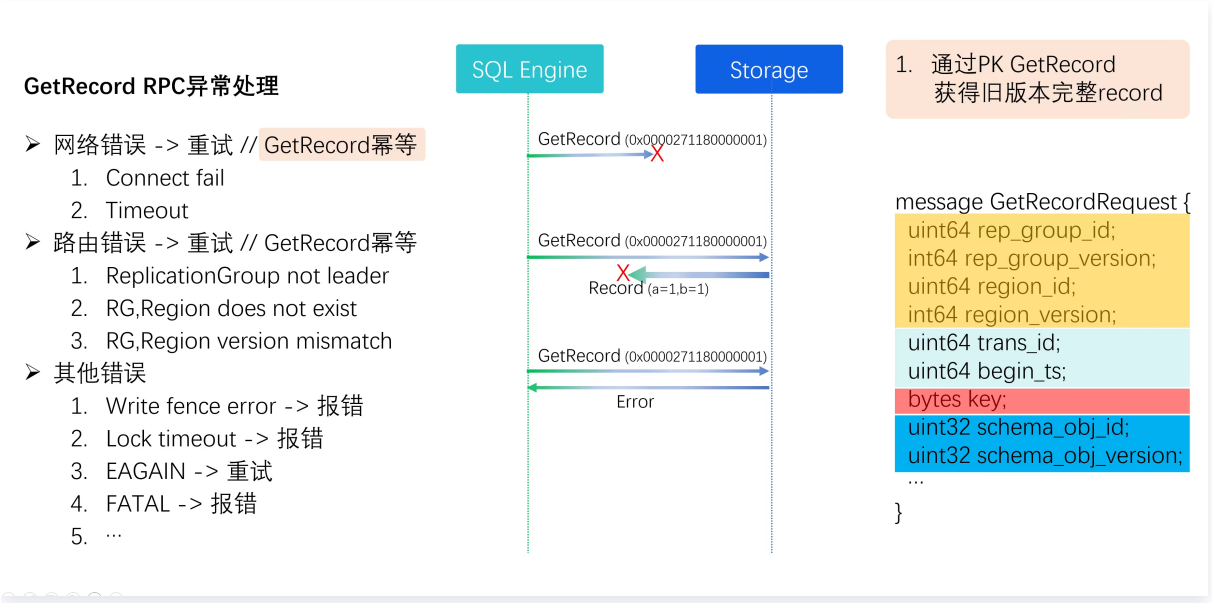
1. 通过PK GetRecord 获得旧版本完整 Record

发起 Get 操作获取旧行数据，即在主键 PK（例如字段 A）上获取完整的旧版本记录。Get RPC 消息的结构可划分为以下四个部分：



- 第一部分（黄色背景）：Replication Group (RG) 元信息，用于校验当前读写操作是否采用了最新的路由。例如存储层接收 Get 请求时，会核验请求中携带的 Region Version 是否与本地最新版本匹配。若版本不匹配，则表明使用了过期的路由访问了错误的节点。此时，SQLEngine 需要从管控节点 MC 获取最新路由，并将 Get 请求重试到正确的新位置。
- 第二部分（绿色背景）：事务快照信息。
- 第三部分（红色背景）：Get 操作的目标 Key 值。
- 第四部分（蓝色背景）：Schema Object ID 及 Version 信息，该部分与 DML/DDL 并发控制相关。

GetRecord RPC 异常处理



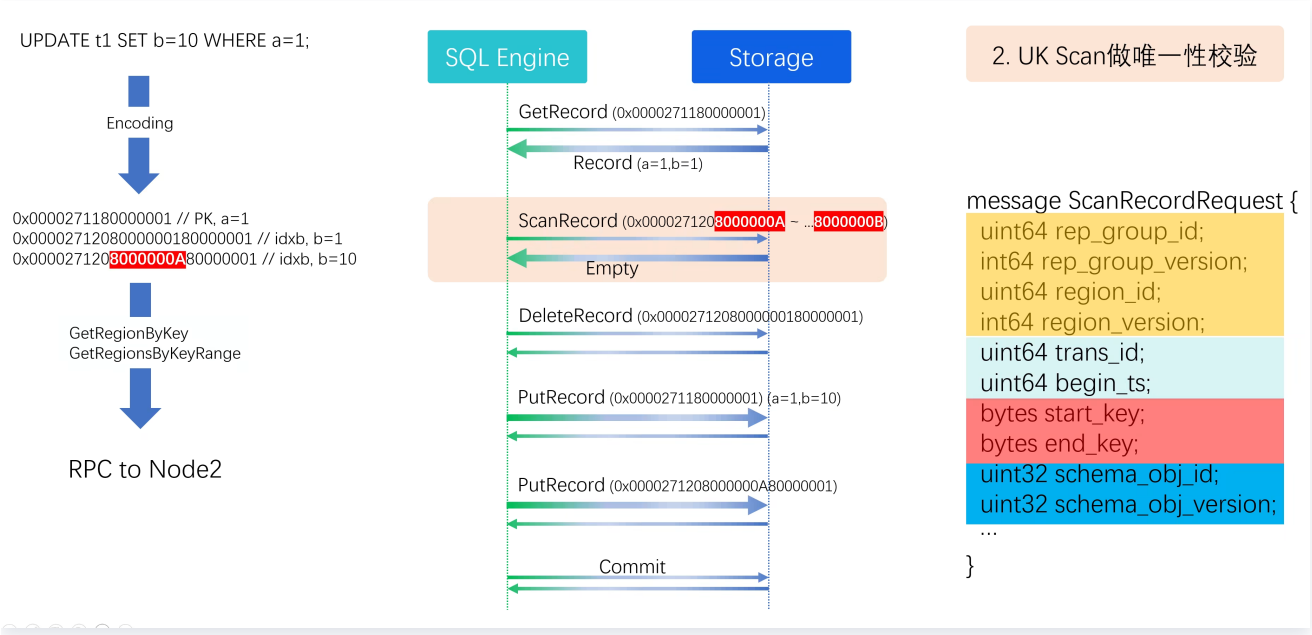
- 网络错误：
  - 错误场景：SQL 节点作为 RPC 发起方，能够区分连接失败或超时，但无法明确具体原因。

- 可能场景
  - 请求传输异常导致延迟，存储层未收到 GET 请求。
  - 存储层收到请求但回包延迟超过 RPC 时限。
  - 存储层收到请求但执行操作耗时过长，未能在时限内返回响应。
- 处理方式：由于 GET RECORD 属于单点只读操作且具有幂等性，可直接重试。

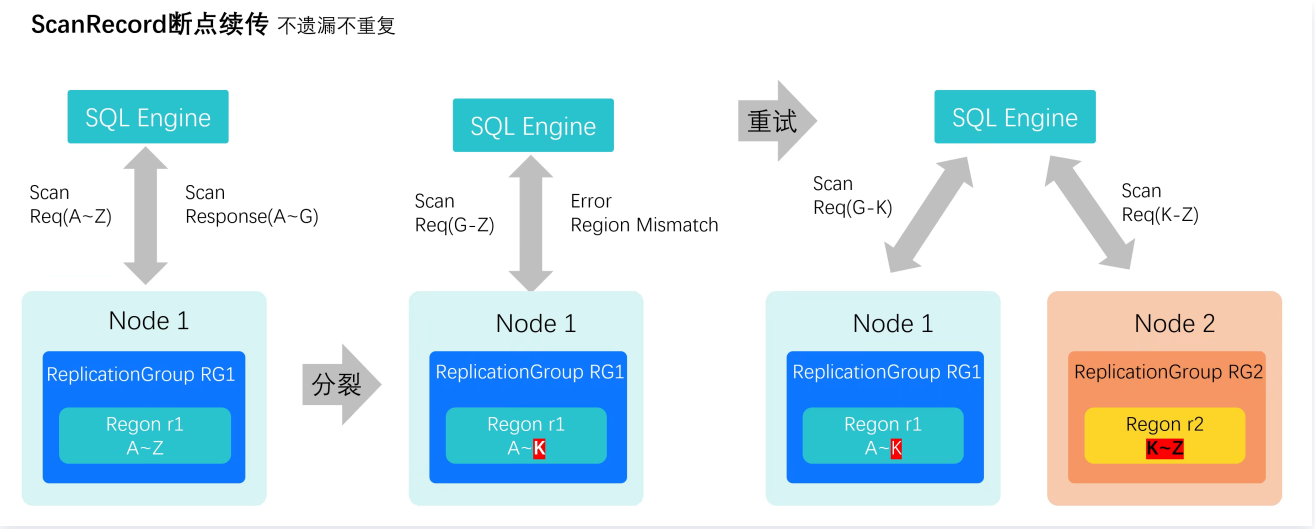
- 路由错误：
  - 处理方式：刷新路由缓存，将 GET RPC 发送至 Key 所在的最新正确路由。
  - 特性：该 RPC 具有幂等性，重试不会引发数据错误。
- 其他错误类型
  - DML 与 DDL 并发报错
  - 锁超时
  - 内部报错

2. UK SCAN 做唯一性检查

由于二级索引 b 是唯一索引，写入新数据 b=10 前需检测是否已存在 b=10 的记录。可通过 SCAN 进行唯一性检查。
SCAN RPC 与 GET 类似，但不具备完全幂等性。在进行网络错误或路由错误重试时，需关注执行进度与偏移量，避免重复查询或遗漏数据。

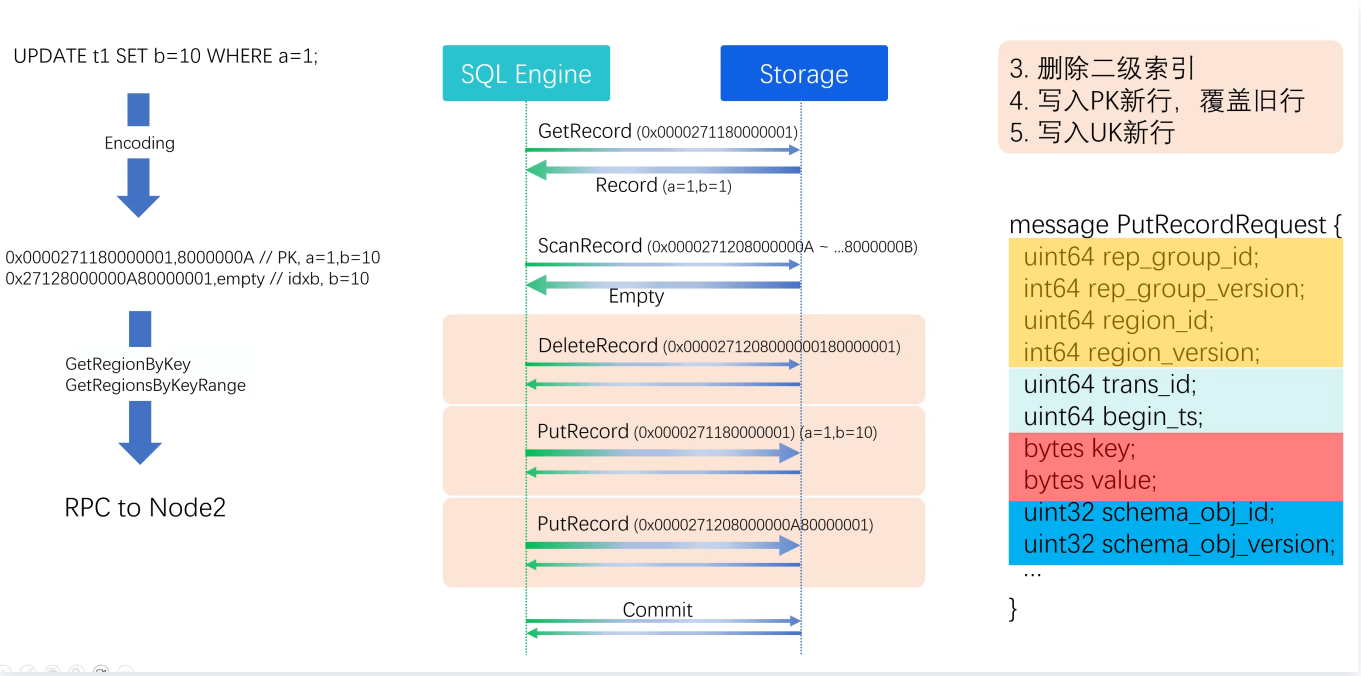


ScanRecord 断点续传示例：



1. 初始计划：扫描 A 到 Z 范围
2. 首次请求：发送至 Region1，返回 A 到 G 范围数据
3. 区域分裂：Region1 发生分裂，不再包含 G 到 Z 的全部数据
4. 错误处理
   - Region1 返回路由变化错误
   - SQL 层更新路由
   - 发现 G 到 Z 范围分布于 Region1 和 Region2
5. 续传机制
   - 拆分请求：G 到 K 发送至 Region1，K 到 Z 发送至 Region2
   - 确保扫描既不重复也不遗漏

3. 删除二级索引



4. 写入 PK 新行，覆盖旧行

写入主键的新记录。

5. 写入 UK 新行

写入二级索引新记录。对于主键来说，由于 UPDATE SQL 没有修改 a=1 的值，只要 PUT 新记录，即可覆盖旧行。

在二级索引的处理过程中，需要进行 DELETE 操作。写操作 RPC 与读操作相比，存在以下关键差异：

网络错误的处理策略

- 当写操作遇到网络错误时，系统必须断开与客户端的连接。
- 基于数据正确性保障考虑，避免重试数据包与原始数据包乱序到达存储层，从而导致数据异常。尽管 TCP 协议本身能保证数据有序传输，但在网络封装或协程框架实现层面，无法完全排除因协程调度延迟等因素，导致重试数据包比首次发送的数据包更早写入 Socket 的可能性。

安全保障机制

- 处理方式：在检测到写 RPC 网络报错时主动断开客户端连接。
- 效果：写入 RPC 连接中断后，整个事务不会提交，从而避免错误数据的产生。

DeleteRecord,PutRecord RPC异常处理

- 网络错误 -> Lost connection to Server
  1. Connect fail
  2. Timeout
- 路由错误 -> 重试
  1. ReplicationGroup not leader
  2. RG,Region does not exist
  3. RG,Region version mismatch
- 其他错误
  1. Write fence error -> 报错
  2. Lock timeout -> 报错
  3. EAGAIN -> 重试
  4. FATAL -> 报错
  5. ···

3. 删除二级索引
4. 写入PK新行，覆盖旧行
5. 写入UK新行

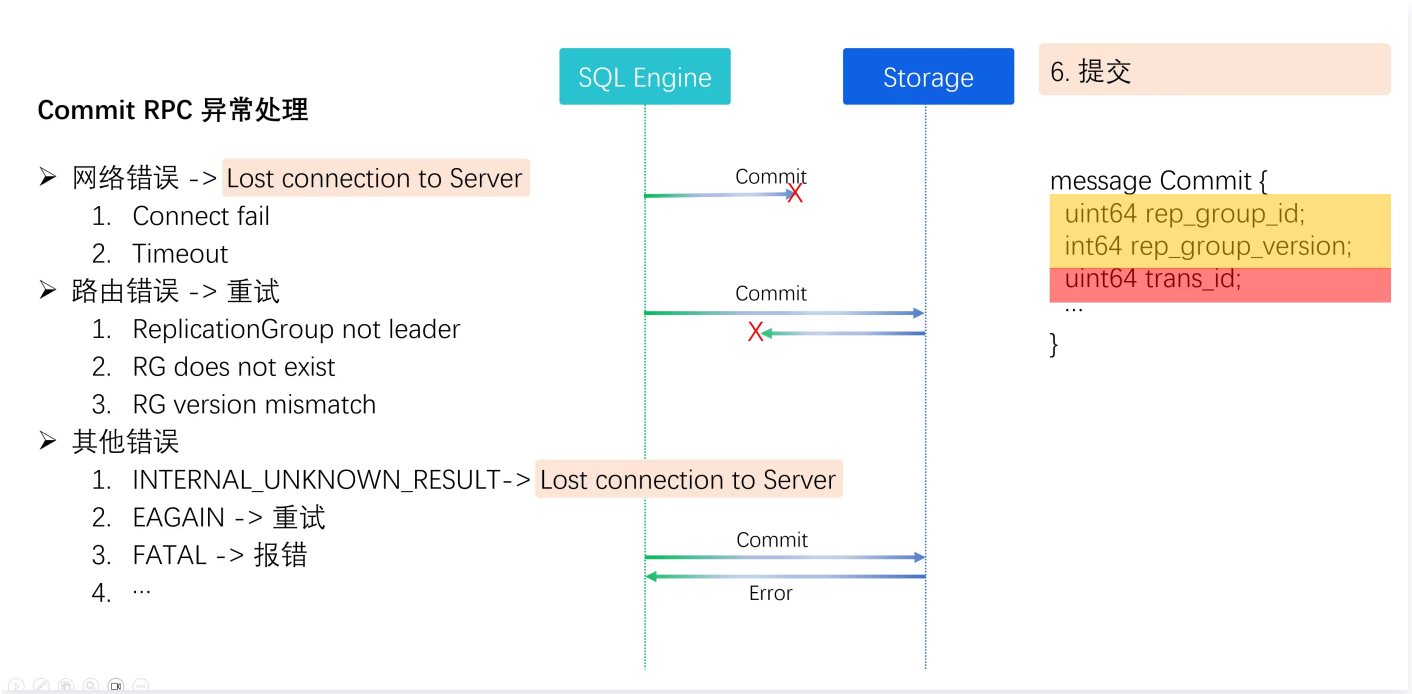
```
message DelRecordRequest {
  uint64 rep_group_id;
  int64 rep_group_version;
  uint64 region_id;
  int64 region_version;
  uint64 trans_id;
  uint64 begin_ts;
  bytes key;
  uint32 schema_obj_id;
  uint32 schema_obj_version;
  ...
}
```

6. 提交

当读写操作全部完成后，系统会提交事务：

- 事务提交的 RPC 消息相比读写操作更为简单，只需提交事务 ID 即可。
- 若事务提交过程中遇到网络错误，同样会断开连接。

状态不确定性处理：类似于单机 MySQL 执行过程中进程被误杀的场景，客户端会收到 "lost connection" 错误，无法确定事务是否执行成功，需要客户确认先前事务的执行结果。



DML 和 DDL 并发

在分布式数据库环境中，必须严格控制 DML（数据操作语言）与 DDL（数据定义语言）的并发执行，核心原因在于：DDL 操作（如增删列、改类型）会使表结构经历一个短暂的“中间状态”。如果此时 DML 操作基于旧的表结构写入数据，那么当 DDL 完成后，无论是新结构还是旧结构，都将无法正确解析这部分“穿越”过来的数据，导致数据损坏。

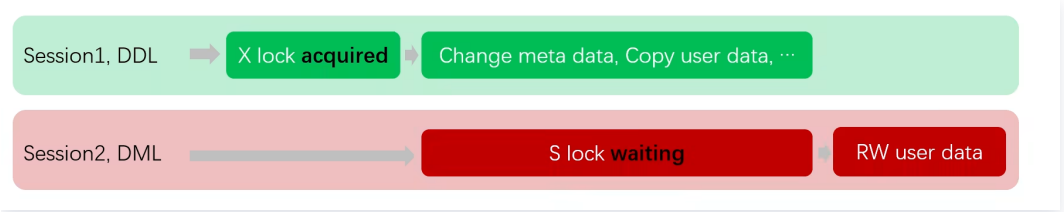
因此，任何一个健壮的数据库系统都必须建立一套机制，确保 DML 和 DDL 操作的严格隔离。

单节点环境：元数据锁 (Metadata Lock, MDL)

在单个数据库节点内，通常使用元数据锁（MDL）来协调。MDL 是一种比行锁粒度更大的锁，它保护的是表的“元数据”，也就是表的结构定义。

场景一：DDL 运行时，DML 等待

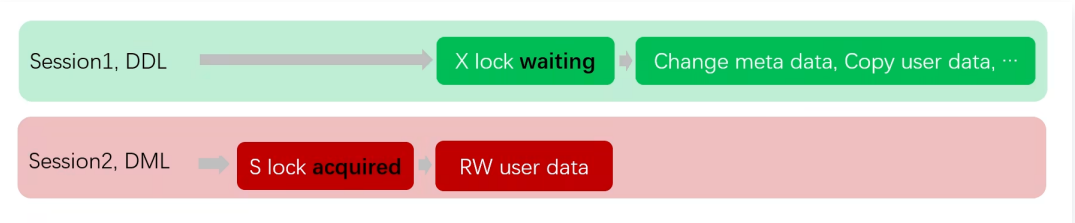
1. Session1 (DDL) 先开始，并且当前没有其他活动事务，它成功获取了表的 X lock，并开始修改元数据或拷贝数据。
2. 此时 Session2 (DML) 尝试对该表进行读写，它需要获取 S lock。
3. 由于 S 锁与 X 锁互斥，Session2 必须进入 waiting 状态，直到 Session1 的 DDL 操作完全结束并释放 X 锁。



场景二：DML 运行时，DDL 等待

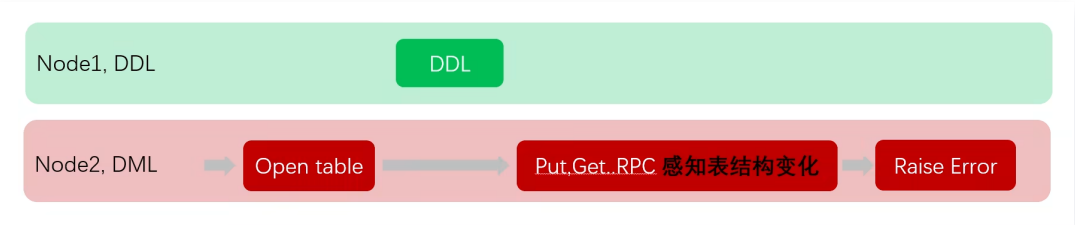
1. Session2 (DML) 先开始一个事务，执行读写操作，并成功获取了表的 S lock。

2. 此时 Session1 (DDL) 尝试修改表结构，它需要获取 X lock。
3. 由于 X 锁与 S 锁互斥，Session1 必须进入 waiting 状态，直到 Session2 提交或回滚事务，释放所有 S 锁。



分布式环境：Write Fence 机制杜绝 DML 写入旧格式数据（DDL 优先级更高）

分布式协调流程：



1. DML 启动 (Node2)：
   - 一个客户端连接到 Node2，发起一个 DML 事务。
   - Node2 执行 `Open table` 操作，从全局元数据中心获取了表的定义（我们称之为 Schema V1）并缓存在本地内存中，准备后续的数据读写。
2. DDL 并发执行 (Node1)：
   - 几乎在同一时间，另一个客户端连接到 Node1，发起一个 DDL 命令（例如 `ALTER TABLE ADD COLUMN ...`）。
   - Node1 成功执行了 DDL，将表的结构更新为 Schema V2，并同步更新了全局元数据中心。
3. 冲突发生 (Node2)：
   - Node2 上的 DML 事务处理完毕，准备通过 Put/Get RPC 调用将数据写入底层存储。
   - 关键冲突点：Node2 准备写入的数据是按照它所缓存的旧结构 Schema V1 进行编码的。但此时，整个系统（由元数据中心定义）已经认为表的结构是 Schema V2。

表版本号 (Table Version) 与写入围栏 (Write Fence)

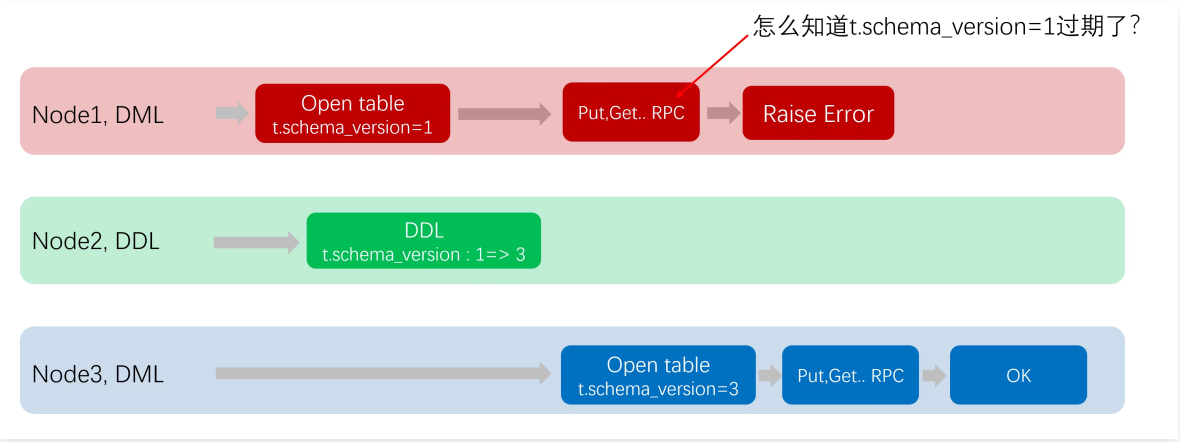
为了解决 DML 与 DDL 的并发冲突，分布式系统引入了以“表版本号”（`schema_version`）为核心的并发控制机制。它就像一个 **“写入围栏”**，能有效阻止 DML 操作向已过时的表结构中写入数据，从而保障数据一致性。其核心设计思想如下：

- **DDL 的职责**：当执行修改表结构的操作时（如新增列、索引等），DDL 事务将该表的 `schema_version` 推进到新版本，并持久化到数据字典（Data Dictionary）中。
- **DML 的职责**：在执行 `Open table` 操作时，不仅需要获取表的分区信息和索引信息，还需要同时获取对应的 `schema_version`。

以下是一个具体示例：

1. **Node1 (DML):** 客户端连接 `Node1` 执行 DML，在 `Open table` 阶段缓存了表的旧版本号 `t.schema_version=1` 。
2. **Node2 (DDL):** 与此同时， `Node2` 上成功执行了 DDL。此操作将表的版本号从 `1` 推进到 `3` 。
3. **Node3 (DML):** `Open table` -> 获取 `t.schema_version=3` -> 发起 `Put/Get RPC(data, version=3)` -> Region Leader 校验 `3 == 3` -> OK。
4. **Node1 (DML):** 发起 `Put/Get RPC(data, version=1)` -> Region Leader 校验 `1 != 3` -> 拒绝请求，返回 Raise Error。

因此，问题进一步转化为：DML 如何判断当前持有的 `schema version` 是否已过期?



DML 如何感知表结构变化

最初，我们考虑让 DML 客户端在每次读写操作前，主动向“数据字典”（Data Dictionary）查询最新的表结构版本。但这个方案存在两个致命缺陷：

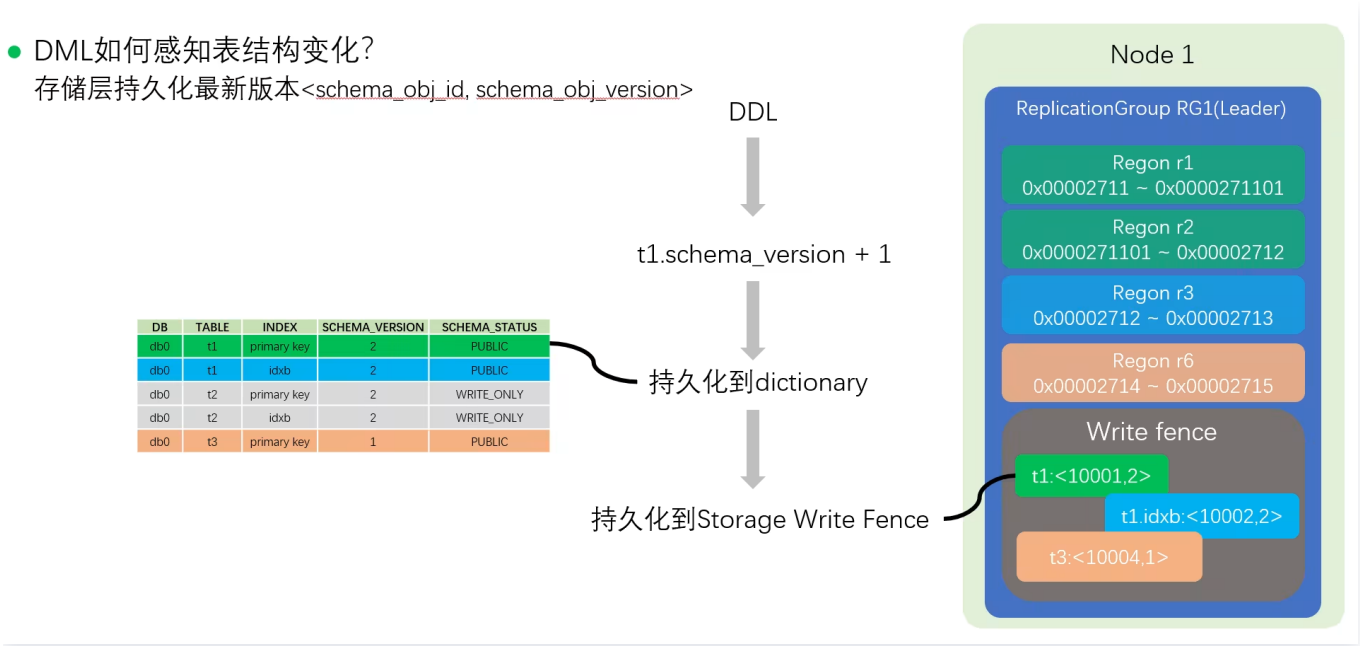
- **性能开销 (Performance Overhead):** 每次 `put` 或 `get` 操作都需要一次额外的 RPC 来访问数据字典，这会极大地增加系统延迟，降低吞吐量。
- **原子性问题 (Atomicity Problem):** 在“查询最新版本”和“执行写入 RPC”之间存在一个时间窗口。如果在这个窗口期内，表结构再次被 DDL 修改，系统依然无法保证数据的一致性。

为了解决上述问题，我们采用了将版本校验责任**下沉至存储层**的方案。该机制的核心是在存储节点上引入一个名为**“写入围栏”（Write Fence）**的结构。

`Write Fence` 的本质是持久化存储了每个表/索引对象（ `schema_obj_id` ）与其当前最新的版本号（ `schema_obj_version` ）的映射关系，形成一个 `<schema_obj_id, schema_obj_version>` 的元组。

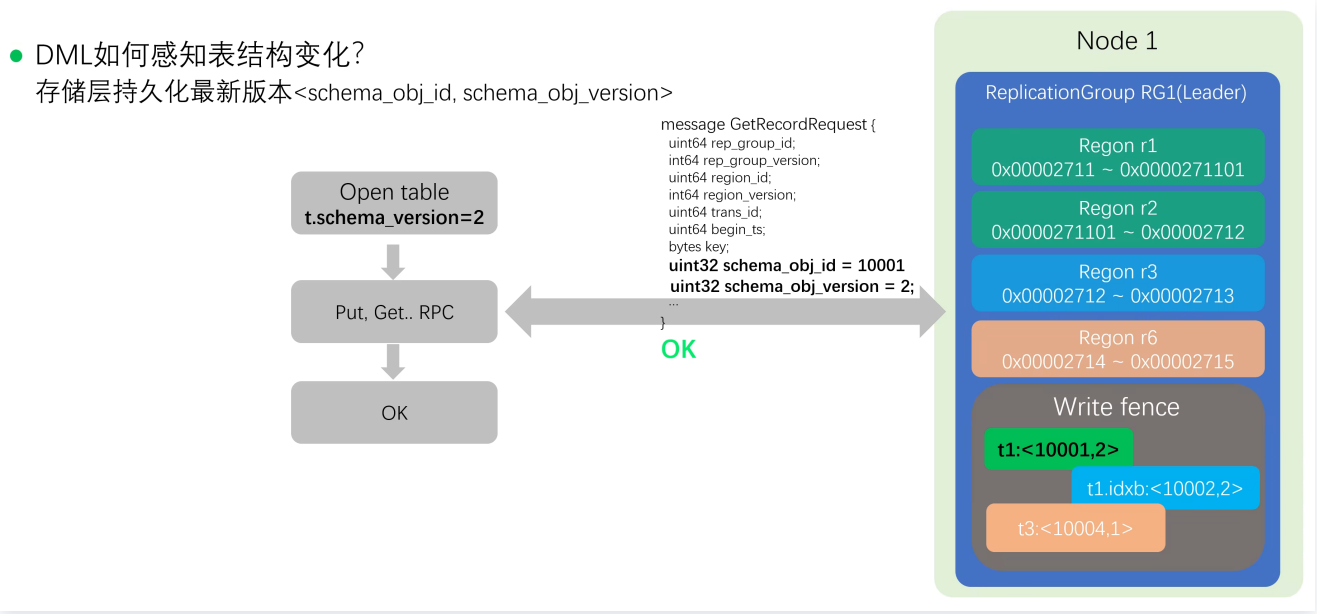
当一个 DDL 事务执行时，它必须原子地完成两件事：

1. **更新数据字典：** 将最新的 `schema_version` （例如从 1 更新到 2）持久化到数据字典中。
2. **更新写入围栏：** 将最新的版本元组（如 `<t1: <10001, 2>>` ）持久化到所有相关数据分片（Region）所在的存储节点（ `Node 1` ）的 `Write Fence` 中。



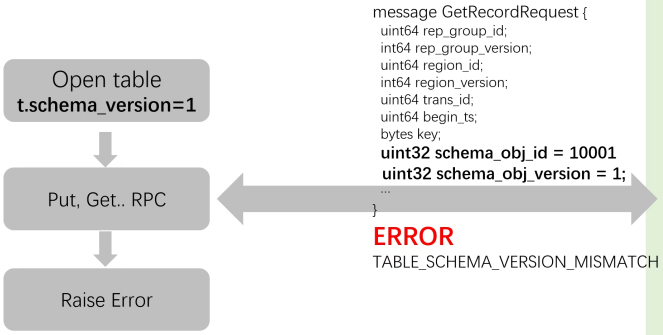
DML 执行与校验流程如下所示：

1. **请求准备：** DML 客户端在 `Open table` 时获取到表结构版本，例如 `t.schema_version=2` 。
2. **RPC 携带版本：** 在发起 `Put/Get` RPC 时，请求体（如 `GetRecordRequest` ）中会携带 `schema_obj_id` 和 `schema_obj_version` （例如 `schema_obj_id = 10001` ，`schema_obj_version = 2` ）。
3. **存储层原子校验：** 存储节点在收到请求后，会将 RPC 中的版本元组 `<10001, 2>` 与其本地 `Write Fence` 中存储的元组进行比对。
   - **版本匹配：** 如果版本一致，说明 Put 请求获取的表结构是最新的，操作被允许执行（ `OK` ）。



   - **版本不匹配：** 如果版本不一致（例如请求携带 `version=1` ，而 `Write Fence` 中是 `version=2` ），说明 DML 持有的表结构已过时。存储层会拒绝该请求，并向上层抛出错误，触发整个 DML 或事务失败并重试。

- DML如何感知表结构变化?
  存储层持久化最新版本<schema_obj_id, schema_obj_version>

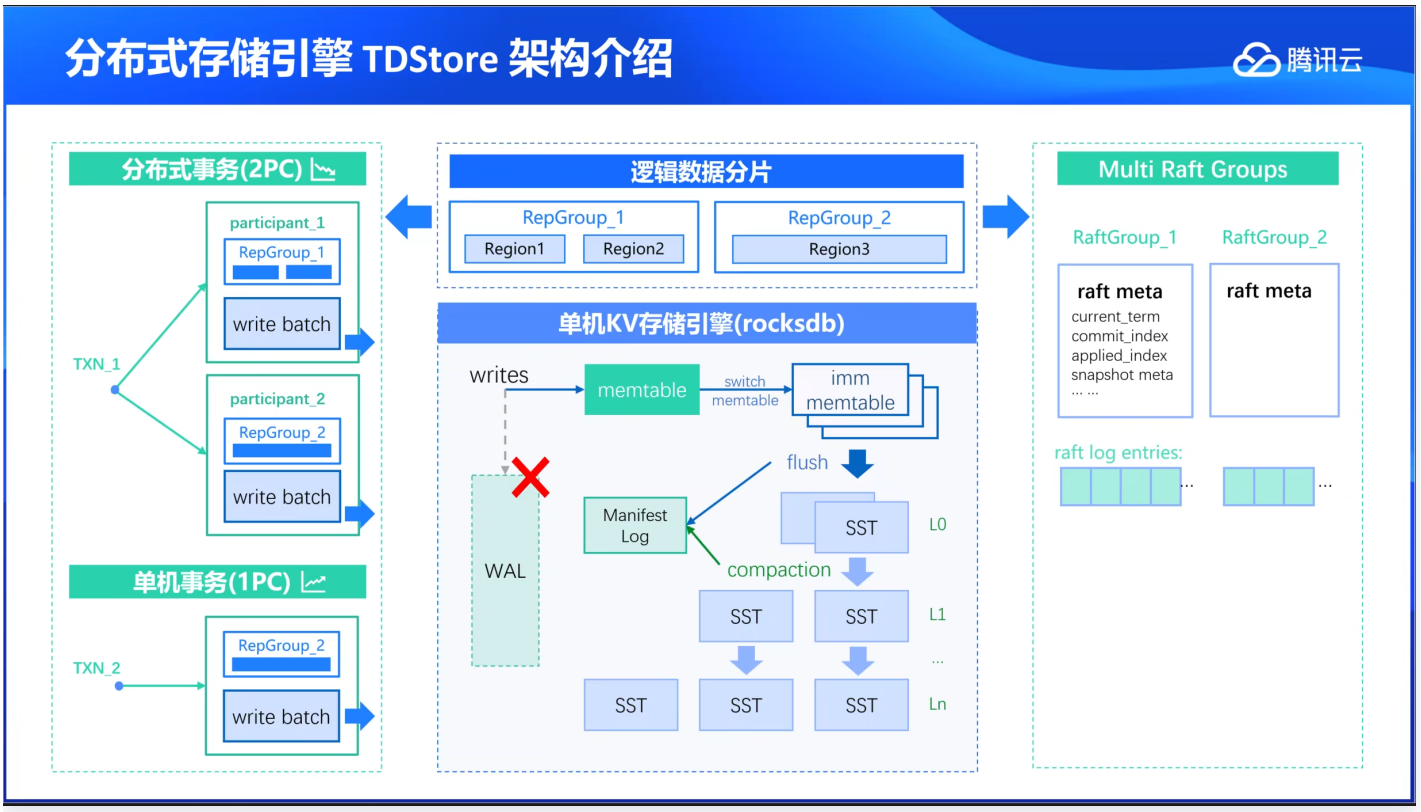


存储引擎TDStore实现之道
存储引擎架构介绍

最近更新时间：2025-12-24 12:03:44

分布式存储引擎 TDStore 架构

TDSQL Boundless 的存储引擎为 TDStore。TDStore 是一款分布式存储引擎，它在 RocksDB 基础上，上层构建了**数据分片模块**、**分布式事务模块**，下层增加了 **Raft 共识协议模块**，把一个单机的存储 KV 引擎改造成了支持高扩展、**高可用**、**分布式事务**、**数据均衡调度**、**多副本强一致**的分布式 KV 存储引擎。



- **单机 KV 存储引擎（RocksDB）**
  - 接收计算层传递的 KV 请求。
  - 使用 LSM-Tree 结构存储数据。
- **逻辑数据分片**
  - 支持以数据分片为单位进行调度迁移。
  - 实现数据在集群节点间的灵活调度。
- **分布式事务**
  - 维护分布式事务在各数据分片上的参与者上下文以及事务状态等信息。
  - 支持将相关性数据调度至同一数据分片，从而将更多的分布式事务转换为单机事务，来提高事务的执行效率。

- **多副本容灾层（Multi-Raft）**
  - 把每个数据分片以 Raft Group 方式在不同 TDStore 节点上创建多副本。
  - 每个数据分片作为独立日志流进行同步。

低成本海量存储

TDStore 存储层基于 LSM-Tree + SSTable 结构存放和管理数据，具有极高的压缩率，能有效降低海量数据的存储成本，单实例可支撑 PB 级别的存储量。
在不同场景中 TDStore 的压缩效率请参见 数据高压缩比 。

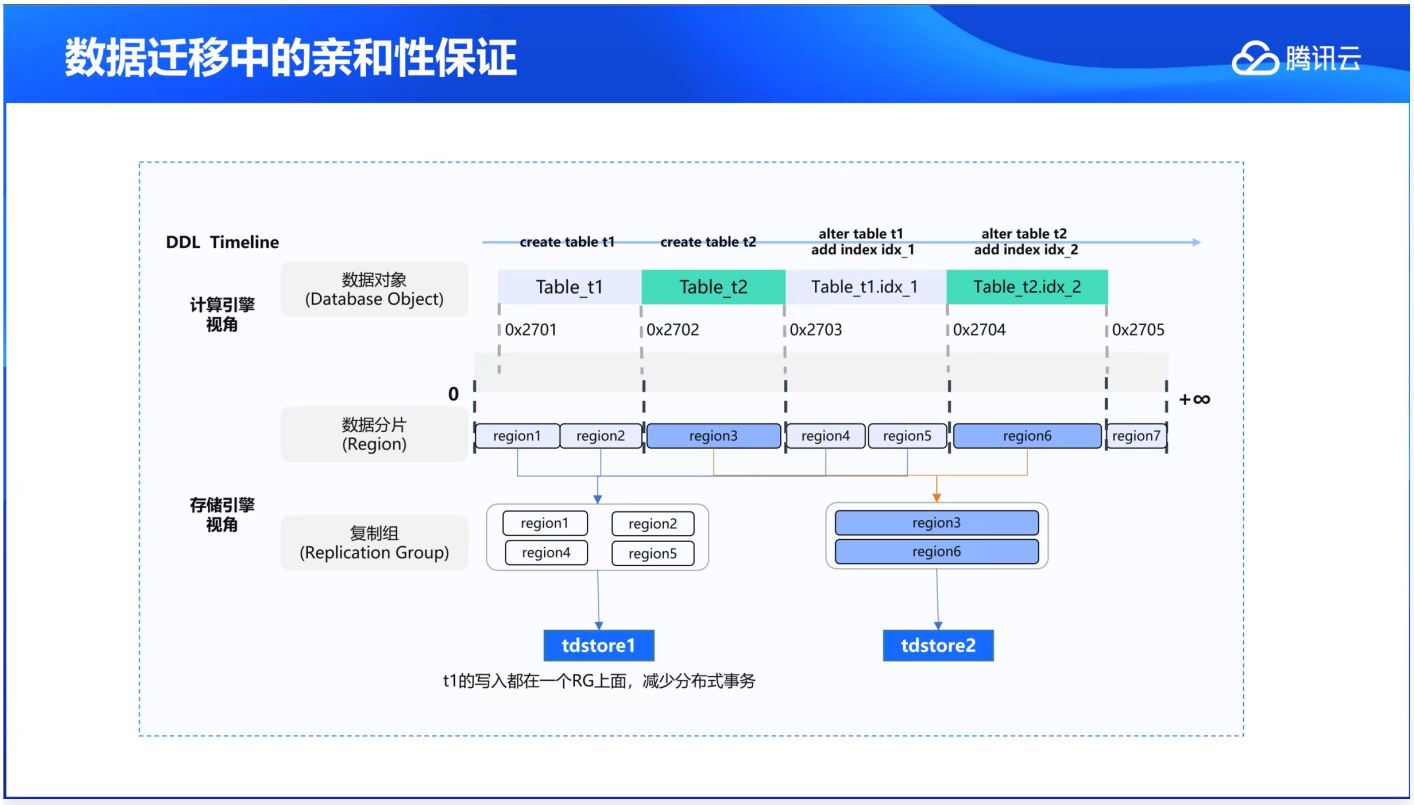
数据分片划拨方式

最近更新时间：2025-12-24 12:03:44

数据迁移中的亲和性保证

- **数据对象（Database Object）**：包含表、分区、索引等数据库对象。每一个表、分区、索引包含了多个行（记录），每条记录以键值对（KV）形式编码，保证同一表或索引的数据编码连续。如图中，`Table_t1` 和 `Table_t2` 分别有主键索引 和二级索引，每个索引对应连续的 key 区间。
- **数据分片（Region）**：一段连续的，左闭右开的 key 空间。单个分片最多包含一个数据对象的数据，支持将数据对象拆分为多个分片，保证数据管理的粒度化和规整性。
- **复制组（Replication Group）**：对应一个 Raft 日志流，管理多个不同 Region 的数据。支持数据亲和性调度，将关联数据置于同一复制组。
- **数据亲和性**：数据之间存在相关性，例如：表内部的主键和二级索引，不同表之间的连接键，将相关的数据调度到同一个复制组，进行优化。

基于**三层数据模型**与**一体化对等架构**两个设计，我们可以做出灵活弹性的调度：包括利用数据亲和性消除分布式事务。



基于数据感知的调度

典型案例分析：

1. 用户创建包含 ID、姓名、年龄的表（表 ID 为100）
2. 在年龄列创建二级索引（索引表 ID 为200）

3. 主键索引 key 前缀为00000064，二级索引 key 前缀000000C8，两者 key 范围相距较远

传统方案：同时向两个独立的 Raft 日志流写入数据，必须通过两阶段提交（2PC）完成分布式事务，事务执行效率显著低于单机事务。

架构演进：基于数据感知的调度

腾讯云

数据的编码与划分策略 – 分布式事务

tindex id = 100

PK=id	ROW
1	{1, “张三” ,15}
2	{2, “李四” ,30}
3	{3, “王五” ,23}
4	{4, “赵六” ,45}

tindex id = 200

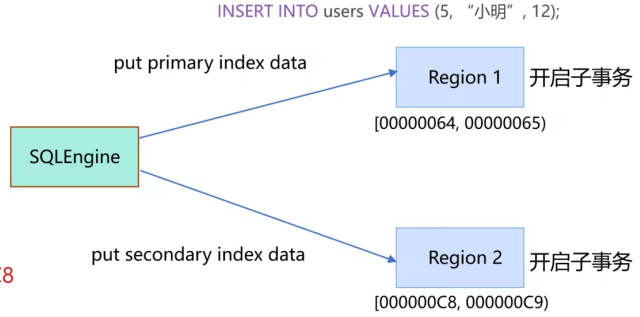
SK = age	PK
15	1
23	2
30	3
45	4

```
CREATE TABLE users(
   id INT NOT NULL,
   name VARCHAR(20) NOT NULL,
   age INT NOT NULL,
   PRIMARY KEY(id)
);
ALTER TABLE users ADD INDEX index_age(age);
```

主键索引的编码
- key = tindex id + primary index value
- value = row data

二级索引的编码
- key = tindex id + secondary index value
- value = primary index value

user 表主键 tindex id = 100，对应 key prefix = 00000064
user 表 age 索引 tindex id = 200，对应 key prefix = 000000C8



TDSQL Boundless 优化方案：

- 将关联数据（主键与二级索引）置于同一复制组。
- SQLEngine 执行查询读写时，仅需向目标复制组发送请求，事务提交时也只需与单个复制组交互，通过 1PC 协议极大提升事务处理效率

架构演进：基于数据感知的调度

腾讯云

数据分片分为两个层级进行调度 – 降低分布式事务比例

PK=id	ROW
1	{1, “张三” ,15}
2	{2, “李四” ,30}
3	{3, “王五” ,23}
4	{4, “赵六” ,45}

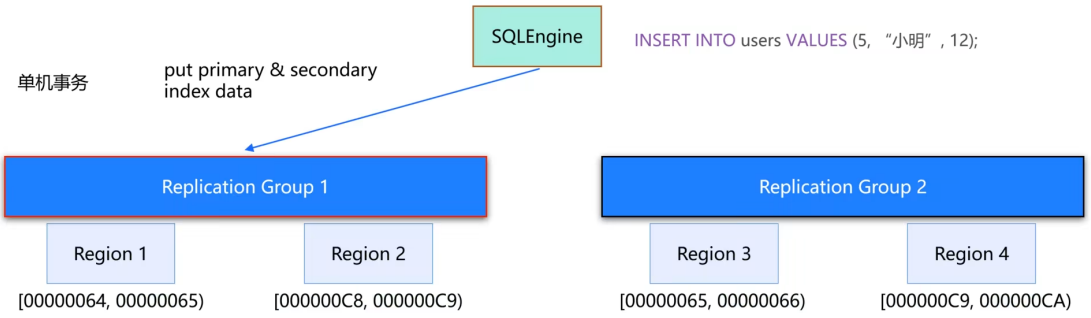
key prefix = 00000064

SK = age	PK
15	1
23	2
30	3
45	4

key prefix = 000000C8

基于数据感知的调度策略将数据管理分为 key range region + replication group两个层次：

- key range region 是**数据划分**单位，每个 region 管理一段连续的数据分片
- replication group 是**数据同步**单位，每个 replication group 对应一个 raft group，包含若干 key range region

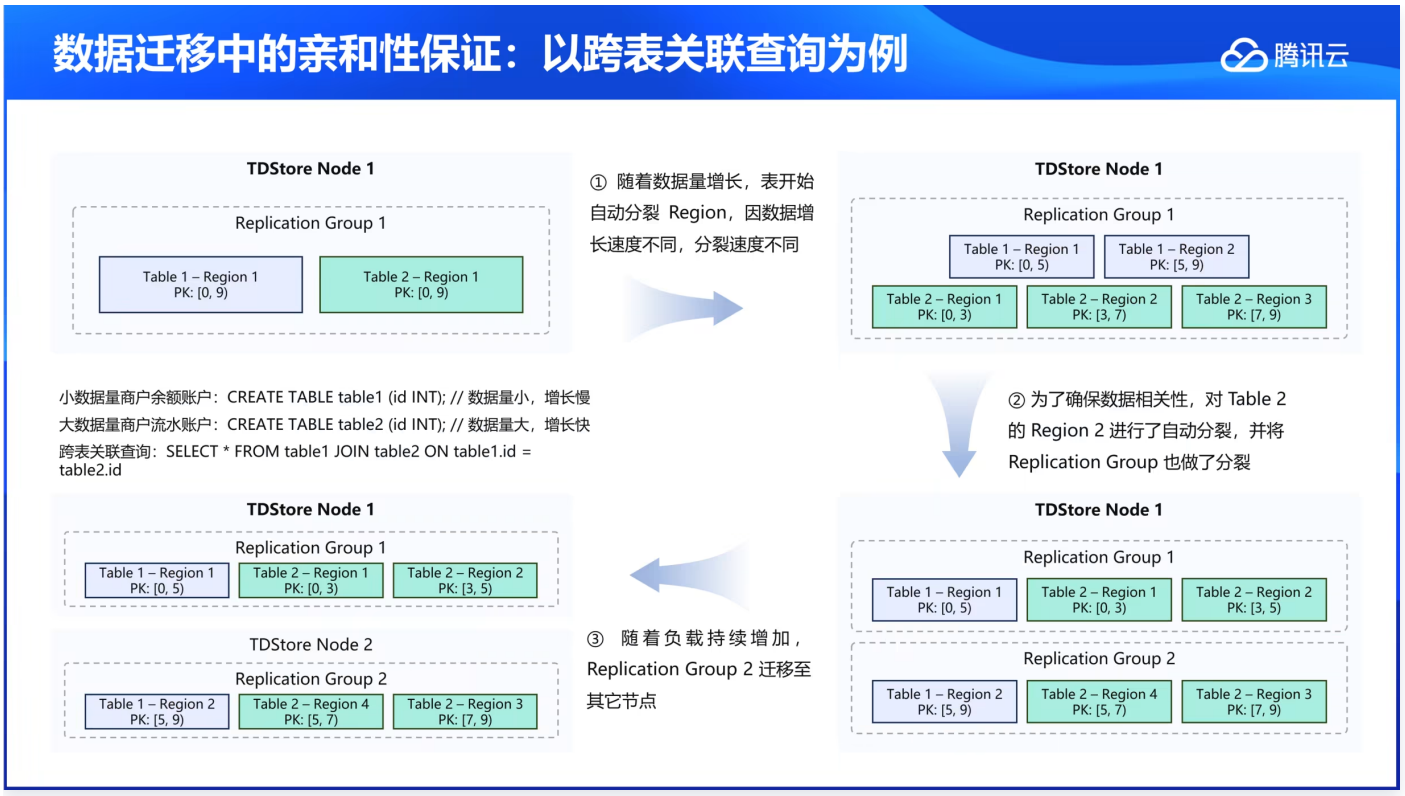


数据迁移中的亲和性保证：以跨表关联查询为例

初始状态：小数据量商户余额账户表与大数据量商户流水账户表共存于同一节点。

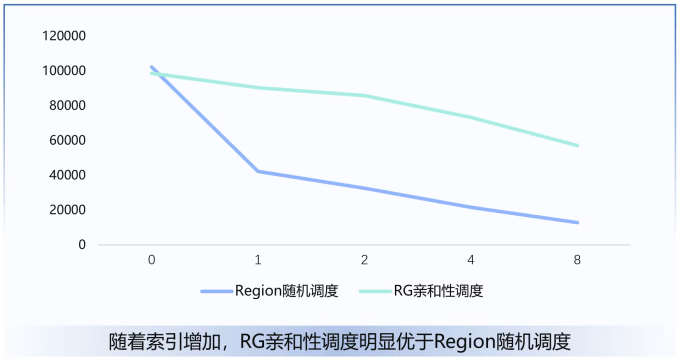
数据增长处理：

1. 随着数据量增长，表开始自动分裂 Region，因数据增长速度不同，分裂速度不同。
2. 为了确保数据相关性，对 Table 2的 Region 2进行了自动分裂，并将复制组也做了分裂。
3. 把相同数据范围的数据调度到同一个复制组（如 RG1：数据范围0-5；RG2：数据范围5-9）。当业务进行跨表关联查询的时候，只需要访问同一个复制组就可以达到查询的目的。
4. 随着负载持续增加，RG2迁移至其它节点。实现数据拆分的同时保证单个复制组内数据亲和性。

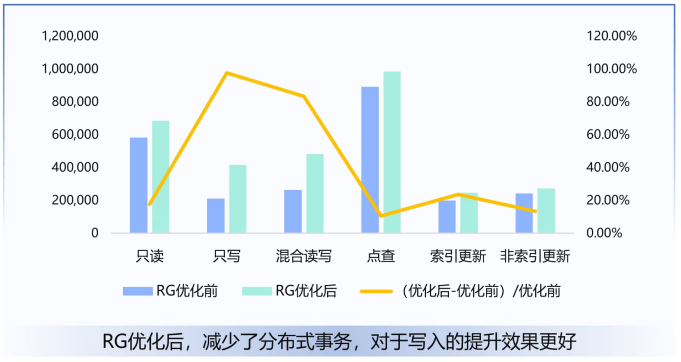


数据迁移中的亲和性保证：收益总结

亲和性保证可以使得经常被共同操作的数据存放在同一个节点，这使得它们的读写涉及更少的数据移动和 RPC 调用，也可以避免分布式事务，大幅度提升性能。



随着索引增加，RG亲和性调度明显优于Region随机调度



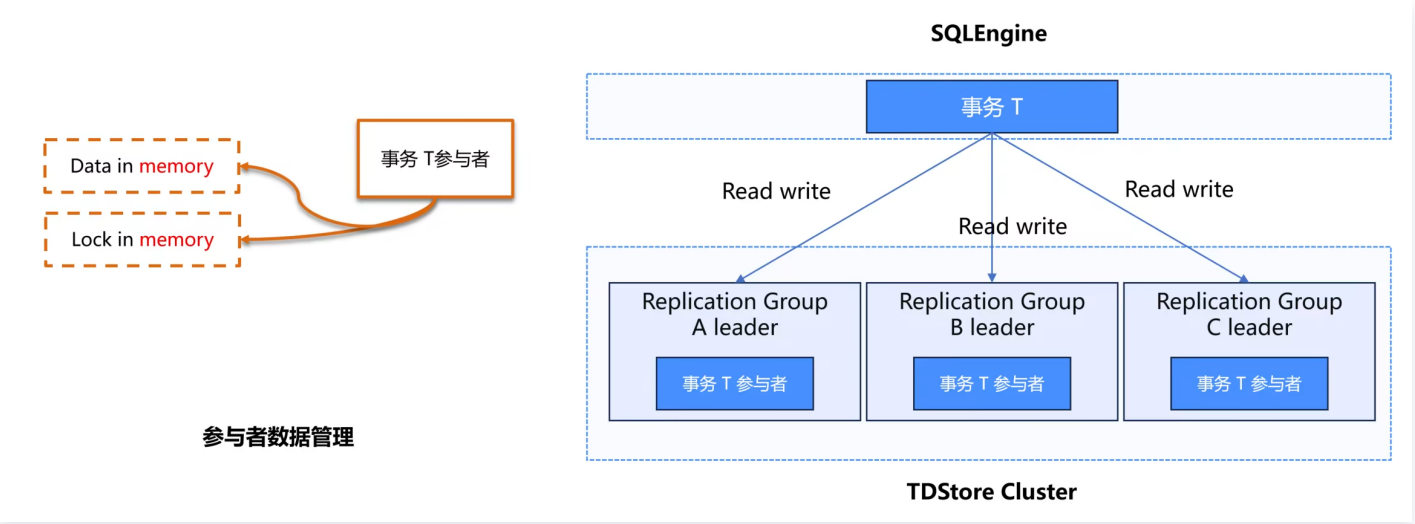
RG优化后，减少了分布式事务，对于写入的提升效果更好

分布式事务原理

最近更新时间：2025-12-24 12:03:44

分布式事务模型：读写阶段

分布式事务设计旨在同时保证正确性与高效性。随着业务规模不断扩大，对数据库系统负载能力提出更高要求。为应对此类高并发场景，必须对事务的读写和提交链路进行全面优化。



- **参与者管理机制**：以复制组（Replication Group）为基本单位管理事务参与者。当事务需要访问特定复制组数据时，自动创建对应的参与者上下文。SQLEngine 的一个事务单元，访问 TDStore 集群上的多个参与者，每个参与者管理一个 Replication Group 的数据。
- **事务模型特性**

  事务在执行第一条语句的时候，会从 MC 获取时间戳，作为读快照。事务执行读写请求的时候，会去访问数据对应的 Replication Group 的 Leader 副本，并在该副本上创建一个参与者上下文。参与者上下文位于内存中，会缓存事务在该 Replication Group 上写入的数据，以及持有的悲观锁。事务提交时，会选择其中某一个参与者担任协调者，并向该协调者发送提交请求，请求中带有该事务访问过的所有 Replication Group 的 ID。协调者收到后，会按照 2PC 流程，保证所有参与者数据提交的原子性。在 2PC 的 commit 阶段开始前，协调者会从 MC 再次获取一个时间戳，作为 commit_ts，这个 commit_ts 会随着 2PC 的 commit 请求下发到各个参与者，并作为数据写入 RocksDB 的 sequence number。在后续做可见性判断的时候，会将读操作的读时间戳和各个版本的 sequence number 做比较，对于 sequence number 小于读时间戳的所有版本，选取 sequence number 最大的版本读取。
  - **数据缓存机制**
    - 事务数据在提交前完全缓存在内存中。
    - SQLEngine 层不缓存事务数据，由 TDStore 实现分散缓存。
    - 仅在事务提交阶段进行数据持久化操作。
  - **内存管理机制**
    - 事务在内存中的数据（包括未提交的数据和悲观锁信息等）大小被严格限制

- 正在推进大事务数据提前落盘功能，减缓内存使用压力，从根本解决问题

分布式事务模型：提交阶段

2PC 下沉

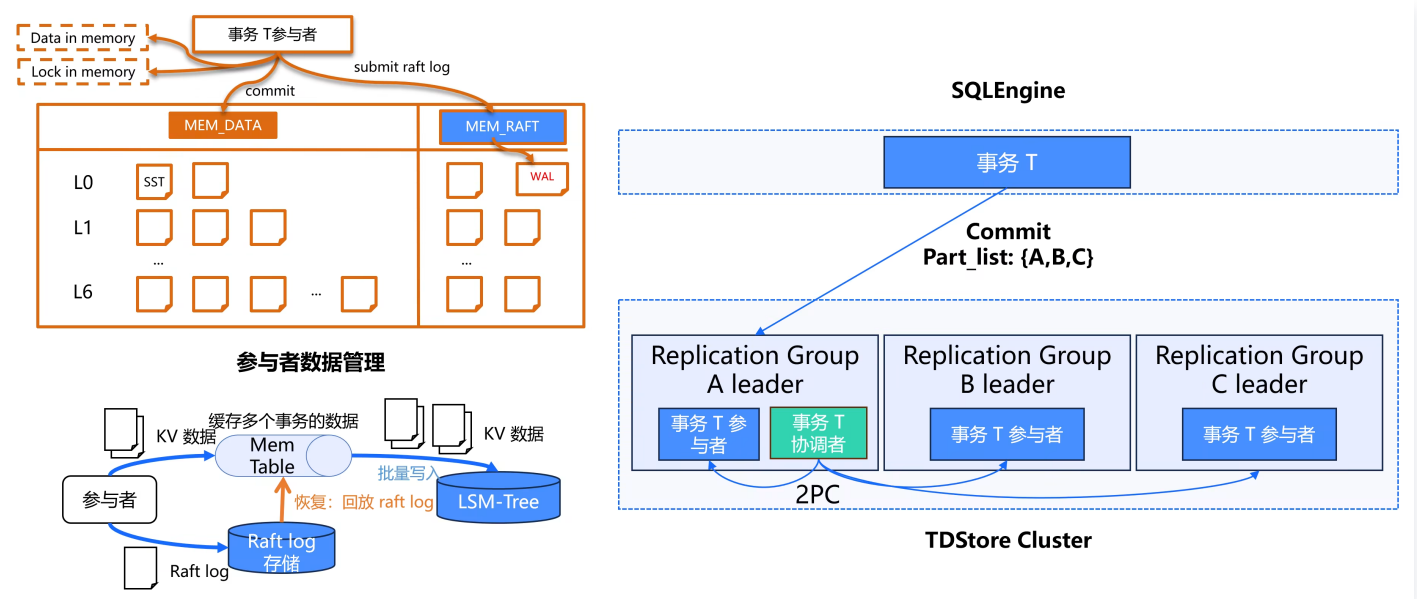
2PC 的实现被完全下沉到 TDStore 层，SQLEngine 不感知事务提交流程。

1. SQLEngine 提交分布式事务时，从参与者中选取一个节点作为协调者。
2. 向选定的参与者节点发送提交请求，请求中包含事务涉及的所有参与者列表。
3. 接收请求的节点在其复制组内创建协调者上下文。
4. 由协调者负责向其他参与者发送读写请求，推进完整的 2PC 流程。

数据持久化

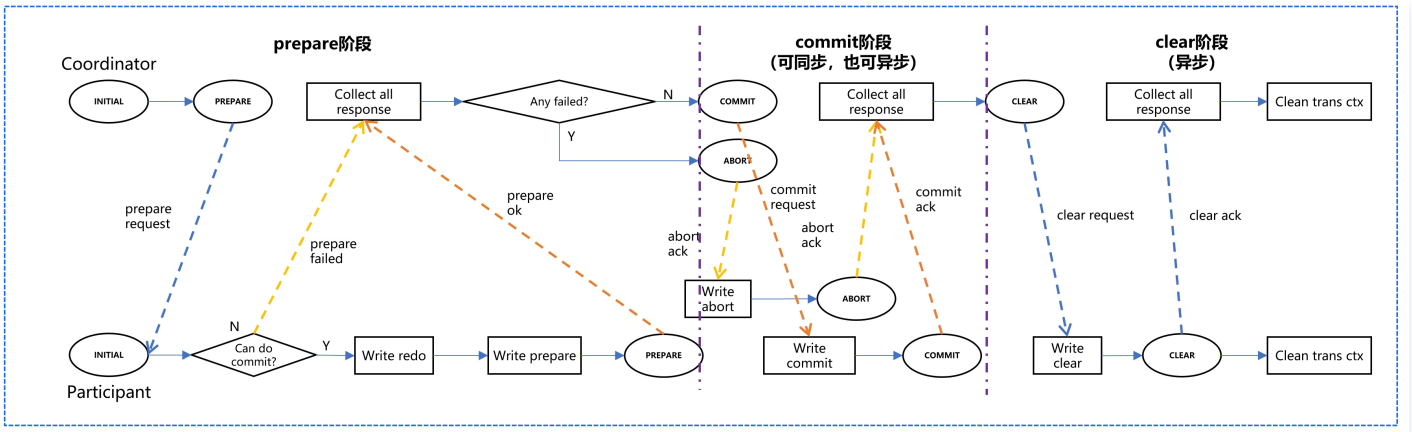
TDStore 直接使用 Raft log 作为 WAL 日志，数据写入 LSM-Tree 不需要额外写 WAL 日志。具体做法：

1. 节点重启时从上一记录点回放 Raft Log。
2. 数据刷新到磁盘后推进日志点，减少宕机时需回放的日志量。
3. 通过单一 Log 实现备机数据同步和故障恢复双重功能。相当于减少了额外写入 WAL 的步骤，也达到了减少写数据量的目的。



分布式事务提交原理

- **传统 2PC 的瓶颈**：在传统的两阶段提交实现中，协调者和参与者都需要同步日志，导致整个流程需要5次日志同步操作，这在性能上存在明显瓶颈。
- **TDSQL Boundless 核心优化：实现了分布式事务下沉，使用协商式 2PC 避免协调者同步日志带来的开销，保证了跨 Replication Group 的事务的原子性。**



	日志式	协商式
RPC 轮数	2轮 RPC：prepare、commit	3轮 RPC：prepare、commit、clear（异步）
日志同步次数	参与者2次 + 协调者3次 = 5次日志	参与者同步3次日志（包含1次异步日志）
故障处理	宕机重启，参与者/协调者回放本地日志即可确定状态	• 宕机重启，参与者回放本地日志即可确定状态，**协调者需要通过参与者发送的消息确定状态** • 参与者必须定时向协调者发消息告知状态

数据分片高效调度

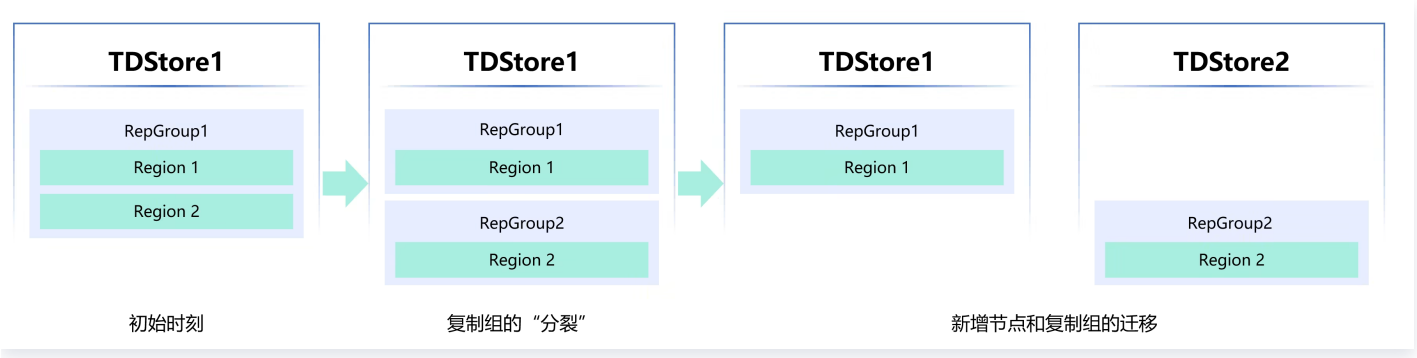
最近更新时间：2025-12-24 12:03:44

弹性扩缩容

TDSQL Boundless 的核心特性之一是实现业务无感知的扩缩容，确保在扩缩容过程中业务请求不受性能抖动影响。下面详细解析数据分片与复制组的调度机制如何实现这一目标。
扩缩容就是数据分片 – 复制组 – 物理节点三者关系的重组，存在以下几种事件：

- 数据逻辑关系变化
  - **数据分片(Region)的分裂和合并。**
  - **数据分片(Region)在复制组(Replication Group)上的转移。**
  - **复制组(Replication Group)创建和删除。**
- 数据物理关系变化
  - **复制组(Replication Group)迁移，即复制组的成员变更。**

无感知调度流程



1. 初始状态：系统包含一个复制组（RG），该复制组内包含两个连续 Region（Region1、Region2）。
2. 复制组分裂：将 Region2 从原复制组中拆分出来，创建新的复制组专门管理 Region2 的数据。
3. 数据迁移：将新复制组的数据迁移到其他物理节点，通过 Raft 协议完成数据同步与切主。

业务价值

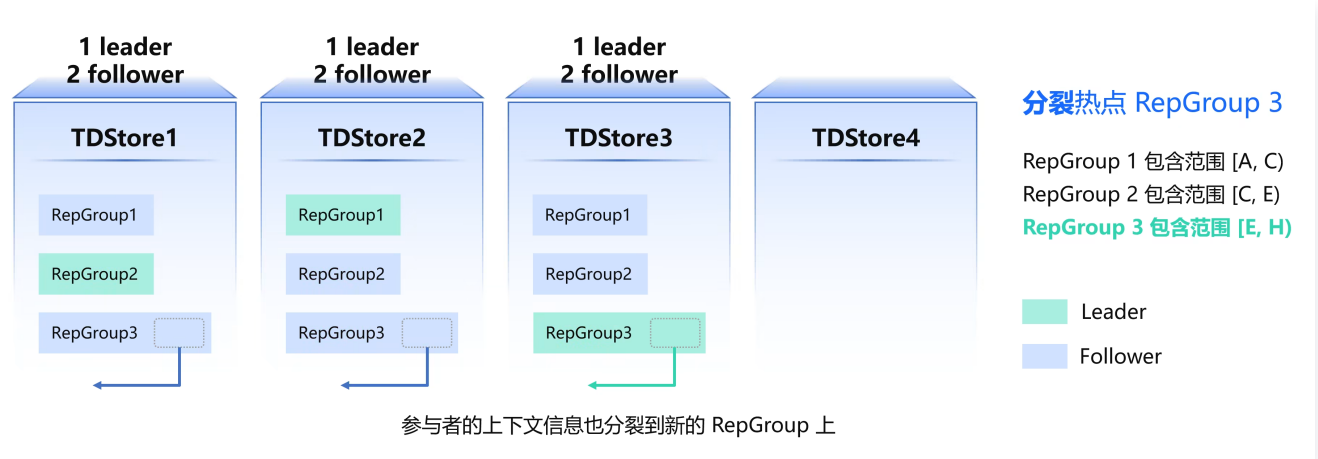
- **业务**：腾讯计费平台发票流水业务，主要为外部用户和腾讯内部系统提供实时交易订单查询及发票开具服务。数据量30T (三副本)，单表80+亿行。写入峰值为1w/s；查询平均300 – 400笔/s，数据量增长迅速。
- **诉求**：需要方便、快速进行扩容。
- **效果**：在线水平扩容从月缩短至分钟级，业务无须感知。

弹性扩缩容：分裂

分裂是将单个 Replication Group 拆分成两个，拆分得到的两个 Replication Group 各自管理拆分前的一半数据。

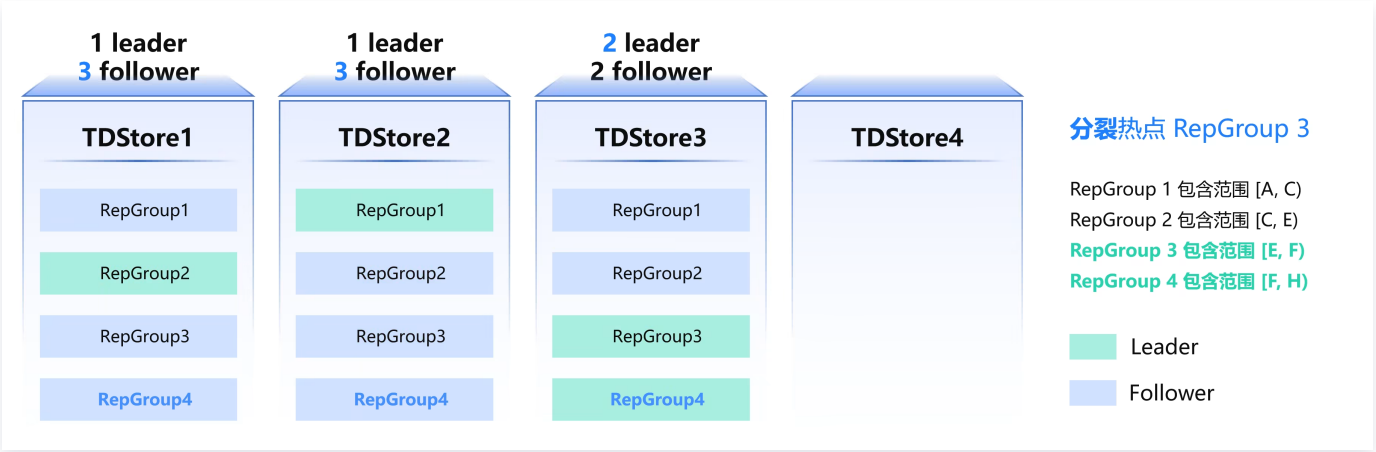
1. 初始状态：3个 TDStore 节点（TDStore1、2、3）；新增 TDStore4 节点，分摊系统压力；TDStore1、2、3 上存在数据量大的 RG3。
2. Replication Group 分裂：
   - Replication Group 分裂时，将一部分 Key Range Region 划分到新 Replication Group 上。
   - Replication Group 分裂时会构建一个新 Raft Group，但是不会对本地数据做修改。

   如下所示，对 RG3 副本执行分裂；将 RG3 上未提交的事务上下文按拆分范围同步迁移至新创建的 RG4 副本，确保所有未提交数据完整迁移，避免事务中断或数据丢失。



3. 分裂得到的两个 Replication Group，将热点分散到多个 Replication Group 上，分别承担分裂前一部分的压力。

   分裂流程仅在上层逻辑模块进行复制组拆分，底层 KV 数据未发生物理变动，对系统 IO 的影响较为有限。



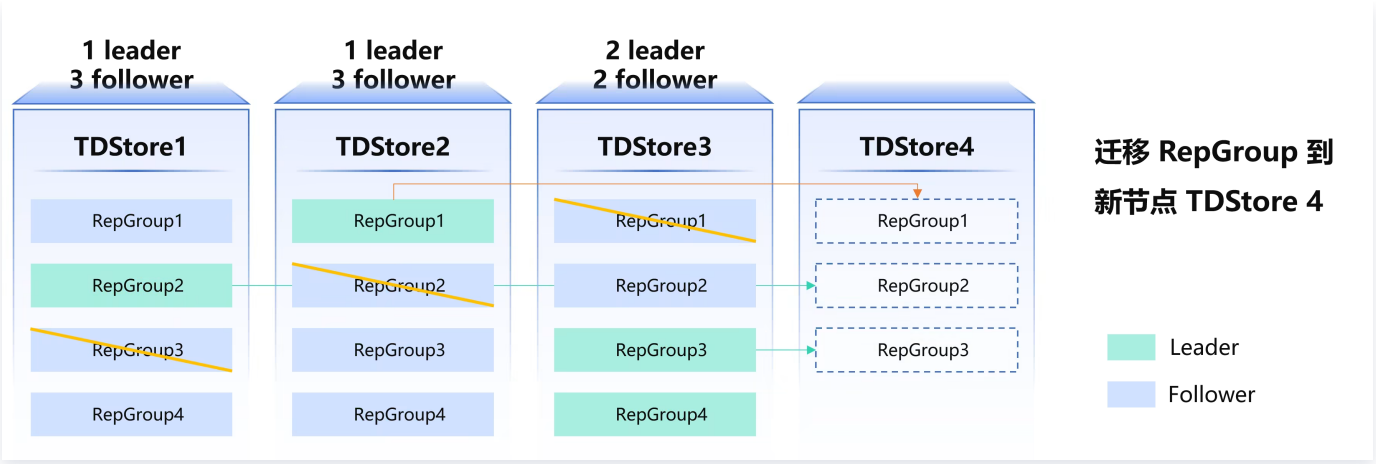
弹性扩缩容：迁移

迁移是指将 Raft Group 的某个 follower 副本转移到新节点的过程，核心价值在于：

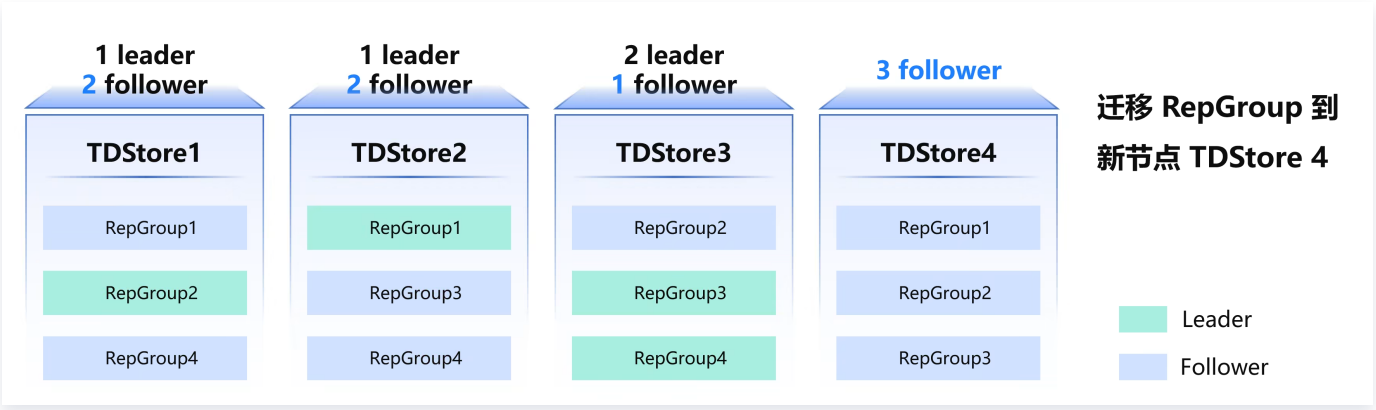
- 数据分散：通过迁移 Replication Group，实现数据在不同 TDStore 节点间的均衡分布
- 读写无感：follower 的创建与销毁不影响 leader 的正常读写操作

MC 下发迁移任务流程如下：

1. 在目标节点为待迁移的复制组创建新副本。
2. Leader 执行 install snapshot 流程，将完整 RG 数据同步至新副本，确保新副本数据达到最新状态



3. 删除待迁移的旧副本，通过"先增后减"策略实现平滑迁移。将 TDStore1、TDStore2、TDStore3 的计算压力均衡分配至 TDStore4，减轻 leader 副本的读写压力，实现集群整体负载的均衡分布。



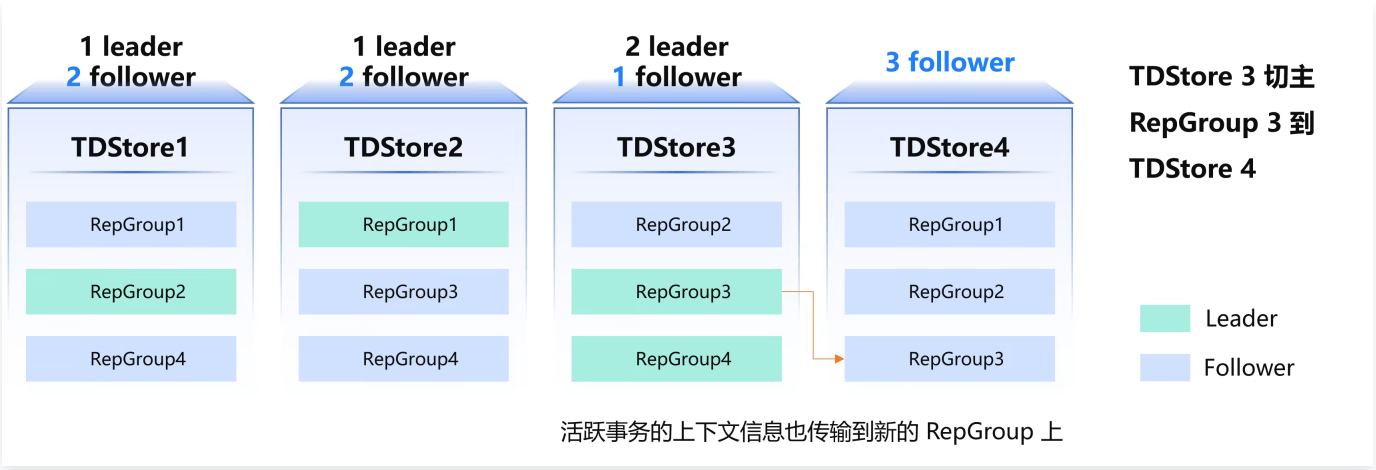
所有迁移过程对业务无感知。所有增减副本操作仅在 follower 进行，迁移全程对访问 leader 的业务完全透明。

弹性扩缩容：切主

切主是指将 Raft Group 中的 Leader 角色切换到另一个 Raft 副本的过程。核心目的是通过分散 Leader 热点，实现负载均衡。

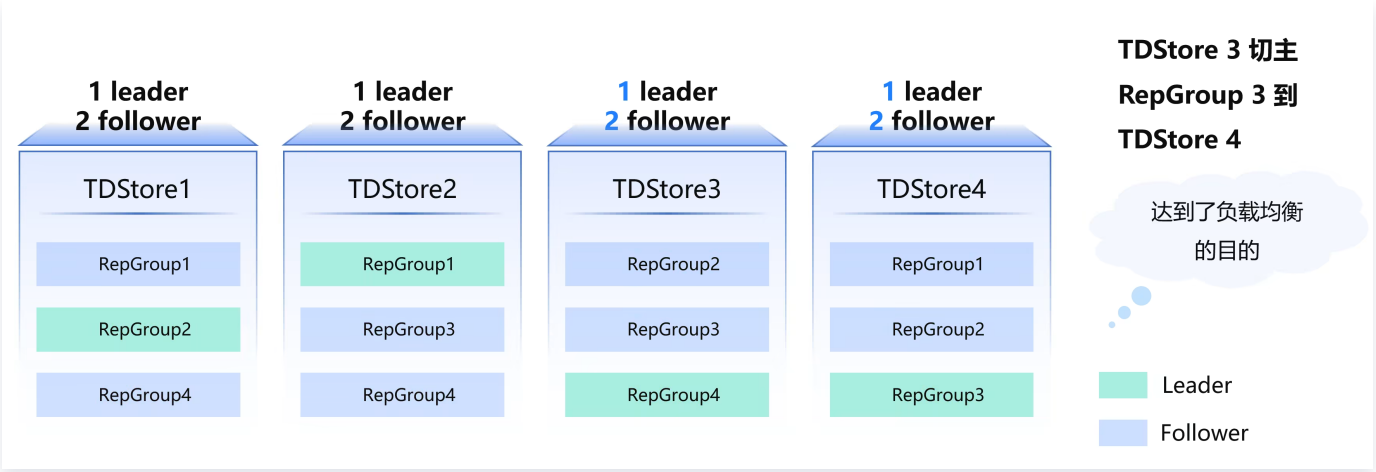
如下所示， TDStore3 节点同时承担两个 Leader 副本和一个 Follower 副本，读写压力集中，可通过切主将部分 Leader 转移至其他节点（如 TDStore4），从而均衡集群负载。切主执行流程：

1. 将原 Leader（如 TDStore3 上的 RG3）的未完成活跃事务上下文信息提前发送给目标节点（如 TDStore4），确保事务状态无缝迁移。



2. 在事务信息同步完成后，于 Raft 层触发正式的 Leader 切换操作。

   由于提前同步了事务信息，切主过程中不会终止任何进行中的事务，保证了业务的连续性。



调度引擎：元数据管理和均衡调度原理剖析

架构/元数据模型

最近更新时间：2025-12-24 12:03:44

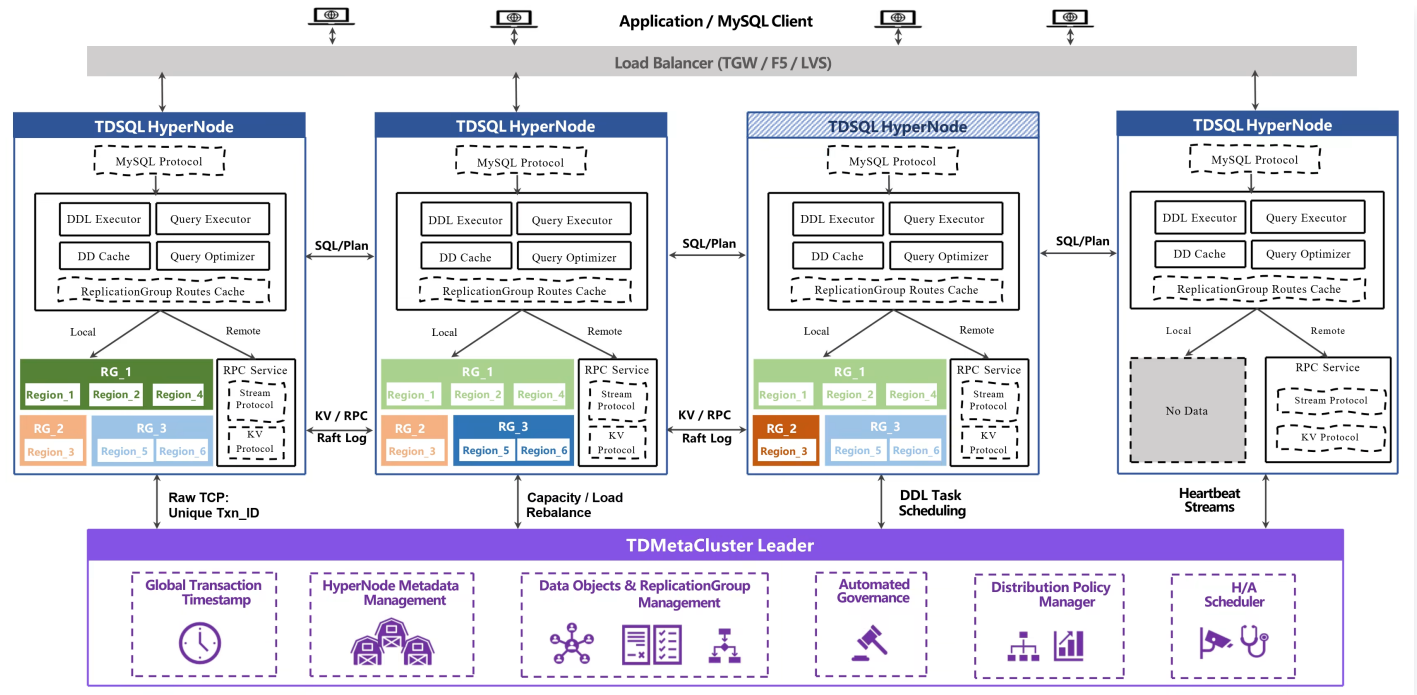
数据库架构演进

- **集中式架构（单机数据库）**
  - 性能与可靠性依赖特定硬件：扩展性差，存在单点故障风险。
  - 弹性极差：当数据量从 GB 级增长至 TB 级（如几十 TB）时，单机硬件无法承载。
- **分布式数据库中间件架构**（TDSQL MySQL（InnoDB））
  - 优势：使用廉价硬件，降低了系统成本。
  - 痛点：
    - 中间件本身复杂：需要大量人力进行维护。
    - 对业务侵入性强：建表时必须显式指定分片键；扩容时需要预先定义和执行复杂的拆表方案。
    - 功能受限：跨库查询、分布式事务实现复杂，影响研发效率
- **原生分布式**（TDSQL Boundless ）
  - 原生 SQL 引擎：业务无需像使用中间件那样指定分片键或修改业务逻辑，对业务完全透明，如同使用单机数据库一样简单。
  - 弹性伸缩架构：基于 TDStore 存储引擎的先进分片管理机制，实现类似 Manner 架构的智能化数据分布，使业务能够无感知地进行扩缩容。
  - 降低综合成本：基于 LSM-Tree 存储结构构建，并且在每一层的数据都提供压缩算法，进一步压缩业务数据，从而降低综合成本。

TDSQL Boundless 总体架构

TDSQL Boundless 一体化存算分离的架构实现，很好地解决了 TDSQL MySQL（ InnoDB 引擎）在语法兼容性、弹性扩缩容能力上的短板。

- **计算引擎**： 访问远端的分布式 KV 存储，每条记录由计算层编码为 `<Key, Value>`；计算层的访问空间为 $[0, +\infty)$ 的线性 key 空间。
- **存储引擎**： 分布式 KV 存储集群，作为事务协调者提供 Get/Put 接口；数据分片以 Key-Range 范围切分，由调度引擎控制分片粒度以及存放位置。
- **调度引擎**： 承接计算引擎和存储引擎的枢纽，维护和更新全局路由信息，调度和管理数据分片生命周期，实现业务无感知扩缩容。



DDL 操作流程（以建表为例）：

1. 业务发送 CREATE TABLE 语句 → 计算引擎
2. 计算引擎解析 DDL → 生成 DDL 任务
3. MC 接收 DDL 任务 → 与 TDStore 交互
4. 创建对应数据分片 → 更新路由信息
5. 返回执行结果 → 业务端

DML 操作流程（以数据插入为例）

1. INSERT 语句 → 计算引擎
2. 数据编码为 KV 格式 → Key: "table:pk", Value: "row_data"
3. 查询 MC 获取路由 → 确定目标分片位置
4. 通过 Put 接口 → 写入对应数据分片
5. 二级索引同步 → 写入索引分片

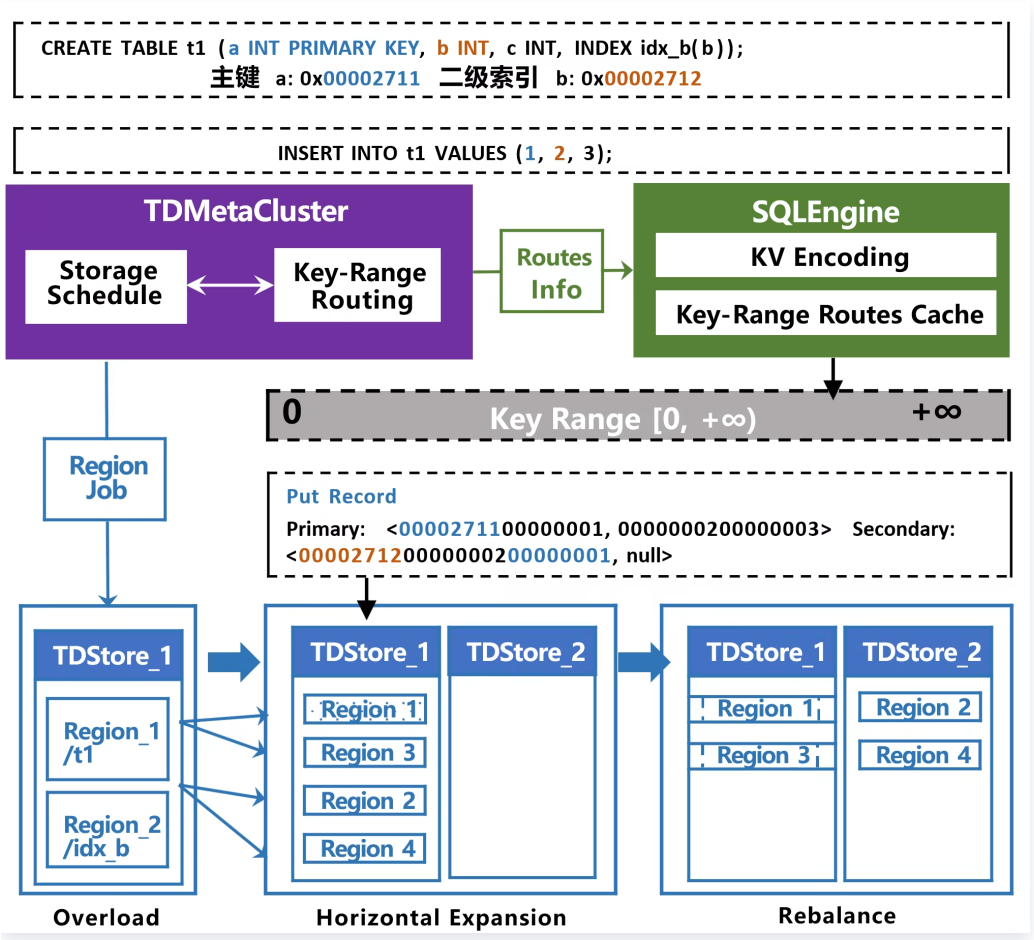
MC（调度引擎）的核心作用是为数据访问提供路由信息，并智能管理数据分片（Region）。当集群需要扩容加入新节点时，MC 会自动、平滑地将部分数据迁移到新节点上，使所有存储节点的负载趋于均衡。整个过程对业务应用完全透明，无需人工干预，实现了真正的“无感知”扩缩容。

TDSQL Boundless 元数据模型

面对分布式架构三大挑战：感知缺失、调度受限、规则固化，TDSQL Boundless 通过三级元数据模型破局。

- **DataObject**：表、索引、自增值等逻辑层面的概念抽象，定义了不同类型的数据结构，用于拓扑感知数据亲和性关系。例如，在创建一张表时，调度引擎会记录其表结构及二级索引等元数据。当后续进行数据分片（Region）的创建与调度时，系统会利用这些信息，尽可能将关联紧密的数据（如一张表的主键数据与其二级索引）放置在同一个物理节点上。这种机制旨在最大程度地减少跨节点访问和分布式事务，从而显著降低 SQL 查询的延迟与网络开销。

- **Replication Group**：基于 Raft 协议，物理层面存储 KV 的数据分片单元。一主 N 备，并保证数据一致性，具备多种不同角色，区分为 Leader、Follower、Learner、Witness，共同构成一个可靠的数据副本集合。
- **Region**：一段连续范围的 Key Range，是数据物理存储的最小单元。一个 RG 可以包含多个 Region。每个 Region 承载着某个 DataObject（例如一张表）的一部分实际数据。调度引擎的核心工作之一，就是根据 DataObject 定义的亲和性规则，将这些细粒度的 Region 智能地调度到合适的 RG 中进行存放与管理，从而实现性能与可用性的最优平衡。



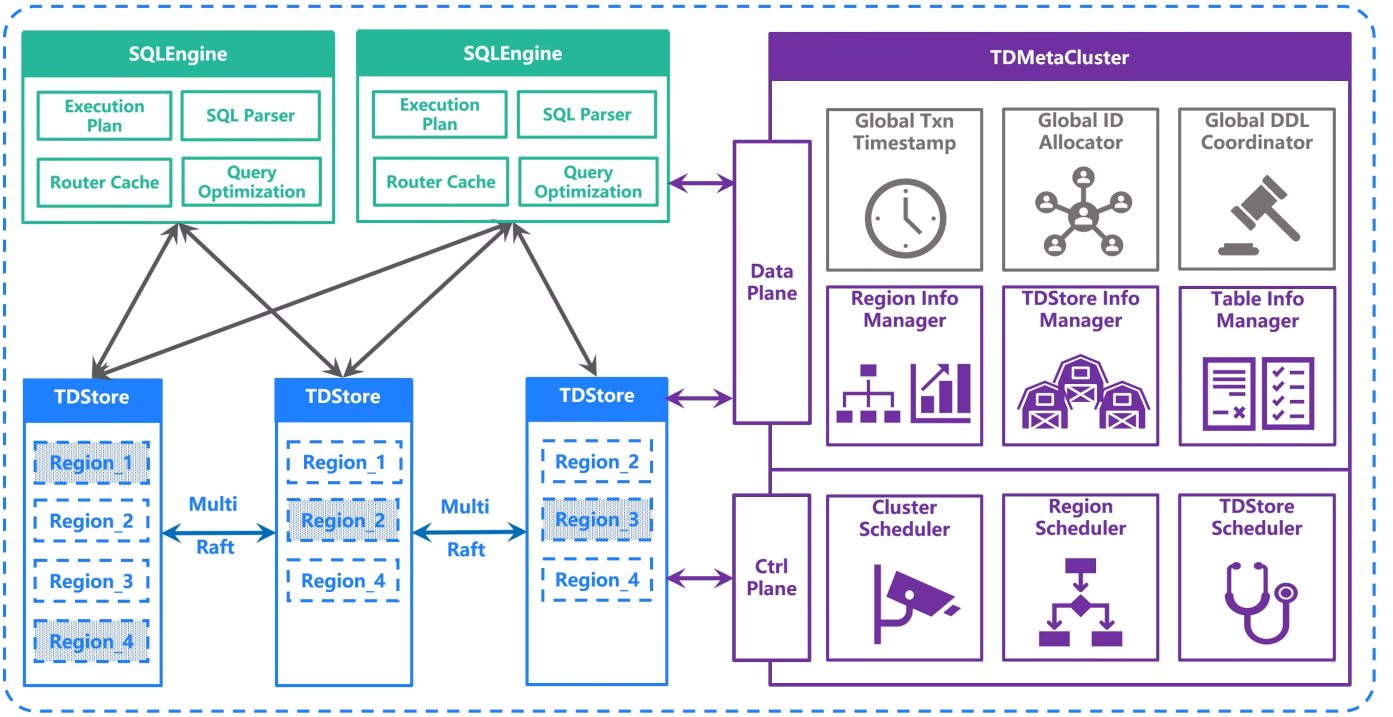
调度/感知能力

最近更新时间：2025-12-24 12:03:44

原子调度

调度能力指调度引擎（MC）与存储引擎（TDStore）之间的交互方式，关注如何高效、可靠地执行数据移动、副本管理等物理操作。

原子调度起源：TDSQL Boundless 实现了 Data/Ctrl Plane Separation 架构



1. **控制平面与数据平面分离 (Ctrl/Data Plane Separation)**：MC 作为集群的“大脑”，承担所有控制平面（Ctrl Plane）的功能，包括监控、决策和调度。这使得计算引擎（SQLEngine）和存储引擎（TDStore）可以专注于数据流相关的操作和性能优化，职责清晰，架构简洁。
2. **MC 负责各层级资源尤其是存储层的监控和调度**。

原子调度起源：TDStore 存储层资源状态与调度任务

调度引擎（MC）需要处理诸如数据分片（Region）跨节点迁移等复杂任务。这类任务从业务视角看是单一的，但在内部涉及多个子步骤，例如：在新节点上创建副本、从主副本同步数据、更新Raft组成员配置、清理原有副本等。以副本迁移（Replica Migration）为例，其内部就包含多达5个子任务。

核心挑战在于若将这类多步骤任务的调度与协调逻辑直接放在存储层（TDStore），会带来显著问题：

1. 逻辑过重：存储层需要同时处理关键的数据读写和复杂的调度逻辑，这会增加其负担，可能影响核心业务的性能与稳定性。

2. 状态混乱：任务的推进、暂停、回滚等流程如果由被执行者（存储节点）自行判断，会引入复杂的二级状态，在异常发生时难以保证状态的一致性，使问题排查和恢复变得异常困难。

为解决上述问题，TDSQL Boundless 将复杂的调度逻辑上移至拥有全局视角的调度引擎（MC）。在 MC 中，我们将每一个调度操作（如迁移、分裂）抽象为一个 Job。每种 Job 类型都有明确的状态机，并被分解为一系列顺序执行的原子步骤（Atomic Step）。

根据调度对象的不同，任务粒度也有所差异：

- Replica 级任务：操作相对集中在单个节点上，例如下线并清理某个特定副本。
- Rep Group 级任务：通常涉及多个节点（因高可用架构为一主多副本），例如需要在一个 Region 的所有副本上同步执行分裂操作。
- Region 级任务：粒度更细，仅涉及 Region 部分元数据的更改。

对象	State	JobType
Rep Group	• UNINIT • NORMAL • OFFLINE	• CREATE • SPLIT • MERGE • OFFLINE • ONLINE • DELETE
Replica	• UNINIT • ONLINE • PENDING • DOWN • OFFLINE	• CREATE • OFFLINE • ONLINE • MIGRATE • REMOVE • DESTROY • PROMOTE/DEMOTE • TRANSFER LEADER
Region	• UNINIT • NORMAL	• CREATE • SPLIT • MERGE • DELETE

解决方案：MC 承担总 Coordinator，TDStore 作为被动执行者

为平衡性能与可扩展性，TDSQL Boundless 确立了明确的职责划分：调度引擎（MC）作为全局的总协调者（Coordinator），而存储引擎（TDStore）作为被动执行者。

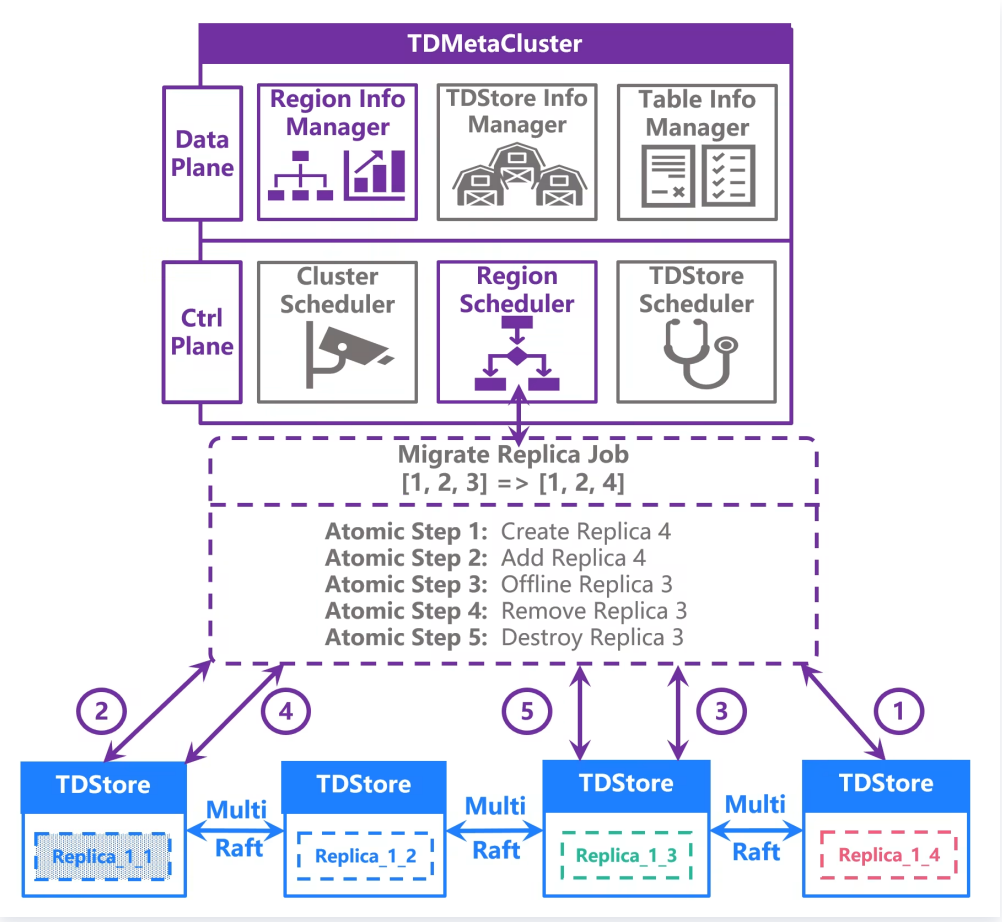
- 以 Region 为单位的细粒度锁，分离 Region 的状态和调度 Job 状态，最大程度减少调度任务对前端请求的影响。

- MC 定义 Atomic Step 任务状态机全集，每个 Job 包含一个或多个 Atomic Step。
- TDStore 的 Region 层在任意时刻，最多只会收到并执行单个 Atomic Step。
- TDStore 对 Job 推进或回滚无感知，正常推进或回滚全由 MC 掌握。

如下所示，一个 RG 副本迁移（Replica Migration）Job 可分解为以下五个原子步骤：

1. Create： 在目标节点创建新副本。
2. Add：新副本从主副本同步数据。
3. Offline：下线旧副本。
4. Remove：变更 Raft 组配置，移除旧节点。
5. Destroy：在旧节点清理副本数据。

执行流程与回滚：MC 依次下发每个步骤。例如，若步骤1（Create）失败，MC 会立即触发预定义的回滚流程，指令 TDStore 清理已创建的资源。TDStore 只负责执行“回滚”这个原子指令本身，而整个失败处理和回滚决策完全由 MC 控制。



数据感知

拓扑感知(Topology Aware)的数据对象生命周期管理，贯通逻辑层<–>物理层链路

1. 管控调度引擎基于 Data Object 抽象，建立**拓扑感知的数据对象管理**体系， 打通数据逻辑层与物理存储层的链路。

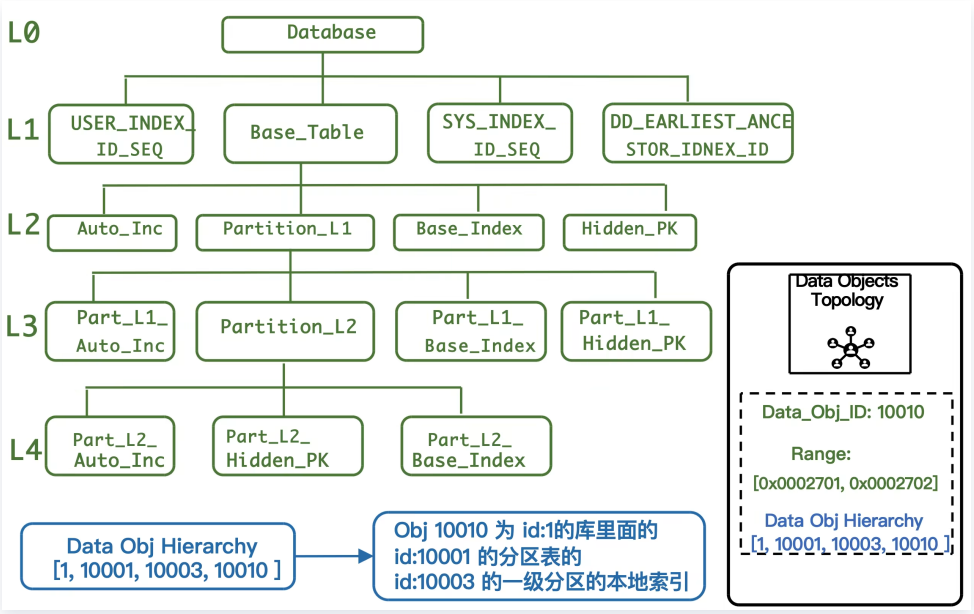
   该体系基于 MySQL 关系型模型，对常见数据类型进行分层定义，形成清晰的拓扑结构：

- L0 级：Database（数据库）
- L1 级：Table（表），隶属于某个 Database
- L2 级：Index/Partition（索引或分区），隶属于某个 Table。若为分区表，则进一步区分一级分区、二级分区及其对应索引。

这种层级编码使系统能够精确定位每一个数据对象的逻辑归属。例如，对象10010可明确表示：它是数据库（id:1）中分区表（id:1001）的一级分区（id:1003）下的一个二级索引。
该体系的核心价值在于感知数据之间的亲和性关系。例如，同一张表的主数据与其二级索引、同一分区下的不同子表，在业务查询中极有可能被同时访问。通过拓扑感知，调度引擎能够识别这些逻辑上紧密关联的数据对象，并在物理调度时尽可能将其放置在相同或相近的存储节点。这样做可显著减少跨节点数据访问、降低分布式事务比例，从而提升查询性能与响应速度。
通过 Data Object 与 Region 的关联，该系统实现了逻辑层与物理层的有效贯通：

- 逻辑层（Data Object）：代表 TDSQL Boundless 中的库、表、索引等逻辑实体。
- 物理层（Region）：对应 TDStore 中管理的一段连续 Key Range，是数据的实际存储单元。

每一个 Region 都明确归属于某个 Data Object，存储其对应的物理数据。调度引擎（MC）借助这套映射关系，既能理解数据的业务语义，又能掌握其物理分布，从而做出兼顾业务亲和性与存储均衡性的智能调度决策。



2. 管控调度引擎切入 **DDL 关键路径**， 将 DDL 任务映射为对存储层的数据分片的原子调度操作，维持了存储层的 KV 语义独立性。

   其核心流程与机制如下：

   ### 2.1 DDL 任务解析与元数据管理

   当计算引擎接收到前端发起的建表、删表等 DDL 语句后，会将其解析为标准的 DDL 任务。该任务本质是对全局元数据进行“增删改”的操作，以此实现**数据对象（Data Object）的生命周期管理**。例如，创建表对应生成一个 `Create` 任务以新增 Data Object，删除表则对应一个 `Destroy` 任务。

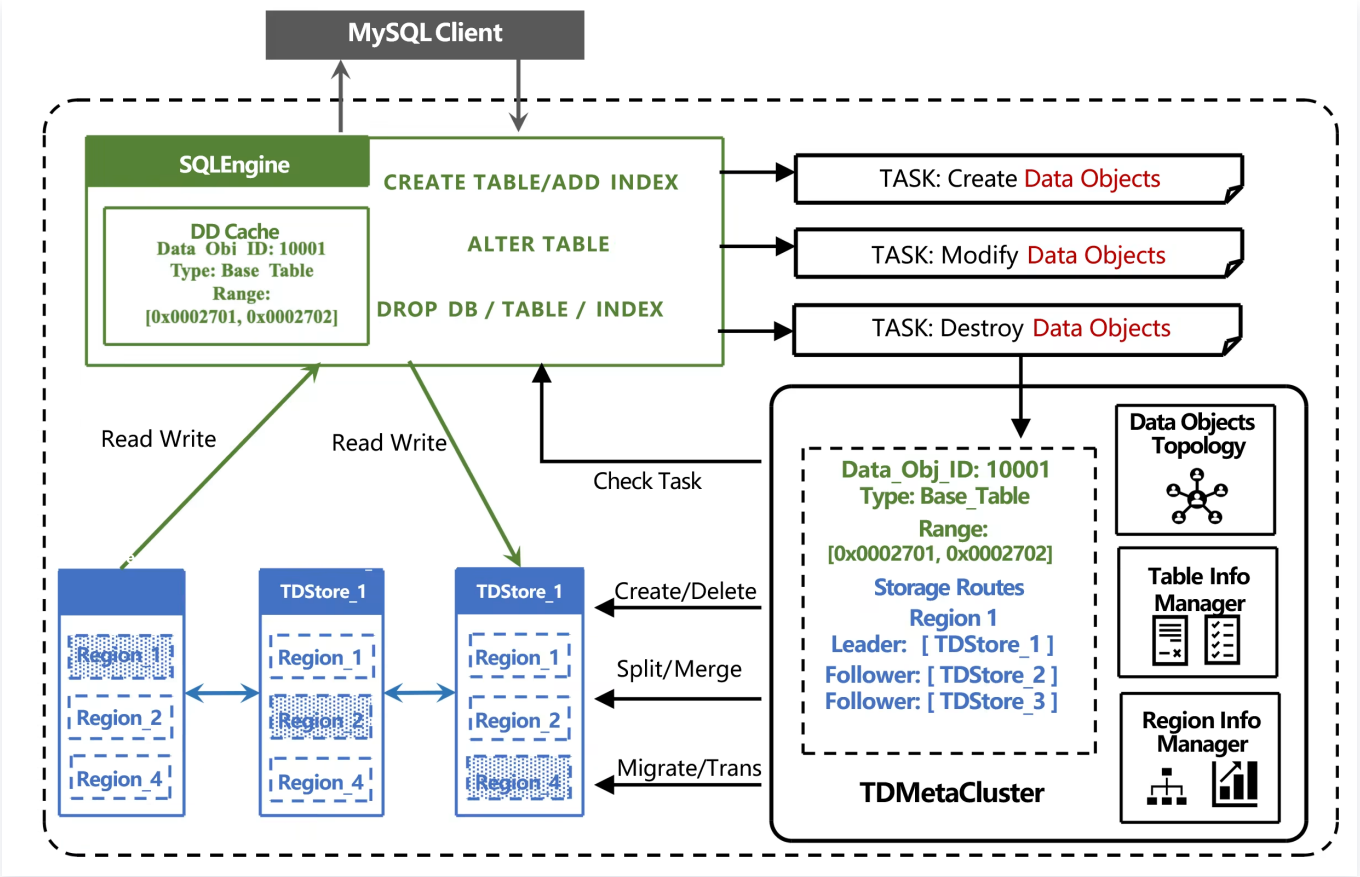
   ### 2.2 拓扑感知与亲和性调度

   调度引擎（MC）通过 RPC 机制感知到数据对象的变更后，会立即将其纳入**拓扑感知数据管理体系**中。该系统能精准识别数据对象间的亲和性关系（如主表与索引）。当需要创建新的数据分片（Region）时，MC 会基于

此亲和性信息，下发调度任务（Job）给存储层（TDStore），智能地将关联紧密的数据对象调度到相同的物理节点上。

2.3 维持存储层语义独立性

整个流程的关键优势在于，存储层（TDStore）完全无需理解复杂的 SQL 语义。它仅需执行由 MC 下发的原子性操作指令（如“创建某个 Key Range 的Region”或“删除某个 Region”）。这种设计将业务逻辑与存储实现彻底解耦，**确保了存储层只需专注于高效的 KV 读写与副本管理，维持了其核心语义的独立性和纯粹性。**通过这一机制，调度引擎既实现了基于业务逻辑的智能数据分布优化，又避免了对底层存储的过度侵入，构建了一个既高效又灵活的存算分离架构。



3. 管控调度引擎维护和更新全局库表对象的物理路由信息，聚合物理层读写统计汇合成对象层的热点统计，为基于**数据感知的调度策略**提供底座能力。

基于复制组（ReplicationGroup）的两层元数据管理以及 On-Demand 分配机制

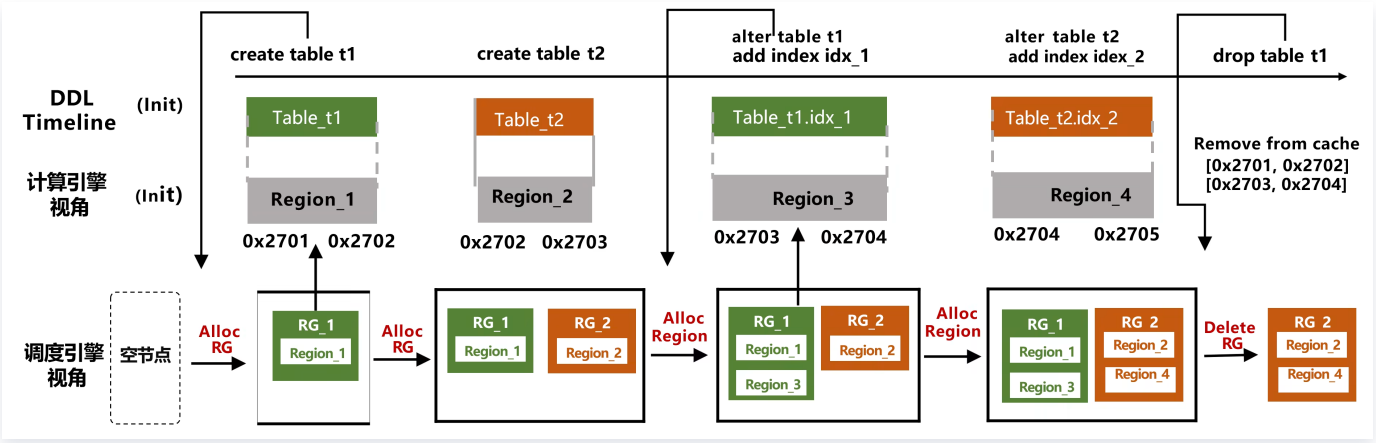
- **基于数据拓扑感知的能力，以 On-Demand 机制为计算引擎分配存储资源，保持存算分离的语义，存储层与元数据框架升级完全解耦。**

  与友商在 Region 过大时简单按范围切分的方式不同，TDSQL Boundless 引入了 On-Demand 分配机制，并与数据拓扑感知能力深度结合：

  - 按需分配资源：在执行 DDL 操作（如建表、创建索引）时，调度引擎会根据数据对象的逻辑含义与亲和性关系，向存储层按需申请 Region。例如，创建一张表时，系统会为其分配专属的 Region 资源。
  - 亲和性驱动调度：借助拓扑感知体系，系统能识别出应放置在一起的数据分片。例如，在为某张表创建二级索引时，调度引擎会依据数据层级关系，将索引所在的 Region 与主表数据所在的 Region 调度至同一节

点，从而极大减少跨节点访问，降低分布式事务比例。

- **基于复制组（ReplicationGroup + Region）的两层元数据管理，支持任意数据分片的分裂与合并，优化分布式架构大部分场景下的分布式事务问题。**



任务/规则

最近更新时间：2025-12-24 12:03:44

调度任务

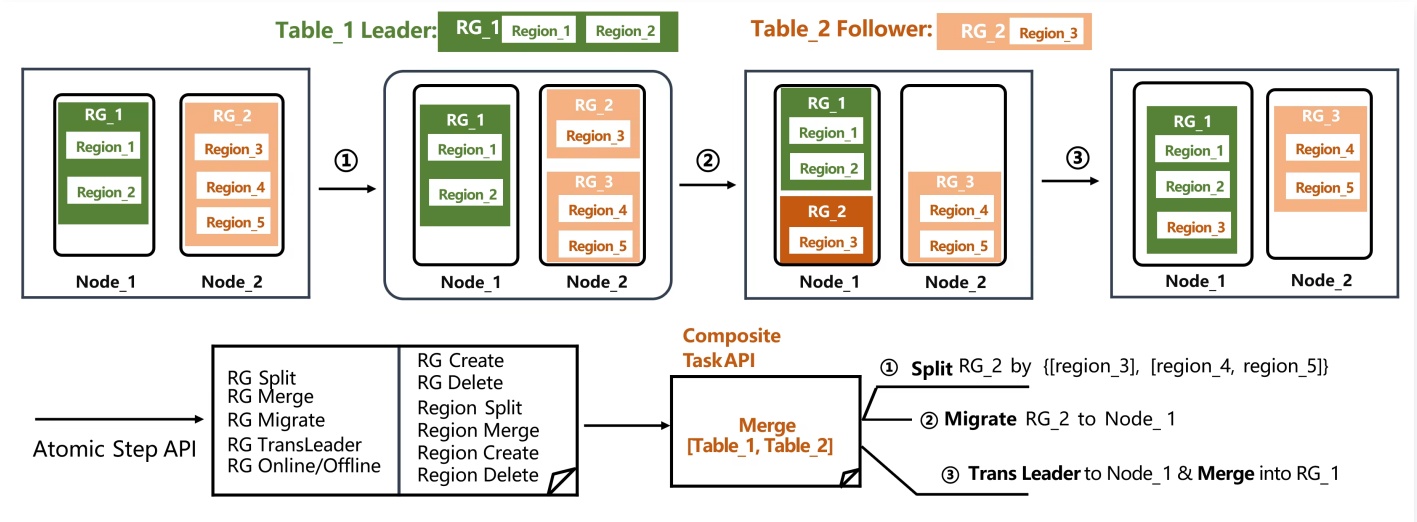
基于执行计划的、可编排的数据分片调度任务执行框架

TDSQL Boundless 设计了一个基于执行计划的可编排任务框架，用于将多个原子调度任务组合成复杂的逻辑流程（如RG + Region 双层元数据，拓扑感知，调度策略任务，复合式任务子步骤等）。该框架的核心是一个称为 Task 的编排器，它负责解析调度目标、规划执行步骤、管理依赖关系，并依次向存储层（TDStore）下发原子指令。

如下所示，由于查询优化需求，需要将逻辑上关联的 table1 和 table2 在物理上调度到同一个节点（Node1），以减少跨节点交互。

1. 初始状态：table1 的数据分片位于 Node1 的 Region1 和 Region2 上；table2 的数据分片（Region3）位于另一个节点 Node2 上。
2. 执行分裂：框架向 TDStore 下发第一个原子任务——“分裂 RG2”。TDStore 执行完毕并上报成功。
3. 执行迁移：框架确认前置条件满足后，下发第二个原子任务——“将 Region3 从 Node2 迁移至 Node1”。TDStore 完成数据迁移。
4. 切主并合并 Region：框架下发第三个原子任务——“将相关 RG 的主副本切换（Transfer Leader）”，以确保两个 RG 的数据处于同分布状态。在所有前置步骤成功后，框架下发最终的原子任务——“合并指定的 RG。

通过此框架，复杂的运维操作被简化为一个自动、可靠的调度任务，无需人工干预每一步。

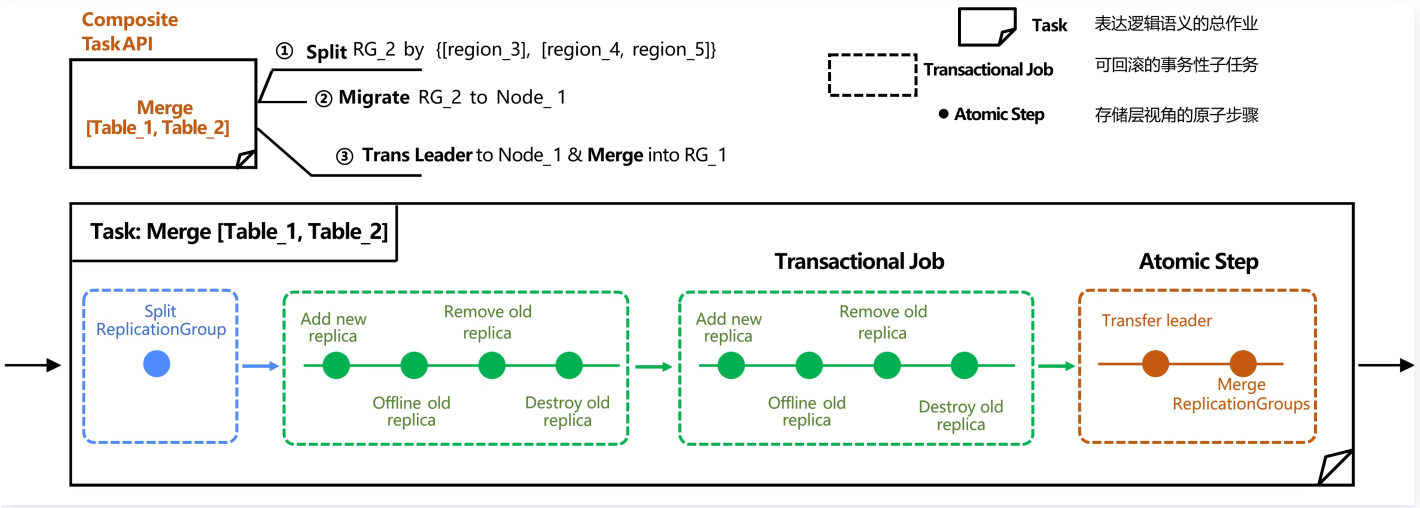


可编排的数据分片调度任务执行框架

管控调度引擎实现了一套可编排的任务执行框架，通过定义事务性子任务，使存储层只需感知KV操作原语，从而保持其语义简洁性。

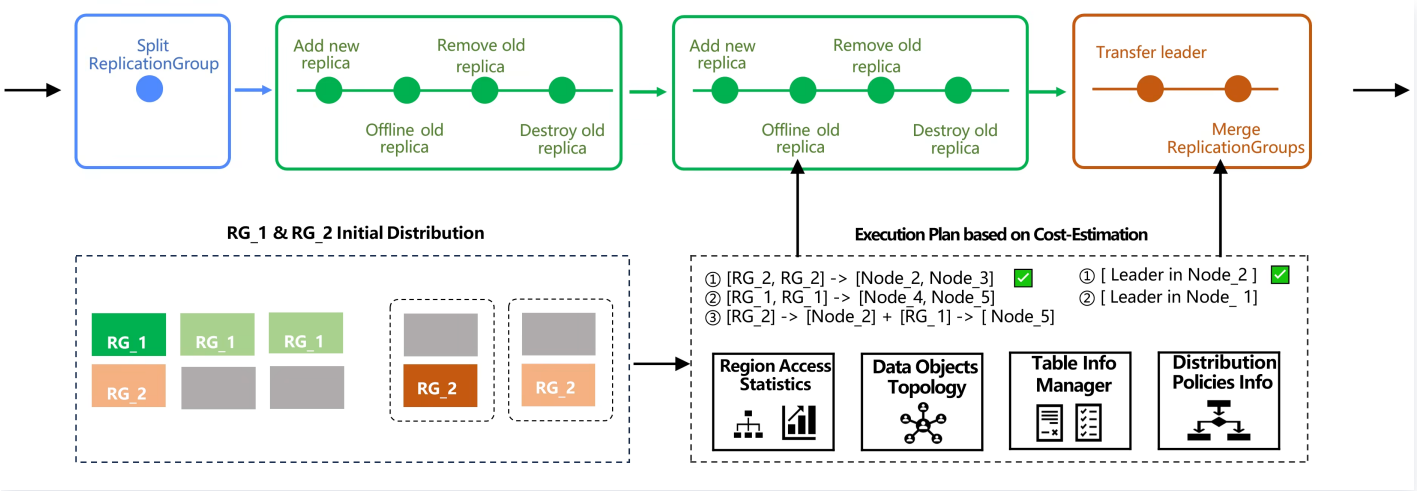
该框架将复杂的调度逻辑抽象为可编排的事务型子任务。以两张表的亲和性调度为例，这是一个逻辑层面的调度需求，框架会将其分解为多个原子步骤（如分裂、迁移、切换主副本等），并确保这些步骤组成的事务要么全部成功，要么在任一环节失败时能够完全回滚。

通过这种设计，复杂的调度逻辑完全由管控层处理，存储层仅需执行简单的KV操作原语，实现了调度复杂性与存储简洁性的有效分离。



基于Cost-Estimation 的执行计划

- 管控调度引擎维护和更新全局资源分布状态和信息 ，计算可能的执行计划及代价 。
- 分布式多节点、 多数据分片场景下，面向全局资源以及读写热点分布寻找最优解。



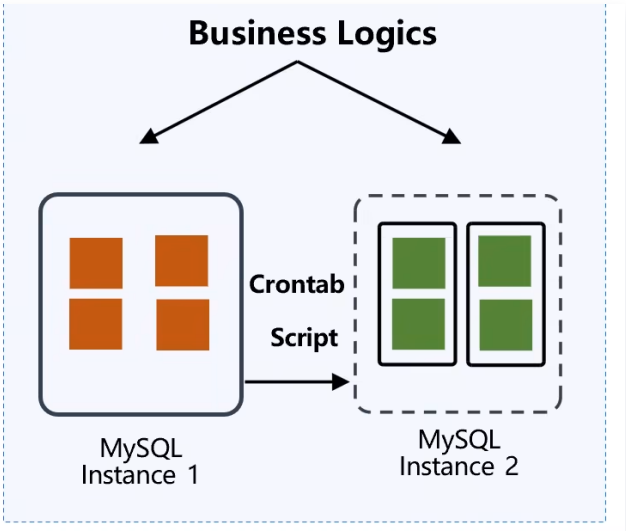
调度规则

典型业务场景

基于TDSQL Boundless的数据感知与调度能力，我们针对典型的业务场景需求，抽象并定义了一套数据生命周期管理与放置规则策略系统。该系统能够直接响应业务诉求，智能地指导数据分片的调度与放置。

- **场景1：流水、账单**

业务痛点：历史数据（如3个月前的子分区）占用高成本存储资源，传统方案需外置脚本定期迁移，业务访问需配置不同端口



- **场景 2：金融，多法人**

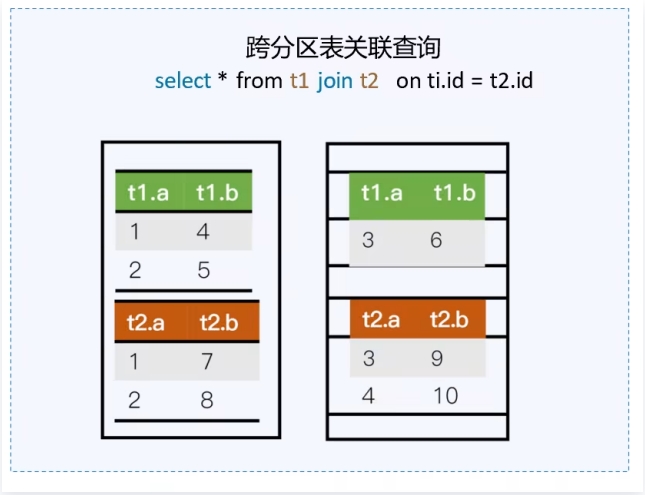
  业务痛点：监管要求不同法人数据必须物理隔离，传统方案通过改写 SQL 引擎实现，扩展性受限，且无法实现细粒度的拓扑感知。

```
case `a`
when g1 values in('qd','jn') then
TDSQL_SUBDISTRIBUTED by HASH( b )(s1,s2), when g2 values
in('wf','rz') then
TDSQL_SUBDISTRIBUTED by HASH( b )(s3,s4) end case
```

- **场景 3：广告，检索**

  业务痛点：

  - **分区表之间，跨表关联查询 ，具有相关性的数据尽可能放在一起 。**
  - **TDSQL MySQL（InnoDB版）支持固定形式。**
  - **无数据拓扑感知 ，无法实现。**



基于当前分布式中间件架构，这一需求尚未得到很好的解决，因为该架构缺乏逻辑层面的感知能力。而在 TDSQL Boundless 数据感知能力体系中，通过构建的逻辑亲和性能力，我们可以将这些表调度到同一个节点上，从而有效解决上述问题。

基于以上几个场景，我们对相关规则进行了抽象，定义了一套生命周期管理的策略系统。该系统用于指定数据的分布位置，并确保表之间具备所需的亲和性。

实现符合通用标准的、可扩展的数据分布与亲和性调度策略系统

基于数据拓扑感知能力，实现可扩展的符合 K8s 语义的策略接口系统 ，作为驱动数据分布调度任务框架的编排输入。

- **结构：** 约束 constraints 由一条或多条固定格式的 `key-op-values` 组成。

  其中， `op` 支持 `in` 、 `notIn` 、 `exists` 、 `notExists` 、 `=` 、 `!=` 等基于等值的和基于集合的标签选择运算符 。
- **含义：** 创建 `table_1` ，绑定 `cross-3-idc` 策略 ，该表有一主两备3个副本，在 gz_1 – 5 可用区中分布，Leader 尽量不选择 gz_3 或 gz_5 ，优先确保 zone 级别的容灾能力， 如 zone 级别容灾无法满足，则自动退化为保证 rack 容灾级别的分布。

```
{
  "policy_name": "cross-3-idc",
  "common_option": {
    "constraints":
      [
        {"key": "replica_count",     "op": "=",     "values": ["3"]},
        {"key": "follower_count",    "op": "=",     "values": ["2"]},
        {"key": "zone",              "op": "in",    "values": ["gz_1", "gz_2", "gz_3", "gz_5"]},
        {"key": "leader_preference", "op": "notIn", "values": ["gz_3", "gz_5"]},
        {"key": "fault_tolerance",   "op": "in",    "values": ["zone", "rack"]}
      ]
  }
}
```

- **策略示例：** 创建表时绑定 `cross-3-idc` 策略。

```
CREATE TABLE table_1 USING DISTRIBUTION_POLICY cross-3-idc
```

- **常用副本放置规则**

key	op	values
region	"in", "notIn", "exists", "notExists"	地域列表，如 ["guangzhou"]
zone	"in", "notIn", "exists", "notExists"	可用区列表，如 ["guangzhou-1"]
rack	"in", "notIn", "exists", "notExists"	机架列表，如 ["rack-1","rack-2"]
host	"in", "notIn", "exists", "notExists"	主机列表，如 ["host-1","host-2","host-3"]
node	"in", "notIn"	节点列表，如 ["node-tdsql3-xxx-001"]
replica-count	"="	副本数量，如 ["3"]
follower-count	"="	RG follower 数量，如 ["2"]
learner-count	"="	RG learner 数量，如 ["1"]
witness-count	"="	RG witness 数量，如 ["1"]
leader-preferences	"="	RG leader 偏好
fault-tolerance-level	"="	容灾级别，如 ["zone"]
tdsql-storage-type	"in", "notIn", "exists", "notExists"	磁盘类型列表，如 ["CLOUD_TCS","CLOUD_BSSD"]

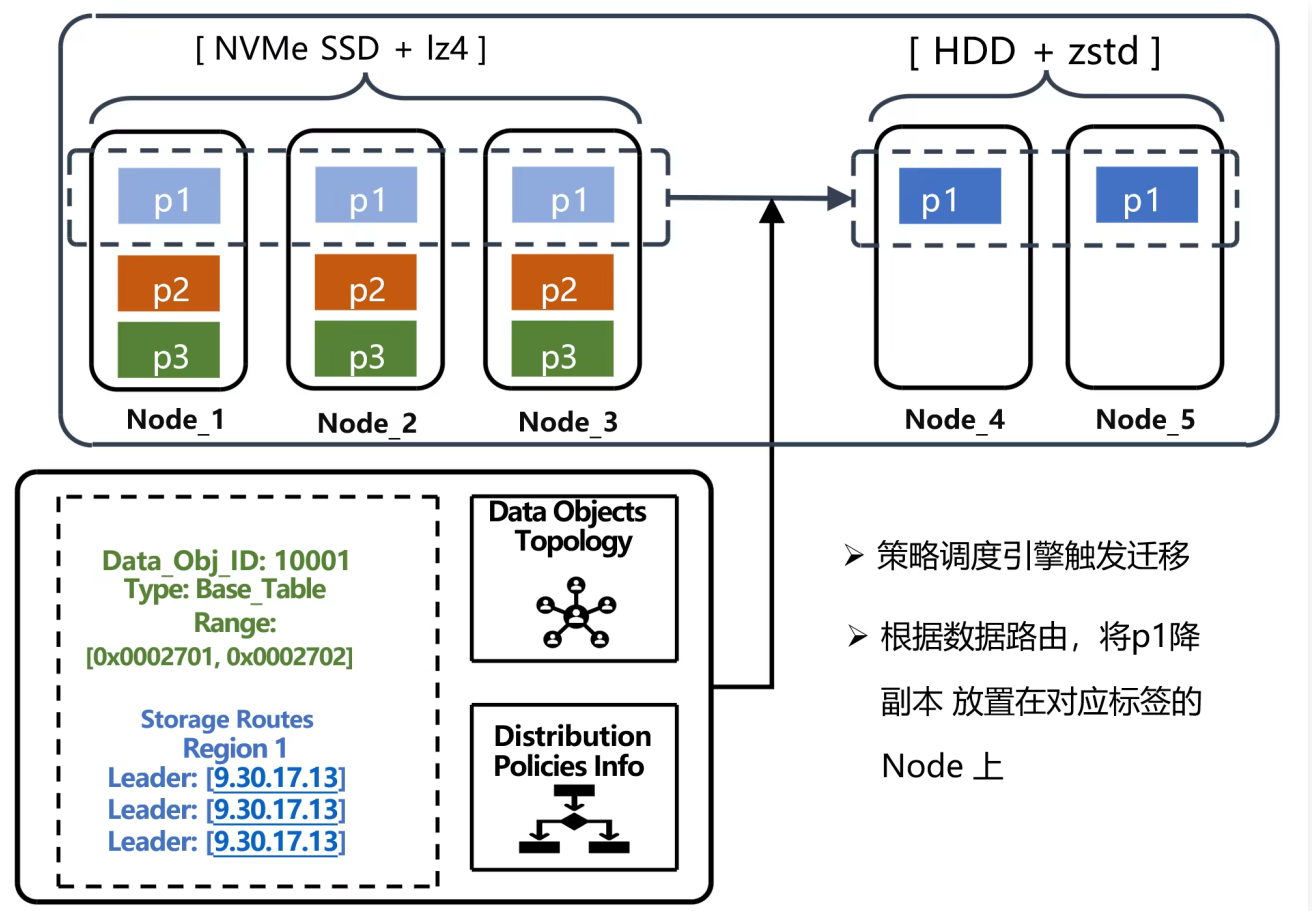
- **策略系统核心价值**
  - 通用标准：符合事实标准， key-op-values 规范对用户友好。
  - 语义可扩展 ： 可以根据不同的业务需求，扩展更多的 PolicyOption。
  - 数据链路全解耦 ： 调度引擎独立管控，计算/存储全解耦，数据拓扑感知 + 调度任务编排框架驱动。
- **典型示例**

对于流水、账单场景，TDSQL Boundless 通过内置策略（如3个月之前的子分区，自动迁移至冷存节点，降副本）实现自动化冷热数据分层，无需人工干预。用户只需通过 SQL 指定数据生命周期策略，系统自动识别过

期数据并将其迁移至低成本存储节点。

```
CREATE TABLE table_1(id INT, trans_date DATETIME)
  PARTITION BY RANGE (TO_DAYS(trans_date) ) (
    PARTITION p1 VALUES LESS THAN (TO_DAYS('2022-11-01')),
    PARTITION p2 VALUES LESS THAN (TO_DAYS('2022-12-01')),
    PARTITION p3 VALUES LESS THAN (TO_DAYS('2023-02-01')),
    PARTITION p4 VALUES LESS THAN (TO_DAYS('2023-03-01')));
ALTER TABLE table_1
   USING DISTRIBUTION_POLICY partition-cool-down
```

```
{
 "policy_name": "partition-cool-down",
 "hibernate_option": {
   "constraints":
   [
     {"key": "exp_period_day", "op": ">=", "values": ["90"]},
     {"key": "dest_labels",    "op": "in", "values": ["hdd,zstd"]},
     {"key": "replica_count",  "op": "=",  "values": ["2"]}
   ]
 }
}
```



隐式/显式地为分区规则相同的表建立亲和性关系

隐式亲和性

在分布式数据库环境中，对于分区规则相同的表，TDSQL Boundless 引入了分区级亲和性调度机制，以优化关联查询性能并降低分布式事务开销。

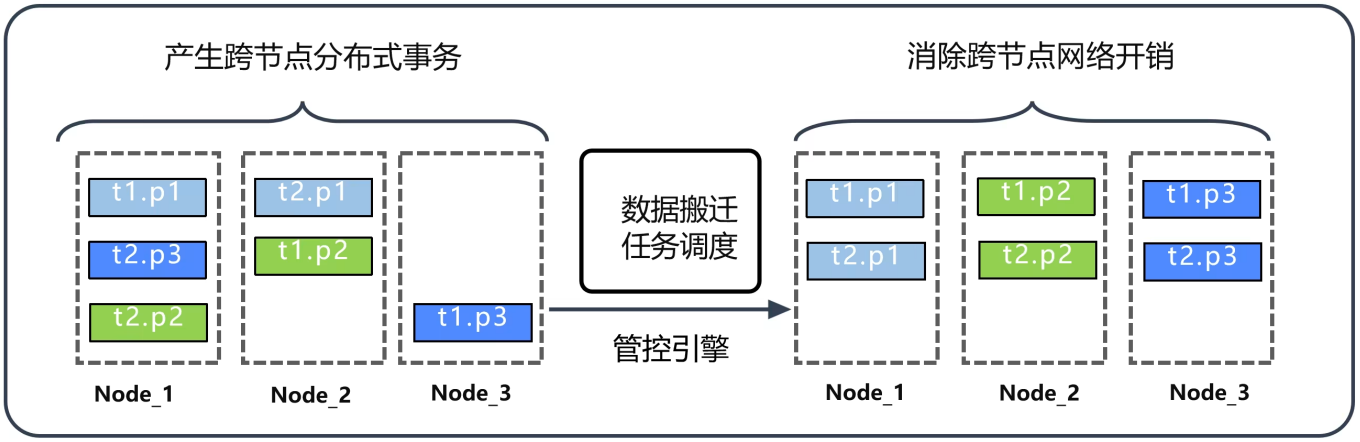
如下所示，当两张表采用相同的分区规则时，系统默认认为其相同编号的分区在业务逻辑上具有强关联性（例如常在同一事务中被同时访问）。基于这一判断，调度引擎会主动将相同编号的分区调度至同一物理节点，从而实现数据本地化访问。

```
CREATE TABLE t1(id INT PRIMARY KEY, f1
 INT,f2 VARCHAR)
PARTITION BY HASH (f1) PARTITIONS 3;

CREATE TABLE t2(id INT PRIMARY KEY, f1
 INT,f3 VARCHAR)
PARTITION BY HASH (f1) PARTITIONS 3;
```

下图左边为未启用亲和性调度时，数据分布分散，关联查询需跨节点访问，网络开销大。集群扩容时，均衡策略可能进一步打散相关数据（如将 t1.p3 迁移至新节点 node3），加剧性能损耗。

右边为启用亲和性调度后，通过调度引擎的智能编排，将关联分区（如 t1.p3 与 t2.p3 ）固定在同一节点，有效避免因扩容或负载均衡导致的数据分散，保障查询性能稳定。



显式亲和性

除了分区表的隐式亲和性外，TDSQL Boundless 还提供了显式亲和性定义机制，允许业务通过 SQL 语法主动配置表间的关联关系，实现更精细化的数据分布优化。

场景一：小表紧密关联

- **适用情况：**有两张数据量不大的小表，业务上紧密相关，频繁做写入或关联查询，要保证在同一个RG。
- **配置语法：**

```
CREATE PARTITION POLICY IF NOT EXISTS pp1
```

- **调度效果：**系统将尽可能将这两张表分配到同一个复制组（RG）中，确保关联操作在节点内完成，消除分布式事务开销。

场景二：大表分区级亲和

- **适用情况：**数据量巨大的表，无法将整表置于同一 RG，但可通过分区实现局部亲和
- **配置语法：**

```
CREATE PARTITION POLICY IF NOT EXISTS pp2 PARTITION BY HASH ( INT ) PARTITIONS 4
```

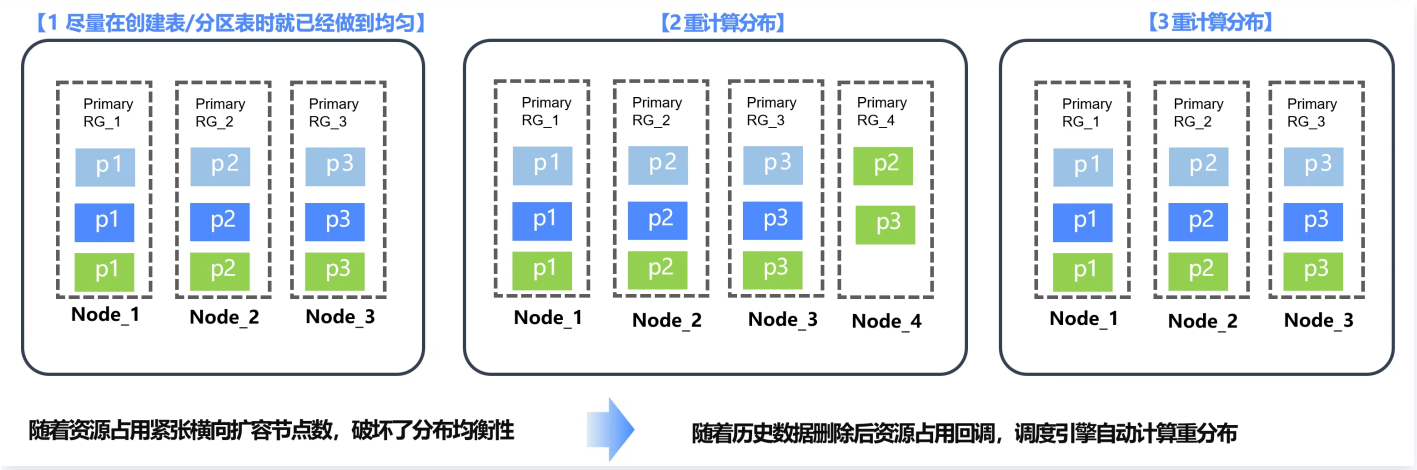
- **调度效果：**系统将两张表相同规则的分区（如相同哈希值的分区）调度至同一 RG，两张表上同一时间的写入和查询没有2PC或跨节点JOIN。

负载均衡

最近更新时间：2025-12-24 12:03:44

负载均衡与数据亲和性协同调度机制

TDSQL Boundless 的管控调度引擎（MC）实现了负载均衡与数据亲和性协同调度机制，在保证业务性能的同时确保集群资源的高效利用。

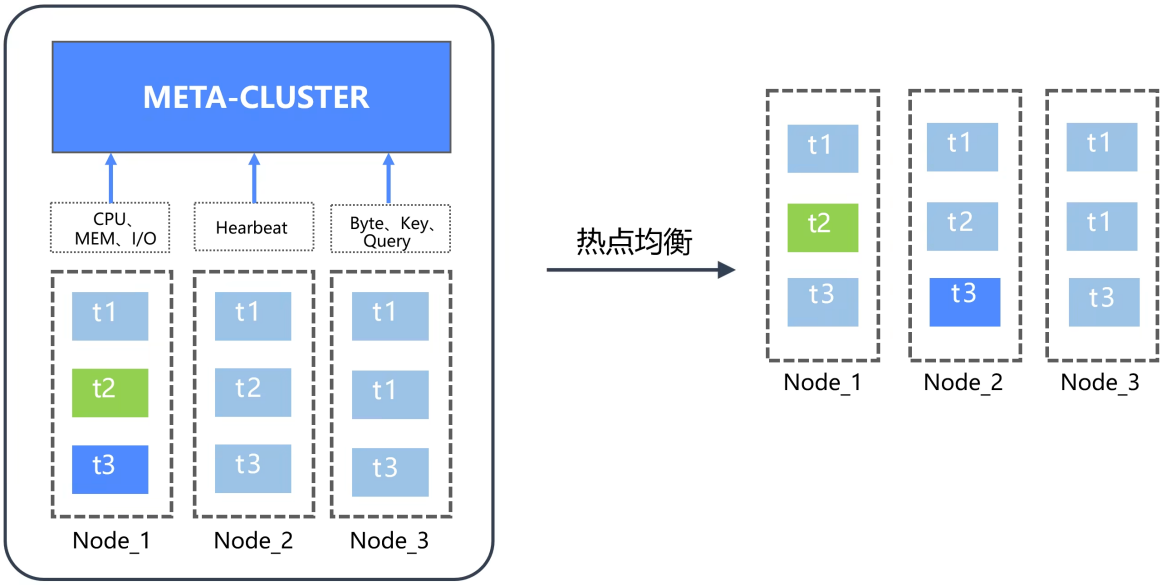


1. 初始分布策略
   - 均匀分布原则：创建表时按节点数自动打散数据分布
   - Leader 均衡：默认每个节点分布一个 Primary RG Leader，确保读写负载分散
   - 副本均衡：表的一级/二级分区均匀分布所有可用节点
2. 动态扩容感知

   随着资源占用紧张横向扩容节点数，破坏了分布均衡性，MC 自动执行以下操作：
   - 计算新增节点的容量状态
   - 重新评估数据分布均衡性
   - 迁移部分数据分片至新节点
   - 维持亲和性规则的同时优化整体分布
3. 持续优化机制
   - 随着业务发展，当数据增删导致资源占用变化时，MC 定期重新计算数据分布。
   - 渐进式优化：通过多次小规模迁移逐步优化分布，避免大规模数据搬迁对业务造成影响。
4. 极端情况下数据倾斜过高，调度引擎依然能够分裂表数据并进行副本搬迁。

热点检测与智能调度机制

TDSQL Boundless 的管控调度引擎（MC）实现了智能化的热点检测与负载均衡系统，通过多维度监控和智能算法有效解决分布式数据库中的热点问题。



t3表所在的 region 从 Node_1 **分裂**出去，并执行**切主**到 Node_2，最后执行**合并**

1. 实时数据上报
   - 节点心跳：Node 周期性上报 Heartbeat，包含节点基本信息、Region 级别的读写流量统计。
   - 资源采集：MC 周期性采集节点资源信息，包含 CPU 使用率、内存占用情况、I/O 负载指标
2. 多维度负载状态由 MC 维护 Multi-TopN 记录，从 Byte（数据量负载）、Key（访问键值分布）、Query（查询频率与复杂度）维度监控节点负载。
3. 智能调度算法
   - 基于 MovingAverage 记录历史均值，避免冲击负载产生冗余调度。
   - 计算各个节点的热度得分，选择最佳均匀分布，并产生调度任务。

SQL 参考
基本元素
Hint
INDEX_FOR_GROUPBY

最近更新时间：2025-11-18 10:10:22

功能描述

`INDEX_FOR_GROUPBY` / `NO_INDEX_FOR_GROUPBY` 提供 Hint 控制 GROUPBY 操作的情况下采用 Loose Index Scan 方式（松散索引扫描）减少扫描开销。

语法

```
/*+ NDEX_FOR_GROUPBY ([@query_block_name] tbl_name [index_name])*/
```

示例

```
CREATE TABLE t1 (
  a INT,
  b INT,
  c INT,
  KEY k2 (a, b, c)
);

INSERT INTO t1 VALUES
(1, 2, 3), (1, 2, 4), (3, 4, 5), (3, 4, 5);

SELECT /*+ INDEX_FOR_GROUPBY(t1 k2)*/DISTINCT a, b
  FROM t1
 WHERE a BETWEEN 1 AND 10
   AND a > 1 AND c = 5
ORDER BY a, b;
```

列存向量化引擎 Join Order Hint

最近更新时间：2025-11-18 10:10:22

功能描述

使用 Join Order Hint 指定列存向量化引擎计划片段生成。在选择向量化引擎作为执行器时，在查询的 SQL 语句中添加 MySQL 风格的 Join Order Hint 即可传递给向量化引擎。目前支持三种 Join Order 相关的 Hint：`join_order`，`join_prefix`，`join_suffix`。

类型	作用范围	位置约束	适用场景
join_order	全表 join 顺序片段	计划树中间位置	明确多表关联路径
join_prefix	前缀表强制左深树顺序	计划树最左端	优先处理维度表
join_suffix	后缀表强制右深树顺序	计划树最右端	延迟处理大表关联

join_order

join_order Hint 类型尝试将 Hint 内指定的几个表 join 顺序固定为一个左深树顺序片段。未指定的表根据默认优化器逻辑和这个片段连接。

- 示例1

```
SELECT /*+join_order(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a;
```

执行计划解析：

```
| physical_plan
 -> Projection: a a (rows=*)
   -> Inner Hash Join (a = a) (rows=*)
     -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
     -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
```

全部表的 join 顺序被指定。Hash Join 情况下，`t2` 是 build side，`t1` 是 probe side，这和 MySQL 计划一致。

- 示例2

```
SELECT /*+join_order(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a JOIN t3
ON t2.a=t3.a;
```

执行计划解析：

```
 -> Inner Hash Join (a = a) (rows=*)
   -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
   -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
```

上面的计划被作为一个计划片段，`t3` 表在计划中所处的位置不确定，由优化器决定。

join_prefix

除了具有 join_order 的作用，同时被指定的计划片段会出现在整个计划树的最左侧（显示的计划的底部）。

- 语法示例

```
SELECT /*+join_prefix(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a JOIN t3
ON t2.a=t3.a;
```

- 执行计划解析：

```
| physical_plan
 -> Projection: a a a (rows=*)
   -> Inner Hash Join (a = a) (rows=*)
     -> Tdstore Scan Text: test.t3 Projections: a (rows=*)
     -> Inner Hash Join (a = a) (rows=*)
       -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
       -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
```

`t1` 和 `t2` 的 join 顺序被指定，同时被固定在计划最左侧。

join_suffix

除了具有 join_order 的作用，同时被指定的计划片段会出现在整个计划树的最右侧（显示的计划的顶部）。

- 语法示例

```
SELECT /*+join_suffix(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a JOIN t3
ON t2.a=t3.a;
```

- 执行计划解析：

```
| physical_plan
 -> Projection: a a a (rows=*)
```

```
    -> Inner Hash Join (a = a) (rows=*)
      -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
      -> Inner Hash Join (a = a) (rows=*)
        -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
        -> Tdstore Scan Text: test.t3 Projections: a (rows=*)
```

`t1` 和 `t2` 的 join 顺序被指定，同时被固定在计划最右侧。

注意事项

- Hint 中的表的大小写不区分，且只能指定表名，而非 query block 的名字。
- 在同一个 Hint 中可以有1个或多个表，其中对 join_order 当指定超过2个表时有效。可以同时有多个 Hint。比如下面的所有 Hint 均生效：

```
SELECT /*+join_prefix(t2,t1) join_order(t3,t5,t4)*/ * FROM  t1 JOIN t2
ON t1.a=t2.a
    JOIN t3 ON  t2.a=t3.a
    JOIN t4 ON  t3.a=t4.a
    JOIN t5 ON t4.a=t5.a;
```

- 每个 SQL 查询中至多允许一个 join_prefix 或一个 join_suffix Hint。若存在多个同类型 Hint，仅第一个生效，剩下的会被忽略。
- 当 Hint 之间有冲突时，后指定的 Hint 会被忽略。比如下面第二个 Hint 不会生效：

```
SELECT /*+join_order(t2,t1) join_order(t1,t2)*/ * FROM t1 JOIN t2 ON
t1.a=t2.a;
```

- 如果两表之间没有 join 条件，这两个表若在 Join Order Hint 中毗邻，则该 Hint 会被忽略。比如下面的 Hint 无效：

```
SELECT /*+join_order(t2,t1)*/ * FROM t1,t2;
```

数据字典
PERFORMANCE_SCHEMA DATA_LOCKS

最近更新时间：2025-11-18 10:10:22

功能

`PERFORMANCE_SCHEMA.DATA_LOCKS` 用于展示集群中所有 hybrid 节点的 TDStore 上所有已成功加锁的悲观锁信息。
在用户没有查询 `DATA_LOCKS` 表的时候， `DATA_LOCKS` 不会带来额外的开销。用户不需要将 `performance_schema` 系统变量设置为 `ON` ，即可查询 `DATA_LOCKS` 表。

字段说明

字段名	类型	描述
ENGINE	varchar(32)	存储引擎。对于 TDStore 为 `RocksDB` 。
ENGINE_LOCK_ID	varchar(128)	悲观锁的唯一标识符。对于 TDStore，其格式为 `<持有锁的事务ID>_<锁范围>` 。 `<锁范围>` 的格式是： • 如果锁的范围是一个单条的 key，则 `<锁范围>` 为 key 的十六进制表示。例如：锁标识符 `28673778183569468_00002797` 表示 ID 为 28673778183569468 的事务在十六进制编码为 `00002797` 的 key 上加了一个悲观锁。 • 如果锁的范围是一个 key 区间，则 `<锁范围>` 格式为 `[<hex_start_key>,<hex_end_key>)` （左闭右开）或 `(<hex_start_key>,<hex_end_key>)` （开区间）。例如：锁标识符 `28673778183569468_[00002797,00002798)` ，表示 ID 为 `28673778183569468` 的事务在 `[00002797,00002798)` 区间加了一个悲观锁。 当 key 的十六进制编码过长的时候，可能无法在128字符内表示完全。这个时候，字段值的后两个字符会被置为 `..` ，表示这不是一个完整的锁标识符。这种情况下，用户通过 `ENGINE_LOCK_ID` 查询悲观锁信息时，也需要使用带有 `..` 的字段值做查询。
ENGINE_TRANSACTION_ID	bigint unsigned	持有悲观锁的事务的唯一标识符。如果这个悲观锁是读锁，被多个事务共同持有，则只会展示其中一个事务的唯一标识符。

THREAD_ID	bigint unsigned	Performance Schema 的内部线程 ID，在 Performance Schema 的生命周期内唯一，可用于可靠地关联 Performance Schema 的各类事件表。
EVENT_ID	bigint unsigned	导致加锁的 Performance Schema 事件 ID。对于 TDStore 为 `NULL`。
OBJECT_SCHEMA	varchar(64)	悲观锁对应的数据库名。对于 TDStore 为 `NULL`。
OBJECT_NAME	varchar(64)	悲观锁对应的表名。对于 TDStore 为 `NULL`。
PARTITION_NAME	varchar(64)	悲观锁对应的分区名。对于 TDStore 为 `NULL`。
SUBPARTITION_NAME	varchar(64)	悲观锁对应的子分区名。对于 TDStore 为 `NULL`。
INDEX_NAME	varchar(64)	悲观锁对应的索引名。对于 TDStore 为 `NULL`。
OBJECT_INSTANCE_BEGIN	bigint unsigned	悲观锁在内存中的地址。
LOCK_TYPE	varchar(32)	悲观锁的类型。对于 TDStore，`KEY` 值代表对单条 key 加锁，`PRE_RANGE` 值代表对 key 区间加锁。
LOCK_MODE	varchar(32)	悲观锁的读写模式。对于 TDStore，`Write` 值代表加写锁，`Read` 值代表加读锁。
LOCK_STATUS	varchar(32)	悲观锁的状态。对于 TDStore，值为 `GRANTED`，即加锁成功。正在加的锁不会展示在 `data_locks` 表中，该信息可以通过查询 `data_lock_waits` 表获取。
LOCK_DATA	varchar(8192)	额外的信息，值由各个存储引擎自行决定。对于 TDStore，值为 `NULL`。
START_KEY	varchar(128)	对于单条 key 上的锁，值为 `NULL`；对于 key 区间上的锁，值代表 key 区间的左边界的十六进制编码。
END_KEY	varchar(128)	对于单条 key 上的锁，值代表该 key 的十六进制编码；对于 key 区间上的锁，值代表 key 区间的右边界的十六进制编码。
EXCLUDE_START_KEY	tinyint(1)	当且仅当悲观锁范围是一个 key 区间，且该区间为开区间时，值为1；否则，值为0。

BLOCKING_TRANSACTION_NUM	bigint unsigned	在该悲观锁上等待加锁的事务数量。
BLOCKING_CHECK_READ_TRANSACTION_NUM	bigint unsigned	在该悲观锁上等待快照读的事务数量。当持有悲观锁的事务进入提交流程的时候，悲观锁会阻塞快照读请求。
READ_LOCKED_NUM	bigint unsigned	该悲观锁上读锁的数量。
TINDEX_ID	int unsigned	悲观锁对应的 tindex ID。通过这个字段的值，我们可以执行 `SELECT * FROM information_schema.statistics WHERE tindex_id=...` 从系统表中获取 tindex ID 对应的表信息。
DATA_SPACE_TYPE	varchar(32)	悲观锁对应的 data space。`DATA_SPACE_TYPE_USER` 值代表对用户表的部分数据加锁，`DATA_SPACE_TYPE_SYSTEM` 值代表对系统表的部分数据加锁。
REPLICATION_GROUP_ID	bigint unsigned	悲观锁对应的 replication group ID。
KEY_RANGE_REGION_ID	bigint unsigned	悲观锁对应的 key range region ID。
PROCESSLIST_ID	bigint unsigned	对应开启该事务的会话连接（session）ID。
NODE_ID	bigint unsigned	开启该事务的 SQLEngine 节点 ID。
NODE_NAME	varchar(64)	开启该事务的 SQLEngine 节点名称。

说明：
TDSQL Boundless 暂未支持在 `data_locks` 表中直接展示库名、表名等信息。如果需要查看表信息，可以根据 `TINDEX_ID` 字段的值，直接从 `information_schema.statistics` 表查询。

示例

- 输出示例：

```
tdsql [(none)]> SELECT * FROM performance_schema.data_locks \G
```

```
*************************** 1. row ***************************
                    ENGINE: RocksDB
            ENGINE_LOCK_ID:
28681376014270466_(0000279780000006,00002798)
      ENGINE_TRANSACTION_ID: 28681376014270466
                 THREAD_ID: 44
                  EVENT_ID: NULL
             OBJECT_SCHEMA:
               OBJECT_NAME:
            PARTITION_NAME: NULL
         SUBPARTITION_NAME: NULL
                INDEX_NAME: NULL
      OBJECT_INSTANCE_BEGIN: 140502282795216
                 LOCK_TYPE: PRE_RANGE
                 LOCK_MODE: Write
               LOCK_STATUS: GRANTED
                 LOCK_DATA: NULL
                 START_KEY: 0000279780000006
                   END_KEY: 00002798
        EXCLUDE_START_KEY: 1
    BLOCKING_TRANSACTION_NUM: 0
BLOCKING_CHECK_READ_TRANSACTION_NUM: 0
           READ_LOCKED_NUM: 0
                 TINDEX_ID: 10135
           DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
       REPLICATION_GROUP_ID: 257
        KEY_RANGE_REGION_ID: 7312
             PROCESSLIST_ID: 2097282
                    NODE_ID: 2
                  NODE_NAME: node-1-002
*************************** 2. row ***************************
                    ENGINE: RocksDB
            ENGINE_LOCK_ID:
28681376014270466_0000279780000006
      ENGINE_TRANSACTION_ID: 28681376014270466
                 THREAD_ID: 44
                  EVENT_ID: NULL
             OBJECT_SCHEMA:
               OBJECT_NAME:
            PARTITION_NAME: NULL
```

```
                  SUBPARTITION_NAME: NULL
                         INDEX_NAME: NULL
               OBJECT_INSTANCE_BEGIN: 140507882943448
                         LOCK_TYPE: KEY
                         LOCK_MODE: Write
                       LOCK_STATUS: GRANTED
                         LOCK_DATA: NULL
                         START_KEY: NULL
                           END_KEY: 0000279780000006
                  EXCLUDE_START_KEY: 0
            BLOCKING_TRANSACTION_NUM: 0
BLOCKING_CHECK_READ_TRANSACTION_NUM: 0
                     READ_LOCKED_NUM: 0
                           TINDEX_ID: 10135
                     DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
                REPLICATION_GROUP_ID: 257
                 KEY_RANGE_REGION_ID: 7312
                      PROCESSLIST_ID: 1048704
                             NODE_ID: 1
                           NODE_NAME: node-1-001
```

- **查询示例：**

```
-- 查询所有节点的悲观锁数量
SELECT COUNT(*) FROM performance_schema.data_locks \G

-- 查询所有节点上，replication group 257 的 key range region 7312 的所有的悲观锁信息（仅指定 key_range_region_id 也可查询，但效率较差，无法利用索引加速查询）
SELECT * FROM performance_schema.data_locks WHERE replication_group_id = 257 AND key_range_region_id = 7312 \G

-- 查询所有节点上，和 [0000279780000003, 0000279780000004] 有交集的所有悲观锁的信息
SET @left = "0000279780000003";
SET @right = "0000279780000004";
SELECT * FROM performance_schema.data_locks WHERE
    (lock_type = 'KEY' AND @left <= end_key AND end_key <= @right) OR
-- 如果是单条 key 上的悲观锁，那么对应的 key 存储在 end_key 字段上，只要
```

```
end_key 在区间内即可
    (lock_type = 'PRE_RANGE' AND @left < end_key AND (start_key <
@right OR (NOT exclude_start_key and start_key = @right))) \G -- 如果是
key 区间上的悲观锁，那么需要根据 exclude_start_key，判断 [start_key,
end_key) 或 (start_key, end_key) 和区间是否有交集

-- 查询所有节点上，testdb.test 表所有悲观锁的信息
-- 第一步：获取 testdb.test 的 tindex_id
SELECT tindex_id FROM information_schema.statistics WHERE table_schema
= 'testdb' AND table_name = 'test';
-- 第二步：使用第一步获取到的 tindex_id 查询锁信息
SELECT * FROM performance_schema.data_locks WHERE tindex_id = 12345;

-- 查询所有节点上，事务 28681535313936395 所加所有悲观锁的信息
SELECT * FROM performance_schema.data_locks WHERE
engine_transaction_id = 28681535313936395 \G

-- 查询所有节点上，阻塞了事务的所有悲观锁的信息
SELECT * FROM performance_schema.data_locks WHERE
blocking_transaction_num > 0 OR blocking_check_read_transaction_num >
0
```

- **展示被阻塞的参与者信息：**

  下面展示了三个并发执行的 session。在这个例子中，session B 和 session C 的查询语句会被 session A 在表 t 上加的范围悲观锁阻塞。

  - session A：

    ```
    BEGIN;
    SELECT a FROM t FOR UPDATE;
    SELECT SLEEP(100);
    ```

  - session B：

    ```
    SELECT b FROM t FOR UPDATE;
    ```

  - session C：

    ```
    SELECT c FROM t FOR UPDATE;
    ```

此时，使用以下查询语句去查询哪些事务正在等待，以及它们正在被哪些事务阻塞。

```
SELECT blocking_engine_transaction_id, blocking_engine_lock_id,
requesting_engine_transaction_id, requesting_engine_lock_id,
tindex_id, replication_group_id, key_range_region_id FROM
performance_schema.data_lock_waits \G
```

结果如下：

blocking_engine_transaction_id	blocking_engine_lock_id	requesting_engine_transaction_id	requesting_engine_lock_id	tindex_id	replication_group_id	key_range_region_id
2868721849519315 5	2868721849519315 5_[0000281 9,0000281A)	2868721889784632 7	2868721889784632 7_[0000281 9,0000281A)	102 65	257	103181
2868721849519315 5	2868721849519315 5_[0000281 9,0000281A)	2868721874685134 9	2868721874685134 9_[0000281 9,0000281A)	102 65	257	103181

DATA_LOCK_WAITS

最近更新时间：2025-11-18 10:10:23

功能

`PERFORMANCE_SCHEMA.DATA_LOCK_WAITS` 用于展示集群中所有 hybrid 节点的 TDStore 上所有悲观锁等待信息。
在用户没有查询 `DATA_LOCK_WAITS` 表的时候， `DATA_LOCK_WAITS` 不会带来额外的开销。用户不需要将 `performance_schema` 系统变量设置为 `ON` ，即可查询 `DATA_LOCK_WAITS` 表。

字段说明

字段名	类型	描述
ENGINE	varchar(32)	存储引擎。对于 TDStore，值是 `RocksDB` 。
REQUESTING_ENGINE_LOCK_ID	varchar(128)	唯一标识被阻塞的事务正在申请的悲观锁，或者正在读取的 key 范围。对于 TDStore，其格式为 `<持有锁的事务 ID>_<锁范围>` 。 `<锁范围>` 的格式是： • 如果锁的范围是一个单条的 key，则 `<锁范围>` 为 key 的十六进制表示。例如：锁标识符 `2867377 8183569468_00002797` 表示 ID 为 2867377818356946 8的事务在十六进制编码为 `00002797` 的 key 上加了一个悲观锁。 • 如果锁的范围是一个 key 区间，则 `<锁范围>` 格式为 `[<hex_start_key>,<hex_end_key>)` （左闭右开）或 `(<hex_start_key>,<hex_end_key>)` （开区间）。例如：锁标识符 `2867377 8183569468_[00002797,00002798)` ，表示 ID 为 2867377818356946 8的事务在 `[0000 2797,00002798)` 区间加了一个悲观锁。 当 key 的十六进制编码过长的时候，可能无法在128字符内表示完全。这个时候，字段值的后两个字符会被置为 `..` ，表示这不是一个完整的锁标识符。这种情况下，用户通过 `ENGINE_LOCK_ID` 查询悲观锁信息时，也需要使用带有 `..` 的字段值做查询。
REQUESTING_NODE_ID	bigint unsigned	开启被阻塞事务的 SQLEngine 节点 ID。

REQUESTING_NODE_NAME	varchar(64)	开启被阻塞事务的 SQLEngine 节点名称。
REQUESTING_ENGINE_TRANSACTION_ID	bigint unsigned	被阻塞的事务的唯一标识符。对于 TDStore，如果被阻塞的是只读请求，则值为 `0` 。
REQUESTING_THREAD_ID	bigint unsigned	执行被阻塞的事务的线程 ID。Performance Schema 内部维护的线程 ID，可用于关联 Performance Schema 的各类事件表。
REQUESTING_EVENT_ID	bigint unsigned	导致被阻塞的事务加锁的 Performance Schema 事件 ID。对于 TDStore，值为 `NULL` 。
REQUESTING_OBJECT_INSTANCE_BEGIN	bigint unsigned	等锁请求在内存中的地址。
BLOCKING_ENGINE_LOCK_ID	varchar(128)	悲观锁的唯一标识。对于 TDStore ，格式参考 `data_locks` 表的 `ENGINE_LOCK_ID` 字段。
BLOCKING_NODE_ID	bigint unsigned	开启持有悲观锁的事务的 SQLEngine 节点 ID。
BLOCKING_NODE_NAME	varchar(64)	开启持有悲观锁的事务的 SQLEngine 节点名称。
BLOCKING_ENGINE_TRANSACTION_ID	bigint unsigned	持有悲观锁的事务的唯一标识符。如果这个悲观锁是读锁，被多个事务共同持有，则只会展示其中一个事务的唯一标识符。
BLOCKING_THREAD_ID	bigint unsigned	执行持有悲观锁的事务的线程 ID，是 Performance Schema 内部维护的线程 ID，可用于关联 Performance Schema 的各类事件表。
BLOCKING_EVENT_ID	bigint unsigned	导致持有悲观锁的事务加锁的 Performance Schema 事件 ID。对于 TDStore ，值为 `NULL` 。
BLOCKING_OBJECT_INSTANCE_BEGIN	bigint unsigned	悲观锁在内存中的地址。
TINDEX_ID	int unsigned	悲观锁对应的 tindex ID。通过这个字段的值，我们可以执行 `SELECT * FROM information_schema.statistics WHERE tindex_id=...` 从系统表中获取 tindex ID 对应的表信息。
DATA_SPACE_TYPE	varchar(32)	悲观锁对应的 data space。 `DATA_SPACE_TYPE_USER` 代表对用户表的部分数据加锁， `DATA_SPACE_T`

		YPE_SYSTEM 代表对系统表的部分数据加锁。
REPLICATION_GROUP_ID	bigint unsigned	悲观锁对应的 replication group ID。
KEY_RANGE_REGION_ID	bigint unsigned	悲观锁对应的 key range region ID。

示例

- 输出示例：

```
tdsql [(none)]> SELECT * FROM performance_schema.data_lock_waits \G
*************************** 1. row ***************************
                           ENGINE: RocksDB
       REQUESTING_ENGINE_LOCK_ID:
28681541555060779_[00002797,00002798)
             REQUESTING_NODE_ID: 2
           REQUESTING_NODE_NAME: node-1-002
REQUESTING_ENGINE_TRANSACTION_ID: 28681541555060779
            REQUESTING_THREAD_ID: 44
             REQUESTING_EVENT_ID: 0
REQUESTING_OBJECT_INSTANCE_BEGIN: 139759771095568
         BLOCKING_ENGINE_LOCK_ID: 28681535313936395_0000279780000001
               BLOCKING_NODE_ID: 1
             BLOCKING_NODE_NAME: node-1-001
  BLOCKING_ENGINE_TRANSACTION_ID: 28681535313936395
             BLOCKING_THREAD_ID: 44
              BLOCKING_EVENT_ID: 0
  BLOCKING_OBJECT_INSTANCE_BEGIN: 139759768476120
                      TINDEX_ID: 10135
                DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
           REPLICATION_GROUP_ID: 257
            KEY_RANGE_REGION_ID: 7312
```

- 查询示例：

```
-- 查询所有节点上，等待悲观锁的请求数量
SELECT COUNT(*) FROM performance_schema.data_lock_wais \G
```

```
-- 查询所有节点上，replication group 257 的 key range region 7312 的所有的悲观锁等待信息（仅指定 key_range_region_id 也可查询，但效率较差，无法利用索引加速查询）
SELECT * FROM performance_schema.data_lock_waits WHERE
replication_group_id = 257 AND key_range_region_id = 7312 \G

-- 查询所有节点上，事务 28682479434989637 正在等待的悲观锁的信息
SELECT blocking_engine_lock_id FROM performance_schema.data_lock_waits
WHERE requesting_engine_transaction_id = 28682479434989637;
-- engine_lock_id 指定为第一条 sql 查询获得的 blocking_engine_lock_id 字段值。
SELECT * FROM performance_schema.data_locks WHERE engine_lock_id =
'29417475275751454_00002C7B80000003';
```

METADATA_LOCKS

最近更新时间：2025-11-18 10:10:23

功能

METADATA_LOCKS 记录着当前 SQLEngine 元数据锁的占用情况，包含：

- 已授予的锁：显示哪些会话当前拥有哪些元数据锁。
- 已请求但尚未授予的锁：显示哪些会话正在等待哪些元数据锁。
- 已被死锁检测器终止的锁定请求。
- 已超时并等待请求会话释放锁的锁请求。

方便用户快速定位元数据锁（Metadata Locks，MDL）相关的问题，提高系统的稳定性和性能。

字段说明

字段名	类型	描述
OBJECT_TYPE	varchar(64)	锁定的对象类型，例如 TABLE、SCHEMA、FUNCTION、MC 等。 其中 MC 是 TDSQL Boundless 新增的全局对象锁，在集群层面对数据库对象加锁。
OBJECT_SCHEMA	varchar(64)	锁定对象所在的数据库名称。
OBJECT_NAME	varchar(64)	锁定对象的名称。
COLUMN_NAME	varchar(64)	锁定对象的具体列名称。
OBJECT_INSTANCE_BEGIN	bigint unsigned	锁定对象在内存中的地址。
LOCK_TYPE	varchar(32)	锁的类型。 • INTENTION_EXCLUSIVE：意图独占锁。这种锁通常用于表示一个事务打算获取一个独占锁，但目前还没有实际获取到。 • SHARED：共享锁。允许多个事务同时持有这种锁，但阻止任何事务获取独占锁。 • SHARED_HIGH_PRIO：高优先级共享锁。这是一种特殊类型的共享锁，用于高优先级的事务。 • SHARED_READ：共享读锁。允许事务读取数据，但阻止其他事务写入或获取独占锁。

		• SHARED_UPGRADABLE：可升级的共享锁。这种锁允许事务在持有共享锁的同时尝试获取独占锁，而不会被阻塞。 • SHARED_NO_WRITE：不允许写的共享锁。这种锁可能阻止事务写入数据，但允许读取。 • SHARED_NO_READ_WRITE：不允许读写的共享锁。这种锁阻止事务读取和写入数据。 • EXCLUSIVE：独占锁。只允许一个事务持有这种锁，阻止其他事务获取任何类型的锁。
LOCK_DURATION	varchar(32)	锁的持续时间，例如 TRANSACTION、LONG 等。
LOCK_STATUS	varchar(32)	表示每个锁的状态： • PENDING：当元数据锁被请求但未立即获得时，插入一行状态为 PENDING 的记录。 • GRANTED：当元数据锁被请求并立即获得时，插入一行状态为GRANTED 的记录。 • VICTIM：当死锁检测器取消挂起的锁请求以打破死锁（ER_LOCK_DEADLOCK）时，将其行状态从 PENDING 更新为VICTIM。这表示该锁行即将被删除。 • TIMEOUT：当挂起的锁请求超时（ER_LOCK_WAIT_TIMEOUT）时，将其行状态从 PENDING 更新为 TIMEOUT。这表示该锁行即将被删除。 • KILLED：当被授予的锁或挂起的锁请求被终止时，将其行状态从GRANTED 或 PENDING 更新为 KILLED。这表示该锁行即将被删除。
SOURCE	varchar(64)	包含生成事件的已检测代码的源文件的名称以及文件中发生检测的行号。
OWNER_THREAD_ID	bigint unsigned	持有锁的线程 ID。
OWNER_EVENT_ID	bigint unsigned	请求元数据锁的事件ID。

示例

```
#session1
BEGIN
```

```
UPDATE test1 SET k = 0 WHERE id = 999;

#session2
ALTER TABLE test1 ADD COLUMN new_column VARCHAR(255);

#查看metadata_locks可以看到ddl获取元数据锁被挂起
tdsql [performance_schema]> select * from metadata_locks where
OBJECT_NAME='test1' \G
*************************** 1. row ***************************
          OBJECT_TYPE: TABLE
        OBJECT_SCHEMA: etest
          OBJECT_NAME: test1
          COLUMN_NAME: NULL
OBJECT_INSTANCE_BEGIN: 140400821752448
            LOCK_TYPE: EXCLUSIVE
        LOCK_DURATION: TRANSACTION
          LOCK_STATUS: PENDING
               SOURCE: mdl.cc:3924
      OWNER_THREAD_ID: 5670
       OWNER_EVENT_ID: 1
1 row in set (0.00 sec)
```

INFORMATION_SCHEMA DDL_JOBS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.DDL_JOBS` 展示每个 DDL 任务的状态信息。

字段说明

字段名	类型	描述
id	bigint unsigned	唯一的 DDL 任务 ID。
schema_name	varchar(64)	DDL 任务对应的 schema 名。
table_name	varchar(64)	DDL 任务对应的 table 名。
version	bigint	DDL 任务对应的版本信息。
ddl_status	enum	DDL 任务执行状态。 • FAIL：失败。 • SUCCESS：成功。 • EXECUTING：正在执行。
start_timestamp	timestamp	DDL 任务开启时间。
last_timestamp	timestamp	最近一次 DDL 任务更新时间。
ddl_sql	longtext	DDL SQL。
info_type	enum	DDL 任务类型。
info	longtext	记录 DDL 任务执行过程中的必要信息。
is_history	bool	DDL 任务是否已经结束。 • 1：已结束。 • 0：尚未结束。

示例

```
tdsql> create table sbtest1 (id int primary key);
tdsql> select * from information_Schema.ddl_jobs order by id desc limit
1\G;
*************************** 1. row ***************************
             ID: 2
    SCHEMA_NAME: tdsql
     TABLE_NAME: sbtest1
        VERSION: 4
     DDL_STATUS: SUCCESS
START_TIMESTAMP: 2025-06-24 15:00:35
 LAST_TIMESTAMP: 2025-06-24 15:00:35
        DDL_SQL: create table sbtest1 (id int primary key)
      INFO_TYPE: CREATE TABLE
           INFO: {"rm_idx":[],"wf_rmed":false,"exec_addr":
{"ip":"10.10.10.10","port":15035},"row_apply_saved":true,"current_schema
_name":"tdsql","ddl_with_complete_info":"","crt_data_obj_task_id":293726
84622102841,"dstr_data_obj_task_id":0,"data_obj_to_be_dstr_arr":
[],"tbl_tidxid":10028,"row_applied":false,"recov_addr":
{"ip":"10.10.10.10","port":15035}}
     IS_HISTORY: 1
1 row in set (0.00 sec)
```

DDL_JOB_STAGE_INFO

最近更新时间：2025-11-18 10:10:23

功能

DDL 操作具有多阶段、长流程的特点，会分成多个阶段执行并且和 TDStore 有多次 RPC 交互。然而，在调试或者线上执行 DDL 时会有概率出现 DDL 卡死的现象。现在可以通过查询 `INFORMATION_SCHEMA.DDL_JOB_STAGE_INFO` 视图，展示每个正在执行的 DDL 任务的运行时信息，包括 DDL 若干阶段的耗时情况，RPC 耗时情况等，方便用户、DBA 或者内核开发人员在 DDL 卡住的时候查询该系统视图就能定位（不用特意去查看 DDL 运行日志）到当前 DDL 执行到的位置。

说明：

在 DDL 主执行节点上的推荐用法： `SELECT * FROM information_schema.DDL_JOB_STAGE_INFO ORDER BY ddl_job_id\G`

在非 DDL 主执行节点上的推荐用法： `/*#all_nodes */ SELECT * FROM information_schema.DDL_JOB_STAGE_INFO ORDER BY ddl_job_id\G`

字段说明

字段名	类型	描述
DDL_JOB_ID	int unsigned	当前正在执行的 DDL job id。
CURRENT_STAGE_TYPE	varchar(64)	• NORMAL：代表正在执行 DDL 前台线程逻辑。 • RECOVERY：代表正在执行 DDL 后台线程逻辑（例如异步 DROP Table 等等）。
CURRENT_STAGE_NAME	varchar(64)	表示当前 DDL 逻辑正在哪个函数中执行。
CURRENT_STAGE_SOURCE	varchar(64)	表示当前 DDL 执行的函数文件名和代码行数。
HISTORY_STAGE_INFOS	varchar(4096)	表示当前 DDL 执行的历史阶段信息，一般由 `START_DDL_JOB` 开始，到END_DDL_JOB 结束。
CURRENT_SQL_QUERY	varchar(2048)	表示当前 DDL 执行的 SQL。
DDL_START_TIME	varchar(64)	表示当前 DDL 开始执行的时间。

STAGE_START_TIME	varchar(64)	表示当前所在 stage 开始的时间。
DDL_DURATION	varchar(64)	表示当前 DDL 开始执行到现在时间节点的时间间隔。
STAGE_DURATION	varchar(64)	表示当前所在 stage 开始执行到现在时间节点的时间间隔。
DDL_ERROR_CODE	int	如果出现异常，会打印当前异常的异常码。反之，则为 `NULL` 。
DDL_ERROR_TEXT	varchar(2048)	如果出现异常，会打印当前异常的文本信息。反之，则为 `NULL` 。

示例

```
tdsql> SELECT * FROM information_schema.DDL_JOB_STAGE_INFO ORDER BY
ddl_job_id\G
*************************** 1. row ***************************
          DDL_JOB_ID: 115
  CURRENT_STAGE_TYPE: NORMAL
  CURRENT_STAGE_NAME: SetDDLJobStatusSucc
CURRENT_STAGE_SOURCE: ddl_common.cc:725
 HISTORY_STAGE_INFOS: [161ms][RPC 18ms][CREATE TABLE DDL][2025-09-01
15:09:37 ~ RUNNING]
 ├──> [OK ][25ms][RPC 1ms][START_DDL_JOB][start_ddl_job]
[ddl_worker.cc:994][2025-09-01 15:09:37 ~ 2025-09-01 15:09:37]
 ├──> [RUN][135ms][RPC 17ms][CREATE_TABLE_NO_LOCK]
[mysql_create_table_no_lock][sql_table.cc:9993][2025-09-01 15:09:37 ~
RUNNING]
      ├──> [RUN][135ms][RPC 17ms][CREATE_TABLE_IMPL][create_table_impl]
[sql_table.cc:9539][2025-09-01 15:09:37 ~ RUNNING]
           ├──> [OK ][112ms][RPC 9ms][CREATE_DATA_OBJECTS_FOR_TABLE]
[CreateDataObjectsForTable][data_object.cc:2150][2025-09-01 15:09:37 ~
2025-09-01 15:09:37]
                ├──> [OK ][103ms][RPC 3ms][CREATE_DATA_OBJECT]
[CreateDataObjects][data_object.cc:1327][2025-09-01 15:09:37 ~ 2025-09-
01 15:09:37]
           ├──> [OK ][0ms][RPC 0ms][CREATE_AUTO_INC_FOR_TABLE]
[CreateAutoIncForTable][auto_inc_index.cc:247][2025-09-01 15:09:37 ~
```

```
2025-09-01 15:09:37]
            ├──> [OK ][5ms][RPC 4ms][CREATE_WRITE_FENCE_FOR_TABLE]
[CreateWriteFenceForTable][ddl_executer.cc:2480][2025-09-01 15:09:37 ~
2025-09-01 15:09:37]
                ├──> [OK ][5ms][RPC 4ms][CREATE_WRITE_FENCE]
[CreateWriteFence][ddl_executer.cc:1269][2025-09-01 15:09:37 ~ 2025-09-
01 15:09:37]
            ├──> [OK ][0ms][RPC 0ms][SET_DDL_JOB_STATUS_SUCCESS]
[SetDDLJobStatusSucc][ddl_common.cc:725][2025-09-01 15:09:37 ~ 2025-09-
01 15:09:37]
   CURRENT_SQL_QUERY: CREATE TABLE sbtest85(
  id INTEGER NOT NULL,
  k INTEGER DEFAULT '0' NOT NULL,
  c CHAR(120) DEFAULT '' NOT NULL,
  pad CHAR(60) DEFAULT '' NOT NULL,
  PRIMARY KEY (id)
) /*! ENGINE = rocksdb */
      DDL_START_TIME: 2025-09-01 15:09:37
    STAGE_START_TIME: 2025-09-01 15:09:37
        DDL_DURATION: 161ms
      STAGE_DURATION: 3ms
      DDL_ERROR_CODE: NULL
      DDL_ERROR_TEXT: NULL
```

LOGSERVICE_MYSQL_CLIENT

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.LOGSERVICE_MYSQL_CLIENT` 展示日志回放 MySQL 模式下每个 MySQL 连接的执行状态。

字段说明

字段名	类型	描述
sid	bigint unsigned	log service 的唯一 ID。
type	smallint	区分是 sys rg 还是 user rg 专用队列。
ip	varchar(64)	MySQL 连接使用的 IP 地址。
port	smallint	MySQL 实例使用的端口。
queue_size	bigint	该连接目前队列中剩余的未消费的消息数量。
err_msg	longtext	该连接状态的错误信息。

示例

```
tdsql> /*all_nodes*/select * from LOGSERVICE_MYSQL_CLIENT limit 1\G
*************************** 1. row ***************************
       sid: 29374627457269768
      type: 1
        ip: domain_10_10_10_10
      port: 9054
queue_size: 0
   err_msg:
1 row in set (0.00 sec)
```

LOGSERVICE_PROCESSLIST

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.LOGSERVICE_PROCESSLIST` 展示日志回放每个 RG 的回放进度。

字段说明

字段名	类型	描述
sid	bigint unsigned	log service 的唯一 ID。
rep_group_id	bigint unsigned	RG 的 ID。
raft_log_index	bigint	当前同步的 Raft log 的 index。
state	varchar(64)	当前 RG 同步的状态。
is_unhealthy	smallint	当前同步状态是否健康。
min_commit_ts_not_send	bigint unsigned	最小的还未回放的日志 commit_ts。
leader_ts	varchar(64)	当前 RG leader 的时间戳。
last_index_in_checkpoint	bigint unsigned	上一个回放的日志 index。
next_index_in_checkpoint	bigint unsigned	下一个将要回放的日志 index。
unconsumed_count_in_checkpoint	bigint unsigned	还未回放的日志数量。
backup_start_index	bigint	获取备份日志的起始 index。
backup_end_index	bigint	获取备份日志的终止 index。
backuped_index	bigint	已经备份的日志 index。

last_leader_change_ts	varchar(64)	RG leader 上一次变化的时间。
err_msg	longtext	RG 回放时错误信息。

示例

```
 tdsql>select * from LOGSERVICE_PROCESSLIST\G
 *************************** 1. row ***************************
                            sid: 29057889255555076
                 rep_group_id: 256
               raft_log_index: 797509
                        state: WaitRaftLog
                 is_unhealthy: 0
             commit_ts_barrier: 29057889222000822
       min_commit_ts_not_send: NULL
                    leader_ts: 2024-11-19 10:59:34
     last_index_in_checkpoint: 797509
     next_index_in_checkpoint: NULL
unconsumed_count_in_checkpoint: 0
            backup_start_index: 0
              backup_end_index: 0
                backuped_index: 0
        last_leader_change_ts: 2024-11-19 10:59:29
                      err_msg:
1 row in set (0.00 sec)
```

LOGSERVICE_STAT

最近更新时间：2025-11-18 10:10:23

功能

LogService 当前状态信息视图。

字段说明

字段名	类型	描述
client_type	varchar(64)	LogService 类型。
sid	bigint unsigned	LogService 唯一 ID。
product_msgs	bigint unsigned	产生的消息数量。
send_success_msgs	bigint unsigned	发送成功的消息数量。
consume_msgs	bigint unsigned	消费的消息数量。
msg_count_in_conflict_detector	bigint unsigned	在冲突处理器缓存中的消息数量（仅 MysqlClient 模式）。
unconsumed_msg_count	bigint unsigned	整体缓存中的消息数量（仅 MysqlClient 模式）。
send_avg_ms	bigint unsigned	每毫秒发送的消息数量。
send_lastest_ms	bigint unsigned	最新一毫秒中发送的消息数量。
send_kb_total	bigint unsigned	总体发送的数据量（单位 KB）。
send_avg_kb_per_second	bigint unsigned	每秒发送的数据量（KB）。

示例

```
tdsql> SELECT * FROM LOGSERVICE_STAT\G
*************************** 1. row ***************************
                   client_type: MYSQL
                           sid: 29472565546188807
                  product_msgs: 112313
             send_success_msgs: 112313
                  consume_msgs: 0
msg_count_in_conflict_detector: 0
           unconsumed_msg_count: 0
                    send_avg_ms: 1657588598
                send_lastest_ms: 270
                 consume_avg_ms: 0
             consume_lastest_ms: 0
                  send_kb_total: 26087
         send_avg_kb_per_second: 0
1 row in set (0.00 sec)
```

META_CLUSTER_DATA_OBJECTS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.META_CLUSTER_DATA_OBJECTS` 展示实例内数据对象相关的元信息。

字段说明

字段名	类型	描述
data_obj_id	int unsigned	数据对象 ID。
data_obj_name	varchar(64)	数据对象命名。
data_obj_type	varchar(64)	数据对象类型。
data_space_type	varchar(64)	数据空间类型。
distribution_policy_id	int unsigned	绑定的 DP ID。
schema_name	varchar(64)	数据对象命名。
table_name	varchar(64)	表名。
partition_name	varchar(64)	一级分区名（如果不是一级分区则为空）。
sub_partition_name	varchar(64)	二级分区名（如果不是二级分区则为空）。
index_name	varchar(64)	索引名（如果为主键索引则为 `PRIMARY`）。

示例

```
tdsql>select * from INFORMATION_SCHEMA.META_CLUSTER_DATA_OBJECTS Limit
1\G
```

```
*************************** 1. row ***************************
           data_obj_id: 268
         data_obj_name: time_zone
         data_obj_type: BASE_TABLE
       data_space_type: DATA_SPACE_TYPE_SYSTEM
distribution_policy_id: 1
           schema_name: mysql
            table_name: time_zone
        partition_name:
    sub_partition_name:
            index_name: PRIMARY
1 row in set (0.01 sec)
```

META_CLUSTER_LEADER_HISTORY

最近更新时间：2025-11-18 10:10:23

功能

`META_CLUSTER_LEADER_HISTORY` 展示 MC Leader 切换记录。

字段说明

字段名	类型	描述
mc_node_id	bigint unsigned	MC 节点 ID。
mc_node_name	varchar(64)	MC 节点名称。
peer_urls	varchar(320)	Raft 成员通信地址。
client_urls	varchar(320)	Client 通信地址。
transfer_time	varchar(64)	切主时间。
election_term	bigint unsigned	选主任期号。

示例

```
tdsql>select * from INFORMATION_SCHEMA.META_CLUSTER_LEADER_HISTORY Limit
1\G
*************************** 1. row ***************************
                mc_node_id: 9245207174208224599
             mc_node_name: mc-Module3-919-0
                peer_urls: [http://10.10.10.10:11002]
              client_urls: [http://10.10.10.10:11001]
            transfer_time: 2024-08-19 11:16:54.507003
            election_term: 2
1 row in set (0.01 sec)
```

META_CLUSTER_NODES

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.META_CLUSTER_NODES` 用于展示实例内节点的元信息。

字段说明

字段名	类型	描述
node_id	bigint unsigned	节点 ID。
node_name	varchar(64)	节点 Name。
conn_id	int unsigned	节点连接 ID。
rpc_addr	varchar(64)	RPC 地址。
sql_addr	varchar(64)	SQL 地址。
node_term	bigint unsigned	节点任期。
node_version	varchar(64)	节点版本号。
node_state	varchar(64)	节点状态。
capacity	bigint unsigned	节点容量。
available	bigint unsigned	节点可用量。
leader_count	int	节点 Leader 数量。
learner_count	int	节点 Learner 数量。
rep_group_count	int	节点 RG 数量。

leader_size	bigint	节点 Leader 类型副本占用的空间。
learner_size	bigint	节点 Learner 类型副本占用的空间。
rep_group_size	bigint	节点所有类型副本占用的空间。
last_report_time	varchar(64)	节点 TDStore 心跳最后上报时间。
global_earliest_snapshot	bigint unsigned	全局最早 snapshot 时间。
local_earliest_snapshot	bigint unsigned	该节点 TDStore 最早 snapshot 时间。
engine_local_earliest_snapshot	bigint unsigned	该节点 SQLEngine 最早 snapshot 时间。
engine_last_report_time	varchar(64)	节点 SQLEngine 心跳最后上报时间。
leader_weight	double	Leader 权重。
replica_weight	double	副本权重。
node_labels	longblob	节点标签。

示例

```
tdsql>select * from information_schema.meta_cluster_nodes limit 1\G
*************************** 1. row ***************************
                  node_id: 1
                node_name: node-tdsql3-86ea1ffe-001
                  conn_id: 1
                 rpc_addr: 10.0.36.9:6005
                 sql_addr: 10.0.36.9:6008
                node_term: 28924771945152603
             node_version: sqlengine-18.1.0
               node_state: NODE_STATE_UP
                 capacity: 21474836480
                available: 8415584256
             leader_count: 0
            learner_count: 0
          rep_group_count: 2
```

```
                    leader_size: 0
                  learner_size: 0
                rep_group_size: 9730233
              last_report_time: 2024-08-19 17:10:46.312314
      global_earliest_snapshot: 28924903780515894
       local_earliest_snapshot: 28924904048951464
engine_local_earliest_snapshot: 28924904048951460
       engine_last_report_time: 2024-08-19 17:10:45.042880
                 leader_weight: 1
                replica_weight: 1
                   node_labels: key: "tdsql-migrate-node-task-dst-node"
1 row in set (0.01 sec)
```

META_CLUSTER_REGIONS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.META_CLUSTER_REGIONS` 表用于存储和展示实例内数据分片（Region）相关的元信息。

字段说明

字段名	类型	描述
region_id	bigint unsigned	Region ID。
rep_group_id	bigint unsigned	数据分片所属的 RG ID。
data_obj_id	bigint unsigned	DataObject ID。
data_obj_type	varchar(64)	DataObject 类型。
start_key	varchar(64)	Key 区间起始位置。
end_key	varchar(64)	Key 区间结束位置。
create_time	varchar(64)	创建时间。
parent_region_id	bigint unsigned	如果发生分裂则为父 Region ID，否则值为 `0`。
region_stats_approximate_size	bigint unsigned	数据分片的近似数据量大小，单位为 bytes。
region_stats_approximate_keys	bigint unsigned	数据分片中 key 的近似数量。

示例

```
tdsql>select * from information_schema.meta_cluster_regions where
region_id=179\G
*************************** 1. row ***************************
                    region_id: 179
                rep_group_id: 1792
                 data_obj_id: 256
               data_obj_type: BASE_TABLE
                   start_key: 00000100
                     end_key: 00000101
                 create_time: 2024-08-19 14:57:42.302908
            parent_region_id: 0
region_stats_approximate_size: 1231749
region_stats_approximate_keys: 1180
1 row in set (0.01 sec)
```

META_CLUSTER_REPLICAS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.META_CLUSTER_REPLICAS` 用于展示实例内 RG 的所有副本相关的元信息。

字段说明

字段名	类型	描述
rep_group_id	bigint unsigned	RG ID。
member_role	varchar(64)	副本角色，支持 `MEMBER_ROLE_FOLLOWER`、`MEMBER_ROLE_LEADER`、`MEMBER_ROLE_LEARNER`。
rep_group_state	varchar(64)	RG 状态。
meta_version	bigint	元信息版本号。
member_version	bigint	成员变更版本号。
key_range_version	bigint	Key 区间版本号。
quorum	int unsigned	法定 Quorum。
node_name	varchar(320)	所在的节点名。
tdstore_ip_port	varchar(64)	所在 TDStore 节点地址。
last_report_time	varchar(64)	最后心跳上报的时间。
last_working_time	varchar(64)	副本为工作状态（`WORKING`、`SPLIT`、`MERGE`）的最后时间。

rep_group_stats_approximate_size	bigint unsigned	副本的近似数据量大小，单位为 bytes。
rep_group_stats_approximate_keys	bigint unsigned	副本中 key 的近似数量。
rep_group_log_info_current_term	bigint unsigned	当前 Leader 任期。
rep_group_log_info_committed_index	bigint unsigned	已提交的日志号。
rep_group_log_info_consecutive_applied_index	bigint unsigned	已应用的日志号。
rep_group_log_info_last_snapshot_index	bigint unsigned	最后一次做快照的位点日志号。
rep_group_log_info_first_index	bigint unsigned	内存中保留的 Raft Log 首位日志号，表示之前的日志都被 Purge 了。
rep_group_log_info_last_index	bigint unsigned	内存中保留的 Raft Log 末位日志号。
rep_group_log_info_disk_index	bigint unsigned	已落盘的 Raft Log 日志号。
rep_group_log_info_applied_index	bigint unsigned	已应用的 Raft Log 日志号。
rep_group_log_info_raft_log_sync_delay_seconds	bigint unsigned	副本日志同步延迟。

示例

```
tdsql>select * from information_schema.META_CLUSTER_REPLICAS limit 1 \G
*************************** 1. row ***************************
                    rep_group_id: 1792
                     member_role: MEMBER_ROLE_FOLLOWER
                rep_group_state: RG_STATE_F_WORKING
                    meta_version: 36
                  member_version: 0
                key_range_version: 36
                          quorum: 3
                       node_name: node-tdsql3-86ea1ffe-001
                  tdstore_ip_port: 10.0.36.9:6005
                 last_report_time: 2024-08-19
17:30:01.444681
                last_working_time: 2024-08-19
17:30:01.444681
      rep_group_stats_approximate_size: 9720817
      rep_group_stats_approximate_keys: 3825
       rep_group_log_info_current_term: 9
    rep_group_log_info_committed_index: 8011
  rep_group_log_info_consecutive_applied_index: 8011
 rep_group_log_info_last_snapshot_index: 7997
        rep_group_log_info_first_index: 2
         rep_group_log_info_last_index: 8011
         rep_group_log_info_disk_index: 8011
      rep_group_log_info_applied_index: 8011
rep_group_log_info_raft_log_sync_delay_seconds: 0
1 row in set (0.01 sec)
```

META_CLUSTER_RGS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.META_CLUSTER_RGS` 用于存储和展示实例中复制组（Replication Group，RG）相关的元信息。

字段说明

字段名	类型	描述
rep_group_id	bigint unsigned	RG ID。
data_space_type	varchar(64)	数据空间类型。 • `DATA_SPACE_TYPE_SYSTEM`：系统空间。 • `DATA_SPACE_TYPE_USER`：用户空间。
rep_group_state	varchar(64)	RG 状态。
meta_version	bigint	元信息版本号。
member_version	bigint	成员变更版本号。
key_range_version	bigint	key 区间版本号。
quorum	int unsigned	法定 Quorum。
member_node_names	varchar(320)	成员列表。
leader_node_name	varchar(320)	Leader 成员。
last_leader_report_time	varchar(64)	最后心跳上报的时间。
create_time	varchar(64)	RG 的创建时间。

parent_rep_group_id	bigint unsigned	如果发生分裂则体现父 RG ID，否则值为 `0` 。
rep_group_stats_approximate_size	bigint unsigned	RG Leader 的近似数据量大小，单位为 bytes。
rep_group_stats_approximate_keys	bigint unsigned	RG Leader 中 key 的近似数量。
rep_group_log_info_current_term	bigint unsigned	当前 Leader 任期。
rep_group_log_info_committed_index	bigint unsigned	已提交的日志号。
rep_group_log_info_consecutive_applied_index	bigint unsigned	已应用的日志号。
rep_group_log_info_last_snapshot_index	bigint unsigned	最后一次做快照的位点日志号。
rep_group_log_info_first_index	bigint unsigned	内存中保留的 Raft Log 首位日志号，表示之前的日志都被 Purge 了。
rep_group_log_info_last_index	bigint unsigned	内存中保留的 Raft Log 末位日志号。
rep_group_log_info_disk_index	bigint unsigned	已落盘的 Raft Log 日志号。
rep_group_log_info_applied_index	bigint unsigned	已应用的 Raft Log 日志号。
rep_group_log_info_raft_log_sync_delay_seconds	bigint unsigned	副本日志同步延迟。

property	varchar(64)	RG属性： • `RG_PROPERTY_DEFAULT`：普通 RG。 • `RG_PROPERTY_BROADCAST`：广播 RG。 • `RG_PROPERTY_SYNC`：同步 RG。 • `RG_PROPERTY_BROADCAST_SYNC`：广播同步 RG。
witness_quorum	int unsigned	witness 法定 Quorum。
witnesses	longtext	witness 成员。

示例

```
tdsql>SELECT * FROM information_schema.meta_cluster_rgs\G;
*************************** 1. row ***************************
                        rep_group_id: 1792
                    data_space_type: DATA_SPACE_TYPE_SYSTEM
                    rep_group_state: RG_STATE_L_WORKING
                       meta_version: 36
                     member_version: 0
                   key_range_version: 36
                              quorum: 3
                   member_node_names: [node-tdsql3-86ea1ffe-
001, node-tdsql3-86ea1ffe-002, node-tdsql3-86ea1ffe-003]
                    leader_node_name: node-tdsql3-86ea1ffe-002
              last_leader_report_time: 2024-08-19
16:31:47.866168
                         create_time: 2024-08-19
14:56:47.692831
                 parent_rep_group_id: 0
     rep_group_stats_approximate_size: 9725997
     rep_group_stats_approximate_keys: 3836
      rep_group_log_info_current_term: 9
    rep_group_log_info_committed_index: 6783
  rep_group_log_info_consecutive_applied_index: 6783
       rep_group_log_info_last_snapshot_index: 6727
        rep_group_log_info_first_index: 2
         rep_group_log_info_last_index: 6783
```

```
                rep_group_log_info_disk_index: 6783
             rep_group_log_info_applied_index: 6783
rep_group_log_info_raft_log_sync_delay_seconds: 11
                                    property: RG_PROPERTY_DEFAULT
                                 witness_quorum: 0
                                      witnesses: []
*************************** 2. row ***************************
                                  rep_group_id: 1281
                               data_space_type: DATA_SPACE_TYPE_USER
                               rep_group_state: RG_STATE_L_WORKING
                                  meta_version: 3
                                member_version: 0
                             key_range_version: 3
                                        quorum: 3
                             member_node_names: [node-tdsql3-86ea1ffe-
001, node-tdsql3-86ea1ffe-002, node-tdsql3-86ea1ffe-003]
                              leader_node_name: node-tdsql3-86ea1ffe-003
                       last_leader_report_time: 2024-08-19
16:31:49.145344
                                   create_time: 2024-08-19
14:56:47.817777
                           parent_rep_group_id: 0
             rep_group_stats_approximate_size: 3070
             rep_group_stats_approximate_keys: 27
              rep_group_log_info_current_term: 3
           rep_group_log_info_committed_index: 864
  rep_group_log_info_consecutive_applied_index: 864
       rep_group_log_info_last_snapshot_index: 831
                rep_group_log_info_first_index: 2
                 rep_group_log_info_last_index: 864
                 rep_group_log_info_disk_index: 864
             rep_group_log_info_applied_index: 864
rep_group_log_info_raft_log_sync_delay_seconds: 18446744073709551615
                                    property: RG_PROPERTY_DEFAULT
                                 witness_quorum: 0
                                      witnesses: []

2 rows in set (0.01 sec)
```

META_CLUSTER_SCHEDULE_CONFIGS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.META_CLUSTER_SCHEDULE_CONFIGS` 用于展示 MC 调度参数。

字段说明

字段名	类型	描述
config_key	varchar(64)	参数名。
config_value	varchar(64)	参数值。
default	varchar(64)	默认值。
unit	varchar(64)	单位。
adjustable	varchar(64)	是否动态可调。
constraint_type	varchar(64)	约束类型，如 `section`、`enum`。
constraint_enum	varchar(64)	enum 列表。
constraint_range_min	varchar(64)	section 范围区间最小值。
constraint_range_max	varchar(64)	section 范围区间最大值。

示例

```
tdsql>select * from information_schema.meta_cluster_schedule_configs
limit 10;
```

```
+-----------------------------------------+-------------+-------------
-+------+-----------+-----------------+---------------------------+---
------------------+----------------------+
| config_key                              | config_value | default
| unit | adjustable | constraint_type | constraint_enum           |
constraint_range_min | constraint_range_max |
+-----------------------------------------+-------------+-------------
-+------+-----------+-----------------+---------------------------+---
------------------+----------------------+
| split-rep-group-job-timeout             | 600          | 600
| s    | 1          | section         |                           | 13
| 604800               |
| advance-global-earliest-snapshot-enabled | 1           | 1
|      | 1          | enum            | 0,1                       | 13
| 604800               |
| min-keys-read                           | 128          | 128
|      | 1          | section         |                           | 1
| 18446744073709551615 |
| max-table-count-per-replication-group   | 100          | 100
|      | 1          | section         |                           | 1
| 10000                |
| replay-job-log-timestamp-guard-interval | 3            | 3
| s    | 1          | section         |                           | 1
| 86400                |
| rebalance-leader-mode                   | leader-count | leader-count
|      | 1          | enum            | hot-rep-group,leader-count | 1
| 86400                |
| check-flush-route-interval              | 300          | 10
| s    | 1          | section         |                           | 1
| 36000                |
| raw-socket-handshake-timeout            | 1000         | 1000
| ms   | 1          | section         |                           | 500
| 2000                 |
| update-hot-data-objects-interval        | 15           | 15
| s    | 1          | section         |                           | 5
| 600                  |
| low-space-ratio                         | 0.95         | 0.95
|      | 1          | section         |                           |
0.25                 | 0.99                 |
```

```
+-----------------------------------------+--------------+-------------
-+-------+------------+-----------------+-----------------------------+---
------------------+----------------------+
10 rows in set (0.01 sec)
```

PARTITION_POLICIES

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.PARTITION_POLICIES` 用于查询系统当中存在的 PARTITION POLICY 规则，`INFORMATION_SCHEMA.PARITTION_POLICIES` 和 `INFORMATION_SCHEMA.PARTITION_POLICY_PARTITIONS` 组成 PARTITION POLICY 规则定义，类似于 `INFORMATION_SCHEMA.TABLES` 和 `INFORMATION_SCHEMA.PARTITIONS` 。

字段说明

字段名	类型	描述
ID	BIGINT UNSIGNED	每个 PARTITION POLICY 拥有唯一 ID。
NAME	VARCHAR(64)	每个 PARTITION POLICY 拥有唯一 NAME。
PARTITION_TYPE	ENUM	分区类型，支持 • HASH：基于给定列的哈希函数结果进行分区。 • KEY_51：类似于 HASH 方式，但可以指定多个列，数据内核计算这些列的哈希函数结果进行分区。 • KEY_55：类似于 KEY_51，KEY_51和 KEY_55 内部采用哈希算法不同 • LINEAR_HASH：线性哈希分区，与 HASH 类似，但使用线性哈希算法，使得数据分布更均匀。 • LINEAR_KEY_51：类似于 KEY_51，但采用线性哈希算法。 • LINEAR_KEY_55：类似于 KEY_55，但采用线性哈希算法。 • RANGE：基于给定列的范围进行分区。每个分区包含某一范围内的数据。 • LIST：基于给定列的离散值列表进行分区。 • RANGE_COLUMNS：类似于 RANGE 分区，但允许基于多个列的范围进行分区。 • LIST_COLUMNS：类似于 LIST 分区，但允许基于多个列的离散值列表进行分区。 • AUTO：自动分区类型，系统会根据数据的特点自动选择合适的分区策略。

		• AUTO_LINEAR：自动线性分区类型，系统会根据数据的特点自动选择合适的线性分区策略。
PARTITION_EXPRESSION	VARCHAR(2048)	创建表的当前分区方案的 `CREATE TABLE` 或 `ALTER TABLE` 语句中使用的分区函数的表达式。
SUBPARTITION_TYPE	ENUM	子分区类型，支持 • HASH • KEY_51 • KEY_55 • LINEAR_HASH • LINEAR_KEY_51 • LINEAR_KEY_55
SUBPARTITION_EXPRESSION	VARCHAR(2048)	子分区表达式，定义方式同 `PARTITION_EXPRESSION` 。
HIDDEN	ENUM('Explicit', 'Implicit')	用户显式创建出的 PARTITION POLICY 或是数据库自动创建的隐式 PARTITION POLICY。
SE_PRIVATE_DATA	MEDIUMTEXT	预留字段。

示例

```
tdsql> SELECT * FROM information_schema.partition_policies;
+----+----------------+----------------+----------------------+---------
----------+--------------------------+----------+-----------------+
| ID | NAME           | PARTITION_TYPE | PARTITION_EXPRESSION |
SUBPARTITION_TYPE | SUBPARTITION_EXPRESSION | HIDDEN   | SE_PRIVATE_DATA
|
+----+----------------+----------------+----------------------+---------
----------+--------------------------+----------+-----------------+
|  1 | impl_pp_hash_4 | HASH           | INTEGER              | NULL
| NULL                     | Implicit | NULL            |
+----+----------------+----------------+----------------------+---------
----------+--------------------------+----------+-----------------+
1 row in set (0.01 sec)
```

PARTITION_POLICY_AFFINITIES

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.PARTITION_POLICY_AFFINITIES` 用于描述 PARTITION POLICY 内的亲和性列表。用户表绑定到 PARTITION POLICY，PARTITION POLICY AFFINITIES 维护哪些用户表、用户分区具有亲和性。

字段说明

字段名	类型	描述
PARTITION_POLICY_ID	BIGINT UNSIGNED	PARTITION POLICY 的唯一 ID。
PARTITION_POLICY_NAME	VARCHAR(64)	PARTITION POLICY 的唯一 NAME。
PARTITION_ID	BIGINT UNSIGNED	PARTITION POLICY PARTITION 的唯一 ID。
PARTITION_NAME	VARCHAR(64)	PARTITION POLICY PARTITION 的唯一 NAME。
SUBPARTITION_NAME	VARCHAR(64)	PARTITION POLICY SUBPARTITION 的唯一 NAME。
DATA_OBJECT_TYPE	ENUM	该数据对象的类型。 • BASE_TABLE：表。 • BASE_INDEX：索引。 • AUTOINC：用户定义自增列的自增值分配器。 • BASE_HIDDEN_PK：隐藏主键的自增值分配器。 • PARTITION_L1：一级分区。 • PARTITION_L1_INDEX：一级分区索引。 • PARTITION_L1_HIDDEN_PK：一级分区隐藏主键的自增值分配器。 • PARTITION_L2：二级分区。 • PARTITION_L2_INDEX：二级分区索引。

		• PARTITION_L2_HIDDEN_PK：二级分区隐藏主键的自增值分配器。 • UNKNOWN：未知类型。
DATA_OBJECT_NAME	VARCHAR(259)	该数据对象的 NAME。
DATA_OBJECT_TINDEX_ID	INT UNSIGNED	该数据对象的全局 ID。
DATA_OBJECT_KEY_RANGE_START	VARCHAR(8)	该数据对象的数据范围起始点（闭区间）。
DATA_OBJECT_KEY_RANGE_END	VARCHAR(8)	该数据对象的数据范围终止点（开区间）。
HIDDEN	ENUM('Explicit', 'Implicit')	用户显式创建出的 PARTITION POLICY 或是数据库自动创建的隐式PARTITION POLICY。
SE_PRIVATE_DATA	MEDIUMTEXT	预留字段。

示例

```
tdsql> SELECT * FROM information_schema.partition_policy_affinities
order by partition_policy_id, partition_id;
+--------------------+----------------------+--------------+----------
------+------------------+-----------------+------------------+-------
---------------+-----------------------------+-------------------------
--+----------+-----------------+
| PARTITION_POLICY_ID | PARTITION_POLICY_NAME | PARTITION_ID |
PARTITION_NAME | SUBPARTITION_NAME | DATA_OBJECT_TYPE | DATA_OBJECT_NAME
| DATA_OBJECT_TINDEX_ID | DATA_OBJECT_KEY_RANGE_START |
DATA_OBJECT_KEY_RANGE_END | HIDDEN    | SE_PRIVATE_DATA |
+--------------------+----------------------+--------------+----------
------+------------------+-----------------+------------------+-------
---------------+-----------------------------+-------------------------
--+----------+-----------------+
|                  7 | impl_pp_hash_2       |           12 | p0
| NULL              | PARTITION_L1      | db0.t_hash2_1.p0 |
```

```
10049 | 00002741                    | 00002742                    |
Implicit | NULL             |
|                   7 | impl_pp_hash_2         |           12 | p0
| NULL              | PARTITION_L1      | db0.t_hash2_2.p0 |
10052 | 00002744                    | 00002745                    |
Implicit | NULL             |
|                   7 | impl_pp_hash_2         |           13 | p1
| NULL              | PARTITION_L1      | db0.t_hash2_1.p1 |
10050 | 00002742                    | 00002743                    |
Implicit | NULL             |
|                   7 | impl_pp_hash_2         |           13 | p1
| NULL              | PARTITION_L1      | db0.t_hash2_2.p1 |
10053 | 00002745                    | 00002746                    |
Implicit | NULL             |
+--------------------+-----------------------+--------------+----------
------+-------------------+------------------+------------------+-------
----------------+----------------------------+-------------------------
--+----------+-----------------+
4 rows in set (0.44 sec)
```

PARTITION_POLICY_PARTITIONS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.PARTITION_POLICY_PARTITIONS` 用于查询系统当中存在的 PARTITION POLICY 规则，`INFORMATION_SCHEMA.PARITTION_POLICIES` 和 `INFORMATION_SCHEMA.PARTITION_POLICY_PARTITIONS` 组成 PARTITION POLICY 规则定义，类似于 `INFORMATION_SCHEMA.TABLES` 和 `INFORMATION_SCHEMA.PARTITIONS` 。

PARTITION POLICY PARTITION 是 PARTITION POLICY 内的一个分区，类似于 PARTITION 和 TABLE 的关系。

字段说明

字段名	类型	描述
PARTITION_POLICY_ID	BIGINT UNSIGNED	PARTITION POLICY 的唯一 ID。
PARTITION_POLICY_NAME	VARCHAR(64)	PARTITION POLICY 的唯一 NAME。
PARTITION_ID	BIGINT UNSIGNED	每个 PARTITION POLICY PARTITION 拥有唯一 ID。
PARTITION_NAME	VARCHAR(64)	分区 NAME。
SUBPARTITION_NAME	VARCHAR(64)	子分区 NAME。
PARTITION_ORDINAL_POSITION	INT UNSIGNED	分区位置，即这是表中的第几个分区。
SUBPARTITION_ORDINAL_POSITION	INT UNSIGNED	子分区位置，即这是分区中的第几个子分区。
PARTITION_METHOD	VARCHAR(13)	分区类型，常见的有 RANGE、LIST、HASH、KEY。
SUBPARTITION_METHOD	VARCHAR(13)	子分区类型。

PARTITION_EXPRESSION	VARCHAR(2048)	创建表的当前分区方案的 `CREATE TABLE` 或 `ALTER TABLE` 语句中使用的分区函数的表达式。
SUBPARTITION_EXPRESSION	VARCHAR(2048)	子分区表达式，定义方式同 `PARTITION_EXPRESSION` 。
PARTITION_DESCRIPTION	TEXT	用于 RANGE 和 LIST 分区描述分区的 values 规则。 • RANGE 分区：包含分区 `VALUES LESS THAN` 子句中设置的值。该值可以是整数，也可以是 `MAXVALUE` 。 • LIST 分区：包含分区 `VALUES IN` 子句中定义的值，该值是逗号分隔的整数值列表。 • 其他分区：除 RANGE 和 LIST 以外的分区方法， `PARTITION_DESCRIPTION` 字段始终为 `NULL` 。
HIDDEN	ENUM	• Explicit：用户显式创建出的 PARTITION POLICY。 • Implicit：数据库自动创建的隐式 PARTITION POLICY。
SE_PRIVATE_DATA	MEDIUMTEXT	预留字段。

示例

```
tdsql> SELECT * FROM information_schema.partition_policy_partitions;
+--------------------+----------------------+--------------+----------
------+------------------+---------------------------+---------------
--------------+------------------+---------------------+---------------
-------+------------------------+----------------------+----------+---
------------------------+
| PARTITION_POLICY_ID | PARTITION_POLICY_NAME | PARTITION_ID |
PARTITION_NAME | SUBPARTITION_NAME | PARTITION_ORDINAL_POSITION |
SUBPARTITION_ORDINAL_POSITION | PARTITION_METHOD | SUBPARTITION_METHOD |
PARTITION_EXPRESSION | SUBPARTITION_EXPRESSION | PARTITION_DESCRIPTION |
HIDDEN    | SE_PRIVATE_DATA            |
+--------------------+----------------------+--------------+----------
------+------------------+---------------------------+---------------
--------------+------------------+---------------------+---------------
```

```
-------+------------------------+----------------------+----------+---
------------------------+
|                    1 | impl_pp_hash_4         |             1 | p0
| NULL               |                             1 |
NULL | HASH              | NULL                   | INTEGER              |
NULL                      | NULL                     | Implicit |
distribution_policy_id=0; |
|                    1 | impl_pp_hash_4         |             2 | p1
| NULL               |                             2 |
NULL | HASH              | NULL                   | INTEGER              |
NULL                      | NULL                     | Implicit |
distribution_policy_id=0; |
|                    1 | impl_pp_hash_4         |             3 | p2
| NULL               |                             3 |
NULL | HASH              | NULL                   | INTEGER              |
NULL                      | NULL                     | Implicit |
distribution_policy_id=0; |
|                    1 | impl_pp_hash_4         |             4 | p3
| NULL               |                             4 |
NULL | HASH              | NULL                   | INTEGER              |
NULL                      | NULL                     | Implicit |
distribution_policy_id=0; |
+---------------------+-----------------------+--------------+----------
------+-------------------+---------------------------+---------------
--------------+------------------+---------------------+--------------
-------+------------------------+----------------------+----------+---
------------------------+
4 rows in set (0.02 sec)
```

PARTITIONS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.PARTITIONS` 用于提供数据库中所有分区表的详细分区信息，包括分区定义、存储参数、统计信息等。

字段说明

字段名	类型	描述
TABLE_CATALOG	varchar(64)	表所属的目录名称，该值始终为 'def'
TABLE_SCHEMA	varchar(64)	表所属的数据库名称
TABLE_NAME	varchar(64)	表名称
PARTITION_NAME	varchar(64)	分区名称
SUBPARTITION_NAME	varchar(64)	如果该行表示一个子分区（subpartition），则为子分区的名称；否则为 `NULL`
PARTITION_ORDINAL_POSITION	int unsigned	分区在表中的位置序号
SUBPARTITION_ORDINAL_POSITION	int unsigned	子分区在分区中的位置序号
PARTITION_METHOD	varchar(13)	分区方法（RANGE、LIST、HASH、KEY 等）
SUBPARTITION_METHOD	varchar(13)	子分区方法
PARTITION_EXPRESSION	varchar(2048)	分区表达式
SUBPARTITION_EXPRESSION	text	子分区表达式

PARTITION_DESCRIPTION	text	分区描述（如 RANGE 分区的边界值）
TABLE_ROWS	bigint unsigned	分区中的估计行数
AVG_ROW_LENGTH	bigint unsigned	分区的平均行长度
DATA_LENGTH	bigint unsigned	分区数据长度（字节）
MAX_DATA_LENGTH	bigint unsigned	分区最大数据长度
INDEX_LENGTH	bigint unsigned	分区索引长度
DATA_FREE	bigint unsigned	分区中未使用的空间
CREATE_TIME	timestamp	分区创建时间
UPDATE_TIME	datetime	分区最后更新时间
CHECK_TIME	datetime	分区最后检查时间
CHECKSUM	bigint	分区校验和值
PARTITION_COMMENT	text	分区注释信息
NODEGROUP	varchar(256)	节点组信息
TABLESPACE_NAME	varchar(268)	表空间名称
TINDEX_ID	int unsigned	表或者分区的 TINDEX_ID
TINDEX_ID_STORAGE_FORMAT	varchar(8)	表索引 ID 存储格式
DATA_SPACE_TYPE	varchar(128)	数据空间类型
SE_PRIVATE_DATA	mediumtext	存储引擎私有数据

TABLE_SCHEMA_VERSION	int unsigned	表结构版本号
TABLE_SCHEMA_STATUS	smallint unsigned	表结构状态
TABLE_EXTRA_INFO	mediumtext	表额外信息（JSON 格式）
TABLE_SE_PRIVATE_DATA	mediumtext	表存储引擎私有数据

示例

1. 创建了一个具有两级分区的 `sales` 表：
   - 一级分区：按年份范围分区（RANGE），共2个分区（p0, p1）。
   - 二级分区：按月哈希分区（HASH），每个一级分区下各有2个子分区。
2. 查询 `PARTITIONS` 视图：显示的是实际的分区/子分区信息，对于有子分区的表，不会显示一级分区的汇总信息。如下所示，查询结果为4行数据，分别对应4个子分区。

```
#创建分区表示例
tdsql >  CREATE TABLE sales (
    id INT NOT NULL,
    sale_date DATE NOT NULL,
    amount DECIMAL(10,2)
)
PARTITION BY RANGE (YEAR(sale_date))
SUBPARTITION BY HASH(MONTH(sale_date))
SUBPARTITIONS 2 (
    PARTITION p0 VALUES LESS THAN (2023),
    PARTITION p1 VALUES LESS THAN (2025)
);

#查询分区信息
tdsql >  select * from INFORMATION_SCHEMA.partitions where
TABLE_SCHEMA="test" and TABLE_NAME="sales"\G;
*************************** 1. row ***************************
               TABLE_CATALOG: def
                TABLE_SCHEMA: test
                  TABLE_NAME: sales
              PARTITION_NAME: p0
```

```
            SUBPARTITION_NAME: p0sp0
   PARTITION_ORDINAL_POSITION: 1
SUBPARTITION_ORDINAL_POSITION: 1
             PARTITION_METHOD: RANGE
          SUBPARTITION_METHOD: HASH
         PARTITION_EXPRESSION: year(`sale_date`)
      SUBPARTITION_EXPRESSION: month(`sale_date`)
        PARTITION_DESCRIPTION: 2023
                   TABLE_ROWS: 0
               AVG_ROW_LENGTH: 0
                  DATA_LENGTH: 0
              MAX_DATA_LENGTH: 0
                 INDEX_LENGTH: 0
                    DATA_FREE: 0
                  CREATE_TIME: 2025-11-09 16:34:32
                  UPDATE_TIME: NULL
                   CHECK_TIME: NULL
                     CHECKSUM: NULL
            PARTITION_COMMENT:
                    NODEGROUP: default
              TABLESPACE_NAME: NULL
                     TINDEX_ID: 10032
     TINDEX_ID_STORAGE_FORMAT: 00002730
             DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
             SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10033;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 2. row ***************************
                TABLE_CATALOG: def
                 TABLE_SCHEMA: test
                   TABLE_NAME: sales
              PARTITION_NAME: p0
           SUBPARTITION_NAME: p0sp1
```

```
   PARTITION_ORDINAL_POSITION: 1
SUBPARTITION_ORDINAL_POSITION: 2
            PARTITION_METHOD: RANGE
         SUBPARTITION_METHOD: HASH
        PARTITION_EXPRESSION: year(`sale_date`)
     SUBPARTITION_EXPRESSION: month(`sale_date`)
       PARTITION_DESCRIPTION: 2023
                  TABLE_ROWS: 0
              AVG_ROW_LENGTH: 0
                 DATA_LENGTH: 0
             MAX_DATA_LENGTH: 0
                INDEX_LENGTH: 0
                   DATA_FREE: 0
                 CREATE_TIME: 2025-11-09 16:34:32
                 UPDATE_TIME: NULL
                  CHECK_TIME: NULL
                    CHECKSUM: NULL
           PARTITION_COMMENT:
                   NODEGROUP: default
             TABLESPACE_NAME: NULL
                   TINDEX_ID: 10034
    TINDEX_ID_STORAGE_FORMAT: 00002732
             DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
             SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10035;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 3. row ***************************
               TABLE_CATALOG: def
                TABLE_SCHEMA: test
                  TABLE_NAME: sales
              PARTITION_NAME: p1
           SUBPARTITION_NAME: p1sp0
   PARTITION_ORDINAL_POSITION: 2
```

```
SUBPARTITION_ORDINAL_POSITION: 1
            PARTITION_METHOD: RANGE
         SUBPARTITION_METHOD: HASH
        PARTITION_EXPRESSION: year(`sale_date`)
     SUBPARTITION_EXPRESSION: month(`sale_date`)
       PARTITION_DESCRIPTION: 2025
                  TABLE_ROWS: 0
              AVG_ROW_LENGTH: 0
                 DATA_LENGTH: 0
             MAX_DATA_LENGTH: 0
                INDEX_LENGTH: 0
                   DATA_FREE: 0
                 CREATE_TIME: 2025-11-09 16:34:32
                 UPDATE_TIME: NULL
                  CHECK_TIME: NULL
                    CHECKSUM: NULL
           PARTITION_COMMENT:
                   NODEGROUP: default
             TABLESPACE_NAME: NULL
                   TINDEX_ID: 10037
    TINDEX_ID_STORAGE_FORMAT: 00002735
             DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
             SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10038;schema_status=0;
        TABLE_SCHEMA_VERSION: 1
         TABLE_SCHEMA_STATUS: 0
            TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
       TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 4. row ***************************
               TABLE_CATALOG: def
                TABLE_SCHEMA: test
                  TABLE_NAME: sales
              PARTITION_NAME: p1
           SUBPARTITION_NAME: p1sp1
   PARTITION_ORDINAL_POSITION: 2
SUBPARTITION_ORDINAL_POSITION: 2
```

```
             PARTITION_METHOD: RANGE
          SUBPARTITION_METHOD: HASH
         PARTITION_EXPRESSION: year(`sale_date`)
      SUBPARTITION_EXPRESSION: month(`sale_date`)
        PARTITION_DESCRIPTION: 2025
                   TABLE_ROWS: 0
               AVG_ROW_LENGTH: 0
                  DATA_LENGTH: 0
              MAX_DATA_LENGTH: 0
                 INDEX_LENGTH: 0
                    DATA_FREE: 0
                  CREATE_TIME: 2025-11-09 16:34:32
                  UPDATE_TIME: NULL
                   CHECK_TIME: NULL
                     CHECKSUM: NULL
            PARTITION_COMMENT:
                    NODEGROUP: default
              TABLESPACE_NAME: NULL
                    TINDEX_ID: 10039
    TINDEX_ID_STORAGE_FORMAT: 00002737
              DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
              SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10040;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
4 rows in set (0.05 sec)
```

PARTITIONS_VERBOSE

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.PARTITIONS_VERBOSE` 用于展示比标准 `PARTITIONS` 表更详细的分区表信息。主要区别在于：

- `PARTITIONS` 视图：仅显示实际的分区/子分区信息，对于有子分区的表，不会显示一级分区的汇总信息。
- `PARTITIONS_VERBOSE` 视图：同时显示一级分区和二级分区的完整信息，提供更全面的分区结构视图。

字段说明

字段名	类型	描述
TABLE_CATALOG	varchar(64)	表所属的目录名称，通常为 'def'
TABLE_SCHEMA	varchar(64)	表所属的数据库名称
TABLE_NAME	varchar(64)	表名称
PARTITION_NAME	varchar(64)	分区名称
SUBPARTITION_NAME	varchar(64)	如果该行表示一个子分区（subpartition），则为子分区的名称；否则为 NULL。
PARTITION_ORDINAL_POSITION	int unsigned	分区在表中的位置序号
SUBPARTITION_ORDINAL_POSITION	int unsigned	子分区在分区中的位置序号
PARTITION_METHOD	varchar(13)	分区方法（RANGE、LIST、HASH、KEY 等）
SUBPARTITION_METHOD	varchar(13)	子分区方法
PARTITION_EXPRESSION	varchar(2048)	分区表达式

SUBPARTITION_EXPRESSION	text	子分区表达式
PARTITION_DESCRIPTION	text	分区描述（如 RANGE 分区的边界值）
TABLE_ROWS	bigint unsigned	分区中的估计行数
AVG_ROW_LENGTH	bigint unsigned	分区的平均行长度
DATA_LENGTH	bigint unsigned	分区数据长度（字节）
MAX_DATA_LENGTH	bigint unsigned	分区最大数据长度
INDEX_LENGTH	bigint unsigned	分区索引长度
DATA_FREE	bigint unsigned	分区中未使用的空间
CREATE_TIME	timestamp	分区创建时间
UPDATE_TIME	datetime	分区最后更新时间
CHECK_TIME	datetime	分区最后检查时间
CHECKSUM	bigint	分区校验和值
PARTITION_COMMENT	text	分区注释信息
NODEGROUP	varchar(256)	节点组信息
TABLESPACE_NAME	varchar(268)	表空间名称
TINDEX_ID	int unsigned	表或者分区的 TINDEX_ID
TINDEX_ID_STORAGE_FORMAT	varchar(8)	表索引 ID 存储格式
DATA_SPACE_TYPE	varchar(128)	数据空间类型

SE_PRIVATE_DATA	mediumtext	存储引擎私有数据
TABLE_SCHEMA_VERSION	int unsigned	表结构版本号
TABLE_SCHEMA_STATUS	smallint unsigned	表结构状态
TABLE_EXTRA_INFO	mediumtext	表额外信息（JSON 格式）
TABLE_SE_PRIVATE_DATA	mediumtext	表存储引擎私有数据

示例

1. 创建了一个具有两级分区的 `sales` 表：
   - 一级分区：按年份范围分区（RANGE），共2个分区（p0, p1）。
   - 二级分区：按月哈希分区（HASH），每个一级分区下各有2个子分区。
2. 查询 `PARTITIONS_VERBOSE` 视图：返回6行数据，包含：
   - 2行一级分区信息（ `SUBPARTITION_NAME` 为 `NULL` ）。
   - 4行二级分区信息。

```
#创建分区表示例
tdsql > CREATE TABLE sales (
    id INT NOT NULL,
    sale_date DATE NOT NULL,
    amount DECIMAL(10,2)
)
PARTITION BY RANGE (YEAR(sale_date))
SUBPARTITION BY HASH(MONTH(sale_date))
SUBPARTITIONS 2 (
    PARTITION p0 VALUES LESS THAN (2023),
    PARTITION p1 VALUES LESS THAN (2025)
);

#查询分区信息
tdsql > select * from INFORMATION_SCHEMA.partitions_verbose where
TABLE_SCHEMA="test" and TABLE_NAME="sales"\G;
*************************** 1. row ***************************
```

```
                TABLE_CATALOG: def
                 TABLE_SCHEMA: test
                   TABLE_NAME: sales
              PARTITION_NAME: p0
           SUBPARTITION_NAME: NULL
   PARTITION_ORDINAL_POSITION: 1
SUBPARTITION_ORDINAL_POSITION: NULL
            PARTITION_METHOD: RANGE
         SUBPARTITION_METHOD: HASH
        PARTITION_EXPRESSION: year(`sale_date`)
      SUBPARTITION_EXPRESSION: NULL
        PARTITION_DESCRIPTION: 2023
                  TABLE_ROWS: NULL
              AVG_ROW_LENGTH: NULL
                 DATA_LENGTH: NULL
             MAX_DATA_LENGTH: NULL
                INDEX_LENGTH: NULL
                   DATA_FREE: NULL
                 CREATE_TIME: 2025-11-09 16:34:32
                 UPDATE_TIME: NULL
                  CHECK_TIME: NULL
                    CHECKSUM: NULL
           PARTITION_COMMENT:
                   NODEGROUP: default
             TABLESPACE_NAME: NULL
                   TINDEX_ID: 10031
     TINDEX_ID_STORAGE_FORMAT: 0000272F
              DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
              SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=0;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 2. row ***************************
                TABLE_CATALOG: def
```

```
                 TABLE_SCHEMA: test
                   TABLE_NAME: sales
               PARTITION_NAME: p0
            SUBPARTITION_NAME: p0sp0
   PARTITION_ORDINAL_POSITION: 1
SUBPARTITION_ORDINAL_POSITION: 1
            PARTITION_METHOD: RANGE
         SUBPARTITION_METHOD: HASH
        PARTITION_EXPRESSION: year(`sale_date`)
     SUBPARTITION_EXPRESSION: month(`sale_date`)
       PARTITION_DESCRIPTION: 2023
                  TABLE_ROWS: 0
              AVG_ROW_LENGTH: 0
                 DATA_LENGTH: 0
             MAX_DATA_LENGTH: 0
                INDEX_LENGTH: 0
                   DATA_FREE: 0
                 CREATE_TIME: 2025-11-09 16:34:32
                 UPDATE_TIME: NULL
                  CHECK_TIME: NULL
                    CHECKSUM: NULL
           PARTITION_COMMENT:
                   NODEGROUP: default
             TABLESPACE_NAME: NULL
                   TINDEX_ID: 10032
    TINDEX_ID_STORAGE_FORMAT: 00002730
             DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
             SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10033;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 3. row ***************************
                TABLE_CATALOG: def
                 TABLE_SCHEMA: test
```

```
                  TABLE_NAME: sales
               PARTITION_NAME: p0
            SUBPARTITION_NAME: p0sp1
   PARTITION_ORDINAL_POSITION: 1
SUBPARTITION_ORDINAL_POSITION: 2
            PARTITION_METHOD: RANGE
         SUBPARTITION_METHOD: HASH
         PARTITION_EXPRESSION: year(`sale_date`)
      SUBPARTITION_EXPRESSION: month(`sale_date`)
        PARTITION_DESCRIPTION: 2023
                  TABLE_ROWS: 0
              AVG_ROW_LENGTH: 0
                 DATA_LENGTH: 0
             MAX_DATA_LENGTH: 0
                INDEX_LENGTH: 0
                   DATA_FREE: 0
                 CREATE_TIME: 2025-11-09 16:34:32
                 UPDATE_TIME: NULL
                  CHECK_TIME: NULL
                    CHECKSUM: NULL
           PARTITION_COMMENT:
                   NODEGROUP: default
             TABLESPACE_NAME: NULL
                   TINDEX_ID: 10034
     TINDEX_ID_STORAGE_FORMAT: 00002732
              DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
              SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10035;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 4. row ***************************
                TABLE_CATALOG: def
                 TABLE_SCHEMA: test
                   TABLE_NAME: sales
```

```
               PARTITION_NAME: p1
            SUBPARTITION_NAME: NULL
   PARTITION_ORDINAL_POSITION: 2
SUBPARTITION_ORDINAL_POSITION: NULL
            PARTITION_METHOD: RANGE
         SUBPARTITION_METHOD: HASH
         PARTITION_EXPRESSION: year(`sale_date`)
      SUBPARTITION_EXPRESSION: NULL
        PARTITION_DESCRIPTION: 2025
                   TABLE_ROWS: NULL
               AVG_ROW_LENGTH: NULL
                  DATA_LENGTH: NULL
              MAX_DATA_LENGTH: NULL
                 INDEX_LENGTH: NULL
                    DATA_FREE: NULL
                  CREATE_TIME: 2025-11-09 16:34:32
                  UPDATE_TIME: NULL
                   CHECK_TIME: NULL
                     CHECKSUM: NULL
            PARTITION_COMMENT:
                    NODEGROUP: default
              TABLESPACE_NAME: NULL
                     TINDEX_ID: 10036
     TINDEX_ID_STORAGE_FORMAT: 00002734
              DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
              SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=0;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 5. row ***************************
               TABLE_CATALOG: def
                TABLE_SCHEMA: test
                  TABLE_NAME: sales
               PARTITION_NAME: p1
```

```
            SUBPARTITION_NAME: p1sp0
   PARTITION_ORDINAL_POSITION: 2
SUBPARTITION_ORDINAL_POSITION: 1
             PARTITION_METHOD: RANGE
          SUBPARTITION_METHOD: HASH
         PARTITION_EXPRESSION: year(`sale_date`)
      SUBPARTITION_EXPRESSION: month(`sale_date`)
        PARTITION_DESCRIPTION: 2025
                   TABLE_ROWS: 0
               AVG_ROW_LENGTH: 0
                  DATA_LENGTH: 0
              MAX_DATA_LENGTH: 0
                 INDEX_LENGTH: 0
                    DATA_FREE: 0
                  CREATE_TIME: 2025-11-09 16:34:32
                  UPDATE_TIME: NULL
                   CHECK_TIME: NULL
                     CHECKSUM: NULL
            PARTITION_COMMENT:
                    NODEGROUP: default
              TABLESPACE_NAME: NULL
                     TINDEX_ID: 10037
     TINDEX_ID_STORAGE_FORMAT: 00002735
             DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
             SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10038;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
*************************** 6. row ***************************
                TABLE_CATALOG: def
                 TABLE_SCHEMA: test
                   TABLE_NAME: sales
               PARTITION_NAME: p1
            SUBPARTITION_NAME: p1sp1
```

```
   PARTITION_ORDINAL_POSITION: 2
SUBPARTITION_ORDINAL_POSITION: 2
            PARTITION_METHOD: RANGE
         SUBPARTITION_METHOD: HASH
        PARTITION_EXPRESSION: year(`sale_date`)
     SUBPARTITION_EXPRESSION: month(`sale_date`)
       PARTITION_DESCRIPTION: 2025
                  TABLE_ROWS: 0
              AVG_ROW_LENGTH: 0
                 DATA_LENGTH: 0
             MAX_DATA_LENGTH: 0
                INDEX_LENGTH: 0
                   DATA_FREE: 0
                 CREATE_TIME: 2025-11-09 16:34:32
                 UPDATE_TIME: NULL
                  CHECK_TIME: NULL
                    CHECKSUM: NULL
           PARTITION_COMMENT:
                   NODEGROUP: default
             TABLESPACE_NAME: NULL
                    TINDEX_ID: 10039
    TINDEX_ID_STORAGE_FORMAT: 00002737
             DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
             SE_PRIVATE_DATA:
create_data_obj_task_id=29572817330634866;distribution_policy_id=1;hid
den_pk_autoinc_tindex_id=10040;schema_status=0;
         TABLE_SCHEMA_VERSION: 1
          TABLE_SCHEMA_STATUS: 0
             TABLE_EXTRA_INFO: {"version":3,"create_ts":0}
        TABLE_SE_PRIVATE_DATA:
autoinc_version=1;create_data_obj_task_id=29572817330634866;distributi
on_policy_id=1;origin_db=test;origin_table=sales;partition_policy_id=0
;sync_table=0;
6 rows in set (0.05 sec)
```

PERSIST

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.PERSIST` 用于展示持久化到 MC 侧的 variables。

字段说明

字段名	类型	描述
VARIABLE_NAME	varchar(64)	变量名字。
VARIABLE_VALUE	varchar(1024)	变量值。

示例

可以通过 `SELECT` 或者 `SHOW` 查询持久化到 MC 侧的变量。

```
tdsql>SELECT * FROM INFORMATION_SCHEMA.PERSIST;
+-------------------------+----------------+
| VARIABLE_NAME           | VARIABLE_VALUE |
+-------------------------+----------------+
| tdstore_deadlock_detect | ON             |
| tdsql_lock_wait_timeout | 50             |
+-------------------------+----------------+
2 rows in set (0.01 sec)

tdsql>SHOW PERSIST VARIABLES;
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| tdstore_deadlock_detect | ON    |
| tdsql_lock_wait_timeout | 50    |
+-------------------------+-------+
2 rows in set (0.00 sec)
```

RANGE_CACHE

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.RANGE_CACHE` 用于展示 Range Cache 中存储的 Block Range Stats，便于进行问题排查。

字段说明

字段名	类型	描述
TABLE_SCHEMA	varchar(192)	数据库名称。
TABLE_NAME	varchar(192)	表名称。
INDEX_NAME	varchar(192)	索引名称。
RANGE_CACHE_MISS	bigint	未命中 Range Cache 的次数。
RANGE_CACHE_SAMPLE_ROWS	bigint	Block Range 的采样行数。
RANGE_CACHE_REFILL_TS	bigint	Block Range refill 的时间戳。
START_KEY	varchar(192)	Block Range 的起始键（Start Key）。
END_KEY	varchar(192)	Block Range 的结束键（End Key）。
RANGE_ROWS	bigint	Block Range 内的估计行数。
RANGE_HINTS	bigint	Block Range 的命中次数。
RANGE_TIMESTAMP	bigint	Block Range Stat 生成的时间戳。

示例

```
tdsql > select * from information_schema.range_cache limit 1 \G;
*************************** 1. row ***************************
           TABLE_SCHEMA: sbtest
             TABLE_NAME: sbtest1
             INDEX_NAME: PRIMARY
       RANGE_CACHE_MISS: 9486
RANGE_CACHE_SAMPLE_ROWS: 323
  RANGE_CACHE_REFILL_TS: 20251024023139
              START_KEY: 000027C280000001
                END_KEY: 000027C280000143
             RANGE_ROWS: 162
            RANGE_HINTS: 1
        RANGE_TIMESTAMP: 20251024023144
1 row in set (11.64 sec)
```

TDSTORE_ACTIVE_COMPACTION_STATS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_ACTIVE_COMPACTION_STATS` 用于展示当前 TDStore 中正在进行的 Compaction 任务。

字段说明

字段名	类型	描述
THREAD_ID	BIGINT	线程 ID。
CF_NAME	VARCHAR(513)	ColumnFamily 名称。
INPUT_LEVEL	INT	Compaction 输入文件的层级。
INPUT_FILES	VARCHAR(513)	Compaction 输入文件的名称。
COMPACTION_REASON	VARCHAR(513)	Compaction 触发的原因。
NODE_NAME	VARCHAR(513)	SQLEngine 节点名称。

示例

```
tdsql>select * from tdstore_active_compaction_stats\G
*************************** 1. row ***************************
        THREAD_ID: 140352025515776
          CF_NAME: user_region
      INPUT_LEVEL: 2
      INPUT_FILES:
022050.sst,022052.sst,022055.sst,022058.sst,022060.sst,022063.sst,022066
.sst,022068.sst,022071.sst,022074.sst,022076.sst,022079.sst,022080.sst,0
22081.sst,022082.sst,022083.sst,022085.sst,022088.sst,022090.sst,022092.
sst,022095.sst,022096.sst,022098.sst,016751.sst,018033.sst,018035.sst,01
```

```
8039.sst,018042.sst,018043.sst,018032.sst,018038.sst,020981.sst,020987.s
st,020994.sst,020999.sst,021006.sst,021012.sst,021019.sst,021026.sst,021
033.sst,021043.sst,018302.sst,018308.sst,018312.sst,018318.sst,018325.ss
t,018333.
COMPACTION_REASON: ManualCompaction
        NODE_NAME: node-Performance-1508-9_109_173_22-001
1 row in set (0.00 sec)
```

TDSTORE_BULK_LOAD_TXN_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_BULK_LOAD_TXN_INFO` 用于查询历史 Bulk Load 事务以及正在执行的 Bulk Load 事务信息。

字段说明

字段名	类型	描述
node_name	varchar(64)	Node 名称。
node_addr	varchar(64)	Node 地址。
trans_id	bigint	事务 ID。
state	varchar(64)	Bulk Load 事务目前的状态。
start_time	varchar(64)	Bulk Load 开始的时间。
end_time	varchar(64)	Bulk Load 结束的时间。
total_file_num	bigint	Bulk Load 事务 SST 文件的数量。
total_file_size	bigint	Bulk Load 事务 SST 文件大小，单位Bytes。
total_cost_ms	bigint	Bulk Load 事务总耗时，单位ms。
detail_cost_info	longtext	Bulk Load 每一个阶段具体的耗时，单位us。
rep_group_ids	longtext	Bulk Load 事务参与的 RG ID。
err_msg	longtext	Bulk Load 失败场景下的报错。

示例

```
tdsql>  select * from INFORMATION_SCHEMA.TDSTORE_BULK_LOAD_TXN_INFO;
*************************** 1. row ***************************
```

```
        node_name: node-******-1-21-001
        node_addr: 10.10.10.10:9806
         trans_id: 29350720562004012
            state: SUCCESS
       start_time: 2025-06-09 11:21:14 231
         end_time: 2025-06-09 11:21:14 541
   total_file_num: 1
  total_file_size: 386413
    total_cost_ms: 310
detail_cost_info:  [BULK_LOAD_TRANS: <PUT_RECORDS: 12333us>,
<EXECUTOR_HANDLE: 76us>, <FINISH_WRITE_AND_SORT_DATA: 5591us>,
<SET_SCHEDULE_REP_GROUP_JOB: 0us>, <GET_REP_GROUP_META_FROM_MC: 1062us>,
<REORG_SST_FILE_BY_REP_GROUP: 210us>, <SEND_SST_FILE_TO_DST_NODE:
1329us>, <PREPARE_PARTS: 56us>, <COMMIT_SYNC_LOG: 28180us>,
<WAIT_CLIENT_BEFORE_COMMITTING: 261958us>, <WAIT_WHEN_COMMITTING:
117us>, <CLEAN_PENDING_COMMIT_DATA_DIR: 0us>]
    rep_group_ids: 257
          err_msg:
*************************** 2. row ***************************
        node_name: node-******-1-21-001
        node_addr: 10.10.10.10:9806
         trans_id: 29350720562004019
            state: SUCCESS
       start_time: 2025-06-09 11:21:14 231
         end_time: 2025-06-09 11:21:14 575
   total_file_num: 1
  total_file_size: 386233
    total_cost_ms: 344
detail_cost_info:  [BULK_LOAD_TRANS: <PUT_RECORDS: 14501us>,
<EXECUTOR_HANDLE: 57us>, <FINISH_WRITE_AND_SORT_DATA: 5831us>,
<SET_SCHEDULE_REP_GROUP_JOB: 0us>, <GET_REP_GROUP_META_FROM_MC: 1482us>,
<REORG_SST_FILE_BY_REP_GROUP: 176us>, <SEND_SST_FILE_TO_DST_NODE:
196us>, <PREPARE_PARTS: 44us>, <COMMIT_SYNC_LOG: 51951us>,
<WAIT_CLIENT_BEFORE_COMMITTING: 269700us>, <WAIT_WHEN_COMMITTING:
150us>, <CLEAN_PENDING_COMMIT_DATA_DIR: 0us>]
    rep_group_ids: 257
          err_msg:
*************************** 3. row ***************************
        node_name: node-******-1-21-001
        node_addr: 10.10.10.10:9806
```

```
        trans_id: 29350720562004034
           state: SUCCESS
      start_time: 2025-06-09 11:21:14 231
        end_time: 2025-06-09 11:21:14 597
  total_file_num: 1
 total_file_size: 386649
   total_cost_ms: 366
detail_cost_info:  [BULK_LOAD_TRANS: <PUT_RECORDS: 17218us>,
<EXECUTOR_HANDLE: 77us>, <FINISH_WRITE_AND_SORT_DATA: 8347us>,
<SET_SCHEDULE_REP_GROUP_JOB: 0us>, <GET_REP_GROUP_META_FROM_MC: 982us>,
<REORG_SST_FILE_BY_REP_GROUP: 1139us>, <SEND_SST_FILE_TO_DST_NODE:
200us>, <PREPARE_PARTS: 55us>, <COMMIT_SYNC_LOG: 66420us>,
<WAIT_CLIENT_BEFORE_COMMITTING: 271738us>, <WAIT_WHEN_COMMITTING:
102us>, <CLEAN_PENDING_COMMIT_DATA_DIR: 0us>]
   rep_group_ids: 257
         err_msg:
3 rows in set (0.01 sec)
```

TDSTORE_CF_OPTIONS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_CF_OPTIONS` 用于展示 SQLEngine 中 ColumnFamily 相关的配置项。

字段说明

字段名	类型	描述
DB_TYPE	VARCHAR(193)	RocksDB 类型，支持 `data_db` 、 `multi_raft_db` 及 `monitor_info_db` 。
CF_NAME	VARCHAR(193)	ColumnFamily 的名称。
OPTION_TYPE	VARCHAR(193)	配置项名称。
VALUE	VARCHAR(193)	配置项的值。
NODE_NAME	VARCHAR(513)	SQLEngine 节点的名称。

示例

```
tdsql>select * from tdstore_cf_options\G
*************************** 1. row ***************************
    DB_TYPE: data_db
    CF_NAME: default
OPTION_TYPE: COMPARATOR
      VALUE: leveldb.BytewiseComparator
  NODE_NAME: node-1-001
*************************** 2. row ***************************
    DB_TYPE: data_db
    CF_NAME: default
OPTION_TYPE: COMPACTION_FILTER
      VALUE: NULL
  NODE_NAME: node-1-001
*************************** 3. row ***************************
```

```
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: COMPACTION_FILTER_FACTORY
       VALUE: NULL
   NODE_NAME: node-1-001
*************************** 4. row ***************************
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: WRITE_BUFFER_SIZE
       VALUE: 67108864
   NODE_NAME: node-1-001
*************************** 5. row ***************************
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: MAX_WRITE_BUFFER_NUMBER
       VALUE: 2
   NODE_NAME: node-1-001
*************************** 6. row ***************************
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: MIN_WRITE_BUFFER_NUMBER_TO_MERGE
       VALUE: 1
   NODE_NAME: node-1-001
*************************** 7. row ***************************
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: NUM_LEVELS
       VALUE: 7
   NODE_NAME: node-1-001
*************************** 8. row ***************************
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: LEVEL0_FILE_NUM_COMPACTION_TRIGGER
       VALUE: 4
   NODE_NAME: node-1-001
*************************** 9. row ***************************
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: LEVEL0_SLOWDOWN_WRITES_TRIGGER
       VALUE: 20
```

```
   NODE_NAME: node-1-001
*************************** 10. row ***************************
     DB_TYPE: data_db
     CF_NAME: default
OPTION_TYPE: LEVEL0_STOP_WRITES_TRIGGER
       VALUE: 36
   NODE_NAME: node-1-001
```

TDSTORE_COLUMNAR_COMPACTION_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_COMPACTION_INFO` 用于记录列存 compaction 和 drop table 任务的执行过程与结果。

字段说明

字段名	类型	描述
node_name	VARCHAR	执行列存 compaction 节点的节点名。
job_description	VARCHAR	列存 compaction 任务描述： • 普通 compaction • drop table
event_type	VARCHAR	列存 compaction 任务类型（TD_EVENT_COLUMNAR_COMPACT）。
table_name	VARCHAR	执行列存 compaction 的表的表名。
tx_id	INT	执行列存 compaction 的列存事务的事务 ID。
tx_seq_num	INT	执行列存 compaction 的列存事务提交后获取到的列存元数据库的最老 snapshot 对应的 sequence number。
ret	VARCHAR	列存 compaction 任务的返回码。
compacted_files	VARCHAR	列存 compaction 任务回收的文件列表。
new_files	VARCHAR	列存 compaction 任务生成的文件列表。
begin_time	VARCHAR	列存 compaction 任务的开始时间。
cost_us	INT	列存 compaction 任务的总耗时。
detailed_info	VARCHAR	列存 compaction 任务每一步的耗时。

示例

```
tdsql3_sys_local@localhost [test_base]> SELECT * FROM
information_schema.TDSTORE_COLUMNAR_COMPACTION_INFO\G
*************************** 1. row ***************************
      node_name: node-run-test_reboot-001
job_description: columnar compaction
     event_type: TD_EVENT_COLUMNAR_COMPACT
     table_name: test_table
          tx_id: 12
     tx_seq_num: 13
            ret: 0(EC_OK)
compacted_files: 10026_2, 10026_1
      new_files: 3
     begin_time: 2025-04-22 18:51:49 215
        cost_us: 4910
  detailed_info:  [COLUMNAR_FILE_DEFRAGMENTATION: <BEGIN: 4695us>,
<REWRITE_FILES: 85us>, <META_MODIFICATION: 123us>, <REMOVE_PKS: 0us>,
<COMMIT: 6us>]
*************************** 2. row ***************************
      node_name: node-run-test_reboot-001
job_description: columnar compaction
     event_type: TD_EVENT_COLUMNAR_COMPACT
     table_name: test_table
          tx_id: 22
     tx_seq_num: 28
            ret: 0(EC_OK)
compacted_files: 10026_3, 10026_5, 10026_4
      new_files: 6
     begin_time: 2025-04-22 18:52:19 235
        cost_us: 6800
  detailed_info:  [COLUMNAR_FILE_DEFRAGMENTATION: <BEGIN: 6563us>,
<REWRITE_FILES: 86us>, <META_MODIFICATION: 141us>, <REMOVE_PKS: 0us>,
<COMMIT: 7us>]
2 rows in set (0.01 sec)
```

TDSTORE_COLUMNAR_FILE_DELETION_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_FILE_DELETION_INFO` 用于记录列存文件删除的过程与结果。

字段说明

字段名	类型	描述
node_name	VARCHAR	执行列存文件删除的节点的节点名。
job_description	VARCHAR	列存文件删除的任务描述。
event_type	VARCHAR	列存文件删除的任务类型。
columnar_files_name	VARCHAR	列存文件删除任务本次删除的文件的列表。
begin_time	VARCHAR	列存文件删除任务的开始时间。
cost_us	INT	列存文件删除任务的总耗时。
detailed_info	VARCHAR	列存文件删除任务各阶段的耗时。

示例

```
tdsql3_sys_local@localhost [test_base]> SELECT * FROM
INFORMATION_SCHEMA.TDSTORE_COLUMNAR_FILE_DELETION_INFO\G
*************************** 1. row ***************************
          node_name: node-run-test_reboot-001
    job_description: delete files
         event_type: TD_EVENT_COLUMNAR_FILE_DELETE
columnar_files_name: 10026_2, 10026_1
         begin_time: 2025-04-22 18:51:49 859
            cost_us: 365
      detailed_info:  [COLUMNAR_FILE_DELETION: <BEGIN: 364us>,
<DELETE_FILES: 0us>]
*************************** 2. row ***************************
```

```
          node_name: node-run-test_reboot-001
    job_description: delete files
         event_type: TD_EVENT_COLUMNAR_FILE_DELETE
columnar_files_name: 10026_3, 10026_5, 10026_4
         begin_time: 2025-04-22 18:52:19 859
            cost_us: 336
      detailed_info:  [COLUMNAR_FILE_DELETION: <BEGIN: 335us>,
<DELETE_FILES: 0us>]
2 rows in set (0.01 sec)
```

TDSTORE_COLUMNAR_FILES_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_FILES_INFO` 用于展示列存节点中列存文件的相关信息。

字段说明

字段名	类型	描述
TABLE_ID	INT	文件对应的表的 TIndexID。
DB	VARCHAR	文件所对应的 DATABASE 名称。
TABLE_NAME	VARCHAR	文件所对应的 TABLE 名称。
FILE_PATH	VARCHAR	文件的绝对路径。
FILE_ID	INT	文件的 ID。
FILE_SIZE_IN_MB	DOUBLE	文件的大小，单位是 MB。
LAST_WRITE_TIME	VARCHAR	文件的最后修改时间。
RECORD_COUNT	INT	文件的总数据行数。
INVALID_RECORD_COUNT	INT	文件无效的数据行数。
INVALID_RATIO	DOUBLE	文件中无效数据的占比，即 `INVALID_RECORD_COUNT/RECORD_COUNT`。

示例

```
SELECT * FROM TDSTORE_COLUMNAR_FILES_INFO;
+----------+----------+------------+-------------------------------------
-------------------------------------------------------------------------
+---------+-------------------+---------------------+--------------+----
-----------------+---------------+
```

```
| TABLE_ID | DB        | TABLE_NAME | FILE_PATH
| FILE_ID | FILE_SIZE_IN_MB    | LAST_WRITE_TIME      | RECORD_COUNT |
INVALID_RECORD_COUNT | INVALID_RATIO |
+----------+----------+------------+-----------------------------------
-----------------------------------------------------------------------
+---------+------------------+--------------------+--------------+----
------------------+---------------+
|    10024 | tpch100g | part        | /data/tdsql/auto-install-clang-
debug-
build/10351/data/10351/tdstore/columnar_data/10024_00000010.parquet |
10 | 25.18278980255127 | 2025-04-16 16:19:25 |       500387 |
0 |             0 |
|    10024 | tpch100g | part        | /data/tdsql/auto-install-clang-
debug-
build/10351/data/10351/tdstore/columnar_data/10024_00000012.parquet |
12 | 25.41611957550049 | 2025-04-16 16:19:32 |       502697 |
0 |             0 |
+----------+----------+------------+-----------------------------------
-----------------------------------------------------------------------
+---------+------------------+--------------------+--------------+----
------------------+---------------+
```

TDSTORE_COLUMNAR_FILE_VERSION_LIST

最近更新时间：2025-11-18 10:10:23

功能

列存文件每一行的版本信息。

字段说明

字段名	类型	描述
TABLE_ID	INT	表对应的 ID。
FILE_ID	INT	文件的 ID。
ROW_ID	INT	记录在文件内的行 ID。
INSERT_TS	BIGINT	记录的写入时间，是一个逻辑时间戳。
DELETE_TS	BIGINT	记录的删除时间，是一个逻辑时间戳。
IS_VALID	INT	在当前时间下，这一行记录是否还有效。

示例

```
tdsql > select * from TDSTORE_COLUMNAR_FILE_VERSION_LIST where
table_id=27452 and file_id=35399;
+----------+---------+--------+------------------+--------------------
+----------+
| TABLE_ID | FILE_ID | ROW_ID | INSERT_TS        | DELETE_TS
| IS_VALID |
+----------+---------+--------+------------------+--------------------
+----------+
|    27452 |   35399 |      0 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |   35399 |      1 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |   35399 |      2 | 29483145577365957 | 9223372036854775807
|        1 |
```

```
|    27452 |    35399 |       3 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |    35399 |       4 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |    35399 |       5 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |    35399 |       6 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |    35399 |       7 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |    35399 |       8 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |    35399 |       9 | 29483145577365957 | 9223372036854775807
|        1 |
|    27452 |    35399 |      10 | 29483145577365957 | 9223372036854775807
|        1 |
+----------+----------+---------+-------------------+---------------------
+----------+
```

TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO` 用于统计一张表的行存数据量和列存数据量以及二者比例。

字段说明

字段名	类型	描述
node_name	VARCHAR	被统计表所在的节点。
db_name	VARCHAR	被统计表所在的数据库名。
table_name	VARCHAR	被统计表的表名。
table_id	INT	被统计表的表 id。
column_data_count	INT	被统计表的列存数据行数。
row_data_count	INT	被统计表的行存数据行数。
data_ratio	DOUBLE	被统计表的列存数据行数与行存数据行数的比例（列存数据行数/行存数据行数）。

示例

```
tdsql3_sys_local@localhost [information_schema]> select * from
TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO\G
*************************** 1. row ***************************
        node_name: node-run-test_reboot-001
          db_name: test_base
       table_name: test_table
         table_id: 10026
column_data_count: 4
   row_data_count: 2702
       data_ratio: 0.0014803849000740192
```

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

TDSTORE_COLUMNAR_REP_GROUP_REPLAY_LATENCY

最近更新时间：2025-11-18 10:10:23

功能

列存各个 RG 的回放延迟。

字段说明

字段名	类型	描述
REP_GROUP_ID	UINT64	RG 的 ID。
REPLAY_LATENCY_IN_SEC	UINT64	RG 的回放延迟，单位是秒。
SAFE_READ_TIME	VARCHAR(173)	目前 RG 回放到的数据时间点。
SAFE_READ_TIMESTAMP	UINT64	目前 RG 回放到的数据时间点对应的毫秒级时间戳。

示例

```
tdsql > select * from
information_schema.tdstore_columnar_rep_group_replay_latency;
+--------------+----------------------+--------------------+----------
-----------+
| REP_GROUP_ID | REPLAY_LATENCY_IN_SEC | SAFE_READ_TIME      |
SAFE_READ_TIMESTAMP |
+--------------+----------------------+--------------------+----------
-----------+
|          513 |                    1 | 2025-09-11 18:35:20 |
1757586920 |
+--------------+----------------------+--------------------+----------
-----------+
```

TDSTORE_COMMON_EVENT_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_COMMON_EVENT_INFO` 用于查询 TDStore 层一些泛用任务的执行结果，包括：

- `transfer leader`
- `add/remove replica`
- `promote leader`
- `promote_demote leader`

字段说明

TDSTORE_COMMON_EVENT_INFO 字段说明

字段名	类型	描述
node_name	varchar(64)	节点名称。
job_description	text	任务描述。
job_id	bigint unsigned	任务 ID
rep_group_id	bigint unsigned	RG ID。
common_column1	text	泛用列，对于不同任务有不同含义，详见表 common_column 列字段说明。
common_column2	text	泛用列，对于不同任务有不同含义，详见表 common_column 列字段说明。
common_column3	text	泛用列，对于不同任务有不同含义，详见表 common_column 列字段说明。
event_type	varchar(64)	任务类型。
begin_time	varchar(64)	任务开始时间。

cost_us	bigint unsigned	任务耗时，单位为微秒。
detailed_info	text	任务及子任务每一步的耗时。

common_column 列字段说明

job	common_column1	common_column2	common_column3
CleanupDataTasks	任务类型	/	/
AddOneReplica	副本节点信息	/	/
TransferLeader	原 Leader 地址	新 Leader 地址	是否为强制
TransferLeaderByLeader	原 Leader 地址	新 Leader 地址	是否为强制
SaveRaftSnapshotForSplit	Region 分裂时的日志位置	/	/
RemoveReplica	被删除副本节点名称	/	/
TryPromoteToLeader	被 promote 的节点名称	/	/
PromoteDemote	被 promote 的所有节点信息	被 demote 的所有节点信息	/

示例

```
tdsql>select * from TDSTORE_COMMON_EVENT_INFO limit 1\G
*************************** 1. row ***************************
      node_name: node-Module13-180-5-004
job_description: transfer leader
         job_id: 20010
   rep_group_id: 70695
 common_column1: node-Module13-180-5-002
 common_column2: TDMember[node_name:node-Module13-180-5-004 uuid:0 ]
 common_column3: 0
     event_type: TD_EVENT_TRANSFER_LEADER
     begin_time: 2024-11-04 20:20:56 330
        cost_us: 993
```

```
  detailed_info:  [TRANSFER_LEADER: <BEGIN: 4us>,
<SEND_TRANSFER_LEADERSHIP_RPC: 989us>, <WAIT_TRANS: 0us>,
<CHECK_REPLICA_EXIST: 0us>, <EXECUTE: 0us>, <REVOKE_LEASE: 0us>,
<SWITCH_LEADER: 0us>, <END_REGION_JOB: 0us>]
```

TDSTORE_COMPACTION_HISTORY

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_COMPACTION_HISTORY` 用于展示 TDStore 的历史 Compaction 任务。

字段说明

字段名	类型	描述
THREAD_ID	VARCHAR(513)	执行 Compaction 任务的线程 ID。
CF_NAME	VARCHAR(513)	Compaction 任务所属的 ColumnFamily 名称。
INPUT_LEVEL	INT	Compaction 输入文件的层级。
OUTPUT_LEVEL	INT	Compaction 输出文件的层级。
INPUT_FILES	VARCHAR(513)	Compaction 输入文件的名称。
OUTPUT_FILES	VARCHAR(513)	Compaction 输出文件的名称。
COMPACTION_REASON	VARCHAR(513)	Compaction 触发的原因。
START_TIMESTAMP	BIGINT	Compaction 开始的时间。
END_TIMESTAMP	BIGINT	Compaction 结束的时间。
INPUT_COMPACTION_BYTES	BIGINT	Compaction 总共输入的数据量。
OUTPUT_COMPACTION_BYTES	BIGINT	Compaction 总共输出的数据量。
INPUT_FILE_COUNT	INT	Compaction 输入的文件数。
OUTPUT_FILE_COUNT	INT	Compaction 输出的文件数。

CPU_MICROS	BIGINT	Compaction 所消耗的 CPU 时间，单位为 `ms` 。
NODE_NAME	VARCHAR(513)	SQLEngine 节点的名称。

示例

```
tdsql>select * from TDSTORE_COMPACTION_HISTORY\G
*************************** 1. row ***************************
              THREAD_ID: 7f77f1fff700
                CF_NAME: user_region
            INPUT_LEVEL: 1
           OUTPUT_LEVEL: 2
            INPUT_FILES: 000212.sst
           OUTPUT_FILES: 000232.sst
      COMPACTION_REASON: LevelMaxLevelSize
        START_TIMESTAMP: 1722946987
          END_TIMESTAMP: 1722946988
 INPUT_COMPACTION_BYTES: 33818511
OUTPUT_COMPACTION_BYTES: 16227005
       INPUT_FILE_COUNT: 1
      OUTPUT_FILE_COUNT: 1
             CPU_MICROS: 420796
              NODE_NAME: node-Performance-1505-9_109_173_22-001
*************************** 2. row ***************************
              THREAD_ID: 7f77f1fff700
                CF_NAME: user_region
            INPUT_LEVEL: 1
           OUTPUT_LEVEL: 2
            INPUT_FILES: 000071.sst
           OUTPUT_FILES: 000234.sst
      COMPACTION_REASON: LevelMaxLevelSize
        START_TIMESTAMP: 1722946988
          END_TIMESTAMP: 1722946988
 INPUT_COMPACTION_BYTES: 33818511
OUTPUT_COMPACTION_BYTES: 16226568
       INPUT_FILE_COUNT: 1
      OUTPUT_FILE_COUNT: 1
             CPU_MICROS: 426266
              NODE_NAME: node-Performance-1505-9_109_173_22-001
```

```
*************************** 3. row ***************************
              THREAD_ID: 7f77f1fff700
                CF_NAME: user_region
            INPUT_LEVEL: 1
           OUTPUT_LEVEL: 2
            INPUT_FILES: 000220.sst
           OUTPUT_FILES: 000235.sst
      COMPACTION_REASON: LevelMaxLevelSize
        START_TIMESTAMP: 1722946988
          END_TIMESTAMP: 1722946989
 INPUT_COMPACTION_BYTES: 33818511
OUTPUT_COMPACTION_BYTES: 16226904
       INPUT_FILE_COUNT: 1
      OUTPUT_FILE_COUNT: 1
             CPU_MICROS: 415835
              NODE_NAME: node-Performance-1505-9_109_173_22-001
```

TDSTORE_INSTALL_SNAPSHOT_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_INSTALL_SNAPSHOT_INFO` 用于查询 TDStore 层 Install Snapshot 任务的执行情况，包括当前正在执行的任务以及执行完毕的历史任务。

字段说明

字段名	类型	描述
isLeader	varchar(64)	该节点是否为 Leader。
source_node	varchar(256)	Install Snapshot 任务的 Leader Node 地址。
dest_node	varchar(256)	Install Snapshot 任务的 Follower Node 地址。
term	bigint unsigned	Raft 任期。

type	varchar(64)	• 如果是 Leader，记录 Install Snapshot 任务触发原因:
		  ○ `Replica`：由 MC 下发任务。
		  ○ `Log Delay`：Follower 节点日志落后。
		• 如果是 Follower，该字段为空。

replication_group_id	bigint unsigned	RG ID。
start_time	varchar(64)	Install Snapshot 任务的开始时间。
end_time	varchar(64)	Install Snapshot 任务的结束时间。
err_msg	varchar(256)	如果任务失败，记录报错信息。
transfered_file_num	bigint	Follower 已拉取的 Snapshot 文件数。(如果是 Leader，该字段为空)
transfered_file_size	bigint	Follower 已拉取的 Snapshot 文件总大小。(如果是 Leader，该字段为空)

total_file_num	bigint	Leader 生成的 Snapshot 文件数。（如果是 Follower，该字段为空）
total_file_size	bigint	Leader 生成的 Snapshot 文件总大小。（如果是 Follower，该字段为空）
stage	varchar(64)	Install Snapshot 任务的状态: • `Doing`：Install Snapshot 任务正在执行。 • `Done`： Install Snapshot 任务已完成。
cost_info	varchar(64)	Install Snapshot 任务耗时: • 如果是 Leader，记录生成 Snapshot 文件的阶段耗时。 • 如果是 Follower，记录从 Leader 拷贝 Snapshot 文件的耗时。

示例

```
tdsql> SELECT * FROM TDSTORE_INSTALL_SNAPSHOT_INFO limit 2\G;
*************************** 1. row ***************************
           is_leader: false
         source_node: 10.10.10.252:20002:768:tpcc_cluster:node-
tpcc_cluster-001:0
           dest_node: 10.10.10.10:20002:768:tpcc_cluster:node-
tpcc_cluster-003:1
                term: 4
                type: NULL
        rep_group_id: 768
          start_time: 2024-08-27 11:15:46 367
            end_time: 2024-08-27 11:15:46 473
             err_msg: NULL
 transfered_file_num: 2
transfered_file_size: 10702037
      total_file_num: NULL
     total_file_size: NULL
               stage: done
           cost_info: DOWNLOAD_SNAPSHOT: 57ms
*************************** 2. row ***************************
           is_leader: true
```

```
          source_node: 10.10.10.252:20002:768:tpcc_cluster:node-
tpcc_cluster-001:0
            dest_node: 10.10.10.10:20002:768:tpcc_cluster:node-
tpcc_cluster-003:1
                 term: 4
                 type: Log Delay
         rep_group_id: 768
           start_time: 2024-08-27 11:15:46 505
             end_time: 2024-08-27 11:15:46 612
              err_msg: NULL
 transfered_file_num: NULL
transfered_file_size: NULL
       total_file_num: 2
     total_file_size: 10702037
               stage: done
           cost_info:  [INSTALL_SNAPSHOT: <COMPACT_SST_TO_LBASE: 35ms>,
<GENERATE_AND_GET_HARD_LINK_SST_FILE: 0ms>]
2 rows in set (0.00 sec)
```

TDSTORE_LOCAL_METADATA_DESTROY_REPLICA

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_LOCAL_METADATA_DESTROY_REPLICA` 用于记录正在进行的数据物理删除相关的信息。当触发物理数据删除（包括副本的删除、replication group 的删除、key range region 的删除）的时候，表中会写入对应记录。当数据完成删除的时候，该记录会从表中移除。

字段说明

字段名	类型	描述
REPLICATION_GROUP_ID	BIGINT UNSIGNED	被删除的数据所属的 replication group 的 ID。
JOB_ID	BIGINT UNSIGNED	执行删除操作的 job ID。
CLEAN_TASK_TYPE	INT	执行删除操作的 job 类型。值可能包括： • 0：表示是对 replication group 做删除。 • 1：表示是对 key range region 做删除。
DATA_SPACE_TYPE	INT	data space 类型。值可能包括： • 1：表示是 system data space。 • 2：表示是 user data space。
META_VERSION	BIGINT	replication group 元数据版本。
MEMBER_VERSION	BIGINT	replication group 的 raft group 成员变动版本。
DATA_REGION_RAFT_LOG_INDEX	BIGINT	执行删除时，replication group 的 raft log index。
REGION_LIST	VARCHAR(1024)	被删除的 region 列表。
TABLE_ID_LIST	VARCHAR(1024)	删除涉及到的 data object ID 列表。

示例

```
tdsql3_sys_local@localhost [information_schema]> select * from
TDSTORE_LOCAL_METADATA_DESTROY_REPLICA \G
*************************** 1. row ***************************
      replication_group_id: 257
                    job_id: 1507
           clean_task_type: 1
           data_space_type: 2
              meta_version: 0
            member_version: 0
data_region_raft_log_index: 6203
               region_list: regions { rep_group_id: 257 region_id: 1503
start_key: "\000\000\'/" end_key: "\000\000\'0" data_space_type:
DATA_SPACE_TYPE_USER }
             table_id_list:
```

TDSTORE_LOCAL_METADATA_TSO_META

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_LOCAL_METADATA_TSO_META` 用于记录各个 replication group 的授时元信息。当授时来源为 TDStore 的时候，可以通过查询表，获取各个 replication group 的授时元信息。

字段说明

字段名	类型	描述
replication_group_id	bigint unsigned	replication group 的 ID。
value	varchar(1024)	授时元信息。

示例

```
tdsql3_sys_local@localhost [information_schema]> select * from
TDSTORE_LOCAL_METADATA_TSO_META;
+----------------------+-------------------------------------------------
-----------------------------------------------+
| replication_group_id | value
|
+----------------------+-------------------------------------------------
-----------------------------------------------+
|                  768 | term: 8 unix_timestamp_ns: 1743422628000000000
start_from_ts: 29249777673699328 enable: true |
+----------------------+-------------------------------------------------
-----------------------------------------------+
```

TDSTORE_LOCAL_METADATA_WRITE_FENCE

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_LOCAL_METADATA_WRITE_FENCE` 用于展示节点各个 replication group 的 write fence 信息。

字段说明

字段名	类型	描述
replication_group_id	bigint unsigned	write fence 所在的 replication group 的 ID。
schema_obj_id	int unsigned	write fence 的 ID。
schema_obj_version	int unsigned	write fence 的 version。
start_key	varchar(128)	write fence 的 start key。
end_key	varchar(128)	write fence 的 end_key。
is_online_ddl	bigint unsigned	是否为 online ddl 创建的 write fence。（目前该字段已被废弃）

示例

```
tdsql3_sys_local@localhost [information_schema]> select * from
TDSTORE_LOCAL_METADATA_WRITE_FENCE limit 3 \G
*************************** 1. row ***************************
replication_group_id: 257
       schema_obj_id: 10001
  schema_obj_version: 3
           start_key: 0x00002711
             end_key: 0x00002712
       is_online_ddl: 0
*************************** 2. row ***************************
replication_group_id: 257
```

```
        schema_obj_id: 10002
  schema_obj_version: 1
           start_key: 0x00002712
             end_key: 0x00002713
       is_online_ddl: 0
*************************** 3. row ***************************
replication_group_id: 257
       schema_obj_id: 10003
  schema_obj_version: 1
           start_key: 0x00002713
             end_key: 0x00002714
       is_online_ddl: 0
```

TDSTORE_PART_CTX

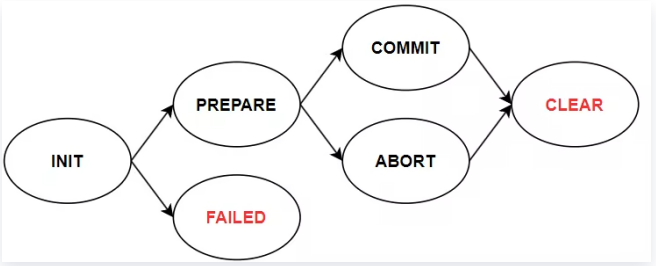
最近更新时间：2025-11-18 10:10:23

功能

`TDSTORE_PART_CTX` 用于展示集群中所有 hybrid 节点的 TDStore 上所有悲观锁等待信息。

字段说明

字段名	类型	是否可以为 NULL	描述
trans_id	bigint unsigned	否	事务的唯一标识符。
state	varchar(16)	否	参与者的状态。 • `RUNNING`：参与者正处于读写阶段（即还未进入提交流程），且未被悲观锁阻塞。 • `WAIT LOCK`：参与者正处于读写阶段，且正在被悲观锁阻塞（此时 `requested_lock_id` 和 `blocking_lock_id` 一定不为 `NULL`）。 • `COMMITTING`：参与者正在尝试提交（并不意味着事务最终一定会提交成功）。 • `ROLLING BACK`：参与者正在回滚。
commit_state	varchar(10)	否	参与者的提交状态。 • `INIT`：参与者还未进入提交流程，或者已进入提交流程但正在同步 redo/prepare log。 • `FAILED`：参与者 prepare 失败。 • `PREPARE`：参与者 2PC prepare 成功。 • `COMMIT`：参与者 2PC commit 成功。 • `ABORT`：参与者 2PC prepare 成功，但在第二阶段 abort 了。 • `CLEAR`：参与者 commit 或 abort 后，即将释放参与者上下文（这个状态十分短暂）。 状态的推进关系如下图所示：

			
snapshot_ts	bigint unsigned	否	参与者的快照读时间戳。
prepare_ts	bigint unsigned	是	事务的 prepare 时间戳。事务的所有已经 prepare 了的参与者，prepare 时间戳相同。当参与者还未 prepare 的时候，值为 `NULL` 。
commit_ts	bigint unsigned	是	事务的 commit 时间戳。事务的所有已经 commit 了的参与者，commit 时间戳相同。当参与者还未 commit 的时候，值为 `NULL` 。
is_replay	int	否	如果值为 `1` ，代表参与者正在通过 replay raft log 来推进 2PC，这一般发生在 Follower 上，但也有可能发生在 Leader 上任之前。
data_space_type	varchar(32)	否	参与者对应的 data space。`DATA_SPACE_TYPE_USER` 代表参与者访问了用户表，`DATA_SPACE_TYPE_SYSTEM` 代表参与者访问了系统表。
replication_group_id	bigint unsigned	否	参与者对应的 replication group ID。
coordinator_replication_group_id	bigint unsigned	是	协调者对应的 replication group ID。当参与者还未 prepare 的时候，值为 `NULL` 。
expired_time	datetime	否	参与者超时释放时间。
requested_lock_id	varchar(128)	是	唯一标识正在申请的悲观锁，或者正在读取的 key 范围。格式参考 `data_locks` 表的 `ENGINE_LOCK_ID` 字段。如果没有正在等待悲观锁，值为 `NULL` 。
blocking_lock_id	varchar(128)	是	唯一标识正在等待的悲观锁。格式参考 `data_locks` 表的 `ENGINE_LOCK_ID` 字段。如果没有正在等待悲观锁，值为 `NULL` 。

lock_wait_milliseconds	bigint unsigned	否	累计等待悲观锁花费的耗时（单位：毫秒）。
single_keys_locked	bigint unsigned	否	持有的单条 key 悲观锁的数量。
ranges_locked	bigint unsigned	否	持有的 key 区间悲观锁的数量。
write_batch_size	bigint unsigned	是	write batch 大小（单位：字节）。`NULL` 值代表着收集参与者信息的 bthread 没能获取到参与者的锁，因此没能获取到相关信息，此时大概率参与者正在等待某个悲观锁的释放。
lock_memory_usage	bigint unsigned	是	悲观锁消耗的内存大小（单位：字节）。`NULL` 值代表着收集参与者信息的 bthread 没能获取到参与者的锁，因此没能获取到相关信息，此时大概率参与者正在等待某个悲观锁的释放。
isolation_level	varchar(16)	否	隔离级别。TDStore 仅支持两种隔离级别，`REPEATABLE_READ` 和 `READ_COMMITTED`。
submit_pending_log_num	bigint unsigned	是	正在同步的 raft 日志数量。`NULL` 值代表着收集参与者信息的 bthread 没能获取到参与者的锁，因此没能获取到相关信息，此时大概率参与者正在等待某个悲观锁的释放。
note	varchar(256)	否	备注信息。 • 空字符串：无备注信息。 • `fail to acquire participant lock`：获取参与者锁失败。这种情况下，部分字段（如 write_batch_size ）的值为 `NULL`。
node_name	varchar(256)	是	开启该事务参与者的 SQLEngine 节点名称。
thread_id	bigint unsigned	否	运行该事务参与者的线程 ID。这是 Performance Schema 的内部线程 ID，在 Performance Schema 的生命周期内唯一，可用于可靠地关联 Performance Schema 的各类事件表。
sql_session_id	bigint unsigned	否	对应开启该事务参与者的会话连接（session）ID。

示例

- 输出示例：

```
tdsql [(none)]> SELECT * FROM information_schema.tdstore_part_ctx
LIMIT 3\G
*************************** 1. row ***************************
                        trans_id: 28707112062290305
                           state: RUNNING
                    commit_state: INIT
                     snapshot_ts: 28707112062290305
                      prepare_ts: NULL
                       commit_ts: NULL
                       is_replay: 0
                data_space_type: DATA_SPACE_TYPE_USER
            replication_group_id: 257
coordinator_replication_group_id: NULL
                    expired_time: 2024-03-22 11:23:37
               requested_lock_id: NULL
                blocking_lock_id: NULL
          lock_wait_milliseconds: 0
              single_keys_locked: 4
                   ranges_locked: 0
                write_batch_size: 3219
               lock_memory_usage: 1008
                 isolation_level: REPEATABLE_READ
          submit_pending_log_num: 0
                            note:
                       node_name: node-1-001
                       thread_id: 65
                  sql_session_id: 1048736
*************************** 2. row ***************************
                        trans_id: 28707112062290205
                           state: COMMITTING
                    commit_state: PREPARE
                     snapshot_ts: 28707112062290205
                      prepare_ts: 28707112062290205
                       commit_ts: 28707112062290368
                       is_replay: 0
                data_space_type: DATA_SPACE_TYPE_USER
            replication_group_id: 257
coordinator_replication_group_id: 257
```

```
                    expired_time: 2024-03-22 11:23:37
                requested_lock_id: NULL
                 blocking_lock_id: NULL
           lock_wait_milliseconds: 0
               single_keys_locked: 7
                    ranges_locked: 0
                 write_batch_size: NULL
                lock_memory_usage: NULL
                  isolation_level: REPEATABLE_READ
           submit_pending_log_num: NULL
                             note: fail to acquire participant lock
                        node_name: node-1-001
                        thread_id: 65
                   sql_session_id: 1048736
*************************** 3. row ***************************
                         trans_id: 28707111676413643
                            state: LOCK WAIT
                     commit_state: INIT
                      snapshot_ts: 28707111676413643
                       prepare_ts: NULL
                        commit_ts: NULL
                        is_replay: 0
                  data_space_type: DATA_SPACE_TYPE_USER
              replication_group_id: 257
coordinator_replication_group_id: NULL
                    expired_time: 2024-03-22 11:23:14
                requested_lock_id: 28707111676413643_0000279C80018657
                 blocking_lock_id: 28707111676413316_0000279C80018657
           lock_wait_milliseconds: 0
               single_keys_locked: 4
                    ranges_locked: 0
                 write_batch_size: NULL
                lock_memory_usage: NULL
                  isolation_level: REPEATABLE_READ
           submit_pending_log_num: NULL
                             note: fail to acquire participant lock
                        node_name: node-1-002
                        thread_id: 65
                   sql_session_id: 1048704
```

- 查询示例：

```
-- 查询当前节点上参与者的数量
SELECT COUNT(*) FROM information_schema.tdstore_part_ctx \G

-- 查询当前节点上，replication group 257 的参与者信息
SELECT * FROM information_schema.tdstore_part_ctx WHERE
replication_group_id = 257 \G

-- 查询当前节点上正在提交流程中的参与者信息
SELECT * FROM information_schema.tdstore_part_ctx WHERE state =
'COMMITTING' \G

-- 查询当前节点上执行时间超过 5s 的事务的信息（目前 trans_id 右移 24 位得到的结果，是事务开始时间的 unix timestamp）
SELECT * FROM information_schema.tdstore_part_ctx WHERE trans_id >> 24
< UNIX_TIMESTAMP(DATE_SUB(NOW(), INTERVAL 5 SECOND)) \G

-- 查询当前节点上 write batch 大小超过 128 MB 的大事务的信息（事务写过的数据，存储于 write batch 中）
SELECT * FROM information_schema.tdstore_part_ctx WHERE
write_batch_size > 128 * 1024 * 1024 \G
```

TDSTORE_PESSIMISTIC_DEADLOCK_DETAIL_INFO

最近更新时间：2025-11-18 10:10:23

功能

在执行死锁检测的节点上查询 `TDSTORE_PESSIMISTIC_DEADLOCK_DETAIL_INFO` 系统表，可以得到历史所有死锁中，构成死锁环的事务详细等待关系：包括申请锁和阻塞的事务 ID，申请和阻塞的锁范围，造成该等待关系的 SQL 运行在哪个节点上，造成该等待关系的事务正在执行的 SQL 语句。

字段说明

字段名	类型	说明
rollback_trans_id	bigint unsigned	本死锁环中，被回滚的事务的 ID。
requesting_node	varchar(64)	锁等待关系中，执行申请悲观锁的事务的 SQLEngine 节点名称。
requesting_trans_id	bigint unsigned	锁等待关系中，申请悲观锁的事务的 ID。
blocking_trans_id	bigint unsigned	锁等待关系中，已经持有悲观锁的事务的 ID。
req_lock_range	varchar(64)	锁等待关系中，申请悲观锁的事务申请的锁范围。
blk_lock_range	varchar(64)	锁等待关系中，已经持有悲观锁的事务此时持有的锁范围。
timestamp	varchar(64)	本死锁环对应的死锁事件的发生时间。
requesting_sql	varchar(128)	锁等待关系中，申请悲观锁的事务此时正在执行的 SQL 语句。

示例

- 输出示例：

```
tdsql> SELECT * FROM
information_schema.tdstore_pessimistic_deadlock_detail_info \G
*************************** 1. row ***************************
  rollback_trans_id: 29468596191101374
    requesting_node: node-1
requesting_trans_id: 29468595838780035
  blocking_trans_id: 29468595939442912
     req_lock_range: 000027308000000F
     blk_lock_range: 000027308000000F
          timestamp: 2025-08-29 19:00:30 82
     requesting_sql: SELECT * FROM deadlock_test WHERE id = 15 FOR
UPDATE
*************************** 2. row ***************************
  rollback_trans_id: 29468596191101374
    requesting_node: node-2
requesting_trans_id: 29468595939442912
  blocking_trans_id: 29468596191101374
     req_lock_range: [0000273080000010,0000273080000015)
     blk_lock_range: [0000273080000010,0000273080000015)
          timestamp: 2025-08-29 19:00:30 82
     requesting_sql: SELECT * FROM deadlock_test WHERE id BETWEEN 16
AND 20 FOR UPDATE
*************************** 3. row ***************************
  rollback_trans_id: 29468596191101374
    requesting_node: node-3
requesting_trans_id: 29468596191101374
  blocking_trans_id: 29468595838780035
     req_lock_range: 000027308000000C
     blk_lock_range: [000027308000000A,000027308000000F)
          timestamp: 2025-08-29 19:00:30 82
     requesting_sql: SELECT * FROM deadlock_test WHERE id = 12 FOR
UPDATE
```

- **查询示例**

```
-- 查询所有节点上，历史发生过的死锁详细信息
SELECT * FROM
information_schema.tdstore_pessimistic_deadlock_detail_info \G
```

```
-- 查询所有节点上，被回滚事务 ID 为 29468596191101374 的死锁事件的详细信息：
SELECT * FROM
information_schema.tdstore_pessimistic_deadlock_detail_info WHERE
rollback_trans_id = 29468596191101374 \G
```

TDSTORE_PESSIMISTIC_DEADLOCK_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_PESSIMISTIC_DEADLOCK_INFO` 用于在执行死锁检测的节点上查询该系统表，可以得到历史所有死锁的发生时间、被回滚的事务的 ID、构成死锁环的各个事务的 ID 以及执行该事务的 SQLEngine 的 Node ID。

字段说明

字段名	类型	说明
rollback_trans_id	bigint unsigned	被回滚的事务的 ID。
cycle_transactions	longtext	构成死锁环的各个事务的 ID 以及执行该事务的 SQLEngine 的 Node ID。
timestamp	varchar(64)	死锁的发生时间。

示例

```
tdsql> SELECT * FROM
information_schema.tdstore_pessimistic_deadlock_info;
+----------------------+-----------------------------------------------------
--------+--------------------------+
| rolled_back_trans_id | cycle_transactions
| timestamp                |
+----------------------+-----------------------------------------------------
--------+--------------------------+
|    29342246289539082 | 29342246289539082 (node 1), 29342246423756948
(node 1) | 2025-06-03 15:03:05 501 |
+----------------------+-----------------------------------------------------
--------+--------------------------+
```

TDSTORE_REGION_EVENT_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_REGION_EVENT_INFO` 用于查询 TDStore 层 REGION 相关任务执行情况，包括 `create region`、`delete region`、`split region`、`merge region`。

字段说明

字段名	类型	描述
node_name	varchar(64)	节点名。
job_description	text	任务描述。
ret	varchar(64)	函数返回值。
job_id	bigint unsigned	任务 ID。
rep_group_id	bigint unsigned	RG ID。
old_key_range_region_ids	text	原 REGION 的 ID 列表。
key_range_region_id	bigint unsigned	• `create region` 场景下，表示新生成的 REGION 的 ID。 • `split region` 场景下，表示分裂生成的 REGION 的 ID。 • `merge region` 场景下，表示合并生成的 REGION 的 ID。
my_job_ret	varchar(64)	任务返回值。
report_mc_ret	varchar(64)	汇报 MC 的返回值。
event_type	varchar(64)	任务类型。

begin_time	varchar(64)	任务开始时间。
cost_us	bigint unsigned	任务耗时，单位为微秒。
detailed_info	text	任务及子任务每一步的耗时。

示例

```
tdsql>select * from TDSTORE_REGION_EVENT_INFO limit 1 \G
*************************** 1. row ***************************
               node_name: node-Module13-180-5-004
         job_description: create key range region
                  job_id: 578
            rep_group_id: 70695
old_key_range_region_ids: NULL
     key_range_region_id: 569
                     ret: 0(EC_OK)
              my_job_ret: 0(EC_OK)
           report_mc_ret: 0(EC_OK)
              event_type: TD_EVENT_CREATE_REGION_MAIN
              begin_time: 2024-11-04 20:02:02 258
                 cost_us: 44083
           detailed_info:  [CREATE_REGION_MAIN: <BEGIN: 2us>,
<CHECK_EXIST: 2us>, <CHECK_OVERLAP: 131us>, <PREPARE: 51us>,
<CHECK_READY: 0us>, <REPORT_PREPARE: 43734us>, <COMMIT: 70us>,
<CLEAN_FAILED_JOB: 0us>, <END: 93us>]
```

TDSTORE_REPLICATION_GROUP_EVENT_INFO

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_REPLICATION_GROUP_EVENT_INFO` 用于查询 TDStore 层 replication_group 相关任务执行情况，包括：

- `create replication_group`
- `delete replication_group`
- `split replication_group`
- `merge replication_group`
- `destroy replica`

字段说明

字段名	类型	描述
node_name	varchar(64)	节点名称。
job_description	text	任务描述。
job_id	bigint unsigned	任务 ID。
old_rep_group_id	bigint unsigned	原 replication_group 的 ID。
rep_group_id	bigint unsigned	• `create replication_group` 场景下，表示新生成的 replication_group 的 ID。 • `split replication_group` 场景下，表示分裂生成的 replication_group 的 ID。 • `merge replication_group` 场景下，表示合并生成的 replication_group 的 ID。
ret	varchar(64)	函数返回值。
my_job_ret	varchar(64)	任务的返回值。

report_mc_ret	varchar(64)	向 mc 汇报的返回值。
extra_info	text	部分任务的额外信息，当前仅支持 destry replication_group，将列出原因。
event_type	varchar(64)	任务类型。
begin_time	varchar(64)	任务开始时间。
cost_us	bigint unsigned	任务耗时，单位为微秒。
detailed_info	text	任务及子任务每一步的耗时。

示例

```
tdsql>select * from TDSTORE_REPLICATION_GROUP_EVENT_INFO limit 1\G
*************************** 1. row ***************************
       node_name: node-Module13-180-5-004
 job_description: merge replication group
          job_id: 31736
old_rep_group_id: 2607222
    rep_group_id: 8101200
             ret: 0(EC_OK)
      my_job_ret: 0(EC_OK)
   report_mc_ret: 0(EC_OK)
      extra_info: NULL
      event_type: TD_EVENT_MERGE_REPLICATION_GROUP
      begin_time: 2024-11-04 20:53:56 294
         cost_us: 3726373
   detailed_info:  [MERGE_REPLICATION_GROUP: <BEGIN: 4120us>,
<WAIT_ALL_TXNS: 3us>, <WAIT_MERGE_LOG_APPLIED: 967033us>,
<FORCE_INSTALL_SNAPSHOT_FOR_VANISHED_RG: 2213872us>, <MERGE_LEASE: 0us>,
<MERGE_WRITE_FENCE: 516515us>, <REPORT_TO_MC: 21880us>, <COMMIT:
2896us>, <END: 54us>]
```

TDSTORE_SST_PROPS

最近更新时间：2025-11-18 10:10:23

功能

`INFORMATION_SCHEMA.TDSTORE_SST_PROPS` 用于展示 TDStore 中 SSTable 的属性。

字段说明

字段名	类型	描述
DB_TYPE	VARCHAR(193)	SSTable 所属的 RocksDB 类型，支持 `data_db` 、 `multi_raft_db` 、 `monitor_info_db` 。
SST_NAME	VARCHAR(193)	SSTable 的名称。
COLUMN_FAMILY	INT	SSTable 对应的 ColumnFamily ID。
LEVEL	INT	SSTable 所处的层级。
MAX_KEY	VARCHAR(193)	SSTable 中最大的 Key。
MIN_KEY	VARCHAR(193)	SSTable 中最小的 Key。
NUM_DATA_BLOCKS	BIGINT	SSTable 中数据块的数量。
NUM_ENTRIES	BIGINT	SSTable 中 Entry 的数量。
NUM_DELETIONS	BIGINT	SSTable 中删除的数量。
RAW_KEY_SIZE	BIGINT	SSTable 未压缩之前 Key 的总大小，单位为字节。
RAW_VALUE_SIZE	BIGINT	SSTable 未压缩之前 Value 的总大小，单位为字节。
DATA_BLOCK_SIZE	BIGINT	SSTable 中数据块的总大小，单位为字节。
INDEX_BLOCK_SIZE	BIGINT	SSTable 中索引块的总大小，单位为字节。
INDEX_PARTITIONS	INT	SSTable 中索引分区的数量。

TOP_LEVEL_INDEX_SIZE	BIGINT	SSTable 中最高层级索引的大小，单位为字节。
FILTER_BLOCK_SIZE	BIGINT	SSTable 中过滤块的总大小，单位为字节。
COMPRESSION_ALGO	VARCHAR(193)	SSTable 所使用的压缩算法。
CREATION_TIME	BIGINT	SSTable 的创建时间。
OLDEST_KEY_TIME	BIGINT	SSTable 最老的 Key 的创建时间。
FILTER_POLICY	VARCHAR(193)	SSTable 所使用的 Filter 名称。
NODE_NAME	VARCHAR(513)	SSTable 所处的 SQLEngine 节点名称。

示例

```
tdsql > select * from tdstore_sst_props\G
*************************** 1. row ***************************
             DB_TYPE: data_db
            SST_NAME: 000019.sst
       COLUMN_FAMILY: 1
               LEVEL: 0
             MAX_KEY: 0000011E00000001
             MIN_KEY: 0000000100000001
     NUM_DATA_BLOCKS: 145
         NUM_ENTRIES: 54974
       NUM_DELETIONS: 794
        RAW_KEY_SIZE: 2256766
      RAW_VALUE_SIZE: 8356920
     DATA_BLOCK_SIZE: 9729222
    INDEX_BLOCK_SIZE: 5047
    INDEX_PARTITIONS: 0
TOP_LEVEL_INDEX_SIZE: 0
   FILTER_BLOCK_SIZE: 0
    COMPRESSION_ALGO: NULL
       CREATION_TIME: 1724913962
     OLDEST_KEY_TIME: 1724913362
```

```
       FILTER_POLICY: NULL
           NODE_NAME: node-1-001
*************************** 2. row ***************************
             DB_TYPE: data_db
            SST_NAME: 000020.sst
       COLUMN_FAMILY: 2
               LEVEL: 0
            MAX_KEY:
000027121E711E951C471E95091CAA1DAA1CAA1DB9091E95020B1E951E33091EB51DB91C
7A1C47091E951CAA020B1D77091CAA1DB90000000004
            MIN_KEY:
000027121C8F1D321C471CF4091DB91DDD1E711E95091D321C7A1E710277091C471D771D
771DDD091EF5020B1D32020B091E71020B1E951C47091C601D771CAA1E7108
     NUM_DATA_BLOCKS: 1
         NUM_ENTRIES: 6
       NUM_DELETIONS: 0
        RAW_KEY_SIZE: 477
      RAW_VALUE_SIZE: 278
     DATA_BLOCK_SIZE: 646
    INDEX_BLOCK_SIZE: 28
    INDEX_PARTITIONS: 0
TOP_LEVEL_INDEX_SIZE: 0
   FILTER_BLOCK_SIZE: 69
    COMPRESSION_ALGO: NULL
       CREATION_TIME: 1724913962
     OLDEST_KEY_TIME: 1724913362
       FILTER_POLICY: NULL
           NODE_NAME: node-1-001
*************************** 3. row ***************************
             DB_TYPE: data_db
            SST_NAME: 000011.sst
       COLUMN_FAMILY: 3
               LEVEL: 0
            MAX_KEY: 00000000000000003000000011D
            MIN_KEY: 00000000000000010100002711
     NUM_DATA_BLOCKS: 1
         NUM_ENTRIES: 165
       NUM_DELETIONS: 0
        RAW_KEY_SIZE: 3465
      RAW_VALUE_SIZE: 2310
```

```
      DATA_BLOCK_SIZE: 4481
     INDEX_BLOCK_SIZE: 28
     INDEX_PARTITIONS: 0
TOP_LEVEL_INDEX_SIZE: 0
    FILTER_BLOCK_SIZE: 0
     COMPRESSION_ALGO: NULL
        CREATION_TIME: 1724913353
      OLDEST_KEY_TIME: 3
        FILTER_POLICY: NULL
            NODE_NAME: node-1-001
3 rows in set (0.00 sec)
```

SYS LOGSERVICE_DD_PARTITIONS

最近更新时间：2025-11-18 10:10:23

功能

存储每个分区的 `tindex_id` 到主表 `tindex_id` 的映射。对于分区表，在解析 Raft log 时可以通过分区的 `tindex_id` 找到对应主表的 `tindex_id` ，进而通过查询 `sys.logservice_dd_tables` 表获得解析 Raft log 所需的表结构。

字段说明

字段名	类型	描述
sid	bigint unsigned	用于标识 `logservice` 的编号。
tindex_id	int unsigned	标识每个分区的 ID。
parent_tindex_id	int unsigned	当前分区对应主表的 `tindex id` 。

示例

```
tdsql> DESC sys.logservice_dd_partitions;
+------------------+-----------------+------+-----+---------+-------+
| Field            | Type            | Null | Key | Default | Extra |
+------------------+-----------------+------+-----+---------+-------+
| sid              | bigint unsigned | NO   | PRI | NULL    |       |
| tindex_id        | int unsigned    | NO   | PRI | NULL    |       |
| parent_tindex_id | int unsigned    | NO   |     | NULL    |       |
+------------------+-----------------+------+-----+---------+-------+

tdsql> SELECT  * FROM sys.logservice_dd_partitions;
+------------------+-----------+------------------+
| sid              | tindex_id | parent_tindex_id |
+------------------+-----------+------------------+
| 29475711576178806 |     10035 |            10034 |
| 29475711576178806 |     10036 |            10034 |
```

```
| 29475711576178806 |     10037 |            10034 |
| 29475711576178806 |     10038 |            10034 |
+-------------------+-----------+------------------+
```

LOGSERVICE_DD_TABLES

最近更新时间：2025-12-24 12:03:42

功能

存储 `logservice` 回放解析 raft log 所需的表结构信息，并在 `logservice` 回放 DDL 的过程中对其中的表结构进行更新。

字段说明

字段名	类型	描述
sid	bigint unsigned	用于标识 `logservice` 的编号。
tindex_id	int unsigned	用于标识每个主表的 ID。
info	longtext	存储序列化之后的表结构元信息。

示例

```
tdsql> DESC sys.logservice_dd_tables;
+-----------+-----------------+------+-----+---------+-------+
| Field     | Type            | Null | Key | Default | Extra |
+-----------+-----------------+------+-----+---------+-------+
| sid       | bigint unsigned | NO   | PRI | NULL    |       |
| tindex_id | int unsigned    | NO   | PRI | NULL    |       |
| info      | longtext        | YES  |     | NULL    |       |
+-----------+-----------------+------+-----+---------+-------+


tdsql> SELECT * FROM sys.logservice_dd_tables WHERE tindex_id = 10029 \G
*************************** 1. row ***************************
      sid: 29473053092085985
tindex_id: 10029
     info:
{"mysqld_version_id":80026,"dd_version":80023,"sdi_version":80019,"dd_ob
ject_type":"Table","dd_object":
{"name":"t2","tindex_id":10029,"schema_version":1,"schema_status":0,"mys
ql_version_id":80026,"created":20250902021912,"last_altered":20250902021
```

```
912,"hidden":1,"options":"avg_row_length=0;explicit_encryption=0;key_blo
ck_size=0;keys_disabled=0;pack_record=0;stats_auto_recalc=0;stats_sample
_pages=0;","columns":
[{"name":"c1","type":4,"is_nullable":false,"is_zerofill":false,"is_unsig
ned":false,"is_auto_increment":false,"schema_version":1,"is_virtual":fal
se,"hidden":1,"ordinal_position":1,"char_length":11,"numeric_precision":
10,"numeric_scale":0,"numeric_scale_null":false,"datetime_precision":0,"
datetime_precision_null":1,"has_no_default":true,"default_value_null":fa
lse,"init_default_value_null":true,"srs_id_null":true,"srs_id":0,"defaul
t_value":"AAAAAA==","init_default_value":"","default_value_utf8_null":tr
ue,"default_value_utf8":"","default_option":"","update_option":"","comme
nt":"","generation_expression":"","generation_expression_utf8":"","optio
ns":"interval_count=0;","se_private_data":"","engine_attribute":"","seco
ndary_engine_attribute":"","column_key":2,"column_type_utf8":"int","elem
ents":[],"collation_id":8,"is_explicit_collation":false},
{"name":"c2","type":4,"is_nullable":true,"is_zerofill":false,"is_unsigne
d":false,"is_auto_increment":false,"schema_version":1,"is_virtual":false
,"hidden":1,"ordinal_position":2,"char_length":11,"numeric_precision":10
,"numeric_scale":0,"numeric_scale_null":false,"datetime_precision":0,"da
tetime_precision_null":1,"has_no_default":false,"default_value_null":tru
e,"init_default_value_null":true,"srs_id_null":true,"srs_id":0,"default_
value":"","init_default_value":"","default_value_utf8_null":true,"defaul
t_value_utf8":"","default_option":"","update_option":"","comment":"","ge
neration_expression":"","generation_expression_utf8":"","options":"inter
val_count=0;","se_private_data":"","engine_attribute":"","secondary_engi
ne_attribute":"","column_key":1,"column_type_utf8":"int","elements":
[],"collation_id":8,"is_explicit_collation":false}],"schema_ref":"test2"
,"autoinc_tindex_id":0,"hidden_pk_autoinc_tindex_id":0,"se_private_id":1
8446744073709551615,"engine":"ROCKSDB","last_checked_for_upgrade_version
_id":0,"comment":"","se_private_data":"autoinc_version=1;create_data_obj
_task_id=29473869991510193;distribution_policy_id=1;origin_db=test2;orig
in_table=t2;partition_policy_id=0;sync_table=0;","engine_attribute":"","
secondary_engine_attribute":"","row_format":1,"partition_type":0,"partit
ion_expression":"","partition_expression_utf8":"","default_partitioning"
:0,"subpartition_type":0,"subpartition_expression":"","subpartition_expr
ession_utf8":"","default_subpartitioning":0,"indexes":
[{"name":"PRIMARY","id":334,"tindex_id":10029,"hidden":false,"is_generat
ed":false,"ordinal_position":1,"comment":"","options":"flags=0;","se_pri
vate_data":"create_data_obj_task_id=29473869991510193;distribution_polic
y_id=1;kv_format_version=1;","type":1,"algorithm":1,"is_algorithm_explic
```

```
it":false,"is_visible":true,"engine":"ROCKSDB","engine_attribute":"","se
condary_engine_attribute":"","elements":
[{"ordinal_position":1,"length":4,"order":2,"hidden":false,"column_opx":
0}]}],"foreign_keys":[],"check_constraints":[],"partitions":
[],"collation_id":33,"extra_info":"{\"version\":3,\"create_ts\":0}"}}
1 row in set, 1 warning (0.00 sec)
```

META_CLUSTER_TASKS

最近更新时间：2025-11-18 10:10:23

功能

MC 的 Task 执行记录。

字段说明

字段名	类型	描述
task_id	bigint unsigned	Task ID。
create_time	datetime	创建 Task 时间。
finish_time	datetime	结束 Task 时间。
task_type	varchar(64)	Task 类型。
task_status	varchar(64)	Task 执行状态。
task_desc	text	Task 执行详情。
task_meta	text	Task 元信息。
task_progress	text	Task 执行阶段。

示例

```
tdsql3_sys_local@localhost [sys]> select * from meta_cluster_tasks limit
1\G
*************************** 1. row ***************************
      task_id: 29071203637395469
  create_time: 2024-11-28 15:26:07
  finish_time: 2024-11-28 15:26:07
    task_type: TASK_TYPE_CREATE_DATA_OBJECTS
  task_status: TASK_STATUS_SUCCEED
    task_desc:
    task_meta: {"CreateDataObjects":
{"task_id":29071203637395469,"data_objects":
```

[{"data_obj_id":2,"data_obj_name":"U1lTVEVNX1NQQUNFX0lOREVYX0lEX1NFUVVFTkNF","data_obj_type":13,"data_obj_id_hierarchy":[2],"binary_key_range":{"start_key":"AAAAAg==","end_key":"AAAAAw=="},"has_data":true,"data_space_type":1,"create_task_id":29071203637395469,"distribution_policy_id":1}],"rep_group_jobs":[{"rep_group_job_id":12,"rep_group_id":1792,"rep_group_job_type":11,"ReplicationGroupJobInfo":{"CreateRegionInRepGroup":{"rep_group_id":1792,"meta_version":1,"new_region_meta_list":[{"region_id":11,"start_key":"AAAAAg==","end_key":"AAAAAw==","data_space_type":1,"member":{"region_id":11,"member_node_names":["node-******-1-001","node-******-1-003","node-******-1-002"]},"region_labels":{"no-auto-merge":"1","no-auto-split":"1"},"rep_group_id":1792,"data_obj_id":2,"has_data":true,"data_obj_type":13}]}}}],"ddl_job_desc":"Empty"}}

```
   task_cause: CAUSE_API_TRIGGER
task_progress: <nil>
1 row in set (0.00 sec)
```

META_CLUSTER_JOBS

最近更新时间：2025-11-18 10:10:23

功能

MC 的调度任务记录。

字段说明

字段名	类型	描述
rep_group_id	bigint unsigned	RepGroup ID。
rep_group_job_id	bigint unsigned	Job ID。
rep_group_job_type	varchar(64)	Job 类型。
rep_group_job_meta	json	Job 元数据。
rep_group_job_state	bigint	Job 执行状态。
create_time	datetime	创建 Job 时间。
finish_time	datetime	结束 Job 时间。
job_raft_log_index	bigint	Job 执行结束后的 raft log 位点。
job_vanished_rg_raft_log_index	bigint	Job 执行结束后 vanished rg 的 raft log 位点（仅针对合并 RG 任务有效）。
job_commit_ts	varchar(64)	Job 提交时间戳。
job_msg	text	Job 信息。
job_desc	text	Job 执行详情。
task_id	bigint unsigned	发起 Job 的 Task 的 ID。

task_type	varchar(64)	发起 Job 的 Task 的 类型。
cause_type	varchar(64)	发起调度 Job 的原因类型。
schedule_rule	text	调度任务具体描述，例如迁移任务中的 MigRule 等。

示例

```
tdsql3_sys_local@localhost [sys]> select * from meta_cluster_jobs limit
1\G
*************************** 1. row ***************************
                  rep_group_id: 1792
              rep_group_job_id: 8
            rep_group_job_type: RG_JOB_TYPE_CREATE_RG
            rep_group_job_meta: {"rep_group_id": 1792,
"rep_group_job_id": 8, "rep_group_job_type": 1,
"ReplicationGroupJobInfo": {"CreateRepGroup": {"new_rep_group_meta":
{"quorum": 1, "learners": ["node-***-1-003", "node-***-1-002"],
"node_name": "node-***-1-001", "create_time": 1732778760675987857,
"member_role": 3, "member_uuids": {"node-***-1-001": 0, "node-***-1-
002": 0, "node-***-1-003": 0}, "rep_group_id": 1792, "learner_quorum":
2, "data_space_type": 1, "rep_group_labels": {"no-auto-merge": "1", "no-
auto-split": "1", "no-auto-migrate": "1"}, "member_node_names": ["node-
***-1-001", "node-***-1-003", "node-***-1-002"]}}}}
           rep_group_job_state: Created
                   create_time: 2024-11-28 15:25:59
                   finish_time: 2024-11-28 15:25:59
            job_raft_log_index: 0
job_vanished_rg_raft_log_index: 0
                job_commit_ts: 29071203519955022
                      job_msg:
                     job_desc: {"mc_host":"mc-***-1-
1","reason":"","reported_by":"","job_detail":{"causeType":22}}
                      task_id: 0
                    task_type: TASK_TYPE_UNKNOWN
                   cause_type: CAUSE_MC_INTERNAL_TRIGGER
                schedule_rule: ""
1 row in set (0.02 sec)
```

RECYCLE_BIN_INFO

最近更新时间：2025-11-18 10:10:23

功能

回收站信息。

字段说明

字段名	类型	描述
tindex_id	int unsigned	数据对象 ID。
recycle_name	varchar(64)	回收站内表名。
origin_schema	varchar(64)	原数据库名。
origin_table	varchar(64)	原数据表名。
drop_time	timestamp	删除表的时间。
purge_time	timestamp	清除表的时间（从回收站内清除）。

示例

```
tdsql3_sys_local@localhost [sys]> select * from recycle_bin_info\G
*************************** 1. row ***************************
    tindex_id: 10039
 recycle_name: bin$hn/tgpuuffyxynootid4jq==$1
origin_schema: test
 origin_table: test3
    drop_time: 2024-12-12 11:58:29
   purge_time: 2024-12-19 11:58:29
1 row in set (0.01 sec)
```

TDSQL_RAFT_LEADER_SWITCH_RES

最近更新时间：2025-11-18 10:10:23

功能

TDStore 和 MC 的 Raft 层 Leader 角色切换记录。

字段说明

字段名	类型	描述
ID	int	自增 ID。
module	ENUM("MC", "TDSTORE")	模块，目前提供 MC、TDStore 的切主信息。
idx	bigint unsigned	raft 节点标识。
switch_begin_time	bigint	切换开始时间（毫秒级时间戳）。
switch_end_time	bigint	切换结束时间（毫秒级时间戳）。
switch_used_time	bigint	切换耗时。
switch_reason	varchar(512)	切换原因。
switch_type	int	切换类型。
switch_status	int	切换状态。
src_host	VARCHAR(128)	原 Leader 节点地址。
dst_host	VARCHAR(128)	新 Leader 节点地址。
is_success	int	是否成功。
failure_reason	varchar(128)	失败原因。

示例

```
tdsql3_sys_local@localhost [sys]> select * from
tdsql_raft_leader_switch_res;
```

```
+-----+---------+-------+------------------+-----------------+---------
---------+---------------+-------------+---------------+---------------
------+----------------------+------------+----------------+
| ID  | module  | idx   | switch_begin_time | switch_end_time |
switch_used_time | switch_reason | switch_type | switch_status |
src_host              | dst_host              | is_success |
failure_reason |
+-----+---------+-------+------------------+-----------------+---------
---------+---------------+-------------+---------------+---------------
------+----------------------+------------+----------------+
| 202 | TDSTORE | 72757 |     1733195593005 |   1733195593022 |
17 |               |           0 |             0 |
| node-******-1-001 |          1 |                |
| 203 | MC      |     0 |     1733280390626 |   1733280388221 |
-2405 |               |           0 |             0 | mc-******-1-1
| mc-******-1-2     |          0 | NULL           |
| 401 | TDSTORE | 72757 |     1733195895171 |   1733195895181 |
10 |               |           1 |             0 | node-******-1-001
| node-******-1-002 |          1 |                |
+-----+---------+-------+------------------+-----------------+---------
---------+---------------+-------------+---------------+---------------
------+----------------------+------------+----------------+
3 rows in set (0.00 sec)
```

SQL 语句
ALTER INSTANCE TRANSFER LEADER

最近更新时间：2025-11-18 10:10:22

功能描述

`ALTER INSTANCE TRANSFER LEADER` 语句用于在实例中将指定复制组（Replication Group，RG）的领导节点（Leader Node）转移到另一个节点。该语句允许用户指定新的领导节点，并可选择强制转移领导权。

权限要求

`ALTER INSTANCE TRANSFER LEADER` 需要当前用户具备 `SUPER` 权限。

语法

```
ALTER INSTANCE TRANSFER LEADER {RG | REP_GROUP} rep_group_id TO [NODE]
new_leader_node [FORCE];
```

参数说明

参数	是否必选	说明
rep_group_id	必选	指定要转移领导节点的复制组 ID。
new_leader_node	必选	指定新的领导节点。
FORCE	可选	表示强制转移领导权。

示例

1. 查询 RG `1792` 的 `leader_node_name` 信息。

   如下所示， `leader_node_name` 为 `node-tdsql3-86ea1ffe-002` 。

```
tdsql>select * from INFORMATION_SCHEMA.META_CLUSTER_RGS where
rep_group_id=1792 \G
*************************** 1. row ***************************
                        rep_group_id: 1792
                    data_space_type: DATA_SPACE_TYPE_SYSTEM
```

```
                              rep_group_state: RG_STATE_L_WORKING
                                 meta_version: 36
                               member_version: 0
                            key_range_version: 36
                                       quorum: 3
                            member_node_names: [node-tdsql3-86ea1ffe-
001, node-tdsql3-86ea1ffe-002, node-tdsql3-86ea1ffe-003]
                             leader_node_name: node-tdsql3-86ea1ffe-
002
                     last_leader_report_time: 2024-08-20
11:30:07.277221
                                 create_time: 2024-08-19
14:56:47.692831
                          parent_rep_group_id: 0
            rep_group_stats_approximate_size: 9722004
            rep_group_stats_approximate_keys: 3826
             rep_group_log_info_current_term: 9
          rep_group_log_info_committed_index: 30704
  rep_group_log_info_consecutive_applied_index: 30704
        rep_group_log_info_last_snapshot_index: 30683
               rep_group_log_info_first_index: 2
                rep_group_log_info_last_index: 30704
                rep_group_log_info_disk_index: 30704
            rep_group_log_info_applied_index: 30704
rep_group_log_info_raft_log_sync_delay_seconds: 11
1 row in set (0.02 sec)
```

2. 将复制组 `1792` 的领导节点转移到 `node-tdsql3-86ea1ffe-003` 。

```
tdsql>ALTER INSTANCE TRANSFER LEADER RG 1792 TO 'node-tdsql3-86ea1ffe-
003';
Query OK, 0 rows affected (0.01 sec)
job_id: 17373
```

3. 再次查看 RG `1792` 的 `leader_node_name` 信息。

如下所示， 领导节点已转移到 `node-tdsql3-86ea1ffe-003` 。

```
tdsql>select * from INFORMATION_SCHEMA.META_CLUSTER_RGS where
rep_group_id=1792 \G
```

```
*************************** 1. row ***************************
                    rep_group_id: 1792
                data_space_type: DATA_SPACE_TYPE_SYSTEM
                rep_group_state: RG_STATE_L_WORKING
                   meta_version: 36
                 member_version: 0
               key_range_version: 36
                         quorum: 3
               member_node_names: [node-tdsql3-86ea1ffe-001, node-tdsql3-86ea1ffe-002, node-tdsql3-86ea1ffe-003]
                leader_node_name: node-tdsql3-86ea1ffe-003
          last_leader_report_time: 2024-08-20 11:30:46.123801
                     create_time: 2024-08-19 14:56:47.692831
                parent_rep_group_id: 0
    rep_group_stats_approximate_size: 9727537
    rep_group_stats_approximate_keys: 3837
     rep_group_log_info_current_term: 10
  rep_group_log_info_committed_index: 30719
rep_group_log_info_consecutive_applied_index: 30719
rep_group_log_info_last_snapshot_index: 30663
       rep_group_log_info_first_index: 2
        rep_group_log_info_last_index: 30719
        rep_group_log_info_disk_index: 30719
    rep_group_log_info_applied_index: 30719
rep_group_log_info_raft_log_sync_delay_seconds: 2
1 row in set (0.01 sec)
```

ALTER TABLE

最近更新时间：2025-11-18 10:10:22

功能描述

修改已存在的表的结构，例如修改绑定的分区亲和性策略等。

语法

```
alter_table_stmt:
  ALTER TABLE table_name USING PARTITION POLICY partition_policy_name
  | ALTER TABLE table_name DROP PARTITION POLICY_ [FORCE]
```

参数说明

参数	是否必选	说明
ALTER TABLE table_name USING PARTITION POLICY partition_policy_name	可选	修改表绑定的分区亲和性策略，其中 `table_name` 跟 `partition_policy_name` 需要显式指定。

ALTER TABLE table_name DROP PARTITION POLICY_ [FORCE]	可选	解绑表绑定的分区亲和性策略。
		• `table_name` 需显式指定。
		• `FORCE` 选项：
		  ○ 若指定，则执行成功后，当前表不再绑定任何亲和性策略。
		  ○ 若不指定，则会尝试解绑当前表已绑定的显式亲和性策略，再将其绑定可能的隐式亲和性策略。

示例

- 修改与解绑亲和性策略的操作。

```
# 创建分区亲和性策略pp1
tdsql [demo]> create partition policy pp1 partition by hash(int)
partitions 4;
Query OK, 0 rows affected (0.02 sec)
```

```
# 创建hash 4分区的表t
tdsql [demo]> create table t(id INT) partition by hash(id) partitions
4;
Query OK, 0 rows affected (0.90 sec)

# 将表t绑定到分区亲和性策略pp1
tdsql [demo]> alter table t using partition policy pp1;
Query OK, 0 rows affected (0.64 sec)

# 解绑表t的分区亲和性，完成后表t会被自动绑定到对应的隐式亲和性策略
tdsql [demo]> alter table t drop partition policy;
Query OK, 0 rows affected (0.66 sec)

# 彻底解绑表t的分区亲和性策略，完成后表t不绑定任何分区亲和性策略
MySQL [demo]> alter table t drop partition policy force;
Query OK, 0 rows affected (0.61 sec)
```

BATCH

最近更新时间：2025-11-18 10:10:22

功能描述

`BATCH` 语句将大事务拆分成若干个小事务，整个流程不保证事务特性，但拆分的小事务需要保证事务特性。目前 `BATCH` 语句支持 `DELETE` 。

注意事项

- 目前只支持单表删除，不能嵌套在多语句事务中， `batch_size` 不为0。
- 删除语句执行时需先找到表对应的 RG Leader 的 SQLEngine，再通过 mysql_client 连接上去执行 `DELETE` 语句，避免过多 RPC 带来的性能损耗。

语法

```
BATCH LIMIT batch_size  delete_stmt
```

参数说明

参数	是否必选	说明
LIMIT batch_size	必选	控制事务拆分的粒度，设置非事务批量删除每批的大小。
delete_stmt	必选	删除语句。

示例

如下所示，删除的数据为10000条， `batch_size` 为2000，则在删除数据时每2000条进行一次事务提交，即一个小事务包含2000条删除数据。

```
BATCH LIMIT 2000 DELETE FROM sbtest1 WHERE id > 1000
```

CREATE DATABASE

最近更新时间：2025-11-18 10:10:22

功能描述

该语句用于创建数据库，并可以指定数据库的默认属性（如数据库默认字符集，校验规则等）。

权限要求

执行 `CREATE DATABASE` 语句创建数据库需要当前用户拥有全局的 `CREATE` 权限。

语法

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] database_name
    [database_option] ...

database_option:
[DEFAULT] {CHARACTER SET | CHARSET} [=] charset_name
| [DEFAULT] COLLATE [=] collate_name
```

参数说明

参数	是否可选	描述
IF NOT EXISTS	可选	用于指示如果数据库已经存在，则不进行创建。创建数据库时，如果数据库存在且没有指定 IF NOT EXISTS，则会报错。
database_name	必选	指定待创建的数据库名称。
[DEFAULT] {CHARACTER SET | CHARSET} [=] charset_name	可选	设置数据库的字符集（charset）。
[DEFAULT] COLLATE [=] collate_name	可选	设置数据库的校对规则（collation）。

示例

```
tdsql [(none)]> CREATE DATABASE IF NOT EXISTS test1 DEFAULT CHARACTER
SET utf8;
```

```
Query OK, 1 row affected
```

CREATE PARTITION POLICY

最近更新时间：2025-12-24 12:03:42

功能描述

显式创建分区亲和性策略。亲和性可用于控制数据的放置方式以及数据之间的亲和关系使其在自动调度时遵守预设的策略。通过亲和性策略，用户可实现将不同表的分区绑定在同一物理节点，减少分布式开销的效果。

权限要求

无。

语法

```
CREATE PARTITION POLICY [IF not EXISTS] partition_policy_name
                        [partition_clause]
[opt_using_distribution_policy];

partition_clause:
     PARTITION BY HASH (INT) PARTITIONS partition_num
    | PARTITION BY KEY COLUMNS columns_num PARTITIONS partition_num

opt_using_distribution_policy:
     USING DISTRIBUTION POLICY distribution_policy_name
```

参数说明

参数	是否必选	说明
partition_policy_name	必选	分区亲和性策略的名称。
partition_clause	可选	用于指定分区亲和性策略的结构。 • 如果不指定，则创建的是非分区结构的亲和性策略（用于普通表的绑定）。 • 如果指定，则创建具有分区结构的亲和性策略（用于分区表的绑定）。 其中 `partition_num` 指定分区数量，`columns_num` 指定分区列的数量，只有当分区表的属性与之相符时，才能绑定成功。

opt_using_distribution_policy	可选	用于指定亲和性策略是否绑定了分布式策略。请提前创建数据分布策略，详细请咨询技术支持工程师。

示例

- 创建非分区结构的亲和性策略。

```
tdsql [(none)]> CREATE PARTITION POLICY pp1;
Query OK, 0 rows affected (0.07 sec)


tdsql [test]> CREATE TABLE  tbl1(id INT) USING PARTITION policy pp1;
Query OK, 0 rows affected

tdsql [test]> CREATE TABLE  tbl2(id INT) USING PARTITION policy pp1;
Query OK, 0 rows affected
```

在上述策略下，tbl1 和 tbl2 会保持在同一个节点存放。

- 创建一级 hash 4分区结构的亲和性策略。

```
tdsql [(none)]> CREATE PARTITION POLICY pp2 PARTITION BY HASH(int)
PARTITIONS 4;
Query OK, 0 rows affected (0.02 sec)
```

- 创建非分区结构的亲和性策略（绑定了数据分布策略）。

```
tdsql [(none)]> CREATE PARTITION POLICY pp3 USING DISTRIBUTION POLICY
dp_1;
Query OK, 0 rows affected (0.01 sec)
```

- 创建一级 hash 4分区结构的亲和性策略（绑定了数据分布策略）。

```
tdsql [(none)]> CREATE PARTITION POLICY pp4
    -> PARTITION BY HASH(int)
    -> PARTITIONS 4
    -> USING DISTRIBUTION POLICY dp_2;
Query OK, 0 rows affected (0.02 sec)
```

- 创建一级 key 分区列数为2，分区数为4的亲和性策略。

```
tdsql [(none)]> CREATE PARTITION POLICY pp2
    -> PARTITION BY KEY COLUMNS 2
    -> PARTITIONS 4;
Query OK, 0 rows affected (0.02 sec)
```

- 创建绑定分区亲和性策略的表。

```
# 创建分区亲和性策略（hash 4分区）
tdsql [test]> CREATE PARTITION policy pp2 PARTITION BY HASH(INT)
partitions 4;
Query OK, 0 rows affected

# 建表
tdsql [test]> CREATE TABLE  orders(id INT) PARTITION BY HASH(id)
partitions 4 USING PARTITION policy pp2;
Query OK, 0 rows affected

tdsql [test]> CREATE TABLE  order_details(id INT, oid, detail TEXT)
PARTITION BY HASH(oid) partitions 4 USING PARTITION policy pp2;
Query OK, 0 rows affected
```

通过上述 SQL，表 orders 和 order_details 的 HASH 分区会进行亲和性绑定，使得 HASH 值一致的分区在放置以及调度迁移的时候保持在相同节点。

CREATE TABLE

最近更新时间：2025-11-18 10:10:22

功能描述

创建表。

语法

```
CREATE TABLE [IF NOT EXISTS] table_name
      (table_definition_list) [table_option_list] [partition_clause]
[opt_using_policy] ;

table_definition_list:
    table_definition [, table_definition ...]

table_definition:
      column_definition_list
    | [CONSTRAINT [constraint_name]] PRIMARY KEY index_desc
    | [CONSTRAINT [constraint_name]] UNIQUE {INDEX | KEY}
            [index_name] index_desc
    | [CONSTRAINT [constraint_name]] CHECK(expression) constranit_state

column_definition_list:
    column_definition [, column_definition ...]

column_definition:
     column_name data_type
         [DEFAULT const_value] [AUTO_INCREMENT]
         [NULL | NOT NULL] [[PRIMARY] KEY] [UNIQUE [KEY]] [COMMENT
string_value]

index_desc:
   (column_desc_list) [index_option_list]

index_option_list:
    index_option [ index_option ...]

index_option:
```

```
      KEY_BLOCK_SIZE [=] INT_VALUE
    | COMMENT string_value

table_option_list:
    table_option [ table_option ...]

table_option:
      [DEFAULT] {CHARSET | CHARACTER SET} [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
    | COMMENT string_value
    | ROW_FORMAT [=] REDUNDANT|COMPACT|DYNAMIC|COMPRESSED|DEFAULT
    | SYNC_LEVEL [=] 'NODE(MAJORITY)|NODE(ALL)'
    | DISTRIBUTION [=] 'NODE(DEFAULT)|NODE(ALL)'

partition_option:
      PARTITION BY HASH(expression)
      [subpartition_option] PARTITIONS partition_count
    | PARTITION BY KEY([column_name_list])
      [subpartition_option] PARTITIONS partition_count
    | PARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
      [subpartition_option] (range_partition_list)
    | PARTITION BY LIST {(expression) | COLUMNS (column_name_list)}
      [subpartition_option] PARTITIONS partition_count
    | PARTITION BY RANGE [COLUMNS]([column_name_list])
(range_partition_list)

subpartition_option:
      SUBPARTITION BY HASH(expression)
      SUBPARTITIONS subpartition_count
    | SUBPARTITION BY KEY(column_name_list)
      SUBPARTITIONS subpartition_count
    | SUBPARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
      (range_subpartition_list)
    | SUBPARTITION BY LIST(expression)

range_partition_list:
    range_partition [, range_partition ...]

range_partition:
    PARTITION partition_name
```

```
    VALUES LESS THAN {(expression_list) | MAXVALUE}

range_subpartition_list:
    range_subpartition [, range_subpartition ...]

range_subpartition:
    SUBPARTITION subpartition_name
    VALUES LESS THAN {(expression_list) | MAXVALUE}

expression_list:
    expression [, expression ...]

column_name_list:
    column_name [, column_name ...]

partition_name_list:
    partition_name [, partition_name ...]

partition_count | subpartition_count:
    INT_VALUE

opt_using_policy:
    USING DISTRIBUTION POLICY distribution_policy_name
   | USING PARTITION POLICY partition_policy_name
```

参数说明

参数	是否可选	描述
IF NOT EXISTS	可选	如果指定 `IF NOT EXISTS`，即使待创建的表已存在，也不会报错；如果不指定且待创建的表已存在，则系统会报错。
PRIMARY KEY	可选	为创建的表指定主键。如果不指定，则使用隐藏主键。
KEY | INDEX	可选	为创建的表指定键或索引。如果不指定索引名，则会使用索引引用的第一列作为索引名，如果命名存在重复，则会使用下划线（_）+ 序号的方式命名。（例如，使用 `c1` 列创建的索引如果命名重复，则会将索引命名为 `c1_2` 。）您可以通过 `SHOW INDEX` 语句查看表上的索引。

ROW_FORMAT	可选	指定表是否开启 Encoding 存储格式。 • redundant：不开启 Encoding 存储格式。 • compact：不开启 Encoding 存储格式。 • dynamic：Encoding 存储格式。 • compressed：Encoding 存储格式。 • default：等价 dynamic 模式。
KEY_BLOCK_SIZE	可选	指定索引块的大小（以字节为单位）。
CHARSET | CHARACTER SET	可选	指定表中列的默认字符集。
COLLATE	可选	指定表中列的默认字符序。
COMMENT	可选	注释。不区分大小写。
CHECK	可选	限制列中的值的范围。 • 如果对单个列定义 `CHECK` 约束，那么该列级约束可以写到列定义中，并且可以指定名称。 • 如果对一个表定义 `CHECK` 约束，那么此约束会应用于表中多个列，且允许出现在列的定义前。在删除表时，表中创建的 `CHECK` 约束也会一起被删除。 可以通过如下方式查看约束信息： • 使用 `SHOW CREATE TABLE` 命令 • 查看 `information_schema.TABLE_CONSTRAINTS` 视图 • 查看 `information_schema.CHECK_CONSTRAINTS` 视图
constraint_name	可选	约束名称，最多包含 64 个字符。 • 约束名称的开头结尾中间都允许有空格，但需要用"`"标识名称的开头和结尾。 • 约束名称可以包含特殊字符"$"。 • 如果约束名称为保留字，需要用"`"标识，否则会报错。 • 在同一 Database 下不允许 `CHECK` 约束的名称重复。
expression	可选	约束表达式。 • `expression` 不允许为空。 • `expression` 结果不能为非布尔类型。 • `expression` 不能包含不存在的列。
SYNC_LEVEL	可选	指定同步属性，取值如下： • `NODE(MAJORITY)`：默认值，表示该表为普通表。

		• `NODE(ALL)` ：表示该表为同步表，leader需要将修改强同步到所有副本（除日志副本） 目前同步表必须和广播表一起使用。
DISTRIBUTION	可选	指定广播属性，取值如下： • `NODE(DEFAULT)` ：默认值，表示该表为普通表。 • `NODE(ALL)` ：表示该表为同步表，会在系统中每个节点上都创建一个副本。 目前广播表必须和同步表一起使用。
distribution_policy_name	可选	指定需绑定的数据分布策略。请提前创建数据分布策略，详细请咨询技术支持工程师。 预置分布策略： • `inner_affinity` ：二级分区表的内部亲和性策略，绑定策略后不同一级分区下的对应二级分区会绑定在同一个 raft group 中，如 p0.sp0 与 p1.sp0 会放置在同一个 raft group 中。
partition_policy_name	可选	指定需绑定的分区亲和性策略 。请提前创建分区亲和性策略，详见 CREATE PARTITION POLICY 。

示例

- 创建数据库表。

```
tdsql [test]> CREATE TABLE tbl1 (c1 INT PRIMARY KEY, c2 VARCHAR(50));
Query OK, 0 rows affected
```

- 创建带索引的表。

```
tdsql [test]> CREATE TABLE tbl2 (c1 INT PRIMARY KEY, c2 INT, c3 INT,
INDEX i1 (c2));
Query OK, 0 rows affected
```

- 创建 Hash 分区，分区数为 `8` 的表。

```
tdsql [test]> CREATE TABLE tbl3 (c1 INT PRIMARY KEY, c2 INT) PARTITION
BY HASH(c1) PARTITIONS 8;
Query OK, 0 rows affected
```

- 创建一级分区为 Range 分区，二级分区为 Key 分区的表。

```
tdsql [test]> CREATE TABLE tbl4 (c1 INT, c2 INT, c3 INT) PARTITION BY
RANGE(c1)
            SUBPARTITION BY KEY(c2, c3) SUBPARTITIONS 5
            (PARTITION p0 VALUES LESS THAN(0), PARTITION p1 VALUES
LESS THAN(100));

Query OK, 0 rows affected
```

- **创建一列为 `gbk`，一列为 `utf8` 的表。**

```
tdsql [test]> CREATE TABLE tbl5 (
           c1 VARCHAR(10),
           c2 VARCHAR(10) CHARSET GBK COLLATE gbk_bin
       ) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
Query OK, 0 rows affected
```

- **使用自增列作为分区键。**

```
tdsql [test]> CREATE TABLE tbl6(inv_id BIGINT NOT NULL
AUTO_INCREMENT,c1 BIGINT,
       PRIMARY KEY (inv_id) ) PARTITION BY HASH(inv_id) PARTITIONS 8;
Query OK, 0 rows affected
```

- **创建带 `CHECK` 约束的表，并查看约束信息。**

```
tdsql [test]> CREATE TABLE tbl7 (col1 INT, col2 INT, col3 INT,
CONSTRAINT equal_check1 CHECK(col1 = col3 * 2));
Query OK, 0 rows affected

tdsql [test]> SHOW CREATE TABLE tbl7;
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
--------------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------
-----------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
--------------------------------+
| tbl10 | CREATE TABLE `tbl7` (
  `col1` int DEFAULT NULL,
  `col2` int DEFAULT NULL,
  `col3` int DEFAULT NULL,
  CONSTRAINT `equal_check1` CHECK ((`col1` = (`col3` * 2)))
) ENGINE=ROCKSDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
--------------------------------+
1 row in set
```

- **创建同步表。**

```
tdsql [test]> create table sync_table(c1 int primary key, c2 int)
sync_level = node(all) distribution = node(all);
Query OK, 0 rows affected
```

- **创建绑定分区亲和性策略的表。**

```
# 创建分区亲和性策略
tdsql [test]> create partition policy pp2 partition by hash(int)
partitions 4;
Query OK, 0 rows affected
# 建表
tdsql [test]> create table tbl8(id INT) partition by hash(id)
partitions 4 using partition policy pp2;
Query OK, 0 rows affected
```

DROP DATABASE

最近更新时间：2025-11-18 10:10:22

功能描述

删除数据库。

语法

```
drop_database_stmt:
DROP DATABASE [IF EXISTS] database_name;
```

参数说明

参数	是否可选	描述
IF EXISTS	可选	用于防止当数据库不存在时发生错误。
database_name	必选	指定待删除的数据库名。

示例

```
tdsql [(none)]> DROP DATABASE test1;
Query OK, 0 rows affected
```

DROP PARTITION POLICY

最近更新时间：2025-11-18 10:10:22

功能描述

删除分区亲和性策略。但是若有任何表已经绑定目标分区亲和性策略，则会删除失败。

权限要求

无。

语法

```
DROP PARTITION POLICY [IF EXISTS] partition_policy_name
```

参数说明

参数	是否必选	说明
partition_policy_name	必选	分区亲和性策略的名称。

示例

```
# 删除已存在的名为 pp1 的分区亲和性策略
tdsql [(none)]> drop partition policy pp1;
Query OK, 0 rows affected (0.01 sec)

# 删除不存在的分区亲和性策略
tdsql [(none)]> drop partition policy pp1;
ERROR 8596 (HY000): Unknown partition policy 'pp1'.

# 指定 IF EXISTS，删除不存在的分区亲和性策略
tdsql [(none)]> drop partition policy if exists  pp1;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

DROP TABLE

最近更新时间：2025-11-18 10:10:22

功能描述

删除表。

权限要求

具有该表 `Drop` 权限。

语法

```
DROP TABLE table_name [PURGE];
```

参数说明

参数	是否必选	说明
table_name	必选	表名。
PURGE	可选	直接删除表数据，不再转存入回收站库。

示例

```
TDSQL [(none)]> drop table test2 purge;
Query OK, 0 rows affected (0.70 sec)
```

FLASHBACK TABLE

最近更新时间：2025-11-18 10:10:22

功能描述

按照指定的方式恢复表。如果有同名表，会报错。

权限要求

具有该表 `Drop` 权限。

语法

```
FLASHBACK TABLE recycle_table_name TO BEFORE DROP [RENAME TO
new_table_name];
```

参数说明

参数	是否必选	说明
recycle_table_name	必选	表进入回收站后的别名，可通过 SHOW RECYCLEBIN 的 `recycle_table` 字段获取。
new_table_name	可选	表恢复时的新命名。

示例

```
TDSQL [(none)]> show recyclebin;
+-----------+---------------+--------------+----------------------------
----+---------------------+---------------------+
| tindex_id | origin_schema | origin_table | recycle_table
| drop_time           | purge_time          |
+-----------+---------------+--------------+----------------------------
----+---------------------+---------------------+
|     10028 | test          | test2        |
bin$/usodaiwngrpdoel2wnlqg==$1 | 2025-01-21 19:12:10 | 2025-01-28
19:12:10 |
+-----------+---------------+--------------+----------------------------
----+---------------------+---------------------+
```

```
TDSQL [(none)]> flashback table `bin$/usodaiwngrpdoel2wnlqg==$1` to
before drop rename to test.test3;
Query OK, 0 rows affected (1.72 sec)

TDSQL [(none)]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| test1          |
| test3          |
+----------------+
2 rows in set (0.01 sec)
```

OPTIMIZE TABLE

最近更新时间：2025-11-18 10:10:22

功能描述

该语句对指定表的数据范围执行 compaction。如果该语句在列存节点上执行，则触发的 compaction 会将指定表的数据转储至列存存储引擎。
该语句的执行不锁表，不阻塞读写请求。

权限要求

`SELECT` 和 `INSERT` 权限。

语法

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
```

参数说明

参数	是否必选	说明
NO_WRITE_TO_BINLOG	可选	保留关键字，无实际功能。
LOCAL	可选	保留关键字，无实际功能。

示例

```
tdsql> optimize table t1;
+---------+----------+----------+----------+
| Table   | Op       | Msg_type | Msg_text |
+---------+----------+----------+----------+
| test.t1 | optimize | status   | OK       |
+---------+----------+----------+----------+
1 row in set (0.01 sec)
```

PURGE RECYCLEBIN

最近更新时间：2025-11-18 10:10:22

功能描述

清理回收站内的某张表，需提供站内表名。
如果不指定站内表名，则表示清空整个回收站。

权限要求

具有该表 `Drop` 权限。

语法

```
PURGE RECYCLEBIN [recyele_table_name];
```

参数说明

参数	是否必选	说明
recyele_table_name	可选	• 指定要删除的表名。表名为表进入回收站后的别名，可通过 SHOW RECYCLEBIN 的 `recycle_table` 字段获取。 • 如果不指定站内表名，则表示清空整个回收站。

示例

```
TDSQL [(none)]> show recyclebin;
+-----------+---------------+--------------+----------------------------
----+---------------------+---------------------+
| tindex_id | origin_schema | origin_table | recycle_table
| drop_time           | purge_time          |
+-----------+---------------+--------------+----------------------------
----+---------------------+---------------------+
|     10028 | test          | test1        |
bin$tchgyzsnht3y0xtxlbitnq==$1 | 2025-01-21 19:12:10 | 2025-01-28
19:12:10 |
+-----------+---------------+--------------+----------------------------
----+---------------------+---------------------+
```

```
TDSQL[(none)]> purge recyclebin `bin$tchgyzsnht3y0xtxlbitnq==$1`;
Query OK, 0 rows affected (0.69 sec)
```

SHOW RECYCLEBIN

最近更新时间：2025-11-18 10:10:22

功能描述

`SHOW RECYCLEBIN` 显示回收站内表信息。

语法

```
SHOW RECYCLEBIN;
```

示例

```
TDSQL [(none)]> show recyclebin;
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
| tindex_id | origin_schema | origin_table | recycle_table
| drop_time           | purge_time          |
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
|     10028 | test          | test2        |
bin$/usodaiwngrpdoel2wnlqg==$1 | 2025-01-21 19:12:10 | 2025-01-28
19:12:10 |
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
```

以上返回结果中的字段描述如下：

- `tindex_id`：表唯一标识。
- `origin_schema`：原库名。
- `origin_table`：原表名。
- `recycle_table`：进入回收站后的别名。
- `drop_time`：删除时间。
- `purge_time`：清除时间（从回收站内清除）。

DDL 操作指引

最近更新时间：2025-11-18 10:10:22

操作前检查

1. 确认是否有足够的空间进行 DDL 操作。

   查看进行 DDL 操作的表当前占用的空间，并通过当前实例的磁盘使用率判断是否有足够的空间进行 DDL 操作。

```
SELECT
    table_name AS '表名',
    ROUND((data_length + index_length) / 1024 / 1024, 2) AS '总空间
(MB)',
    ROUND(data_length / 1024 / 1024, 2) AS '数据空间(MB)',
    ROUND(index_length / 1024 / 1024, 2) AS '索引空间(MB)',
    ROUND(data_free / 1024 / 1024, 2) AS '碎片空间(MB)'
FROM information_schema.TABLES
WHERE table_schema = '数据库名'
  AND table_name = '表名';
```

> **说明：**
> 部分 DDL 可以使用 instant DDL，不涉及空间占用，详细请参见 OnlineDDL 说明 。

2. 查看同名表是否有超长数据的慢查询。

```
SELECT * FROM information_schema.processlist WHERE INFO LIKE "%表名%"
ORDER BY TIME_MS DESC LIMIT 10;
```

   若有，则等慢查询结束后再执行 DDL。

3. 针对大单表，查看数据分布，确认 IO 压力。

```
SELECT
    SUM(region_stats_approximate_size) AS size,
    COUNT(b.rep_group_id) AS region_nums,
    sql_addr,
    c.leader_node_name,
    b.rep_group_id
```

```
FROM
    information_schema.META_CLUSTER_DATA_OBJECTS a
    JOIN information_schema.META_CLUSTER_REGIONS b
    JOIN information_schema.META_CLUSTER_RGS c
    JOIN information_schema.META_CLUSTER_NODES d
    ON a.data_obj_id = b.data_obj_id
    AND b.rep_group_id = c.rep_group_id
    AND c.leader_node_name = d.node_name
WHERE
    a.schema_name = '数据库名'
    AND a.table_name = '表名'
GROUP BY
    rep_group_id
ORDER BY
    leader_node_name;
```

如果数据全部倾斜在某一个节点，建议将 `max_parallel_ddl_degree` 从默认8设置为4或者2，减少 DDL 并发线程以降低 IO 压力。

4. 通过租户端监控指标确认 CPU/IO 负载。

   建议在业务低峰期进行 DDL 操作。

操作中查看进度

```
-- 查看DDL，是否在执行，LAST_TIMESTAMP 若在更新，则表明 DDL 在执行，如果长时间没有更新，则可能 hang 住
SELECT * FROM information_schema.ddl_jobs WHERE is_history = 0;

-- 查看进度则关注 information_Schema.ddl_jobs INFO 字段中的"progress"信息
              ID: 13
     SCHEMA_NAME: tdstore
      TABLE_NAME: sbtest1
         VERSION: 13
      DDL_STATUS: SUCCESS
START_TIMESTAMP: 2025-08-08 14:29:35
 LAST_TIMESTAMP: 2025-08-08 14:29:35
         DDL_SQL: alter table tdstore.sbtest1 add index idx(v)
       INFO_TYPE: ALTER TABLE
            INFO: {"tmp_tbl":{"db":"tdstore","table":"#sql-d_1000be_6895994e0000ad_1"},"alt_type":1,"alt_tid_upd":
```

```
{"tid_from":10039,"tid_to":10039},"cr_idx":
[{"id":10040,"ver":4,"stat":0,"tbl_type":1,"idx_type":2}],"rm_idx":
[],"init":false,"tmp_tab":false,"online_op":true,"wf_rmed":false,"online
_copy_stage":0,"idx_op":true,"row_applied":true,"row_apply_saved":true,"
current_schema_name":"tdstore","crt_data_obj_task_id":29437883249066288,
"dstr_data_obj_task_id":0,"alt_tbl_pp_stage":0,"alt_tbl_policy_option":0
,"data_obj_to_be_dstr_arr":[],"progress":"total: 1, scanned: 1
(100.00%)","fillback_mode":"ThomasWrite","exec_addr":
{"ip":"10.10.10.10","port":15035},"recov_addr":
{"ip":"10.10.10.10","port":15035}}
     IS_HISTORY: 1

--通过租户端观察 CPU/IO 负载以及慢查询告警;
```

`INFO` 字段中的 `progress` 即是进度说明： `"progress":"total: 1, scanned: 1 (100.00%)"`

查看任务结果

```
--返回empty， 则表明 DDL job 执行结束
SELECT * FROM information_schema.ddl_jobs WHERE is_history = 0;
```

PL 系统包 DBMS_ADMIN REMOVE_PERSIST_VARIABLE

最近更新时间：2025-11-18 10:10:22

`DBMS_ADMIN.REMOVE_PERSIST_VARIABLE` 用于删除持久化到 MC 侧的 variables。

语法

```
call dbms_admin.remove_persist_variable('variable_name');
```

参数说明

参数	说明
variable_name	需要删除的持久化到 MC 侧的变量名。

示例

```
call dbms_admin.remove_persist_variable('max_connections');
```

SCATTER_PARTITION

最近更新时间：2025-11-18 10:10:22

`DBMS_ADMIN.SCATTER_PARTITION` 用于将分区表按照默认建表策略打散到所有节点，合并到 Primary RG 中。

语法

```
call dbms_admin.scatter_partition('db_name','table_name');
```

参数说明

参数	说明
db_name	数据库名。
table_name	分区表名。

示例

```
call dbms_admin.scatter_partition('test','sbtest1');
```

SCATTER_SUBPARTITION

最近更新时间：2025-11-18 10:10:22

`DBMS_ADMIN.SCATTER_SUBPARTITION` 用于将某个一级分区下的所有二级分区按照默认建表策略打散到所有节点，合并到 Primary RG 中。

语法

```
call
dbms_admin.scatter_subpartition('db_name','table_name','partition_name')
;
```

参数说明

参数	说明
db_name	数据库名。
table_name	分区表名。
partition_name	一级分区名。

示例

```
call dbms_admin.scatter_subpartition('test','sbtest1','sbtest1.p0');
```

TDSTORE_FORCE_FULL_COMPACTION

最近更新时间：2025-11-18 10:10:22

`DBMS_ADMIN.TDSTORE_FORCE_FULL_COMPACTION()` 用于触发一次全量 compaction，即将所有数据 compact 到 l sm tree 的最下层，在列存节点执行还会将所有符合要求的数据都下沉到列存文件中。

语法

```
CALL DBMS_ADMIN.TDSTORE_FORCE_FULL_COMPACTION();
```

示例

```
tdsql3_sys_local@localhost [test_base]> CALL
DBMS_ADMIN.TDSTORE_FORCE_FULL_COMPACTION();
Query OK, 0 rows affected (0.48 sec)
```

错误码信息

最近更新时间：2025-12-24 12:03:43

TDSQL Boundless 错误码及说明如下：

> **说明：**
> 当您遇到如下错误码需要处理时，您可以通过 在线支持 进行处理。

1004-1100

错误码	错误描述	示例
1004	无法创建文件	无法创建文件 `test.txt`（错误号：2 - 路径不存在）
1005	无法创建表	无法创建表 `users`（错误号：13 - 权限不足）
1006	无法创建数据库	无法创建数据库 `my_db`（错误号：28 - 磁盘空间不足）
1007	无法创建数据库，数据库已存在	无法创建数据库 `my_db`；数据库已存在
1008	无法删除数据库，数据库不存在	无法删除数据库 `my_db`；数据库不存在
1009	删除数据库时出错	删除数据库时出错（无法删除 `my_db`，错误号：13 - 权限不足）
1010	删除数据库目录时出错	删除数据库时出错（无法删除目录 `/var/lib/mysql/my_db`，错误号：39 - 目录不为空）
1011	删除对象时出错	删除 `temp_table` 时出错（错误号：2 - 文件不存在）
1012	无法读取系统表中的记录	-
1013	无法获取对象状态	无法获取 `users.ibd` 的状态（错误号：2 - 文件不存在）
101	无法获取工作目录	无法获取工作目录（错误号：2 - 路径不存在）

4		
1015	无法锁定文件	无法锁定文件（错误号：11 – 资源暂时不可用）
1016	无法打开文件	无法打开文件 `query.log`（错误号：24 – 打开文件过多）
1017	找不到文件	找不到文件 `backup.sql`（错误号：2 – 文件不存在）
1018	无法读取目录	无法读取目录 `/var/lib/mysql`（错误号：13 – 权限不足）
1019	无法切换到目录	无法切换到目录 `/tmp`（错误号：2 – 目录不存在）
1020	记录自上次读取后已在表中更改	记录自上次读取后已在表 `orders` 中更改
1021	磁盘已满，等待释放空间	磁盘已满 `/var/lib/mysql`；等待释放空间...（错误号：28 – 磁盘空间不足）
1022	无法写入，表中存在重复键	无法写入；表 `users` 中存在重复键
1023	关闭对象时出错	关闭 `transaction_log` 时出错（错误号：9 – 文件描述符错误）
1024	读取文件时出错	读取文件 `config.cnf` 时出错（错误号：5 – 输入输出错误）
1025	重命名文件时出错	重命名 `temp.table` 为 `users.table` 时出错（错误号：17 – 文件已存在）
1026	写入文件时出错	写入文件 `binlog.0001` 时出错（错误号：28 – 磁盘空间不足）
1027	对象被锁定，无法修改	`orders` 被锁定，无法修改
1028	排序中止	–
1029	视图不存在于数据库中	视图 `user_summary` 不存在于 `sales_db`

1030	存储引擎返回错误	从存储引擎收到错误141 – 表损坏
1031	表的存储引擎不支持此选项	表 `logs` 的存储引擎不支持此选项
1032	在表中找不到记录	在 `products` 中找不到记录
1033	文件中的信息不正确	文件 `index.MYI` 中的信息不正确
1034	表的键文件不正确，尝试修复	表 `customers` 的键文件不正确；尝试修复它
1035	表的旧键文件需要修复	表 `orders` 的旧键文件；修复它！
1036	表是只读的	表 `archive_data` 是只读的
1037	内存不足，重启服务器并重试	内存不足；重启服务器并重试（需要104857600字节）
1038	排序内存不足，考虑增加排序缓冲区大小	–
1039	读取文件时发现意外 EOF	读取文件 `data.txt` 时发现意外 EOF（错误号：84 – 文件格式错误）
1040	连接过多	–
1041	内存不足，检查系统内存使用情况	–
1042	无法获取地址的主机名	–
1043	握手失败	–
1044	用户拒绝访问数据库	用户 `john@localhost` 拒绝访问数据库 `hr_db`
1045	用户拒绝访问	用户 `jane@192.168.1.100` 拒绝访问（使用密码：是）

1046	未选择数据库	–
1047	未知命令	–
1048	列不能为空	列 `email` 不能为空
1049	未知数据库	未知数据库 `non_existent_db`
1050	表已存在	表 `products` 已存在
1051	未知表	未知表 `old_table`
1052	列在表中不明确	列 `id` 在 `orders` 中不明确
1053	服务器正在关闭	–
1054	表中存在未知列	在 `users` 中未知列 `phone_number`
1055	列不在 GROUP BY 中	`address` 不在 GROUP BY 中
1056	不能在列上进行分组	不能在 `salary` 上进行分组
1057	语句中同时有聚合函数和列	–
1058	列计数与值计数不匹配	–
1059	标识符名称太长	标识符名称 `very_long_table_name_here` 太长
1060	重复的列名	重复的列名 `username`
1061	重复的键名	重复的键名 `idx_user`

1062	键有重复条目	键1有重复条目 `john.doe@example.com`
1063	列说明符不正确	列 `age` 的列说明符不正确
1064	SQL 语法错误	SQL 语法错误在 `SELECT * FROM` 附近，第1行
1065	查询为空	–
1066	不唯一的表或别名	不唯一的表/别名 `t1`
1067	默认值无效	`created_at` 的默认值无效
1068	定义了多个主键	–
1069	指定了太多键	指定了太多键；最多允许64个键
1070	指定了太多键部分	指定了太多键部分；最多允许16个部分
1071	指定的键太长	指定的键太长；最大键长度为3072字节
1072	键列在表中不存在	键列 `department` 在表中不存在
1073	BLOB 列不能在使用表类型的键规范中使用	BLOB 列 `photo` 不能在使用表类型的键规范中使用
1074	列的长度太大，改用 BLOB 或 TEXT	列 `description` 的长度太大（最大值 = 65535）；改用 BLOB 或 TEXT
1075	表定义不正确，只能有一个自动列	–
1076	服务器准备接受连接	mysqld：准备接受连接。套接字 `/tmp/mysql.sock` 端口：3306
1077	服务器正常关闭	mysqld：正常关闭

1078	服务器收到信号，正在中止	mysqld：收到信号15。正在中止！
1079	服务器关闭完成	mysqld：关闭完成
1080	服务器强制关闭线程	mysqld：强制关闭线程12345用户 `admin`
1081	无法创建 IP 套接字	–
1082	表没有类似 `CREATE INDEX` 中使用的索引	表 `old_table` 没有类似 `CREATE INDEX` 中使用的索引；重新创建表
1083	字段分隔符参数不是预期值	–
1084	不能将固定行长度与 BLOB 一起使用	–
1085	文件必须在数据库目录中或对所有用户可读	文件 `import.csv` 必须在数据库目录中或对所有用户可读
1086	文件已存在	文件 `backup.sql` 已存在
1087	记录处理统计信息	记录: 1000 已删除: 500 已跳过: 50 警告: 10
1088	重复记录统计信息	记录: 1000 重复: 100
1089	前缀键不正确	–
1090	不能使用 `ALTER TABLE` 删除所有列	–
1091	无法 DROP 对象，检查列或键是否存在	无法 DROP `index_name`；检查列/键是否存在
1092	记录处理统计信息	记录: 1000 重复: 100 警告: 5
1093	不能在 FROM 子句中为更新指定目标表	不能在 FROM 子句中为更新指定目标表 `employees`

1094	未知线程 ID	未知线程 ID: 12345
1095	您不是线程的所有者	您不是线程 12345 的所有者
1096	未使用表	–
1097	列和 SET 的字符串太多	列 id 和 SET 的字符串太多
1098	无法生成唯一的日志文件名	无法生成唯一的日志文件名 `mysql-bin`.(1-999)
1099	表用 READ 锁锁定，无法更新	表 `orders` 用 READ 锁锁定，无法更新
1100	表未用 `LOCK TABLES` 锁定	表 `products` 未用 `LOCK TABLES` 锁定

1101-1200

错误码	错误描述	示例
1101	BLOB、TEXT、GEOMETRY 或 JSON 列不能有默认值	BLOB、TEXT、GEOMETRY 或 JSON 列 `content` 不能有默认值
1102	数据库名称不正确	数据库名称 `my-database` 不正确
1103	表名称不正确	表名称 `user-table` 不正确
1104	SELECT 将检查超过 MAX_JOIN_SIZE 行	–
1105	未知错误	–
1106	未知过程	未知过程 `calculate_stats`
11	过程的参数计数不正确	过程 `get_user` 的参数计数不正确

07		
1108	过程的参数不正确	过程 `update_record` 的参数不正确
1109	在数据库中未知表	在 `sales_db` 中未知表 `old_customers`
1110	列被指定两次	列 `username` 被指定两次
1111	组函数的无效使用	–
1112	表使用了此 MySQL 版本中不存在的扩展	表 `archive` 使用了此 MySQL 版本中不存在的扩展
1113	表必须至少有1列	–
1114	表已满	表 `logs` 已满
1115	未知字符集	未知字符集 `utf8mb4_unicode`
1116	表太多；MySQL 在连接中只能使用有限个表	表太多；MySQL 在连接中只能使用64个表
1117	列太多	–
1118	行大小太大	行大小太大。所用表类型的最大行大小为65535
1119	线程堆栈溢出	线程堆栈溢出：已使用8192的1024堆栈
1120	在 OUTER JOIN 中发现交叉依赖	–
1121	表处理器不支持给定索引中的 NULL	请将列 `nullable_field` 更改为 NOT NULL
1122	无法加载函数	无法加载函数 `custom_hash`

1123	无法初始化函数	无法初始化函数 `validate_data`；缺少依赖项
1124	不允许共享库的路径	–
1125	函数已存在	函数 `calculate_total` 已存在
1126	无法打开共享库	无法打开共享库 `libmysqludf.so`（错误号：2 – 文件不存在）
1127	在库中找不到符号	在库中找不到符号 `mysql_query`
1128	函数未定义	函数 `generate_report` 未定义
1129	主机因多次连接错误被阻止	主机 `192.168.1.100` 因多次连接错误被阻止
1130	主机不允许连接到此 MySQL 服务器	主机 `10.0.0.50` 不允许连接到此 MySQL 服务器
1131	您正在使用 MySQL 作为匿名用户	–
1132	您必须具有更新 MySQL 数据库中表的权限	–
1133	在用户表中找不到任何匹配的行	–
1134	匹配的行统计信息	匹配的行: 1000 已更改: 500 警告: 10
1135	无法创建新线程	无法创建新线程（错误号 11）；系统资源限制
1136	列计数与值计数不匹配	列计数与行 5 的值计数不匹配
1137	无法重新打开表	无法重新打开表 `temp_data`
1138	NULL 值的无效使用	–

1139	从正则表达式收到错误	从正则表达式收到错误 `[a-z]+`
1140	在没有 GROUP BY 子句的情况下混合 GROUP 列	–
1141	用户没有定义这样的授权	用户 `john` 在主机 `localhost` 上没有定义这样的授权
1142	用户在表上无权执行命令	用户 `jane@192.168.1.100` 对表 `sensitive_data` 拒绝 SELECT 命令
1143	用户在表列上无权执行命令	用户 `admin@%` 对表 `users` 中的列 `password` 拒绝 UPDATE 命令
1144	非法的 GRANT/REVOKE 命令	–
1145	GRANT 的主机或用户参数太长	–
1146	表不存在	表 `hr_db.employees` 不存在
1147	用户对表没有定义这样的授权	用户 `guest` 在主机 `%` 上对表 `config` 没有定义这样的授权
1148	使用的命令在此 MySQL 版本中不允许	–
1149	SQL 语法错误	–
1150	延迟插入线程无法获取表的请求锁	延迟插入线程无法获取表 `orders` 的请求锁
1151	使用的延迟线程太多	–
1152	中止的连接	中止的连接12345到数据库： `shop_db` 用户： `cashier` (连接超时)
1153	收到大于 max_allowed_packet 字节的数据包	–
11	从连接管道读取时出错	–

54		
1155	从 `fcntl()` 收到错误	–
1156	数据包顺序错误	–
1157	无法解压通信数据包	–
1158	读取通信数据包时出错	–
1159	读取通信数据包超时	–
1160	写入通信数据包时出错	–
1161	写入通信数据包超时	–
1162	结果字符串长于 max_allowed_packet 字节	–
1163	使用的表类型不支持 BLOB/TEXT 列	–
1164	使用的表类型不支持 AUTO_INCREMENT 列	–
1165	INSERT DELAYED 不能与表一起使用	INSERT DELAYED 不能与表 `locked_table` 一起使用
1166	列名不正确	列名 `select` 不正确
1167	使用的存储引擎无法索引列	使用的存储引擎无法索引列 `large_text`
1168	无法打开不同定义、非 MyISAM 类型或不存在的底层表	–
1169	由于唯一约束，无法写入表	由于唯一约束，无法写入表 `users`

1170	在键规范中使用的 BLOB/TEXT 列没有键长度	在键规范中使用的 BLOB/TEXT 列 `description` 没有键长度
1171	PRIMARY KEY 的所有部分必须为 NOT NULL	–
1172	结果包含多行	–
1173	此表类型需要主键	–
1174	此版本的 MySQL 未编译支持 RAID	–
1175	您正在使用安全更新模式	–
1176	键在表中不存在	键 `idx_email` 在表 `customers` 中不存在
1177	无法打开表	–
1178	表的存储引擎不支持操作	表 `archive` 的存储引擎不支持事务
1179	不允许在事务中执行此命令	–
1180	在 COMMIT 期间收到错误	在 COMMIT 期间收到错误1213 – 死锁检测
1181	在 ROLLBACK 期间收到错误	在 ROLLBACK 期间收到错误1205 – 锁等待超时
1182	在 FLUSH_LOGS 期间收到错误	在 FLUSH_LOGS 期间收到错误28
1183	在 CHECKPOINT 期间收到错误	在 CHECKPOINT 期间收到错误13
1184	中止的连接	中止的连接54321到数据库： `backup_db` 用户： `backup_user` 主机： `10.0.1.100` (连接重置)

1185	表的存储引擎不支持二进制表转储	–
1186	二进制日志已关闭，无法 RESET MASTER	–
1187	重建转储表的索引失败	重建转储表 `old_data` 的索引失败
1188	来自主服务器的错误	来自主服务器的错误：`Duplicate entry`
1189	从主服务器读取时网络错误	–
1190	写入主服务器时网络错误	–
1191	找不到匹配列列表的 FULLTEXT 索引	–
1192	无法执行给定命令，因为有活动的锁定表或活动事务	–
1193	未知系统变量	未知系统变量 `query_cache`
1194	表标记为崩溃，应修复	表 `customer_data` 标记为崩溃，应修复
1195	表标记为崩溃，上次修复失败	表 `order_history` 标记为崩溃，上次修复失败
1196	一些非事务性更改的表无法回滚	–
1197	多语句事务需要超过 max_binlog_cache_size 字节的存储	–
1198	无法在运行从服务器时执行此操作	–
1199	此操作需要运行从服务器	–
1200	服务器未配置为从服务器	–

1201-1300

错误码	错误描述	示例
1201	无法初始化主服务器信息结构；更多错误消息可在 MySQL 错误日志中找到	–
1202	无法创建从服务器线程；检查系统资源	–
1203	用户已有超过 max_user_connections 活动连接	用户 `web_user` 已经有超过100活动连接
1204	只能对 SET 使用常量表达式	–
1205	锁等待超时；尝试重新启动事务	–
1206	锁总数超过锁表大小	–
1207	在 READ UNCOMMITTED 事务期间无法获取更新锁	–
1208	线程持有全局读锁时不允许 DROP DATABASE	–
1209	线程持有全局读锁时不允许 CREATE DATABASE	–
1210	参数不正确	`SET GLOBAL` 的参数不正确
1211	用户不允许创建新用户	`'app_user'@'localhost'` 不允许创建新用户
1212	表定义不正确；所有 MERGE 表必须在同一数据库中	–
1213	尝试获取锁时发现死锁；尝试重新启动事务	–
1214	使用的表类型不支持 FULLTEXT 索引	–
1215	无法添加外键约束	–
1216	无法添加或更新子行：外键约束失败	–
1217	无法删除或更新父行：外键约束失败	–
1218	连接到主服务器时出错	连接到主服务器时出错：连接被拒绝
1219	在主服务器上运行查询时出错	在主服务器上运行查询时出错：表不存在

1220	执行命令时出错	执行命令 `DROP TABLE` 时出错：权限不足
1221	错误用法	`SET` 和 `SELECT` 的错误用法
1222	使用的 SELECT 语句具有不同数量的列	–
1223	无法执行查询，因为您有冲突的读锁	–
1224	禁止混合事务性和非事务性表	–
1225	选项在语句中使用了两次	选项 `ENGINE` 在语句中使用了两次
1226	用户已超过资源限制	用户 `api_user` 已超过 `max_connections` 资源（当前值：500）
1227	拒绝访问；您需要权限才能执行此操作	拒绝访问；您需要 `SUPER` 权限才能执行此操作
1228	变量是 SESSION 变量，不能与 SET GLOBAL 一起使用	变量 `wait_timeout` 是 SESSION 变量，不能与 SET GLOBAL 一起使用
1229	变量是 GLOBAL 变量，应使用 SET GLOBAL 设置	变量 `innodb_buffer_pool_size` 是 GLOBAL 变量，应使用 SET GLOBAL 设置
1230	变量没有默认值	变量 `sql_mode` 没有默认值
1231	变量不能设置为值	变量 `max_connections` 不能设置为 `unlimited` 的值
1232	变量的参数类型不正确	变量 `port` 的参数类型不正确
1233	变量只能设置，不能读取	变量 `last_insert_id` 只能设置，不能读取
1234	错误用法/放置	`LIMIT` 的错误用法/放置
1235	此版本的 MySQL 尚不支持该功能	此版本的 MySQL 尚不支持 `JSON_TABLE`
1236	从二进制日志读取数据时从主服务器收到致命错误	从二进制日志读取数据时从主服务器收到致命错误 1236：`日志文件损坏`
1237	从服务器 SQL 线程因 replicate-*-table 规则忽略查询	–
1238	变量是某种类型，所以不让操作	变量 `system_time_zone` 是只读变量

1239	外键定义不正确	`orders.user_id` 的外键定义不正确：引用列不存在
1240	键引用和表引用不匹配	–
1241	操作数应包含指定数量的列	操作数应包含2列
1242	子查询返回超过1行	–
1243	输入的预处理语句没有找到	给 `EXECUTE` 的未知预处理语句处理程序（stmt_1）
1244	帮助数据库损坏或不存在	–
1245	子查询上的循环引用	–
1246	将列从一种类型转换为另一种类型	将列 `price` 从 `DECIMAL` 转换为 `VARCHAR`
1247	不支持该引用	不支持引用 `VIEW`（`v_user_summary`）
1248	每个派生表必须有自己的别名	–
1249	选择在优化期间被减少	选择2在优化期间被减少
1250	来自 SELECT 的表不能在操作中使用	来自 SELECT 之一的表 `temp` 不能在 `JOIN` 中使用
1251	客户端不支持服务器请求的身份验证协议	–
1252	SPATIAL 索引的所有部分必须为 NOT NULL	–
1253	排序规则对指定字符集无效	排序规则 `utf8_general_ci` 对字符集 `latin1` 无效
1254	从服务器已在运行	–
1255	从服务器已停止	–
1256	未压缩数据大小太大	未压缩数据大小太大；最大大小为16777216
1257	ZLIB：内存不足	–
1258	ZLIB：输出缓冲区空间不足	–
1259	ZLIB：输入数据损坏	–

1260	行被 GROUP_CONCAT() 截断	行5被 GROUP_CONCAT() 截断
1261	行不包含所有列的数据	行10不包含所有列的数据
1262	行被截断；它包含的数据比输入列多	行15被截断；它包含的数据比输入列多
1263	列设置为默认值；NOT NULL 列提供了 NULL	列设置为默认值；第3行的 NOT NULL 列 `email` 提供了 NULL
1264	指定行和列的值超出范围	第7行列 `age` 的值超出范围
1265	指定行和列的数据被截断	第12行列 `name` 的数据被截断
1266	对表使用存储引擎	对表 `logs` 使用存储引擎 `MyISAM`
1267	操作排序规则混合非法	操作 `JOIN` 的排序规则混合非法（utf8, utf8_general_ci）和（latin1, latin1_swedish_ci）
1268	无法删除一个或多个请求的用户	–
1269	无法撤销一个或多个请求用户的所有权限	–
1270	操作排序规则混合非法	操作 `UNION` 的排序规则混合非法（utf8, utf8_general_ci）、（latin1, latin1_swedish_ci）、（utf8mb4, utf8mb4_unicode_ci）
1271	操作排序规则混合非法	操作 `COMPARE` 的排序规则混合非法
1272	变量不是变量组件	变量 `performance_schema.max_digest_length` 不是变量组件
1273	未知排序规则	未知排序规则：`utf8_custom_ci`
1274	CHANGE MASTER 中的 SSL 参数被忽略	–
1275	服务器以 secure-auth 模式运行，但用户有旧格式密码	服务器以 secure-auth 模式运行，但 `'old_user'@'localhost'` 有旧格式密码
1276	字段或引用在 SELECT 中被解析	SELECT #1的字段或引用 `users.id` 在 SELECT #2 中被解析
1277	START SLAVE UNTIL 的参数或参数组合不正确	–

1278	建议在使用 START SLAVE UNTIL 时使用 skip-replica-start	-
1279	SQL 线程未启动，因此忽略 UNTIL 选项	-
1280	索引名称不正确	索引名称 `my-index` 不正确
1281	目录名称不正确	目录名称 `data-dir` 不正确
1282	查询缓存设置大小失败	查询缓存设置大小 16777216 失败；新查询缓存大小为 134217728
1283	列不能是 FULLTEXT 索引的一部分	列 `binary_data` 不能是 FULLTEXT 索引的一部分
1284	未知键缓存	未知键缓存 `custom_cache`
1285	MySQL 以 skip-name-resolve 模式启动	-
1286	未知存储引擎	未知存储引擎 `CustomEngine`
1287	功能已弃用，将在未来版本中移除	`PASSWORD()` 已弃用，将在未来版本中移除。请改用其他方法
1288	目标表不可更新	`UPDATE` 的目标表 `view_users` 不可更新
1289	功能已禁用；需要构建时带有特定选项	`COMPRESS()` 功能已禁用；您需要构建时带有 `ZLIB` 的 MySQL 才能使其工作
1290	MySQL 服务器以选项运行，因此无法执行此语句	MySQL 服务器以 `read_only` 选项运行，因此无法执行此语句
1291	列中有重复值	列 `username` 在 `users` 中有重复值 `john_doe`
1292	截断的不正确值	截断的不正确 `DATE` 值：`2023-02-30`
1293	表定义不正确；只能有一个带有 CURRENT_TIMESTAMP 的 TIMESTAMP 列	-
1294	列的 ON UPDATE 子句无效	`created_at` 列的 ON UPDATE 子句无效
1295	此命令在预处理语句协议中尚不支持	-

1296	从源收到错误	从 `master` 收到错误1062 `Duplicate entry`
1297	从源收到临时错误	从 `slave` 收到临时错误1213 `Deadlock found`
1298	未知或不正确的时区	未知或不正确的时区：`UTC+8`
1299	指定行和列的 TIMESTAMP 值无效	第25行列 `last_login` 的 TIMESTAMP 值无效
1300	无效的字符串	无效的 `UUID` 字符串：`invalid-uuid-string`

1301–1400

错误码	错误描述	示例
1301	函数结果大于 max_allowed_packet – 被截断	`CONCAT()` 的结果大于 max_allowed_packet（16777216）– 被截断
1302	冲突的声明	冲突的声明：`VARCHAR(255)` 和 `TEXT`
1303	不能从另一个存储例程内部创建存储过程	–
1304	对象已存在	存储过程 `calculate_salary` 已存在
1305	对象不存在	函数 `get_user_count` 不存在
1306	无法 DROP 对象	无法 DROP 触发器 `before_insert_user`
1307	无法 CREATE 对象	无法 CREATE 视图 `user_summary`
1308	没有匹配的标签	`LOOP` 没有匹配的标签：`exit_loop`
13	重新定义标签	重新定义标签 `start_label`

09		
13 10	结束标签没有匹配	结束标签 `end_loop` 没有匹配
13 11	引用未初始化的变量	引用未初始化的变量 `counter`
13 12	过程不能在给定上下文中返回结果集	过程 `get_users` 不能在给定上下文中返回结果集
13 13	RETURN 仅允许在 FUNCTION 中	–
13 14	语句不允许在存储过程中使用	`CREATE TABLE` 不允许在存储过程中使用
13 15	更新日志已弃用，由二进制日志替代	–
13 16	更新日志已弃用，SET SQL_LOG_UPDATE 已转换为 SET SQL_LOG_BIN	–
13 17	查询执行被中断	–
13 18	参数数量不正确	函数 `validate_user` 的参数数量不正确；期望 2，收到 3
13 19	未定义的 CONDITION	未定义的 CONDITION：`duplicate_key`
13 20	在 FUNCTION 中未找到 RETURN	在 FUNCTION `calculate_total` 中未找到 RETURN
13 21	FUNCTION 结束而没有 RETURN	FUNCTION `get_price` 结束而没有 RETURN
13 22	游标语句必须是 SELECT	–
13 23	游标 SELECT 不能有 INTO	–
13 24	未定义的 CURSOR	未定义的 CURSOR：`user_cursor`

1325	游标已打开	–
1326	游标未打开	–
1327	未声明的变量	未声明的变量：`temp_result`
1328	FETCH 变量数量不正确	–
1329	没有数据 – 获取、选择或处理了零行	–
1330	重复参数	重复参数：`user_id`
1331	重复变量	重复变量：`counter`
1332	重复条件	重复条件：`not_found`
1333	重复游标	重复游标：`data_cursor`
1334	无法 ALTER 对象	无法 ALTER 函数 `old_function`
1335	不支持子查询值	–
1336	语句不允许在存储函数或触发器中	`LOCK TABLES` 不允许在存储函数或触发器中
1337	游标或处理程序声明后的变量或条件声明	–
1338	处理程序声明后的游标声明	–
1339	CASE 语句的 CASE 未找到	–
1340	配置文件太大	配置文件 `my.cnf` 太大

1341	文件中的文件类型头格式错误	文件 `data.bin` 中的文件类型头格式错误
1342	解析注释时意外文件结束	解析注释 `# Configuration` 时意外文件结束
1343	解析参数时出错	解析参数 `port=3306` 时出错（行：`port=3306`）
1344	跳过未知参数时意外文件结束	跳过未知参数 `unknown_setting` 时意外文件结束
1345	无法发出 EXPLAIN/SHOW；缺少底层表的权限	–
1346	文件在其头中有未知类型	文件 `backup.dat` 在其头中有未知类型 `CUSTOM`
1347	对象不是指定类型	`'hr_db.users'` 不是 `VIEW`
1348	列不可更新	列 `user_id` 不可更新
1349	视图的 SELECT 在 FROM 子句中包含子查询	–
1350	视图的 SELECT 包含子句	视图的 SELECT 包含 `GROUP BY` 子句
1351	视图的 SELECT 包含变量或参数	–
1352	视图的 SELECT 引用临时表	视图的 SELECT 引用临时表 `temp_users`
1353	SELECT 列表和列名列表具有不同的列计数	–
1354	目前不能在此处使用视图合并算法	–
1355	正在更新的视图在其中没有底层表的完整键	–
1356	视图引用无效的表、列、函数或视图的定义者/调用者缺乏使用它们的权限	视图 `'sales_db.summary'` 引用无效的表、列、函数或视图的定义者/调用者缺乏使用它们的权限

1357	不能从另一个存储例程内部删除或更改存储过程	–
1358	GOTO 不允许在存储过程处理程序中使用	–
1359	触发器已存在	–
1360	触发器不存在	–
1361	触发器的目标表是视图或临时表	触发器的 `'user_logs'` 是视图或临时表
1362	不允许在触发器中更新行	不允许在 `BEFORE` 触发器中更新 `NEW` 行
1363	在触发器中不存在行	在 `AFTER` 触发器中不存在 `OLD` 行
1364	字段没有默认值	字段 `email` 没有默认值
1365	除以 0	–
1366	不正确的值对于列	不正确的 `INTEGER` 值: `'abc'` 对于列 `'age'` 在第 5 行
1367	解析期间发现非法的值	解析期间发现非法的 `DATE` `'2023-02-30'` 值
1368	不可更新视图上的 CHECK OPTION	不可更新视图 `'hr_db.employee_view'` 上的 CHECK OPTION
1369	CHECK OPTION 失败	CHECK OPTION 失败 `'sales_db.order_view'`
1370	用户对例程拒绝命令	用户 `'john'@'localhost'` 对例程 `'calculate_bonus'` 拒绝 `EXECUTE` 命令
1371	清除旧中继日志失败	清除旧中继日志失败: `权限不足`
1372	密码哈希应为指定位数十六进制数	密码哈希应为41位十六进制数

1373	在二进制日志索引中找不到目标日志	-
1374	读取日志索引文件时发生 I/O 错误	-
1375	服务器配置不允许清除二进制日志	-
1376	fseek() 失败	-
1377	日志清除期间发生致命错误	-
1378	可清除日志正在使用，不会清除	-
1379	日志清除期间发生未知错误	-
1380	初始化中继日志位置失败	初始化中继日志位置失败：文件损坏
1381	您未使用二进制日志	-
1382	语法保留给 MySQL 服务器内部使用	语法 `INTERNAL` 保留给 MySQL 服务器内部使用
1383	WSAStartup 失败	-
1384	尚不能处理具有不同组的程序	-
1385	使用此程序时选择必须有分组	-
1386	不能使用 ORDER 子句与此程序	-
1387	二进制日志和复制禁止更改全局服务器变量	二进制日志和复制禁止更改全局服务器 `sql_mode`
1388	无法映射文件	无法映射文件：`/var/lib/mysql/data.ibd`，错误号：2

1389	魔数错误	配置文件中的魔数错误
1390	预处理语句包含太多占位符	–
1391	键部分长度不能为 0	键部分 `username` 长度不能为0
1392	视图文本校验和失败	–
1393	不能通过连接视图修改多个基表	不能通过连接视图 `'db.user_order_view'` 修改多个基表
1394	不能在没有字段列表的情况下插入连接视图	不能在没有字段列表的情况下插入连接视图 `'db.user_details'`
1395	不能从连接视图删除数据	不能从连接视图 `'db.composite_view'` 删除数据
1396	操作失败	操作 `DROP` 失败，对于 `table users`
1397	XAER_NOTA: 未知 XID	–
1398	XAER_INVAL: 无效参数（或不支持的命令）	–
1399	XAER_RMFAIL: 当全局事务处于某种状态时无法执行命令	XAER_RMFAIL: 当全局事务处于 `PREPARED` 状态时无法执行命令
1400	XAER_OUTSIDE: 一些工作在全局事务外部完成	–

1401–1500

错误码	错误描述	示例
1401	XAER_RMERR: 事务分支中发生致命错误 – 检查数据一致性	–
1402	XA_RBROLLBACK: 事务分支已回滚	–

1403	用户在主机上对例程没有定义这样的授权	用户 `john` 在主机 `192.168.1.100` 上对例程 `calculate_bonus` 没有定义这样的授权
1404	授予 EXECUTE 和 ALTER ROUTINE 权限失败	–
1405	撤销对已删除例程的所有权限失败	–
1406	指定行和列的数据太长	第15行列 `description` 的数据太长
1407	错误的 SQL 状态	错误的 SQL 状态： `42S02`
1408	服务器准备接受连接	mysqld：准备接受连接。版本： `8.0.28` 套接字： `/tmp/mysql.sock` 端口：3306
1409	无法从固定大小行的文件加载值到变量	–
1410	您不允许使用 GRANT 创建用户	–
1411	函数的不正确类型值	函数 `validate_email` 的不正确 `BOOLEAN` 值： `invalid`
1412	表定义已更改，请重试事务	表定义已更改，请重试事务。表结构已修改
1413	在同一块中声明了重复的处理程序	–
1414	例程的 OUT 或 INOUT 参数不是变量或 BEFORE 触发器中的 NEW 伪变量	例程 `process_data` 的 OUT 或 INOUT 参数2不是变量或 BEFORE 触发器中的 NEW 伪变量
1415	不允许从函数返回结果集	不允许从 `FUNCTION` 返回结果集
1416	无法从发送到 GEOMETRY 字段的数据中获取几何对象	–
1417	例程失败且在其声明中没有 NO SQL 或 READS SQL DATA，并且启用了二进制日志记录	–

1418	此函数在其声明中没有 DETERMINISTIC、NO SQL 或 READS SQL DATA，并且启用了二进制日志记录	–
1419	您没有 SUPER 权限，并且启用了二进制日志记录	–
1420	您无法执行已有关联打开游标的预处理语句	–
1421	语句没有打开的游标	语句（12345）没有打开的游标
1422	在存储函数或触发器中不允许显式或隐式提交	–
1423	视图的底层表字段没有默认值	视图 `hr_db.employee_view` 的底层表字段没有默认值
1424	不允许递归存储函数和触发器	–
1425	为列指定的小数位数太大，超过了最大允许值	为列 `price` 指定的小数位数为10，最大允许为6
1426	为列指定的精度太大	为 `decimal_column` 指定的精度 100 太大。最大为65
1427	对于 float(M,D)、double(M,D) 或 decimal(M,D)，M 必须 >= D	对于 float(M,D)、double(M,D) 或 decimal(M,D)，M 必须 >= D（列 `coordinate`）
1428	您不能将系统表的写锁与其他表或锁类型结合使用	–
1429	无法连接到外部数据源	无法连接到外部数据源： `ODBC_Connection`
1430	处理外部数据源查询时出现问题	处理外部数据源查询时出现问题。数据源错误：连接超时
1431	您尝试引用的外部数据源不存在	您尝试引用的外部数据源不存在。数据源错误：无效数据源
1432	无法创建联合表。数据源连接字符串格式不正确	无法创建联合表。数据源连接字符串 `DSN=my_dsn` 格式不正确

1433	数据源连接字符串格式不正确	数据源连接字符串 `invalid_connection` 格式不正确
1434	无法创建联合表。外部数据源错误	无法创建联合表。外部数据源错误：权限不足
1435	触发器在错误的模式中	–
1436	线程堆栈溢出	线程堆栈溢出：使用了196608字节的262144字节堆栈，需要32768 字节
1437	存储过程、函数或触发器的定义体过长，超过了最大允许长度	例程 `complex_calculation` 的体太长
1438	无法删除默认键缓存	–
1439	列的显示宽度超出范围	列 `id` 的显示宽度超出范围（最大值 = 255）
1440	XAER_DUPID：XID 已存在	–
1441	日期时间函数字段溢出	日期时间函数： `DATE_ADD` 字段溢出
1442	无法在存储函数/触发器中更新表，因为它已被调用此存储函数/触发器的语句使用	无法在存储函数/触发器中更新表 `users`，因为它已被调用此存储函数/触发器的语句使用
1443	表的定义阻止了对表的操作	表 `archive` 的定义阻止了对表 `logs` 的操作 `DELETE`
1444	预处理语句包含引用同一语句的存储例程调用	–
1445	不允许在存储函数或触发器中设置自动提交	–
1446	定义者未完全限定	–
1447	视图没有定义者信息（旧表格式）	视图 `sales.report_view` 没有定义者信息（旧表格式）
1448	您需要 SUPER 权限才能使用定义者创建视图	您需要 SUPER 权限才能使用定义者 `admin@`

		`localhost` 创建视图
1449	指定为定义者的用户不存在	指定为定义者的用户（`old_user@192.168.1.100`）不存在
1450	不允许将模式从一种更改为另一种	不允许将模式从 `MyISAM` 更改为 `InnoDB`
1451	无法删除或更新父行：外键约束失败	无法删除或更新父行：外键约束失败（`约束名fk_user_id`）
1452	无法添加或更新子行：外键约束失败	无法添加或更新子行：外键约束失败（`约束名fk_order_user`）
1453	变量必须用反引号引用或重命名	变量 `select` 必须用 \`select\` 引用或重命名
1454	触发器没有定义者属性	触发器 `db.audit_trigger` 没有定义者属性
1455	对象有旧格式，您应该重新创建	`users` 有旧格式，您应该重新创建 `TABLE` 对象
1456	例程超出了递归限制	例程 `recursive_function` 超出了递归限制100
1457	加载例程失败。数据字典表缺失、损坏或包含错误数据	加载例程 `calculate_stats` 失败。数据字典表缺失、损坏或包含错误数据（内部代码 5）
1458	不正确的例程名称	不正确的例程名称 `invalid-name`
1459	表需要升级	表需要升级。请执行 `"REPAIR TABLE  customers"` 或转储/重新加载以修复它！
1460	存储函数不支持 AGGREGATE	–
1461	无法创建超过 max_prepared_stmt_count 个语句	无法创建超过 max_prepared_stmt_count 个语句（当前值：16384）
1462	视图包含视图递归	`db1.view1` 包含视图递归

1463	非分组字段在子句中使用	非分组字段 `salary` 在 `HAVING` 子句中使用
1464	使用的表类型不支持 SPATIAL 索引	–
1465	不能在系统表上创建触发器	–
1466	名称的前导空格被删除	名称 ` table` 的前导空格被删除
1467	无法从存储引擎读取自增值	–
1470	字符串对类型来说太长	字符串 `very_long_string_value` 对 `VARCHAR` 来说太长（不应超过 255）
1471	目标表不可插入	`UPDATE` 的目标表 `readonly_view` 不可插入
1472	表定义不同或是非 MyISAM 类型或不存在	表 `backup_data` 定义不同或是非 MyISAM 类型或不存在
1473	SELECT 的嵌套级别太高	–
1474	名称已变为空字符串	名称 `invalid_name` 已变为 ``
1475	FIELDS TERMINATED 字符串的第一个字符不明确	–
1476	您尝试创建的外部服务器已存在	您尝试创建的外部服务器 `external_db` 已存在
1477	您尝试引用的外部服务器名称不存在	您尝试引用的外部服务器名称不存在。数据源错误： 服务器未找到
1478	表存储引擎不支持创建选项	表存储引擎 `InnoDB` 不支持创建选项 `COMPRESSED`
1479	语法错误：PARTITIONING 要求为每个分区定义 VALUES	语法错误： `RANGE` PARTITIONING 要求为每个分区定义 VALUES `LESS THAN`

1480	只有特定 PARTITIONING 可以在分区定义中使用 VALUES	只有 `RANGE` PARTITIONING 可以在分区定义中使用 VALUES `LESS THAN`
1481	MAXVALUE 只能在最后的分区定义中使用	–
1482	子分区只能是哈希分区和键值	–
1483	如果在一个分区上定义了子分区，必须在所有分区上定义	–
1484	定义的分区数量错误，与之前的设置不匹配	–
1485	定义的子分区数量错误，与之前的设置不匹配	–
1486	分区函数中不允许常量、随机或时区相关的表达式	–
1487	RANGE/LIST VALUES 中的表达式必须是常量	–
1488	在分区函数字段列表中找不到字段	–
1489	字段列表仅允许在 KEY 分区中使用	–
1490	frm 文件中的分区信息与可以写入 frm 文件的内容不一致	–
1491	函数返回错误的类型	`PARTITION` 函数返回错误的类型
1492	对于分区，每个分区必须被定义	对于 `LIST` 分区，每个分区必须被定义
1493	每个分区的 VALUES LESS THAN 值必须严格递增	–
1494	VALUES 值必须与分区函数类型相同	–
1495	列表分区中相同常量的多重定义	–

1496	分区不能在查询中单独使用	–
1497	此版本的 MySQL 不允许在分区中混合处理器	–
1498	对于分区引擎，必须定义所有分区	对于分区引擎，必须定义所有 `PARTITIONS`
1499	定义了太多分区（包括子分区）	–
1500	只能将 RANGE/LIST 分区与 HASH/KEY 分区混合用于子分区	–

1501–1600

错误码	错误描述	示例
1501	创建特定处理器文件失败	–
1502	分区函数中不允许 BLOB 字段	–
1503	索引必须包括表分区函数中的所有列	`PRIMARY KEY` 必须包括表分区函数中的所有列
1504	分区数量为0不是允许的值	–
1505	无法在未分区的表上进行分区管理	–
1506	分区还不支持外键	–
1507	要操作的分区列表错误	要 `DROP` 的分区列表错误
1508	无法删除所有分区，请改用 DROP TABLE	–
1509	COALESCE PARTITION 只能用于 HASH/KEY 分区	–

1510	REORGANIZE PARTITION 只能用于重新组织分区，不能更改其数量	–
1511	没有参数的 REORGANIZE PARTITION 只能用于使用 HASH PARTITION 的自动分区表	–
1512	特定分区操作只能用于 RANGE/LIST 分区	`REBUI-LD` PARTITION 只能用于 RANGE/LIST 分区
1513	尝试添加具有错误子分区数量的分区	–
1514	必须至少添加一个分区	–
1515	必须至少合并一个分区	–
1516	要重新组织的分区比现有分区多	–
1517	重复的分区名称	重复的分区名称 `p2023`
1518	不允许在此命令上关闭二进制日志	–
1519	重新组织一组分区时，它们必须按顺序排列	–
1520	范围分区的重新组织不能更改总范围，除了最后一个分区可以扩展范围	–
1521	此版本的此处理器不支持分区函数	–
1522	无法从 CREATE/ALTER TABLE 定义分区状态	–
1523	处理器只支持 VALUES 中的 32 位整数	`HASH` 处理器只支持 VALUES 中的32整数
1524	插件未加载	插件 `audit_log` 未加载
152	不正确的值	不正确的 `INTEGER` 值： `abc`

5		
1526	表没有值对应的分区	表没有值 `2023-12-31` 的分区
1527	不允许多次指定选项	不允许多次指定 `ENGINE`
1528	创建对象失败	创建 `TABLE` 失败
1529	删除对象失败	删除 `INDEX` 失败
1530	处理器不支持表空间的自动扩展	–
1531	大小参数指定不正确，要么是数字，要么是 10M 的形式	–
1532	大小数字正确，但我们不允许数字部分超过 20 亿	–
1533	更改失败	更改失败：权限不足
1534	写入基于行的二进制日志中的一行失败	–
1535	主服务器和从服务器上的表定义不匹配	主服务器和从服务器上的表定义不匹配：列数不同
1536	使用 --log-replica-updates 运行的从服务器必须使用基于行的二进制日志记录	–
1537	事件已存在	事件 `daily_backup` 已存在
1538	存储事件失败。来自存储引擎的错误代码	存储事件 `cleanup` 失败。来自存储引擎的错误代码150
1539	未知事件	未知事件 `monthly_report`
1540	更改事件失败	更改事件 `data_export` 失败

1541	删除对象失败	删除 `EVENT` 失败
1542	INTERVAL 要么不是正数，要么太大	–
1543	ENDS 要么无效，要么在 STARTS 之前	–
1544	事件执行时间已过。事件已被禁用	–
1545	无法打开 `mysql.event`	–
1546	未提供日期时间表达式	–
1547	`mysql.events` 的列数错误	`mysql.events` 的列数错误。预期5，找到4。表可能已损坏
1548	无法从 `mysql.events` 加载。表可能已损坏	–
1549	从 `mysql.event` 中删除事件失败	–
1550	编译事件体期间出错	–
1551	相同的新旧事件名称	–
1552	列的数据太长	列 `description` 的数据太长
1553	无法删除索引：外键约束需要它	无法删除索引 `idx_user_id`：外键约束需要它
1554	语法已弃用，将在 MySQL 版本中移除	语法 `TYPE=MyISAM` 已弃用，将在 MySQL 8.0中移除。请改用 `ENGINE=MyISAM`
1555	您不能写锁定日志表。只允许读访问	–
1556	不能在日志表上使用锁	–

1557	维护表的外键约束会导致重复条目	维护表 `orders` 的外键约束，条目 `user_id=123`，键1会导致重复条目
1558	`mysql.events` 的列数错误。预期与实际不符	`mysql.events` 的列数错误。预期5，找到4。使用 MySQL 5.7创建，现在运行8.0。请执行 MySQL 升级过程
1559	当会话有打开的临时表时，无法切换到行式二进制日志格式	–
1560	无法在存储函数或触发器内更改二进制日志记录格式	–
1561	NDB 集群引擎尚不支持动态更改二进制日志格式	–
1562	无法创建带分区的临时表	–
1563	分区常量超出分区函数域	–
1564	不允许使用此分区函数	–
1565	DDL 日志错误	–
1566	不允许在 VALUES LESS THAN 中使用 NULL 值	–
1567	不正确的分区名称	–
1568	事务正在进行时无法更改事务特征	–
1569	ALTER TABLE 导致自动增量重新排序，导致重复条目	ALTER TABLE 导致自动增量重新排序，导致键 `PRIMARY` 的重复条目 `1001`
1570	内部调度程序错误	内部调度程序错误500
1571	启动/停止调度程序期间出错	启动/停止调度程序期间出错。错误代码13

1572	引擎不能在分区表中使用	–
1573	无法激活日志	无法激活 `general_log` 日志
1574	服务器构建时不支持基于行的复制	–
1575	解码 base64 字符串失败	–
1576	不允许递归创建事件	–
1577	无法继续，因为在服务器启动时发现事件调度程序使用的系统表已损坏	–
1578	这里只允许整数作为数字	–
1579	此存储引擎不能用于日志表	–
1580	如果启用了日志记录，则不能操作日志表	如果启用了日志记录，则不能 `DROP` 日志表
1581	无法重命名日志表。启用日志记录时，重命名到/从日志表必须重命名两个表	无法重命名 `general_log` 。启用日志记录时，重命名到/从日志表必须重命名两个表
1582	调用原生函数时参数个数不正确	调用原生函数 `SHA256` 时参数个数不正确
1583	调用原生函数时参数不正确	调用原生函数 `AES_ENCRYPT` 时参数不正确
1584	调用存储函数时参数不正确	调用存储函数 `validate_data` 时参数不正确
1585	此函数与原生函数同名	此函数 `SHA1` 与原生函数同名
1586	键的重复条目	键 `uk_email` 的重复条目 `john@example.com`
1587	打开的文件太多，请再次执行命令	–

1588	事件执行时间已过，并且设置了 ON COMPLETION NOT PRESERVE。事件创建后立即被删除	-
1589	事件执行时间已过，并且设置了 ON COMPLETION NOT PRESERVE。事件未更改。请指定未来的时间	-
1590	主服务器上发生了事件	主服务器上发生了事件 `backup_event` 。消息：表不存在
1591	表对一些现有值没有分区	-
1592	使用语句格式写入二进制日志的不安全语句	使用语句格式写入二进制日志的不安全语句，因为 BINLOG_FORMAT = STATEMENT。 `INSERT ... SELECT`
1593	致命错误	致命错误：内存分配失败
1594	中继日志读取失败	中继日志读取失败：文件损坏
1595	中继日志写入失败	中继日志写入失败：磁盘空间不足
1596	创建对象失败	创建 `USER` 失败
1597	主服务器命令失败	主服务器命令 `DROP DATABASE` 失败： 数据库不存在
1598	无法进行二进制日志记录	无法进行二进制日志记录。消息： 权限不足
1599	视图没有创建上下文	视图 `db1.view1` 没有创建上下文
1600	视图的创建上下文无效	视图 `db2.view2` 的创建上下文无效

1601-1700

错误码	错误描述	示例

1601	存储例程的创建上下文无效	存储例程 `db1.procedure1` 的创建上下文无效
1602	表的 TRG 文件已损坏	表 `db2.table1` 的 TRG 文件已损坏
1603	表的触发器没有创建上下文	表 `db3.users` 的触发器没有创建上下文
1604	表的触发器创建上下文无效	表 `db4.orders` 的触发器创建上下文无效
1605	事件的创建上下文无效	事件 `db5.backup_event` 的创建上下文无效
1606	无法为触发器打开表	无法为触发器 `db6.audit_trigger` 打开表
1607	无法创建存储例程。检查警告	无法创建存储例程 `calculate_stats` 。检查警告
1608	模糊的从服务器模式组合	模糊的从服务器模式组合。 `混合复制模式`
1609	指定类型的 BINLOG 语句前面缺少格式描述 BINLOG 语句	类型为 ROW 的 BINLOG 语句前面没有格式描述 BINLOG 语句 详细解释
1610	检测到损坏的复制事件	–
1611	LOAD DATA 中的无效列引用	LOAD DATA 中的无效列引用（ `@variable` ）
1612	未找到正在清除的日志	未找到正在清除的日志 `binlog.000015`
1613	XA_RBTIMEOUT：事务分支已回滚：耗时太长	–
1614	XA_RBDEADLOCK：事务分支已回滚：检测到死锁	–
1615	预处理语句需要重新准备	–
161	表不支持 DELAYED 选项	表 `archive_data` 不支持 DELAYED 选项

6		
1617	主服务器信息结构不存在	–
1618	选项被忽略	`<skip-slave-start>` 选项被忽略
1619	内置插件无法删除	–
1620	插件正忙，将在关闭时卸载	–
1621	变量是只读的。使用 SET 赋值	`GLOBAL` 变量 `innodb_buffer_pool_size` 是只读的。使用 SET GLOBAL 赋值
1622	存储引擎不支持此语句的回滚	存储引擎 `MyISAM` 不支持此语句的回滚
1623	意外的主服务器心跳数据	意外的主服务器心跳数据： 无效格式
1624	请求的心跳周期值要么为负，要么超过允许的最大值	请求的心跳周期值要么为负，要么超过允许的最大值（3600 秒）
1625	`mysql.ndb_replication` 表的模式错误	`mysql.ndb_replication` 表的模式错误。消息： 列缺失
1626	解析冲突函数时出错	解析冲突函数时出错。消息：语法错误
1627	写入异常表失败	写入异常表失败。消息：权限不足
1628	表的注释太长	表 `logs` 的注释太长（最大 = 2048）
1629	字段的注释太长	字段 `description` 的注释太长（最大 = 1024）
1630	函数不存在	函数 `custom_function` 不存在
1637	活动并发事务太多	–

1638	不完全支持非 ASCII 分隔符参数	-
1639	调试同步点等待超时	-
1640	调试同步点达到命中限制	-
1641	重复的条件信息项	重复的条件信息项 `SQLSTATE`
1642	未处理的用户定义的警告条件	-
1643	未处理的用户定义的未找到的条件	-
1644	未处理的用户定义的异常条件	-
1645	处理程序未激活时的 RESIGNAL	-
1646	SIGNAL/RESIGNAL 只能使用 SQLSTATE 定义的条件	-
1647	条件项的数据被截断	条件项 `MESSAGE_TEXT` 的数据被截断
1648	条件项的数据太长	条件项 `MESSAGE_TEXT` 的数据太长
1649	未知区域设置	未知区域设置： `zh_CN.GB18030`
1650	请求的服务器 ID 与从服务器启动选项冲突	请求的服务器 ID 100与从服务器启动选项 `--replicate-same-server-id` 冲突
1651	查询缓存已禁用；使用 query_cache_type=1 重启服务器以启用它	-
1652	重复的分区字段名称	重复的分区字段名称 `region`
165	分区列列表使用不一致	-

3		
1654	分区列值的类型不正确	–
1655	字段太多	`CREATE TABLE` 中的字段太多
1656	不能在 VALUES IN 中使用 MAXVALUE 作为值	–
1657	此类分区不能有多个值	`RANGE` 分区不能有多个值
1658	VALUES IN 中的行表达式仅允许多字段列分区	–
1659	字段对此类分区的类型不允许	字段 `description` 对此类分区的类型不允许
1660	分区字段的总长度太大	–
1661	无法执行语句：无法写入二进制日志，因为涉及既不能基于行又不能基于语句的引擎	–
1662	无法执行语句：无法写入二进制日志，因为 BINLOG_FORMAT = ROW 且至少有一个表使用限制于基于语句日志记录的存储引擎	–
1663	无法执行语句：无法写入二进制日志，因为语句不安全，存储引擎限制于基于语句日志记录，且 BINLOG_FORMAT = MIXED	无法执行语句：无法写入二进制日志，因为语句不安全，存储引擎限制于基于语句日志记录，且 BINLOG_FORMAT = MIXED。`INSERT ... SELECT`
1664	无法执行语句：无法写入二进制日志，因为语句采用行格式且至少有一个表使用限制于基于语句日志记录的存储引擎	–
1665	无法执行语句：无法写入二进制日志，因为 BINLOG_FORMAT = STATEMENT 且至少有一个表使用限制于基于行日志记录的存储引擎	–
1666	无法执行语句：无法写入二进制日志，因为语句采用行格式且 BINLOG_FORMAT	–

	= STATEMENT	
1667	无法执行语句：无法写入二进制日志，因为涉及多个引擎且至少一个引擎是内部日志系统	–
1668	语句不安全，因为它使用 LIMIT 子句	–
1669	语句不安全，因为它使用 INSERT DELAYED	–
1670	语句不安全，因为它使用通用日志、慢查询日志或 performance_schema 表	–
1671	语句不安全，因为它调用触发器或存储函数插入 AUTO_INCREMENT 列	–
1672	语句不安全，因为它使用可能在从服务器上返回不同值的 UDF	–
1673	语句不安全，因为它使用可能在从服务器上有不同值的系统变量	–
1674	语句不安全，因为它使用可能在从服务器上返回不同值的系统函数	–
1675	语句不安全，因为它在同一事务中访问事务表后访问非事务表	–
1676	语句类型和详细信息	`UNSAFE` 语句：`INSERT ... SELECT`
1677	表的列无法从一种类型转换为另一种类型	表 `db1.table1` 的列3无法从类型 `VARCHAR` 转换为类型 `INTEGER`
1678	无法为表创建转换表	无法为表 `db2.users` 创建转换表
1679	无法在事务内部修改会话级别的 binlog_format 设置	–
1680	为对象指定的路径太长	为 `DATABASE` 指定的路径太长
1681	功能已弃用，将在未来版本中移除	`OLD_PASSWORD()` 已弃用，将在未来版本中移除

1682	原生表结构错误	原生表 `mysql.user` 结构错误
1683	无效的 performance_schema 使用	–
1684	表被跳过，因为其定义正在被并发的 DDL 语句修改	表 `temp.temp_table` 被跳过，因为其定义正在被并发的 DDL 语句修改
1685	无法在事务内部修改会话级别的 `binlog_direct_non_transactional_updates` 设置	–
1686	无法在存储函数或触发器内部更改二进制日志直接标志	–
1687	SPATIAL 索引只能包含几何类型列	–
1688	索引的注释太长	索引 `idx_location` 的注释太长（最大 = 1024）
1689	等待锁因挂起的排他锁而中止	–
1690	值超出范围	`DECIMAL` 中的 `PRECISION` 值超出范围
1691	LIMIT 子句中的非整数类型变量	–
1692	在语句中混合自日志记录和非自日志记录引擎是不安全的	–
1693	语句访问非事务表以及事务表或临时表，并对其中任何一个进行写入	–
1694	无法在事务内部修改 `@@session.sql_log_bin`	–
1695	无法在存储函数或触发器内部更改 sql_log_bin	–
1696	从 .par 文件读取失败	–

1697	分区的 VALUES 值必须为 INT 类型	分区 `p2023` 的 VALUES 值必须为 INT 类型
1698	用户被拒绝访问	用户'test'@'127.0.0.1'被拒绝访问
1699	对通过插件进行身份验证的用户，SET PASSWORD 没有意义	_
1700	带有 IDENTIFIED WITH 的 GRANT 是非法的，因为用户已存在	带有 IDENTIFIED WITH 的 GRANT 是非法的，因为用户 `jane` 已存在

1701-1800

错误码	错误描述	示例
1701	无法截断在外键约束中引用的表	无法截断在外键约束中引用的表（ `orders.user_id_fk` ）
1702	插件是 `force_plus_permanent` ，无法卸载	插件 `audit_log` 是 `force_plus_permanent` ，无法卸载
1703	请求的心跳周期值小于 1 毫秒。该值重置为 0，意味着心跳将有效禁用	_
1704	请求的心跳周期值超过 `replica_net_timeout` 秒的值	请求的心跳周期值超过 `replica_net_timeout` 60秒的值
1705	多行语句需要超过 `max_binlog_stmt_cache_size` 字节的存储	多行语句需要超过 `max_binlog_stmt_cache_size` 16777216 字节的存储
1706	不允许主键/分区键更新，因为表通过别名同时作为多个表被更新	不允许主键/分区键更新，因为表同时作为 `users` 和 `profiles` 被更新
1707	需要重建表。请执行 `ALTER TABLE FORCE` 或转储/重新加载以修复它！	需要重建表。请执行 `ALTER TABLE customers FORCE` 或转储/重新加载以修复它！
1708	变量的值不应小于另一个变量的值	`max_connections` 的值不应小于 `max_user_connections` 的值
1709	索引列大小太大。最大列大小为指定的字节	索引列大小太大。最大列大小为767字节

1710	触发器在其体中有错误	触发器 `audit_trigger` 在其体中有错误：语法错误
1711	未知触发器在其体中有错误	未知触发器在其体中有错误： 未定义变量
1712	索引已损坏	索引 `idx_email` 已损坏
1713	撤销日志记录太大	–
1714	`INSERT IGNORE... SELECT` 不安全，因为 `SELECT` 检索行的顺序无法预测	–
1715	`INSERT... SELECT... ON DUPLICATE KEY UPDATE` 不安全，因为 `SELECT` 检索行的顺序无法预测	–
1716	`REPLACE... SELECT` 不安全，因为 `SELECT` 检索行的顺序无法预测	–
1717	`CREATE... IGNORE SELECT` 不安全，因为 `SELECT` 检索行的顺序无法预测	–
1718	`CREATE... REPLACE SELECT` 不安全，因为 `SELECT` 检索行的顺序无法预测	–
1719	`UPDATE IGNORE` 不安全，因为更新行的顺序无法预测	–
1720	插件标记为不能动态卸载。您必须停止服务器才能卸载它	插件 `keyring_file` 标记为不能动态卸载
1721	插件标记为不能动态安装。您必须停止服务器才能安装它	插件 `validate_password` 标记为不能动态安装
1722	从另一个表选择后在具有自增列的表中写入的语句不安全	–
1723	在具有自增列的表上执行 `CREATE TABLE... SELECT...` 不安全	–
1724	在具有多个 `UNIQUE KEY` 的表上执行 `INSERT... ON DUPLICATE KEY UPDATE` 不安	–

	全	
1725	表正被用于外键检查	–
1726	存储引擎不支持系统表	存储引擎 `MyISAM` 不支持系统表。[`mysql.user`]
1727	插入到非组合主键第一部分的自动增量字段中是不安全的	–
1728	无法从系统表中加载。表可能已损坏	无法从 `mysql.plugin` 中加载。表可能已损坏
1729	请求的主服务器延迟值超过最大值	请求的主服务器延迟值 86400 超过最大值 3600
1730	`BINLOG` 语句中只允许 `Format_description_log_event` 和行事件	`BINLOG` 语句中只允许 `Format_description_log_event` 和行事件（但提供了 `Query_log_event` ）
1731	分区和表之间的属性不匹配	分区和表之间的属性 `ENGINE` 不匹配
1732	要与分区交换的表已分区	要与分区交换的表已分区： `partitioned_table`
1733	要与分区交换的表是临时表	要与分区交换的表是临时表： `temp_data`
1734	子分区表，请使用子分区而不是分区	–
1735	表中的未知分区	表 `sales_data` 中的未知分区 `p2024`
1736	表具有不同的定义	–
1737	找到与分区不匹配的行	–
1738	选项 `binlog_cache_size` 大于 `max_binlog_cache_size`	选项 `binlog_cache_size` （33554432）大于 `max_binlog_cache_size` （16777216）

1739	由于字段上的类型或排序规则转换，无法在索引上使用访问方法	由于字段 `name` 上的类型或排序规则转换，无法在索引 `idx_name` 上使用 `RANGE` 访问
1740	要与分区交换的表有外键引用	要与分区交换的表有外键引用： `orders_fk`
1741	在表中未找到键值	在表 `db.users` 中未找到键值 `user123`
1742	列的数据太长	列 `content` 的数据太长
1743	从网络读取时复制事件校验和验证失败	–
1744	从日志文件读取时复制事件校验和验证失败	–
1745	选项 `binlog_stmt_cache_size` 大于 `max_binlog_stmt_cache_size`	选项 `binlog_stmt_cache_size`（33554432）大于 `max_binlog_stmt_cache_size`（16777216）
1746	创建对象时无法更新表	创建 `INDEX` 时无法更新表 `large_table`
1747	非分区表上的 `PARTITION()` 子句	–
1748	找到与给定分区集不匹配的行	–
1749	分区不存在	分区 `p2023_q4` 不存在
1750	更改复制存储库类型时失败	更改复制存储库类型时失败： `权限不足`
1751	无法回滚某些临时表的创建	–
1752	删除了一些临时表，但这些操作无法回滚	–
1753	操作在多线程从服务器模式下不支持	`LOAD DATA` 在多线程从服务器模式下不支持。 `需要串行执行`

1754	修改的数据库数量超过最大值；数据库名称将不包含在复制事件元数据中	修改的数据库数量超过最大值 10；数据库名称将不包含在复制事件元数据中
1755	无法在并行模式下执行当前事件组	无法在并行模式下执行当前事件组。遇到事件 `Query_log_event`，中继日志名称 `relay-bin.00002`，位置 12345
1756	并行回放错误	协调者遇到错误或者在调度 `relay-log log.00001` 位置 `0` 的event时被杀死
1757	分区表不支持 `FULLTEXT` 索引	–
1758	无效的条件编号	–
1759	在不使用 SSL/TLS 的情况下以明文发送密码极其不安全	–
1760	在主信息存储库中存储 MySQL 用户名或密码信息不安全	–
1761	表的外键约束将导致表中的重复条目	表 `orders` 的外键约束，记录 `user_id=123` 将导致表 `users` 中的重复条目，键 `PRIMARY`
1762	表的外键约束将导致子表中的重复条目	表 `parent_table` 的外键约束，记录 `id=456` 将导致子表中的重复条目
1763	仅启动从服务器 SQL 线程时无法设置身份验证选项	–
1764	表没有支持此查询的 `FULLTEXT` 索引	–
1765	系统变量不能在存储函数或触发器中设置	系统变量 `autocommit` 不能在存储函数或触发器中设置
1766	系统变量在有进行中的事务时不能设置	系统变量 `transaction_isolation` 在有进行中的事务时不能设置
1767	系统变量 `GTID_NEXT` 的值未列在 `GTID_NEXT_LIST` 中	`GTID_NEXT` 的值为 '123e4567-e89b-12d3-a456-426614174000:1'，但该值不在 `GTID_NEXT_LIST` 中

1768	系统变量 `GTID_NEXT` 不能在事务内部更改	–
1769	语句 `SET` 不能调用存储函数	语句 `SET @var = func()` 不能调用存储函数
1770	当 `GTID_NEXT_LIST` 非空时，`GTID_NEXT` 不能设置为 'AUTOMATIC'	–
1771	跳过事务，因为它已被执行和记录	跳过事务 `uuid:100-200`，因为它已被执行和记录
1772	格式错误的 `GTID` 集规范	格式错误的 `GTID` 集规范 `uuid:100-50`
1773	格式错误的 `GTID` 集编码	–
1774	格式错误的 `GTID` 规范	格式错误的 `GTID` 规范 `invalid_gtid`
1775	无法生成 `GTID`：整数部分达到最大值	–
1776	当 `MASTER_AUTO_POSITION` 激活时，不能设置参数 `MASTER_LOG_FILE`、`MASTER_LOG_POS` 等	–
1777	无法执行 `CHANGE MASTER TO MASTER_AUTO_POSITION = 1`，因为 `@@GLOBAL.GTID_MODE = OFF`	–
1778	不能在事务内部执行具有隐式提交的语句当 `@@SESSION.GTID_NEXT == 'UUID:NUMBER'` 时	–
1779	`GTID_MODE = ON` 要求 `ENFORCE_GTID_CONSISTENCY = ON`	–
1780	`@@GLOBAL.GTID_MODE = ON` 或 `ON_PERMISSIVE` 或 `OFF_PERMISSIVE` 要求 `--log-bin` 和 `--log-replica-updates`	–
1781	当 `@@GLOBAL.GTID_MODE = OFF` 时，`@@SESSION.GTID_NEXT` 不能设置为 `UUID:NUM`	–

	BER	
1782	当 @@GLOBAL.GTID_MODE = ON 时，@@SESSION.GTID_NEXT 不能设置为 ANONYMOUS	–
1783	当 @@GLOBAL.GTID_MODE = OFF 时，@@SESSION.GTID_NEXT_LIST 不能设置为非 NULL 值	–
1784	当 @@GLOBAL.GTID_MODE = OFF 时找到 Gtid_log_event	–
1785	语句违反 GTID 一致性：对非事务表的更新	–
1786	语句违反 GTID 一致性：CREATE TABLE ... SELECT	–
1787	语句违反 GTID 一致性：CREATE TEMPORARY TABLE 和 DROP TEMPORARY TABLE	–
1788	@@GLOBAL.GTID_MODE 的值只能一次更改一步	–
1789	无法复制，因为主服务器清除了所需的二进制日志	无法复制，因为主服务器清除了所需的二进制日志。从其他地方复制缺失的事务
1790	@@SESSION.GTID_NEXT 不能被拥有 GTID 的客户端更改	@@SESSION.GTID_NEXT 不能被拥有 GTID 的客户端更改。客户端拥有 uuid:100
1791	未知的 EXPLAIN 格式名称	未知的 EXPLAIN 格式名称：EXTENDED
1792	无法在只读事务中执行语句	–
1793	表分区的注释太长	表分区 p2023 的注释太长（最大 = 1024）
1794	从服务器未正确配置或初始化失败	–
1795	InnoDB 目前一次支持创建一个 FULLTEXT 索引	–

1796	无法在临时 `InnoDB` 表上创建 `FULLTEXT` 索引	–
1797	doc id列对 `InnoDB FULLTEXT` 索引的类型错误	列 `binary_data` 对 `InnoDB FULLTEXT` 索引的类型错误
1798	doc id索引对 `InnoDB FULLTEXT` 索引的类型错误	索引 `spatial_idx` 对 `InnoDB FULLTEXT` 索引的类型错误
1799	创建索引需要超过 `innodb_online_alter_log_max_size` 字节的修改日志	创建索引 `idx_content` 需要超过 `innodb_online_alter_log_max_size` 134217728 字节的修改日志
1800	未知的 `ALGORITHM`	未知的 `ALGORITHM` `CUSTOM`

1801–1886

错误码	错误描述	示例
1801	未知的 `LOCK` 类型	未知的 `LOCK` 类型 `CUSTOM_LOCK`
1802	当从服务器在 MTS 模式下因错误停止或被杀死时，无法执行 `CHANGE MASTER`	–
1803	在并行执行模式下，Slave 出错后无法恢复	–
1804	无法清理工作进程信息表	–
1805	表的列数错误	表 `mysql.user` 的列数错误。预期5列，发现4列
1806	Slave 必须静默重试当前事务	–
1807	表上正在运行外键检查。无法丢弃该表	表 `orders` 上正在运行外键检查。无法丢弃该表
18	模式不匹配	模式不匹配（字符集不一致）

08		
18 09	表位于系统表空间中	表 `mysql.innodb_table_stats` 位于系统表空间中
18 10	I/O 读取错误	I/O 读取错误：(2, 文件不存在) `/var/lib/mysql/data.ibd`
18 11	I/O 写入错误	I/O 写入错误：(28, 磁盘空间不足) `/var/lib/mysql/logfile`
18 12	表缺少表空间	表 `users` 缺少表空间
18 13	表空间已存在	表空间 `users_tablespace` 已存在
18 14	表的表空间已被丢弃	表 `archive_data` 的表空间已被丢弃
18 15	内部错误	内部错误： 内存分配失败
18 16	`ALTER TABLE IMPORT TABLESPACE` 失败	`ALTER TABLE users IMPORT TABLESPACE` 失败，错误号 150： 表空间损坏
18 17	索引损坏	索引损坏： `idx_name` 结构错误
18 18	无效的显示宽度。请改用 `YEAR` 类型	–
18 19	您的密码不满足当前策略要求	–
18 20	执行此语句前，您必须使用 `ALTER USER` 语句重置密码	–
18 21	添加外键约束失败。外键表中缺少约束的索引	添加外键约束失败。外键表 `orders` 中缺少约束 `fk_user_id` 的索引
18 22	添加外键约束失败。被引用表中缺少约束的索引	添加外键约束失败。被引用表 `users` 中缺少约束 `fk_user_id` 的索引
18 23	将外键约束添加到系统表失败	将外键约束 `sys_fk` 添加到系统表失败

1824	无法打开被引用的表	无法打开被引用的表 `referenced_table`
1825	添加外键约束失败。外键约束中的选项不正确	在表 `child_table` 上添加外键约束失败。外键约束 `parent_fk` 中的选项不正确
1826	重复的外键约束名称	重复的外键约束名称 `duplicate_fk`
1827	密码哈希不具有预期的格式	–
1828	无法删除列：外键约束需要此列	无法删除列 `user_id`：外键约束 `fk_user` 需要此列
1829	无法删除列：表的外键约束需要此列	无法删除列 `order_id`：表 `order_items` 的外键约束 `fk_order` 需要此列
1830	列不能为 `NOT NULL`：外键约束需要 `SET NULL`	列 `parent_id` 不能为 `NOT NULL`：外键约束 `fk_parent` 需要 `SET NULL`
1831	表上定义了重复索引	表 `db.users` 上定义了重复索引 `idx_email`
1832	无法更改列：外键约束使用了此列	无法更改列 `customer_id`：外键约束 `fk_customer` 使用了此列
1833	无法更改列：表的外键约束使用了此列	无法更改列 `product_id`：表 `order_details` 的外键约束 `fk_product` 使用了此列
1834	无法从作为表的外键约束父表的表中删除行	无法从作为表 `orders` 的外键约束 `fk_user` 父表的表中删除行
1835	格式错误的通信包	–
1836	以只读模式运行	–
1837	当 `@@SESSION.GTID_NEXT` 设置为 GTID 时，在 `COMMIT` 或 `ROLLBACK` 后必须显式将其设置为不同的值	–
18	存储过程中无法设置系统变量	存储过程中无法设置系统变量 `autocommit`

38		
1839	仅当 `@@GLOBAL.GTID_MODE = ON` 时才能设置 `@@GLOBAL.GTID_PURGED`	–
1840	仅当 `@@GLOBAL.GTID_EXECUTED` 为空时才能设置 `@@GLOBAL.GTID_PURGED`	–
1841	仅当没有进行中的事务时才能设置 `@@GLOBAL.GTID_PURGED`	–
1842	`@@GLOBAL.GTID_PURGED` 已更改	`@@GLOBAL.GTID_PURGED` 已从 `uuid:1-100` 更改为 `uuid:1-200`
1843	`@@GLOBAL.GTID_EXECUTED` 已更改	`@@GLOBAL.GTID_EXECUTED` 已从 `uuid:1-50` 更改为 `uuid:1-150`
1844	无法执行语句：由于 `BINLOG_FORMAT = STATEMENT`，并且同时写入了复制和非复制表，因此无法写入二进制日志	–
1845	此操作不支持。请尝试其他方法	此操作不支持 `ALTER TABLE`。请尝试 `CREATE TABLE`
1846	不支持。请尝试其他方法	不支持 `DROP DATABASE`。原因：数据库正在使用。请尝试 `STOP SLAVE`
1847	`COPY` 算法需要锁	–
1848	分区特定操作尚不支持 `LOCK/ALGORITHM`	–
1849	参与外键的列已被重命名	–
1850	无法原地更改列类型	–
1851	添加外键需要设置 `foreign_key_checks=OFF`	–

1852	使用 `IGNORE` 创建唯一索引需要 `COPY` 算法来移除重复行	–
1853	不允许在不添加新主键的情况下删除主键	–
1854	添加自增列需要锁	–
1855	不能用用户可见的列替换隐藏的 `FTS_DOC_ID`	–
1856	无法删除或重命名 `FTS_DOC_ID`	–
1857	全文索引创建需要锁	–
1858	当服务器以 `@@GLOBAL.GTID_MODE = ON` 运行时，不能设置 `sql_replica_skip_counter`	–
1859	键中包含重复值	键 `PRIMARY` 的重复条目 `1001`
1860	长数据库名和对象标识符导致路径长度超过限制	路径： `/var/lib/mysql/very_long_database_name_table_name`
1861	无法按要求在此 `SQL_MODE` 下静默转换 `NULL` 值	–
1862	您的密码已过期。要登录，您必须使用支持过期密码的客户端更改密码	–
1863	在错误的分区中发现一行数据	在错误的分区 `p2023` 中发现一行数据
1864	无法将事件调度给工作线程，因为其大小超过 `replica_pending_jobs_size_max`	无法将事件 `Query_event` 、中继日志名称 `relay-bin.00003` 、位置12345调度给工作线程，因为其大小104857600超过了 `replica_pending_jobs_size_max` 的 16777216
1865	`CREATE FULLTEXT INDEX WITH PARSER` 不能在 `InnoDB` 表上创建	–

1866	二进制日志文件逻辑上已损坏	二进制日志文件 `binlog.000015` 逻辑上已损坏：`事件顺序错误`
1867	文件未被清除，因为它正在被线程读取	文件 `binlog.000010` 未被清除，因为它正在被5个线程读取，仅清除了10个文件中的3个
1868	文件未被清除，因为它是活动的日志文件	文件 `binlog.000020` 未被清除，因为它是活动的日志文件
1869	`UPDATE` 中的自增值与内部生成的值冲突	–
1870	对于以行格式修改 `BLACKHOLE` 表的语句，不记录行事件	对于以行格式修改 `BLACKHOLE` 表的 `INSERT` 语句，不记录行事件。表：`test.blackhole_table`
1871	Slave 无法从存储库初始化主信息结构	–
1872	Slave 无法从存储库初始化中继日志信息结构	–
1873	尝试切换到用户时访问被拒绝	尝试切换到用户 `john`@`localhost` 时访问被拒绝（使用密码：是）
1874	`InnoDB` 处于只读模式	–
1875	`STOP SLAVE` 命令执行未完成：Slave SQL 线程已收到停止信号，线程正忙	–
1876	`STOP SLAVE` 命令执行未完成：Slave IO 线程已收到停止信号，线程正忙	–
1877	无法执行操作。表缺失、损坏或包含错误数据	无法执行操作。表 `mysql.plugin` 缺失、损坏或包含错误数据
1878	临时文件写入失败	–
1879	升级索引名称失败，请使用 `create index`（`alter table`）`algorithm copy` 重建索引	–

1880	旧格式的 `TIME/TIMESTAMP/DATETIME` 列已升级为新格式	-
1881	当 `innodb_force_recovery > 0` 时，不允许执行此操作	-
1882	提供给函数的初始化向量太短	提供给 `AES_DECRYPT` 的初始化向量太短。必须至少为 16 字节长
1883	现在无法卸载插件	现在无法卸载插件 `audit_log` 。插件正忙
1884	无法执行该语句，因为它需要作为多个语句写入二进制日志	-
1885	从服务器包含的GTID数量超过了主服务器（基于相同SERVER_UUID），这可能表明：二进制日志的末端被截断，或者最后一个二进制日志文件已丢失（例如在发生电源或磁盘故障且sync_binlog不等于1时）。主服务器可能已回滚了某些已复制到从服务器的事务，也可能没有。建议将主服务器已回滚但从服务器中仍存在的事务反向复制回主服务器，和/或在主服务器上提交空事务，以弥补那些已在主服务器提交但未包含在GTID_EXECUTED中的事务。	-
1886	表没有定义必要的键	表 `db.temp_table` 没有定义必要的键

3000-3100

错误码	错误描述	示例
3000	文件已损坏	文件 `/var/lib/mysql/data.ibd` 已损坏
3001	查询在主库上部分完成并被中止	查询在主库上部分完成（主库错误：1213）并被中止。查询： `UPDATE users SET status=1`

3002	查询在主库和从库上导致不同的错误	主库错误：消息= 死锁检测 错误代码=1213；从库错误：实际消息= 表不存在 ，错误代码=1146。默认数据库： test_db 。查询： DELETE FROM temp_table
3003	表的存储引擎未加载	表 db1.logs 的存储引擎 CustomEngine 未加载
3004	当错误处理程序未激活时不能使用 GET STACKED DIAGNOSTICS	–
3005	功能不再支持。该语句已转换	TYPE=MyISAM 不再受支持。该语句已转换为 ENGINE=MyISAM
3006	语句不安全，因为它使用了全文解析器插件	–
3007	无法丢弃/导入与临时表关联的表空间	–
3008	外键级联删除/更新超过最大深度	外键级联删除/更新超过最大深度15
3009	表的列数错误。创建时与当前版本不匹配	表 mysql.user 的列数错误。预期5列，发现4列。创建时 MySQL 版本为5.7，当前运行版本为8.0
3010	触发器没有 CREATED 属性	触发器 db1.schema1.audit_trigger 没有 CREATED 属性
3011	给定操作时间和事件类型的被引用触发器不存在	–
3012	EXPLAIN FOR CONNECTION 命令仅支持 SELECT/UPDATE/INSERT/DELETE/REPLACE 语句	–
3013	列的大小无效	列 description 的大小无效
3014	发现表存储引擎缺少必需的创建选项	发现表存储引擎 InnoDB 缺少必需的创建选项 COMPRESSED
3015	存储引擎内存不足	存储引擎 Memory 内存不足

3016	匿名用户的密码不能过期	–
3017	无法在 Slave SQL 线程运行时执行此操作	–
3018	无法在物化子查询上创建 `FULLTEXT` 索引	–
3019	撤销日志错误	撤销日志错误： `日志损坏`
3020	对数的参数无效	–
3021	无法在 Slave IO 线程运行时执行此操作	无法在 Slave IO 线程运行时执行此操作；请先运行 `STOP SLAVE IO_THREAD FOR CHANNEL 'channel1'`
3022	当从库存在临时表时，此操作可能不安全	–
3023	`CHANGE MASTER TO` 带有 `MASTER_LOG_FILE` 子句但没有 `MASTER_LOG_POS` 子句可能不安全	–
3024	查询执行被中断，超过最大语句执行时间	–
3025	`SELECT` 不是只读语句，禁用计时器	–
3026	重复条目	重复条目 `john@example.com`
3027	SQL 模式不再有任何效果	`NO_AUTO_CREATE_USER` 模式不再有任何效果。请改用 `STRICT_ALL_TABLES` 或 `STRICT_TRANS_TABLES`
3028	`ORDER BY` 的表达式包含聚合函数并应用于 `UNION`	`ORDER BY` 的表达式 #1 包含聚合函数并应用于 `UNION`
3029	`ORDER BY` 的表达式包含聚合函数并应用于非聚合查询的结果	`ORDER BY` 的表达式 #2 包含聚合函数并应用于非聚合查询的结果

3030	当启用 `replica-preserve-commit-order` 时，Slave 工作线程已停止	-
3031	不支持 `replica_preserve_commit_order`	不支持 `replica_preserve_commit_order` `ON`
3032	服务器当前处于离线模式	-
3033	二进制几何函数给了两个不同 SRID 的几何对象	二进制几何函数 `ST_Intersects` 给了两个不同 SRID 的几何对象：4326 和 3857
3034	使用不支持类型的参数调用几何函数	使用不支持类型的参数调用几何函数 `ST_Area`
3035	在几何函数中发生未知的 GIS 错误	在函数 `ST_Union` 中发生未知的 GIS 错误
3036	在 GIS 函数中捕获到未知异常	在函数 `ST_Buffer` 中捕获到未知异常
3037	提供给 GIS 函数的几何数据无效	提供给函数 `ST_IsValid` 的 GIS 数据无效
3038	在函数中，几何对象没有数据	在函数 `ST_Length` 中，几何对象没有数据
3039	无法计算质心，因为几何对象为空	无法计算质心，因为函数 `ST_Centroid` 中的几何对象为空
3040	几何叠加计算错误：几何数据无效	几何叠加计算错误：函数 `ST_Overlap` 中的几何数据无效
3041	几何转向信息计算错误：几何数据无效	几何转向信息计算错误：函数 `ST_IsSimple` 中的几何数据无效
3042	函数中的交点分析过程意外中断	函数 `ST_Intersection` 中的交点分析过程意外中断
3043	在函数中抛出未知异常	在函数 `ST_Difference` 中抛出未知异常
3044	函数中内存分配错误	内存分配错误：函数 `ST_Union` 中的 几何对象合并

3045	域错误	域错误：函数 `ST_ConvexHull` 中的 无效坐标
3046	长度错误	长度错误：函数 `ST_Length` 中的 线串太短
3047	参数无效错误	参数无效错误：函数 `ST_Buffer` 中的 负半径
3048	范围错误	范围错误：函数 `ST_Scale` 中的 缩放因子过大
3049	溢出错误	溢出错误：函数 `ST_Area` 中的 面积计算溢出
3050	范围错误	范围错误：函数 `ST_Distance` 中的 距离超出范围
3051	下溢错误	下溢错误：函数 `ST_Precision` 中的 精度下溢
3052	逻辑错误	逻辑错误：函数 `ST_Simplify` 中的 拓扑矛盾
3053	运行时错误	运行时错误：函数 `ST_Transform` 中的 坐标转换失败
3054	未知异常	未知异常：函数 `ST_GeomFromText` 中的 `WKT` 解析异常
3055	几何字节字符串必须为小端序	–
3056	为复制用户提供的密码超过最大长度 32 个字符	–
3057	用户级锁名称不正确	用户级锁名称 `my_lock` 不正确
3058	尝试获取用户级锁时发现死锁	–
3059	无法执行 `REPLACE`，因为它需要删除不在视图中的行	–
3060	不支持对具有 GIS 索引的表进行在线操作	–

3061	用户变量名非法	用户变量名 `@123var` 非法
3062	当 `GTID_MODE = OFF` 时，无法执行操作	当 `GTID_MODE = OFF` 时，无法 `SET GTID_NEXT`
3063	无法从复制从库线程中执行操作	无法从复制从库线程中 `CREATE USER`
3064	函数中参数的类型不正确	函数 `ST_Contains` 中参数 `geometry1` 的类型不正确
3065	`ORDER BY` 子句的表达式不在 `SELECT` 列表中，引用了不在 `SELECT` 列表中的列	`ORDER BY` 子句的表达式 #3 不在 `SELECT` 列表中，引用了不在 `SELECT` 列表中的列 `email`；这与 `only_full_group_by` 不兼容
3066	`ORDER BY` 子句的表达式不在 `SELECT` 列表中，包含聚合函数	`ORDER BY` 子句的表达式 #4 不在 `SELECT` 列表中，包含聚合函数；这与 `only_full_group_by` 不兼容
3067	提供的过滤器列表包含一个不符合要求的格式的值	提供的过滤器列表包含一个不符合要求的 `db_pattern.table_pattern` 格式的值
3068	`OK` 包太大	–
3069	提供给函数的 `JSON` 数据无效	提供给函数 `JSON_EXTRACT` 的 `JSON` 数据无效：`无效格式`
3070	提供给函数的 `GeoJSON` 数据无效：缺少必需的成员	提供给函数 `ST_GeomFromGeoJSON` 的 `GeoJSON` 数据无效：缺少必需的成员 `type`
3071	提供给函数的 `GeoJSON` 数据无效：成员必须是特定类型	提供给函数 `ST_GeomFromGeoJSON` 的 `GeoJSON` 数据无效：成员 `coordinates` 必须是 `array` 类型
3072	提供给函数的 `GeoJSON` 数据无效	提供给函数 `ST_GeomFromGeoJSON` 的 `GeoJSON` 数据无效
3073	函数中坐标维度数不支持	函数 `ST_GeomFromText` 中坐标维度数不受支持：发现 4，预期 2 或 3
3074	Slave 通道不存在	Slave 通道 `channel1` 不存在
3075	对于给定的主机和端口组合，Slave 通道已存在	对于给定的主机和端口组合，Slave 通道 `channel2` 已存在

3076	无法创建通道：通道名称无效或过长	-
3077	要拥有多个通道，存储库不能是 `FILE` 类型	-
3078	无法删除通道的 Slave 信息对象	无法删除通道 `channel3` 的 Slave 信息对象
3079	Slave 上存在多个通道。请提供通道名称作为参数	-
3080	超过允许的最大复制通道数	-
3081	无法在复制线程运行时执行此操作	无法在复制线程运行时执行此操作；请先运行 `STOP SLAVE FOR CHANNEL 'channel1'`
3082	此操作需要运行的复制线程	此操作需要运行的复制线程；请配置 Slave 并运行 `START SLAVE FOR CHANNEL 'channel1'`
3083	通道的复制线程已经在运行	通道 `channel1` 的复制线程已经在运行
3084	通道的复制线程已停止	通道 `channel2` 的复制线程已停止
3085	无法在 Slave SQL 线程运行时执行此操作	无法在 Slave SQL 线程运行时执行此操作；请先运行 `STOP SLAVE SQL_THREAD FOR CHANNEL 'channel1'`
3086	当 `sql_replica_skip_counter > 0` 时，不允许使用 `START SLAVE [SQL_THREAD]` 启动多个 SQL 线程	-
3087	`GROUP BY` 子句中不包含表达式，并且它包含非聚合列	`GROUP BY` 子句中不包含 `SELECT` 的表达式 #1，并且它包含非聚合列 `department`，该列在功能上不依赖于 `GROUP BY` 子句中的列；这与 `sql_mode=only_full_group_by` 不兼容
3088	在没有 `GROUP BY` 的聚合查询中，表达式包含非聚合列	在没有 `GROUP BY` 的聚合查询中，`SELECT` 的表达式 #2 包含非聚合列 `username`；这与 `sql_mode=only_full_group_by` 不兼容

3089	更新已弃用	更新 `mysql.user` 已弃用。未来版本中将使其变为只读
3090	更改 SQL 模式已弃用	更改 SQL 模式 `NO_ZERO_DATE` 已弃用。未来版本中将被移除
3091	`DROP DATABASE` 失败；数据库目录仍然存在	`DROP DATABASE` 失败；请按以下步骤修复：(1) 从数据库目录 `/var/lib/mysql/test_db` 中删除所有文件；(2) `SET GTID_NEXT='uuid:100'`；(3) `DROP DATABASE test_db`
3092	服务器未正确配置为组的活动成员	–
3093	`START GROUP_REPLICATION` 命令失败，因为组已在运行	–
3094	`START GROUP_REPLICATION` 命令失败，因为应用器模块启动失败	–
3095	`STOP GROUP_REPLICATION` 命令执行未完成：应用器线程在忙时收到了停止信号	–
3096	`START GROUP_REPLICATION` 命令失败，因为初始化组通信层时出错	–
3097	`START GROUP_REPLICATION` 命令失败，因为加入通信组时出错	–
3098	表不符合外部插件的要求	–
3099	不能在没有二进制日志格式为 `ROW` 的情况下更改变量的值	不能在没有二进制日志格式为 `ROW` 的情况下更改变量 `binlog_format` 的值
3100	运行复制钩子时，观察器出错	运行复制钩子 `before_commit` 时，观察器出错

3101-3200

错误码	错误描述	示例
3101	插件指示服务器回滚当前事务	–
3102	生成列的表达式包含不允许的函数	生成列 `full_name` 的表达式包含不允许的函数 `UUID()`
3103	虚拟列的 `INPLACE ADD` 或 `DROP` 不能与其他 `ALTER TABLE` 操作结合	–
3104	不能在生成列上定义带有子句的外键	不能在生成列上定义带有 `CASCADE` 子句的外键
3105	为表中的生成列指定的值不被允许	为表 `users` 中的生成列 `age_category` 指定的值不被允许
3106	生成列不支持特定功能或操作	生成列不支持 `AUTO_INCREMENT`
3107	生成列只能引用在其之前定义的生成列	–
3108	列有其他生成列依赖，不能直接修改或删除	列 `first_name` 有生成列依赖
3109	生成列不能引用自增列	生成列 `user_info` 不能引用自增列 `id`
3110	功能不可用；您需要移除功能或使用内置该功能的 MySQL	`compression` 功能不可用；您需要移除 `zlib` 或使用内置了 `zlib` 的 MySQL
3111	不允许设置全局 GTID_MODE 为特定值，因为有特定限制	不允许 `SET @@GLOBAL.GTID_MODE = ON`，因为存在违反GTID一致性的进行中的事务
3112	复制接收线程无法以 `AUTO_POSITION` 模式启动	复制接收线程 `IO_THREAD` 无法以 `AUTO_POSITION` 模式启动：此服务器使用 `@@GLOBAL.GTID_MODE = OFF`
3113	当 `AUTO_POSITION = 1` 时，无法复制匿名事务	当 `AUTO_POSITION = 1` 时，无法复制匿名事务，文件 `binlog.000015`，位置 123456
3114	当 `@@GLOBAL.GTID_MODE = ON` 时，无法复制匿名事务	当 `@@GLOBAL.GTID_MODE = ON` 时，无法复制匿名事务，文件 `binlog.000016`，位置

		234567
3115	当 `@@GLOBAL.GTID_MODE = OFF` 时，无法复制 GTID 事务	当 `@@GLOBAL.GTID_MODE = OFF` 时，无法复制 GTID 事务，文件 `binlog.000017`，位置 345678
3116	不能设置 `ENFORCE_GTID_CONSISTENCY = ON`，因为存在违反 GTID 一致性的进行中的事务	–
3117	存在违反 GTID 一致性的进行中的事务	–
3118	用户访问被拒绝。账户被锁定	用户 `john` @ `localhost` 访问被拒绝。账户被锁定
3119	表空间名称不正确	表空间名称 `invalid_tablespace` 不正确
3120	表空间非空	表空间 `users_tablespace` 非空
3121	文件名不正确	文件名 `../data.ibd` 不正确
3122	交点不一致	–
3123	优化器提示语法错误	–
3124	不支持的 `MAX_EXECUTION_TIME`	–
3125	`MAX_EXECUTION_TIME` 提示仅支持顶层的独立 `SELECT` 语句	–
3126	提示因冲突/重复而被忽略	提示 `INDEX_MERGE` 因冲突/重复而被忽略
3127	在提示中未找到查询块名称	在 `JOIN_ORDER` 提示中未找到查询块名称 `block1`
3128	在提示中无法解析名称	在 `INDEX_HINT` 提示中无法解析名称 `table_alias`

3129	请勿修改表。这是 MySQL 内部系统表	请勿修改 `gtid_executed` 表。这是 MySQL 内部系统表，用于存储已提交事务的 GTID
3130	可插拔协议不支持该命令	–
3131	锁服务锁名称不正确	锁服务锁名称 `my_lock` 不正确
3132	尝试获取锁定服务锁时发现死锁	–
3133	服务锁等待超时	–
3134	函数中参数超过了几何图形中的最大点数	函数 `ST_ConvexHull` 中参数 `geometry` 超过了几何图形中的最大点数 (1000)
3135	SQL 模式应与严格模式一起使用	`NO_ZERO_DATE` 、`NO_ZERO_IN_DATE` 和 `ERROR_FOR_DIVISION_BY_ZERO` SQL 模式应与严格模式一起使用
3136	版本令牌不匹配	`db1` 的版本令牌不匹配。正确值为 `v2.0`
3137	未找到版本令牌	未找到版本令牌 `token123`
3138	拥有 GTID 的客户端无法更改变量	拥有 GTID 的客户端无法更改变量 `autocommit` 。该客户端拥有 `uuid:100`
3139	无法在通道上执行操作	无法在通道 `channel1` 上执行 `START SLAVE` 操作
3140	JSON 文本无效	JSON 文本无效：在列 `user_data` 的值中位置 5 处的 `"invalid"`
3141	函数参数中的 JSON 文本无效	函数 `JSON_EXTRACT` 的参数 1 中的 JSON 文本无效：位置 10 处的 `"malformed"`
3142	JSON 二进制值包含无效数据	–
3143	JSON 路径表达式无效	JSON 路径表达式无效。错误在字符位置 5 附近。`$.user..name`

3144	无法从特定字符集的字符串创建 JSON 值	无法从字符集为 `latin1` 的字符串创建 JSON 值
3145	提供给函数的 JSON 字符数据无效	提供给函数 `JSON_OBJECT` 的 JSON 字符数据无效： `非UTF8字符` ；要求使用 `utf8`
3146	函数参数中的 JSON 数据类型无效	函数 `JSON_MERGE` 的参数 2 中的 JSON 数据类型无效；需要 JSON 字符串或 JSON 类型
3147	无法将值 `CAST` 为 JSON	–
3148	路径表达式必须使用 `UTF-8` 字符集编码	路径表达式 `$.name` 使用了字符集 `latin1` ，但要求必须是 `UTF-8`
3149	路径表达式不能包含 `*` 和 `**` 标记或数组范围	–
3150	JSON 值太大，无法存储在 JSON 列中	–
3151	JSON 对象包含过长的键名	–
3152	JSON 列仅支持通过指定 JSON 路径上的生成列进行索引	JSON 列 `user_profile` 仅支持通过指定 JSON 路径上的生成列进行索引
3153	在此上下文中不允许路径表达式 `$`	–
3154	JSON 函数的 oneOrAll 参数只能取 `one` 或 `all`	`JSON_SEARCH` 的 oneOrAll 参数只能取 `one` 或 `all`
3155	从指定行和列的 JSON 值转换为目标类型时超出范围	从第10行列 `age` 的 JSON 值转换为 INT 时超出范围
3156	从指定行和列的 JSON 值转换为目标类型时无效	从第15行列 `price` 的 JSON 值转换为 DECIMAL 时无效
3157	JSON 文档超过最大深度	–
3158	JSON 文档不能包含 `NULL` 成员名称	–

3159	当 `--require_secure_transport=ON` 时，禁止使用不安全传输的连接	–
3160	未配置安全传输（SSL 或共享内存），无法设置 `--require_secure_transport=ON`	–
3161	存储引擎被禁用（不允许创建表）	存储引擎 `MyISAM` 被禁用（不允许创建表）
3162	授权 ID 不存在	授权 ID `user123` 不存在
3163	授权 ID 已存在	授权 ID `admin` 已存在
3164	被审计 API 中止	被审计 API 中止（`access_denied`；403）
3165	路径表达式不是指向数组中的单元格的路径	–
3166	另一个缓冲池调整大小操作已在进行中	–
3167	功能被禁用	`query_cache` 功能被禁用；请参阅 `query_cache_type` 的文档
3168	服务器不可用	–
3169	会话被终止	–
3170	内存容量超出限制	`sort_buffer` 的内存容量 268435456 字节超出限制。减少缓冲区大小
3171	未对此查询进行范围优化	–
3172	需要分区升级	需要分区升级。请执行以下命令：`ALTER TABLE db1.sales_data UPGRADE PARTITIONING`
3173	客户端持有 GTID 的所有权	客户端持有 GTID `uuid:100` 的所有权。因此，`WAIT_FOR_EXECUTED_GTID_SET` 不能等待此 GTID

3174	无法在索引虚拟列的基础列上添加外键	–
3175	无法在基础列有外键约束的虚拟列上创建索引	–
3176	请勿使用 XA 事务修改表	请勿使用 XA 事务修改 `gtid_executed` 表
3177	存储引擎拒绝获取锁	–
3178	不支持 `ADD COLUMN col...VIRTUAL, ADD INDEX(col)`	–
3179	存储引擎不支持主密钥轮换	–
3180	存储引擎报告的加密密钥轮换错误	存储引擎报告的加密密钥轮换错误：密钥不存在
3181	写入二进制日志失败。然而，主密钥轮换已成功完成	–
3182	存储引擎不可用	–
3183	此表空间无法加密	–
3184	加密选项无效	–
3185	无法从密钥环中找到主密钥	–
3186	解析器放弃解析此查询	–
3187	无法通过原地算法更改加密属性	–
3188	函数失败，因为密钥环服务返回错误	函数 `keyring_key_store` 失败，因为密钥环服务返回错误
3189	数据库schema似乎已过时	数据库schema似乎已过时。`password` 列长度为 77 个字符，应为 93 个字符

3190	不允许 `RESET MASTER`	不允许 `RESET MASTER`，因为 存在进行中的GTID事务
3191	`START GROUP_REPLICATION` 命令失败，因为组已有 9 个成员	–
3192	无法在存储列的基础列上添加外键	–
3193	无法完成此操作，因为表被另一个连接引用	–
3194	表使用的分区引擎已弃用	表 `db1.archive` 使用的分区引擎已弃用，将在未来版本中移除
3195	函数已弃用	`GeomFromWKB(geometry)` 已弃用，将在未来版本中被 `ST_SRID(geometry, 0)` 替换
3196	函数已弃用	`GeomFromWKB(geometry, srid)` 已弃用，将在未来版本中被 `ST_SRID(geometry, srid)` 替换
3197	资源管理器此时无法提交事务分支	–
3198	函数失败	函数 `ST_Intersects` 失败：几何数据无效
3199	语句不安全，因为它在 XA 事务内部使用	–
3200	UDF 失败	`custom_function` UDF 失败；内存分配错误

3201-3231

错误码	错误描述	示例
3201	无法执行密钥环迁移	无法执行密钥环迁移：密钥环插件未加载
3202	访问被拒绝；您需要权限来执行此操作	访问被拒绝；您需要 `AUDIT_ADMIN` 权限来执行此操作

3203	密钥环迁移状态	密钥环迁移已完成
3204	无法打开过滤器表	无法打开 `audit_log_filter` 过滤器表
3205	无法打开表	无法打开 `mysql.audit_log_user` 审计日志表
3206	未安装密钥环插件	–
3207	未设置审计日志加密密码；将自动生成	–
3208	无法创建 AES 密钥。OpenSSL 的 `EVP_BytesToKey` 函数失败	–
3209	无法从密钥环获取审计日志加密密码	–
3210	尚未安装审计日志过滤功能	–
3211	忽略对用户的请求。需要 `SUPER` 权限或 `AUDIT_ADMIN` 角色	忽略对 `'john'@'localhost'` 的请求
3212	用户需要 `SUPER` 权限或 `AUDIT_ADMIN` 角色	用户 `'jane'@'192.168.1.100'` 需要 `SUPER` 权限或 `AUDIT_ADMIN` 角色
3213	无法重新初始化审计日志过滤器	–
3214	参数类型无效	–
3215	参数数量无效	–
3216	尚未使用 `INSTALL PLUGIN` 语法安装 `audit_log` 插件	–
3217	`"max_array_length"` 参数类型无效	–
32	`"max_array_length"` 参数值无效	–

18		
3220	过滤器名称不能为空	–
3221	用户不能为空	–
3222	未找到指定的过滤器	–
3223	用户名的首字符必须是字母数字	–
3224	用户名中包含无效字符	–
3225	主机名中包含无效字符	–
3226	启用 `MAXDB SQL` 模式后，`TIMESTAMP` 与 `DATETIME` 相同	–
3227	不支持将复制过滤器与 XA 事务一起使用	–
3228	无法打开文件进行错误记录	无法打开文件 `/var/log/mysql/error.log` 进行错误记录
3229	对于具有夏令时（DST）的时区，对时间的分组是不确定的	–
3230	无法启动服务器：命名管道已被使用	无法启动服务器：命名管道 `MySQL` 已被使用
3231	当前事务的写集数据大小超过了外部组件施加的限制	–

3500-3600

错误码	错误描述	示例
3500	不允许在使用 `ROW_FORMAT=COMPRESSED` 或 `KEY_BLOCK_SIZE` 时创建临时表	–

3501	由于存储引擎的以下错误，ACL 操作失败	由于存储引擎的以下错误，ACL 操作失败：错误代码 150 – 表空间损坏
3502	存储引擎不支持指定的索引算法，已改用存储引擎默认算法	此存储引擎不支持 `HASH` 索引算法，已改用存储引擎默认算法
3503	数据库不存在	数据库 `temp_db` 不存在
3504	列的枚举值过多	列 `status` 的枚举值过多
3505	列的枚举/集合值过长	列 `permissions` 的枚举/集合值过长
3506	字典对象无效	`TABLE` 字典对象无效 (`users` 表定义错误)
3507	无法更新字典对象	无法更新 `INDEX` 字典对象
3508	字典对象 ID 不存在	字典对象 ID (12345) 不存在
3509	字典对象名称无效	字典对象名称 `invalid_name` 无效 (包含非法字符)
3510	表空间不存在	表空间 `users_tablespace` 不存在
3511	例程的注释过长	例程 `calculate_stats` 的注释过长 (最大 = 1024)
3512	无法加载例程	无法加载例程 `process_data`
3513	位运算符的二进制操作数必须等长	–
3514	聚合位函数不能接受长度超过 511 字节的参数	–
3515	提示不支持在特定上下文中使用	提示不支持在 `UNION` 中使用
3516	在函数中，值不是期望类型的几何对象	在 `ST_Area` 中， `geometry` 值是意外类型 `POIN`

		T 的几何对象
3517	无法解析 SRID 的空间参考系定义	无法解析 SRID 3857的空间参考系定义
3518	SRID 的空间参考系定义未指定必需的投影参数	SRID 4326 的空间参考系定义未指定必需的 latitude_of_origin (EPSG 9802) 投影参数
3519	没有指定 SRID 的空间参考系	没有 SRID 为9999的空间参考系
3520	函数仅定义为笛卡尔空间参考系，但其参数之一在非笛卡尔系 SRID 中	函数 ST_Distance 仅定义为笛卡尔空间参考系，但其参数之一在 SRID 4326 中，而 SRID 4326不是笛卡尔系
3521	函数仅定义为笛卡尔空间参考系，但其参数之一在未定义 SRID 中	函数 ST_Length 仅定义为笛卡尔空间参考系，但其参数之一在 SRID 10000中，而 SRID 10000尚未定义
3522	主键索引不能为不可见	–
3523	未知的授权 ID	未知的授权 ID john@localhost
3524	授予权限失败	授予 SELECT 给 user123 失败
3525	无法打开安全系统表	–
3526	无法设置默认角色	–
3527	无法在指定的 URN 中找到模式	无法在指定的 URN urn:mysql:component:1.0 中找到模式
3528	无法为指定 URN 中的模式获取模式加载服务实现	无法为指定 URN urn:mysql:component:1.0 中的模式 mysql 获取模式加载服务实现
3529	无法从指定的 URN 加载组件	无法从指定的 URN urn:mysql:audit_log:1.0 加载组件
3530	用户未授予权限给另一个用户	john@localhost 未授予权限给 jane@192.168.1.100

3531	无法从用户撤销角色	无法从 `admin@localhost` 撤销角色
3532	禁止重命名角色标识符	–
3533	无法获取指定的服务实现	无法获取指定的服务实现：`audit_log_filter`
3534	无法满足组件所需的服务依赖关系	无法满足组件 `audit_log` 所需的服务 `keyring` 的依赖关系
3535	无法注册组件提供的服务实现	无法注册组件 `audit_log` 提供的服务实现 `filter`
3536	组件提供的初始化方法失败	组件 `keyring_file` 提供的初始化方法失败
3537	要卸载的 URN 指定组件之前未被加载过	要卸载的 URN `urn:mysql:component:1.0` 指定组件之前未被加载过
3538	组件提供的卸载方法失败	组件 `audit_log` 提供的卸载方法失败
3539	释放先前获取的服务实现失败	–
3540	卸载组件期间，撤销组件提供的服务实现失败	卸载组件期间，撤销组件 `audit_log` 提供的服务实现 `filter` 失败
3541	无法从指定的 URN 卸载组件	无法从指定的 URN `urn:mysql:component:1.0` 卸载组件
3542	持久动态加载器被用来卸载组件，但之前并未用它加载该组件	持久动态加载器被用来卸载组件 `audit_log`，但之前并未用它加载该组件
3543	`mysql.component` 表缺失或定义不正确	–
3544	无法处理组件的持久化数据	无法处理组件 `audit_log` 的持久化数据。存储引擎返回错误代码 150
3545	指定的 URN 的组件在组中被指定了多次	指定的 URN `urn:mysql:component:1.0` 组件在组中被指定了多次

3546	无法更改 `@@GLOBAL.GTID_PURGED`	无法更改 `@@GLOBAL.GTID_PURGED`：存在进行中的事务
3547	无法为用户管理缓存加锁进行处理	–
3548	没有指定 SRID 的空间参考系	没有 SRID 为 10000 的空间参考系
3549	无法持久化变量。请重试	–
3550	INFORMATION_SCHEMA 查询不支持特定子句	INFORMATION_SCHEMA 查询不支持 `WHERE` 子句
3551	无法将动态统计信息存储到数据字典中	无法将动态 `TABLE` 统计信息存储到数据字典中
3552	拒绝访问系统模式	拒绝访问系统模式 `mysql`
3553	拒绝访问系统表空间	拒绝访问系统表空间 `innodb_system`
3554	拒绝访问对象	拒绝访问 TABLE `mysql.user`
3555	拒绝访问数据字典表	–
3556	拒绝访问系统表	–
3557	拒绝访问表	–
3558	函数中的选项键无效	函数 `JSON_EXTRACT` 中的选项键 `invalid_key` 无效
3559	函数中选项的值无效	函数 `ST_Buffer` 中选项 `distance` 的值 `-10` 无效
3560	字符串在函数中不是有效的键值对	字符串 `key=value` 在函数 `JSON_OBJECT` 中不是有效的键值对
3561	函数中的 options 参数以无效字符开头	函数 `JSON_OBJECT` 中的 options 参数以无效字符

		`#` 开头
3562	函数中的 options 参数以无效字符结尾	函数 `JSON_OBJECT` 中的 options 参数以无效字符 `;` 结尾
3563	函数中的 options 参数包含无效字符序列	函数 `JSON_OBJECT` 中的 options 参数包含无效字符序列 `;;`
3564	函数中存在重复的选项键	函数 `JSON_OBJECT` 中存在重复的选项键 `name`
3565	没有指定 SRID 的空间参考系。轴顺序未知	没有 SRID 为 10000 的空间参考系。轴顺序未知
3566	拒绝访问原生函数	拒绝访问原生函数 `AES_ENCRYPT`
3567	请求的下一个二进制日志索引值超出范围	请求的下一个二进制日志索引值 `1000000` 超出范围。请使用介于 `1` 和 `999999` 之间的值
3568	锁定子句中的表名未解析	锁定子句中的表名 `alias_table` 未解析
3569	表出现在多个锁定子句中	表 `users` 出现在多个锁定子句中
3570	语句不安全，因为它使用了 `SKIP LOCKED`	–
3571	语句不安全，因为它使用了 `NOWAIT`	–
3572	由于无法立即获取锁且设置了 `NOWAIT`，语句被中止	–
3573	递归公共表表达式应包含 `UNION`	递归公共表表达式 `cte1` 应包含 `UNION`
3574	递归公共表表达式应有非递归查询块后跟递归查询块	递归公共表表达式 `cte2` 应有一个或多个非递归查询块，后跟一个或多个递归查询块
3575	递归公共表表达式的递归查询块中不能包含聚合或窗口函数	递归公共表表达式 `cte3` 的递归查询块中不能包含聚合或窗口函数
3576	在递归公共表表达式的递归查询块中，递归表不能在 `LEFT JOIN` 的右侧参	在递归公共表表达式 `cte4` 的递归查询块中，递归表不能在 `LEFT JOIN` 的右侧参数中

	数中	
3577	在递归公共表表达式的递归查询块中，递归表必须仅被引用一次	在递归公共表表达式 `cte5` 的递归查询块中，递归表必须仅被引用一次，且不能在任何子查询中
3578	操作要求 `@@internal_tmp_disk_storage_engine=InnoDB`	`CREATE TEMPORARY TABLE` 要求 `@@internal_tmp_disk_storage_engine=InnoDB`
3579	窗口名称未定义	窗口名称 `win1` 未定义
3580	窗口依赖图中存在循环	–
3581	依赖于其他窗口的窗口不能定义分区	–
3582	窗口有框架定义，因此不能被其他窗口引用	窗口 `win2` 有框架定义，因此不能被其他窗口引用
3583	窗口不能继承另一个窗口，因为两者都包含 `ORDER BY` 子句	窗口 `win3` 不能继承 `win4`，因为两者都包含 `ORDER BY` 子句
3584	窗口：框架起始边界不能是 `UNBOUNDED FOLLOWING`	窗口 `win5`：框架起始边界不能是 `UNBOUNDED FOLLOWING`
3585	窗口：框架结束边界不能是 `UNBOUNDED PRECEDING`	窗口 `win6`：框架结束边界不能是 `UNBOUNDED PRECEDING`
3586	窗口：框架起始或结束边界为负数、NULL 或非整数类型	窗口 `win7`：框架起始或结束边界为负数、NULL 或非整数类型
3587	具有 `RANGE N PRECEDING/FOLLOWING` 框架的窗口要求恰好有一个 `ORDER BY` 表达式	具有 `RANGE N PRECEDING/FOLLOWING` 框架的窗口 `win8` 要求恰好有一个 `ORDER BY` 表达式，且为数值或时间类型
3588	具有 `RANGE` 框架的窗口的 `ORDER BY` 表达式为日期时间类型	具有 `RANGE` 框架的窗口 `win9` 的 `ORDER BY` 表达式为日期时间类型。仅允许 `INTERVAL` 边界值
3589	具有 `RANGE` 框架的窗口的 `ORDER BY` 表达式为数值类型	具有 `RANGE` 框架的窗口 `win10` 的 `ORDER BY` 表达式为数值类型，不允许 `INTERVAL` 边界值
3590	窗口具有非常量框架边界	窗口 `win11` 具有非常量框架边界

3591	窗口被定义了两次	窗口 `win12` 被定义了两次
3592	窗口：`ORDER BY` 或 `PARTITION BY` 使用了不再支持的旧式位置指示符	窗口 `win13`：`ORDER BY` 或 `PARTITION BY` 使用了不再支持的旧式位置指示符，请使用表达式
3593	不能在此上下文中使用窗口函数	不能在此上下文中使用窗口函数 `ROW_NUMBER`
3594	不能在此上下文中使用包含窗口函数的表达式的别名	不能在此上下文中使用包含窗口函数的表达式的别名 `row_num`
3595	不能在窗口的规范中嵌套窗口函数	不能在窗口 `win14` 的规范中嵌套窗口函数
3596	窗口：`INTERVAL` 只能与 `RANGE` 框架一起使用	窗口 `win15`：`INTERVAL` 只能与 `RANGE` 框架一起使用
3597	窗口函数不允许在 `GROUP BY` 中使用 `ASC` 或 `DESC`	-
3598	要获取关于窗口函数的信息，请使用 `EXPLAIN FORMAT=JSON`	-
3599	窗口函数忽略了窗口的框架子句，并对整个分区进行聚合	窗口函数 `SUM` 忽略了窗口 `win16` 的框架子句，并对整个分区进行聚合
3600	窗口操作要求 `@@internal_tmp_mem_storage_engine=TempTable`	-

3601-3700

错误码	错误描述	示例
3601	函数参数过多	函数 `JSON_OBJECT` 的参数过多：25；最多允许 20 个
3602	`GROUPING` 函数的参数不在 `GROUP BY` 中	`GROUPING` 函数的参数 #1 不在 `GROUP BY` 中
3603	表空间的注释过长	表空间 `users_tablespace` 的注释过长（最大 = 2048）

3604	存储引擎无法删除表	存储引擎无法删除表 `db1.temp_table`
3605	存储引擎无法删除表，因为该表缺失	存储引擎无法删除表 `db2.missing_table`，因为该表缺失
3606	表空间的文件名重复	表空间 `shared_tablespace` 的文件名重复
3607	删除数据库时出现问题。无法移除数据库目录	删除数据库时出现问题。无法移除数据库目录 (`/var/lib/mysql/test_db`)
3608	没有 SDI 文件匹配模式	没有 SDI 文件匹配模式 `*.sdi`
3609	SDI 中引用的模式不存在	SDI 中引用的模式 `old_schema` 不存在
3610	SDI 中引用的表已存在	SDI 中引用的表 `db1.users` 已存在
3611	导入的 `sqlengine_version` 与当前版本不兼容	导入的 `sqlengine_version` (80016) 与当前版本 (80028) 不兼容
3612	导入的 `dd` 版本与当前版本不兼容	导入的 `dd` 版本 (1) 与当前版本 (2) 不兼容
3613	导入的 `sdi` 版本与当前版本不兼容	导入的 `sdi` 版本 (1) 与当前版本 (2) 不兼容
3614	提示的参数数量无效	提示 `INDEX_MERGE` 的参数数量无效
3615	变量在 `TDMetaCluster` 中不存在	变量 `cluster_mode` 在 `TDMetaCluster` 中不存在
3606	函数中的经度超出范围	函数 `ST_Point` 中的经度181.0超出范围。必须在 (-180.0, 180.0] 之间
3617	函数中的纬度超出范围	函数 `ST_Point` 中的纬度91.0超出范围。必须在 [-90.0, 90.0] 之间
3618	尚未为地理空间参考系统实现函数	尚未为地理空间参考系统实现 `ST_Area` (`POLYGON`)
3619	为权限指定的权限级别非法	为 `SELECT` 指定的权限级别非法
3620	拒绝访问系统视图	拒绝访问系统视图 `INFORMATION_SCHEMA.COLUMNS`
3621	日志过滤器组件出现混淆	日志过滤器组件 `audit_log` 在 `filter_init` 处出现混淆

3622	分区表达式过长	–
3623	`DROP FUNCTION` 无法删除动态注册的用户定义函数	–
3624	无法为表中的列存储列统计信息	无法为表 `db1.users` 中的列 `email` 存储列统计信息
3625	无法更新表中的列的列统计信息	无法更新表 `db2.orders` 中的列 `amount` 的列统计信息
3626	无法移除表中的列的列统计信息	无法移除表 `db3.products` 中的列 `price` 的列统计信息
3627	无法为表中的列构建直方图统计信息	无法为表 `db4.customers` 中的列 `age` 构建直方图统计信息
3628	角色是强制角色，不能被撤销或删除	角色 `admin_role` 是强制角色，不能被撤销或删除
3629	表空间没有指定名称的文件	表空间 `data_tablespace` 没有名为 `data_file.ibd` 的文件
3630	访问被拒绝；您需要权限来执行此操作	访问被拒绝；您需要 `CREATE USER` 权限来执行此操作
3631	执行命令需要 `SUPER` 权限	执行命令 `CHANGE MASTER` 需要 `SUPER` 权限
3632	路径位于当前数据目录内	路径位于当前数据目录 `/var/lib/mysql` 内
3633	克隆操作期间执行了并发 DDL。请重试	–
3634	并发克隆操作过多	并发克隆操作过多。最大允许 – 5
3635	事务中的表不符合外部插件的要求	事务 `xa123` 中的表不符合外部插件的要求
3636	递归查询在指定次数迭代后中止	递归查询在100次迭代后中止。请尝试增加 `@@cte_max_recursion_depth` 到一个更大的值
3637	变量不能使用 `SET_VAR` 提示设置	变量 `sql_mode` 不能使用 `SET_VAR` 提示设置
3638	不能使用这些凭证，因为它们与密码历史策略相矛盾	不能使用这些凭证用于 `'john'@'localhost'`，因为它们与密码历史策略相矛盾
3639	非零密码历史子句被忽略，因为用户身份验证插件不支持密码历史	非零密码历史子句被忽略，因为用户 `'jane'@'192.168.1.100'` 的身份验证插件 `sha256_password` 不支

		持密码历史
3640	客户端不支持特定功能	客户端不支持 SSL
3641	表空间被跳过，因为其定义正被并发 DDL 语句修改	表空间 temp_tablespace 被跳过，因为其定义正被并发 DDL 语句修改
3642	引擎与表空间存储的引擎不匹配	引擎 InnoDB 与表空间 shared_tablespace 存储的引擎 MyISAM 不匹配
3643	几何对象的 SRID 与列的 SRID 不匹配	几何对象的 SRID 是4326，但列的 SRID 是3857
3644	无法更改列上的 SRID 规范，因为该列上存在空间索引	无法更改列 location 上的 SRID 规范，因为该列上存在空间索引
3645	选项 binlog_row_value_options 将被忽略	当 BINLOG_FORMAT=STATEMENT 时，选项 binlog_row_value_options=PARTIAL_JSON 将被忽略
3646	选项 log_bin_use_v1_row_events=1 将被忽略	当 BINLOG_FORMAT=ROW 时，选项 log_bin_use_v1_row_events=1 将被忽略
3647	选项 binlog_row_value_options 将仅用于后映像	当 部分更新 时，选项 binlog_row_value_options=PARTIAL_JSON 将仅用于后映像
3648	无法在表的列中应用 JSON 差异	无法在表 users 的列 preferences 中应用 JSON 差异
3649	表的列的 JSON 差异已损坏	表 settings 的列 config 的 JSON 差异已损坏
3650	资源组已存在	资源组 batch_group 已存在
3651	资源组不存在	资源组 realtime_group 不存在
3652	无效的 CPU ID	无效的 CPU ID 100
3653	无效的 VCPU 范围	无效的 VCPU 范围10-5
3654	资源组的线程优先级值无效	SYSTEM 资源组 high_priority 的线程优先级值20无效。允许的范围是 [-20, 0]
3655	不允许在资源组上执行操作	不允许在 SYSTEM 上执行 SET 操作
3656	资源组正忙	资源组 app_group 正忙
3657	资源组被禁用	资源组 temp_group 被禁用

3658	功能不受支持	功能 `CPU_AFFINITY` 不受支持 (硬件不支持)
3659	属性被忽略	属性 `THREAD_PRIORITY` 被忽略 (操作系统限制)
3660	无效的线程 ID	无效的线程 ID (0)
3661	无法将资源组与线程 ID 绑定	无法将资源组 `group1` 与线程 ID (12345) 绑定 (线程不存在)
3662	由于未指定 `DISABLE` 选项，选项 `FORCE` 无效	–
3663	命令遇到失败	`ALTER RESOURCE GROUP` 命令遇到失败。资源组正忙
3664	无法在表空间中操作 SDI	无法在表空间 `users_tablespace` 中 `CREATE` SDI `db1.users`
3665	`JSON_TABLE` 列缺少值	`JSON_TABLE` 列 `user_name` 缺少值
3666	不能在标量 `JSON_TABLE` 列中存储数组或对象	不能在标量 `JSON_TABLE` 列 `address` 中存储数组或对象
3667	每个表函数都必须有一个别名	–
3668	对于 LATERAL 引用，必须使用 `INNER` 或 `LEFT JOIN`	对于 `users` 所做的 LATERAL 引用，必须使用 `INNER` 或 `LEFT JOIN`
3669	`JSON_TABLE` 的列的值超出范围	`JSON_TABLE` 的列 `age` 的值超出范围
3670	在 `JSON_TABLE` 中发现了超过支持数量的 `NESTED PATH`	在 `JSON_TABLE` `user_data` 中发现了超过支持数量10的 `NESTED PATH`
3671	所选的身份验证方法不支持密码过期	所选的身份验证方法 `mysql_native_password` 不支持密码过期
3672	提供给函数的 GeoJSON 数据无效：必须在顶级对象中指定成员 `crs`	提供给函数 `ST_GeomFromGeoJSON` 的 GeoJSON 数据无效：必须在顶级对象中指定成员 `crs`
3673	列不能为 null	列 `user_id` 不能为 null
3674	查询优化器将不会使用列上的空间索引，因为该列没有 SRID 属性	查询优化器将不会使用列 `coordinates` 上的空间索引，因为该列没有 SRID 属性
3675	创建表/表空间失败，因为磁盘已满	创建表/表空间 `large_table` 失败，因为磁盘已满

3676	无法解析摘要函数的参数	无法解析摘要函数的参数：`"invalid_format"`
3677	无法解析摘要函数的参数	–
3678	模式目录已存在	模式目录 `/var/lib/mysql/schema1` 已存在
3679	模式目录不存在	模式目录 `/var/lib/mysql/schema2` 不存在
3680	无法创建模式目录	无法创建模式目录 `/var/lib/mysql/schema3`（错误号：13 – 权限不足）
3681	模式不存在，但找到了模式目录	模式 `old_schema` 不存在，但找到了模式目录 `/var/lib/mysql/old_schema`
3682	函数仅针对 SRID 0 和 SRID 4326 定义	函数 `ST_Distance` 仅针对 SRID 0和 SRID 4326定义
3683	选项 `expire_logs_days` 和 `binlog_expire_logs_seconds` 不能同时使用	–
3684	结果字符串大于结果缓冲区	–
3685	正则表达式的参数非法	–
3686	正则表达式搜索中的索引越界	–
3687	正则表达式库中的内部错误	–
3688	正则表达式语法错误	第5行第3个字符处的正则表达式语法错误
3689	正则表达式中无法识别的转义序列	–
3690	正则表达式中包含此库版本未实现的功能	–
3691	正则表达式中括号不匹配	–
3692	`{min,max}` 区间的描述不正确	–
3693	`{min,max}` 区间中的最大值小于最小值	–
3694	正则表达式中无效的反向引用	–
3695	正则表达式中的后视断言超过了限制	–

3696	正则表达式包含未闭合的括号表达式	–
3697	正则表达式包含一个 `[x-y]` 字符范围，其中 x 在 y 之后	–
3698	正则表达式回溯栈溢出	–
3699	正则表达式匹配超时	–
3700	正则表达式模式在大小或复杂度上超过了限制	–

3701-3800

错误码	错误描述	示例
3701	组件的值在特定位置出现混淆	`audit_log` 的值在 `filter_init` 处出现混淆
3702	设置空的管道会禁用错误日志记录	设置空的 `log_error_services` 管道会禁用错误日志记录
3703	过滤器错误	过滤器 `audit_log`：初始化失败
3704	尚未为笛卡尔空间参考系统实现函数	尚未为笛卡尔空间参考系统实现 `ST_Area`（`POLYGON`）
3705	尚未为投影空间参考系统实现函数	尚未为投影空间参考系统实现 `ST_Length`（`LINESTRING`）
3706	提供给函数的半径无效	提供给函数 `ST_Buffer` 的半径无效：半径必须大于零
3707	重启服务器失败	重启服务器失败 (权限不足)
3708	缺少必需的属性	缺少必需的属性 `SRID`
3709	属性有多重定义	属性 `name` 有多重定义
3710	空间参考系统名称不能是空字符串或以空白字符开头或结尾	–

3711	组织名称不能是空字符串或以空白字符开头或结尾	–
3712	已经存在指定 SRID 的空间参考系统	已经存在 SRID 为4326的空间参考系统
3713	已经存在指定 SRID 的空间参考系统	已经存在 SRID 为3857的空间参考系统
3714	SRID 0 不可修改	–
3715	SRID 范围已被保留供系统使用	SRID 范围 [0, 1000] 已被保留供系统使用
3716	无法修改 SRID，因为至少有一列依赖于它	无法修改 SRID 4326。至少有一列依赖于它
3717	属性中包含无效字符	属性 `description` 中包含无效字符
3718	属性过长	属性 `name` 过长。最大长度为255个字符
3719	`utf8` 当前是字符集 UTF8MB3 的别名	–
3720	`NATIONAL` / `NCHAR` / `NVARCHAR` 隐含字符集 UTF8MB3	–
3721	无效的默认排序规则	无效的默认排序规则 `utf8mb4_unicode_ci`
3722	无法为列收集信息	无法为列 `email` 收集信息：数据损坏
3723	表不能在保留的表空间中创建	表 `temp_table` 不能在保留的表空间 `innodb_system` 中创建
3724	此选项不能设置为特定值	此选项不能设置为 `OFF`
3725	主库和从库上的原子 DDL 语句提交失败	–
3726	函数仅针对地理空间参考系统定义	函数 `ST_Distance` 仅针对地理空间参考系统定义，但其参数之一位于 SRID 0，而 SRID 0 不是

		地理参考系统
3727	函数遇到了一个太大的多边形	函数 `ST_Area` 遇到了一个太大的多边形
3728	空间索引不能是主键或唯一索引	–
3729	空间索引不支持索引类型	空间索引不支持索引类型 `UNIQUE`
3730	无法删除表，因为外键约束引用了它	无法删除表 `users`，因为表 `orders` 上的外键约束 `fk_user_id` 引用了它
3731	函数参数包含经度超出范围的几何对象	函数 `ST_Distance` 的一个参数包含经度为 181.0 的几何对象，该值超出范围
3732	函数参数包含纬度超出范围的几何对象	函数 `ST_Distance` 的一个参数包含纬度为 91.0 的几何对象，该值超出范围
3733	外键使用了不支持的虚拟列	外键 `fk_virtual` 使用了不支持的虚拟列 `virtual_col`
3734	添加外键约束失败。被引用表中缺少约束的列	添加外键约束失败。被引用表 `users` 中缺少约束 `fk_user` 的列 `id`
3735	无法为错误代码添加抑制规则	`audit_log`：无法为代码 `"ACCESS_DENIED"` 添加抑制规则
3736	空间参考系定义指定了无效的地理轴	SRID 4326 的空间参考系定义指定了无效的地理轴 `NORTH` 和 `WEST`
3737	半长轴的长度必须为正数	–
3738	反扁率必须大于 1.0，如果是球体则为 0.0	–
3739	角度单位转换因子必须为正数	–
3740	本初子午线必须在 (−180, 180] 度之间	–
3741	不支持从 SRID 转换	不支持从 SRID 10000转换

3742	不支持转换到 SRID	不支持转换到 SRID 20000
3743	不支持从 SRID 转换。该空间参考系没有 TOWGS84 子句	不支持从 SRID 30000转换。该空间参考系没有 TOWGS84 子句
3744	不支持转换到 SRID。该空间参考系没有 TOWGS84 子句	不支持转换到 SRID 40000。该空间参考系没有 TOWGS84 子句
3745	当会话存在打开的临时表时，禁止更改 `@@session.binlog_format`	–
3746	当任何复制通道存在打开的临时表时，禁止更改 `@@global.binlog_format` 或 `@@persist.binlog_format`	–
3747	当任何复制通道应用线程正在运行时，禁止更改 `@@global.binlog_format` 或 `@@persist.binlog_format`	–
3748	语句违反 GTID 一致性	语句违反 GTID 一致性：当 `@@session.binlog_format=STATEMENT` 时，不允许在事务内使用 `CREATE TEMPORARY TABLE`
3749	XA：在客户端断开连接期间，无法为 PREPARED XA 事务备份 MDL 锁	–
3750	当 `sql_require_primary_key` 设置时，无法创建或更改没有主键的表	–
3751	函数索引数据被截断	第10行的函数索引 `idx_name_length` 数据被截断
3752	函数索引值超出范围	第15行的函数索引 `idx_age` 值超出范围
3753	无法在返回 JSON 或 GEOMETRY 值的函数上创建函数索引	–
3754	函数索引不能引用自增列	函数索引 `idx_auto` 不能引用自增列 `id`
3755	无法删除列，因为它被函数索引使用	无法删除列 `email`，因为它被函数索引 `idx_email_length` 使用

3756	主键不能是函数索引	-
3757	无法在返回 BLOB 或 TEXT 的表达式上创建函数索引	无法在返回 `TEXT` 的表达式上创建函数索引
3758	函数索引的表达式包含不允许的函数	函数索引 `idx_expression` 的表达式包含不允许的函数 `UUID()`
3759	不支持全文函数索引	-
3760	不支持空间函数索引	-
3761	使用的存储引擎无法索引表达式	使用的存储引擎 `MyISAM` 无法索引表达式 `LOWER(name)`
3762	不支持在列上创建函数索引	不支持在列 `content` 上创建函数索引
3763	生成列的表达式包含不允许的函数	生成列 `full_name` 的表达式包含不允许的函数：`CONCAT()`
3764	生成列的表达式不能引用行值	生成列 `row_hash` 的表达式不能引用行值
3765	生成列的表达式不能引用用户或系统变量	生成列 `current_user` 的表达式不能引用用户或系统变量
3766	表列具有默认值表达式依赖，因此无法删除或重命名	表 `users` 的列 `username` 具有默认值表达式依赖，因此无法删除或重命名
3767	列的默认值表达式不能引用在它之后定义的列	列 `full_name` 的默认值表达式不能引用在它之后定义的列 `last_name`
3768	列的默认值表达式不能引用自增列	列 `user_ref` 的默认值表达式不能引用自增列 `id`
3769	列的默认值表达式包含不允许的函数	列 `created_at` 的默认值表达式包含不允许的函数
3770	列的默认值表达式包含不允许的函数	列 `updated_at` 的默认值表达式包含不允许的函数：`NOW()`

3771	列的默认值表达式不能引用行值	列 `row_version` 的默认值表达式不能引用行值
3772	列的默认值表达式不能引用用户或系统变量	列 `current_time` 的默认值表达式不能引用用户或系统变量
3773	`DEFAULT` 函数不能用于默认值表达式	–
3774	默认值表达式不支持特定功能	默认值表达式不支持 `AUTO_INCREMENT`
3775	语句违反 GTID 一致性	语句违反 GTID 一致性：使用表达式作为 `DEFAULT` 的 `ALTER TABLE ... ADD COLUMN`
3776	无法更改表的存储引擎，因为该表参与了外键约束	–
3777	在表达式中设置用户变量已弃用	–
3778	排序规则是已弃用字符集 UTF8MB3 的排序规则	`utf8_general_ci` 是已弃用字符集 UTF8MB3 的排序规则
3779	嵌套注释语法已弃用	–
3780	外键约束中的引用列和被引用列不兼容	外键约束 `fk_product` 中的引用列 `product_code` 和被引用列 `id` 不兼容
3781	在新组复制主成员应用积压时，已持有语句超时	–
3782	当新主成员应用积压时，由于组复制插件关闭或线程被终止，已持有语句被中止	–
3783	在组复制主选举期间应用积压时，由于成员处于错误状态，已持有语句被中止	–
3784	无法从密钥环获取密钥	–
3785	无法从密钥环找到密钥	–
3786	从密钥环获取了无效的密钥	–

3787	读取复制日志加密头时出错	读取复制日志加密头时出错：文件损坏
3788	更改二进制日志加密设置后，轮转某些日志失败	–
3789	密钥意外存在	密钥 `binlog_key_1` 意外存在
3790	生成密钥失败	–
3791	存储密钥失败	–
3792	移除密钥失败	–
3793	无法更改 `binlog_encryption` 值	无法更改 `binlog_encryption` 值。密钥环未加载
3794	无法恢复二进制日志加密主密钥	–
3795	慢查询日志文件格式已按要求更改	–
3796	选项 `group_replication_consistency` 不能用于当前成员状态	–
3797	在 `group_replication_consistency='BEFORE'` 时等待组事务提交出错	–
3798	等待具有 `group_replication_consistency='AFTER'` 的事务提交时出错	–
3799	组复制插件正在停止，因此不允许新事务启动	–
3800	函数索引的表达式不能引用行值	函数索引 `idx_row_hash` 的表达式不能引用行值

3801-3900

错误	错误描述	示例

码		
3801	加密要写入二进制日志文件的内容失败	加密要写入二进制日志文件的内容失败：密钥环错误
3802	页面跟踪尚未启动	–
3803	未为指定的 LSN 范围启用跟踪	–
3804	当并发克隆正在进行时，无法清除数据	–
3805	当 `binlog-encryption` 关闭时，无法轮转二进制日志主密钥	–
3806	无法恢复二进制日志主密钥，组合错误	无法恢复二进制日志主密钥，组合 new_master_key_seqno=2, master_key_seqno=1 和 old_master_key_seqno=0 错误
3807	在密钥环上操作二进制日志主密钥失败	–
3808	轮转一个或多个二进制或中继日志文件失败	–
3809	二进制日志主密钥轮转部分成功	部分文件轮转失败。已生成新的二进制日志主密钥
3810	无法从未使用的密钥环中移除二进制日志加密密钥	–
3811	无法从密钥环移除辅助二进制日志加密密钥	–
3812	为检查约束指定了非布尔类型的表达式	为检查约束 `chk_age` 指定了非布尔类型的表达式
3813	列检查约束引用了其他列	列检查约束 `chk_salary` 引用了其他列
3814	检查约束的表达式包含不允许的函数	检查约束 `chk_email` 的表达式包含不允许的函数： `UUID()`
38	检查约束的表达式包含不允许的函数	检查约束 `chk_date` 的表达式包含不允许的函数

15		
3816	检查约束的表达式不能引用用户或系统变量	检查约束 `chk_limit` 的表达式不能引用用户或系统变量
3817	检查约束不能引用行值	检查约束 `chk_row` 不能引用行值
3818	检查约束不能引用自增列	检查约束 `chk_id` 不能引用自增列
3819	违反检查约束	违反检查约束 `chk_age_positive`
3820	检查约束引用了不存在的列	检查约束 `chk_dept` 引用了不存在的列 `department_id`
3821	在表中未找到检查约束	在表中未找到检查约束 `chk_status`
3822	重复的检查约束名称	重复的检查约束名称 `chk_duplicate`
3823	列不能在检查约束中使用：外键约束需要此列	列 `user_id` 不能在检查约束 `chk_user` 中使用：外键约束 `fk_user` 的引用操作需要此列
3824	在启用了默认加密的数据库中创建未加密的表	–
3825	请求在使用特定表空间时创建表	请求在使用 `ENCRYPTED` 表空间时创建 `UNENCRYPTED` 表
3826	表加密与其数据库默认加密不同，并且用户权限不足	–
3827	数据库默认加密与 `default_table_encryption` 设置不同，并且用户权限不足	–
3828	表空间加密与 `default_table_encryption` 设置不同，并且用户权限不足	–
3829	此表空间无法加密，因为其中一个表的模式默认加密为 OFF	–

3830	此表空间无法解密，因为其中一个表的模式默认加密为 ON	–
3831	无法确定表空间的类型	无法确定名为 `shared_tablespace` 的表空间的类型
3832	源表空间已加密但目标表空间未加密	–
3833	`ENCRYPTION` 子句对表空间无效	`ENCRYPTION` 子句对 `TEMPORARY` 表空间无效
3834	多个子句	多个 `ENCRYPTION` 子句
3835	`GRANT ... AS` 目前仅支持全局权限	–
3836	`AS` 子句中的某些授权 ID 无效	–
3837	列具有函数索引依赖，因此无法删除或重命名	列 `email` 具有函数索引依赖，因此无法删除或重命名
3838	插件不会被用作 "早期" 插件	插件 `keyring_file` 不会被用作 "早期" 插件
3839	重做日志归档启动禁止在 `subdir` 参数中使用路径名	–
3840	重做日志归档启动超时	–
3841	服务器变量 `innodb_redo_log_archive_dirs` 为 NULL 或为空	–
3842	在服务器变量 `innodb_redo_log_archive_dirs` 中未找到标签	在服务器变量 `innodb_redo_log_archive_dirs` 中未找到标签 `archive1`
3843	服务器变量 `innodb_redo_log_archive_dirs` 中标签后为空	服务器变量 `innodb_redo_log_archive_dirs` 中标签 `archive2` 后为空
3844	重做日志归档目录不存在或不是目录	重做日志归档目录 `/archive/logs` 不存在或不是目录

3845	重做日志归档目录位于服务器目录之内、之下或之上	重做日志归档目录 `/archive` 位于服务器目录 `/var/lib/mysql` 之内
3846	重做日志归档目录对所有操作系统用户可访问	重做日志归档目录 `/tmp/archive` 对所有操作系统用户可访问
3847	无法创建重做日志归档文件	无法创建重做日志归档文件 `/archive/redo.log`（操作系统错误号：13 - `权限不足`）
3848	重做日志归档已在特定路径上启动	重做日志归档已在 `/archive/redo.log` 上启动
3849	重做日志归档未激活	-
3850	重做日志归档失败	重做日志归档失败：`磁盘空间不足`
3851	重做日志归档不是由本会话启动的	-
3852	正则表达式错误	正则表达式错误：函数 `REGEXP_INSTR` 中的无效模式
3853	函数参数中的 JSON 类型无效	函数 `JSON_EXTRACT` 的参数 1 中的 JSON 类型无效；需要 `OBJECT`
3854	无法将字符串进行字符集转换	无法将字符串 `'中文'` 从 `utf8mb4` 转换为 `latin1`
3855	列具有分区函数依赖，因此无法删除或重命名	列 `region` 具有分区函数依赖，因此无法删除或重命名
3856	`FLOAT/DOUBLE` 列的 `AUTO_INCREMENT` 支持已弃用	请考虑从列 `float_id` 中移除 `AUTO_INCREMENT`
3857	当实例因备份被锁定时，无法停止从库 SQL 线程	-
3858	为浮点数据类型指定位数已弃用	-
3859	`DECIMAL` 和浮点数据类型的 `UNSIGNED` 已弃用	-

3860	整数显示宽度已弃用	–
3861	`ZEROFILL` 属性已弃用	–
3862	克隆提供者错误	克隆提供者错误：网络连接失败
3863	克隆从提供者收到意外响应	克隆从提供者收到意外响应：协议错误
3864	克隆提供者 MySQL 版本与接收者不同	克隆提供者 MySQL 版本：`8.0.25` 与接收者 MySQL 版本 `8.0.28` 不同
3865	克隆提供者操作系统与接收者不同	克隆提供者操作系统：`Linux` 与接收者操作系统：`Windows` 不同
3866	克隆提供者平台与接收者不同	克隆提供者平台：`x86_64` 与接收者平台：`ARM64` 不同
3867	克隆提供者排序规则在接收者中不可用	克隆提供者排序规则：`utf8mb4_0900_ai_ci` 在接收者中不可用
3868	克隆配置值不同	克隆配置 `innodb_page_size`：提供者值：`16384` 与接收者值：`65536` 不同
3869	克隆系统配置	克隆系统配置：内存不足
3870	克隆提供者插件在接收者中未激活	克隆提供者插件 `audit_log` 在接收者中未激活
3871	克隆到当前数据目录时，克隆无法使用环回连接	–
3872	克隆加密表需要 SSL 连接	–
3873	克隆估算的数据库大小不足	克隆估算的数据库大小为100GB。可用空间50GB不足
3874	并发克隆正在进行中	–
38	当特定条件满足时，无法执行克隆操	当备份锁定激活时，无法执行克隆操作

75	作	
3876	无法向匿名用户授予角色	–
3878	空密码不能设置为用户的第二个密码	空密码不能设置为用户 `'john'@'localhost'` 的第二个密码
3879	访问被拒绝，认证 ID 无法访问数据库	访问被拒绝，认证 ID `user123` @ `192.168.1.100` 无法访问数据库 `secure_db`
3880	无法设置 `mandatory_roles`：认证 ID 拥有权限	无法设置 `mandatory_roles`：认证 ID `admin` @ `localhost` 拥有 `SUPER` 权限
3881	GTID 表尚未准备就绪，无法使用	无法打开表 `mysql.gtid_executed`
3882	传递给函数的几何对象位于 SRID 0 中	传递给函数 `ST_Length` 的几何对象位于 SRID 0 中，该 SRID 未指定长度单位
3883	安装插件时出错	安装插件 `audit_log` 时出错：版本不兼容
3884	存储引擎无法为此会话分配临时表空间	–
3885	收到未知错误	收到未知错误：9999
3886	无法更改表的列。索引结果大小将超过最大键长度	无法更改表 `users` 的列 `name`。索引 `idx_name` 的结果大小将超过最大键长度 767字节
3887	捕获组具有无效的名称	–
3888	无法设置 SSL	由于以下 SSL 库错误，无法设置 SSL：证书无效
3889	辅助引擎操作失败	辅助引擎操作失败。存储空间不足
3890	不允许对定义了辅助引擎的表执行 DDL	–
3891	当前密码不正确	–

3892	需要在 `REPLACE` 子句中指定当前密码才能更改它	–
3893	为其他用户更改密码时，请勿指定当前密码	–
3894	不能为用户设置当前密码，因为身份验证插件正在更改	不能为用户 `'jane'@'192.168.1.100'` 设置当前密码，因为身份验证插件正在更改
3895	不能为用户设置当前密码，因为新密码为空	不能为用户 `'john'@'localhost'` 设置当前密码，因为新密码为空
3896	数据库上至少存在一个部分撤销，系统变量 `@@partial_revokes` 必须设置为 ON	–
3897	认证 ID 被设置为 `mandatory_roles`，无法授予指定权限	认证 ID `user123`@`localhost` 被设置为 `mandatory_roles`。无法授予 `SELECT` 权限
3898	不支持将复制过滤器与 XA 事务一起使用	–
3899	不支持 `sql_mode`	不支持 `sql_mode=0x00010000`
3900	正则表达式中的匹配模式标志无效	–

3901-4000

错误码	错误描述	示例
3901	数据库的权限同时作为部分权限撤销和 `mysql.db` 中的记录存在	数据库 `test_db` 的 `SELECT` 权限同时作为部分权限撤销和 `mysql.db` 中的记录存在
3902	没有名为某个的度量单位	没有名为 `kilometer` 的度量单位
3903	函数索引的CAST的JSON值无效	函数索引 `idx_json` 的CAST的JSON值无效
3904	函数索引的CAST的JSON值超出范围	函数索引 `idx_number` 的CAST的JSON值超出范围

3905	多值索引的每个记录的值数量超出限制	多值索引 `idx_tags` 的每个记录的值数量超出1000个
3906	多值索引的每个记录的总值长度超出限制	多值索引 `idx_data` 的每个记录的总值长度超出10000字节
3907	函数索引的数据过长	函数索引 `idx_long_text` 的数据过长
3908	不能在索引的标量键部分中存储数组或对象	不能在索引 `idx_json_key` 的标量键部分中存储数组或对象
3909	由于类型或排序规则转换，无法使用函数索引	由于类型或排序规则转换，无法使用函数索引 `idx_name`
3910	函数失败	函数 `ST_Area` 失败。几何对象无效
3911	无法在组复制插件运行时更新 `GTID_PURGED`	–
3912	对于具有夏令时（DST）的时区，对时间的分组是不确定的。请考虑为此查询切换到UTC。	–
3913	长数据库名和对象标识符导致表的路径过长	长数据库名和对象标识符导致表 `very_long_table_name` 的路径过长
3914	忽略对某用户的请求。执行操作需要特定角色	忽略对 `'john'@'localhost'` 的请求。执行操作需要角色： `'admin'`
3915	`audit_log` 密码已复制到角色中，并将在首次清除密码时移除	`audit_log` 密码已复制到 `'auditor'` 中，并将在首次清除密码时移除
3916	开始自动重新加入过程尝试	开始自动重新加入过程尝试1，共3次
3917	在查询期间加载或卸载了插件，系统变量表已更改	–
3918	在查询期间加载或卸载了插件，全局状态变量已更改	–
3919	`START GROUP_REPLICATION` 命令无法启动其消息服务	–

3920	通道的 MASTER_COMPRESSION_ALGORITHMS 无效	通道 channel1 的 MASTER_COMPRESSION_ALGORITHMS 无效
3921	通道的 MASTER_ZSTD_COMPRESSION_LEVEL 无效	通道 channel2 的 MASTER_ZSTD_COMPRESSION_LEVEL 100无效
3922	压缩算法无效	压缩算法 invalid_algo 无效
3923	算法的zstd压缩级别无效	算法 zstd 的zstd压缩级别100无效
3924	指定的压缩算法列表超过了通道的总数	指定的压缩算法列表 zlib,lz4,zstd,snappy 超过了通道 channel3 的总数3
3925	复制通道的 PRIVILEGE_CHECKS_USER 被设置为匿名账户	复制通道 channel4 的 PRIVILEGE_CHECKS_USER 被设置为 @localhost
3926	复制通道的 PRIVILEGE_CHECKS_USER 被设置为不存在的用户	复制通道 channel5 的 PRIVILEGE_CHECKS_USER 被设置为 nonexistent@host
3927	在复制通道的复制配置存储库中发现无效的 PRIVILEGE_CHECKS_USER	在复制通道 channel6 的复制配置存储库中发现无效的 PRIVILEGE_CHECKS_USER
3928	复制通道的 PRIVILEGE_CHECKS_USER 没有 REPLICATION_APPLIER 权限	复制通道 channel7 的 PRIVILEGE_CHECKS_USER 被设置为 user@host ，但此用户没有 REPLICATION_APPLIER 权限
3929	动态权限未在服务器上注册	动态权限 BACKUP_ADMIN 未在服务器上注册
3930	函数失败，因为密钥无效	函数 AES_DECRYPT 失败，因为密钥无效
3931	函数失败，因为密钥类型无效	函数 AES_ENCRYPT 失败，因为密钥类型无效
3932	函数失败，因为密钥过长	函数 KEY_GENERATE 失败，因为密钥过长
3933	函数失败，因为密钥类型过长	函数 KEY_STORE 失败，因为密钥类型过长
3934	JSON SCHEMA VALIDATION 执行错误	-

3935	指定了无效的字符集	指定了无效的字符集 `invalid_charset`
3936	指定了无效的字符集名称	指定了无效的字符集名称 `nonexistent_charset`
3937	指定了无效的排序规则	指定了无效的排序规则 `invalid_collation`
3938	指定了无效的扩展参数类型	指定了无效的扩展参数类型 `invalid_type`
3939	表有多个名为名称的约束，请用特殊的约束子句区分	表有多个名为 `chk_duplicate` 的约束
3940	约束不存在	约束 `chk_missing` 不存在
3941	不支持更改约束的强制状态	不支持更改约束 `pk_primary` 的强制状态
3942	`VALUES` 子句的每一行必须至少有一列	–
3943	`VALUES` 子句不能使用 `DEFAULT` 值	–
3944	查询不符合变量 `require_row_format` 的限制	–
3945	`REQUIRE_ROW_FORMAT` 的请求值无效	`REQUIRE_ROW_FORMAT` 的请求值 `2` 无效，必须为0或1
3946	无法获取用户管理服务的锁，无法确定角色是否为强制角色	无法获取用户管理服务的锁，无法确定角色 `admin` @ `localhost` 是否为强制角色
3947	无法获取用户管理服务的锁，无法获取强制角色列表	–
3948	加载本地数据被禁用	–
3949	导入失败，因为CFG文件版本不兼容	导入 `table` 失败，因为CFG文件版本(1)与当前版本(2)不兼容
395	内存不足	–

0		
3951	字符集只能设置类型为 `STRING` 的 `UDF` 参数	–
3952	字符集只能设置类型为 `STRING` 的 UDF RETURN	–
3953	一个查询块中有多个 `INTO` 子句	–
3954	`INTO` 子句位置错误	–
3955	用户访问被拒绝。账户已被封锁	用户 `'john'@'localhost'` 的访问被拒绝。由于连续5次登录失败，账户已被封锁1天（剩余0.5天）
3956	`YEAR` 数据类型的 `UNSIGNED` 已弃用	–
3957	克隆需要 `max_allowed_packet` 值为特定值或更高	克隆需要 `max_allowed_packet` 值为67108864或更高。当前值为16777216
3958	由于记录缺失，无法在表中操作SDI	由于记录缺失，无法在 `MySQL` 中为 `db1`.`table1` 表sdi
3959	检查约束使用了列，因此无法删除或重命名该列	检查约束 `chk_age` 使用了列 `age`，因此无法删除或重命名该列
3960	此操作无法在组复制运行时执行	–
3961	在 `JSON_TABLE` 列定义中，在 `ON ERROR` 子句后指定 `ON EMPTY` 子句是已弃用语法	–
3962	查询表达式的查询块内的 `INTO` 子句已弃用	–
3963	`VALUES` 函数已弃用	–
3964	`SQL_CALC_FOUND_ROWS` 已弃用	–

3965	`FOUND_ROWS()` 已弃用	–
3966	在指定路径上未找到任何值	在指定路径上未找到 `$.user.name` 的任何值
3967	在指定路径上找到多个值	在指定路径上找到多个 `$.tags` 的值
3968	主机名长度不能超过特定字符数	主机名长度不能超过255个字符
3969	分区函数忽略了列的前缀键部分	分区函数忽略了列 `db1`.`table1`.`col1` 的前缀键部分 `prefix`(10)
3970	`START GROUP_REPLICATION` 命令失败，因为为恢复通道提供的用户名为空	–
3971	`START GROUP_REPLICATION` 命令失败，因为未为恢复通道提供带有 `PASSWORD` 的 `USER` 选项	–
3972	`START GROUP_REPLICATION` 命令失败，因为为恢复通道提供的密码超过了最大长度	–
3973	语句要求将子查询转换为非SET操作	–
3974	恢复套接字端点的输入值无效	恢复套接字端点 `invalid_endpoint` 的输入值无效
3975	服务器未在端点上监听	服务器未在端点 `192.168.1.100:3306` 上监听
3976	变量不能设置为值	变量 `innodb_buffer_pool_size` 不能设置为值 `invalid_value` 。 值过大
3977	在带有 `START TRANSACTION` 语句的 `CREATE TABLE` 之后，只允许特定语句	–
3978	不允许在使用 `CREATE TABLE as SELECT` 和带有 `START TRANSACTION` 语句的 `CREATE TABLE` 时创建外键	–

3979	`START TRANSACTION` 子句不能用于特定语句	`START TRANSACTION` 子句不能用于 `SELECT`
3980	JSON属性无效	JSON属性无效，错误： 无效格式 ，位置5： `"invalid"`
3981	存储引擎不支持 `ENGINE_ATTRIBUTE`	存储引擎 `MyISAM` 不支持 `ENGINE_ATTRIBUTE`
3982	用户属性必须是有效的JSON对象	–
3983	由于InnoDB重做日志记录被禁用，无法执行操作	–
3984	由于InnoDB正在归档重做日志，无法执行操作	–
3985	没有足够的资源来完成锁定请求	–
3986	在SQL布尔上下文中评估JSON值会隐式地与JSON整数0比较	–
3987	函数不支持字符集	函数 `REGEXP` 不支持字符集 `utf16`
3988	在函数中不可能将排序规则转换为另一个	从排序规则 `utf8mb4_general_ci` 转换为 `utf8mb4_bin` 对 `CONVERT` 不可能
3989	SCHEMA处于只读模式	模式 `production_db` 处于只读模式
3990	启用异步复制连接故障转移功能失败，CHANGE MASTER TO SOURCE_CONNECTION_AUTO_FAILOVER = 1 只有在 @@GLOBAL.GTID_MODE = ON 时才能配置	–
3991	启用异步复制连接故障转移功能失败，CHANGE MASTER TO SOURCE_CONNECTION_AUTO_FAILOVER = 1 只有在CHANGE MASTER TO 的	–

	MASTER_AUTO_POSITION 选项生效的情况下才能设置	
3992	无法执行 `@@GLOBAL.GTID_MODE` 更改，因为启用了异步复制连接故障转移	–
3993	无法执行 `CHANGE MASTER TO MASTER_AUTO_POSITION = 0`，因为启用了异步复制连接故障转移	–
3994	在函数中参数的使用无效	在 `JSON_EXTRACT` 中参数的使用无效
3995	在调用函数时，字符集不能结合使用	在调用 `CONVERT` 时，字符集 `utf8mb4` 不能与 `latin1` 结合使用
3996	在此平台上更改变量不受支持	在此平台上更改 `innodb_page_size` 不受支持
3997	无效的时区间隔	无效的时区间隔：`2:30`
3998	无法将值转换为特定类型	无法将值转换为 `DATE`
3999	hypergraph优化器尚不支持特定功能	hypergraph优化器尚不支持 `子查询`
4000	hypergraph优化器是高度实验性的，仅用于测试	–

4001-4051

错误码	错误描述	示例
4001	使用 `--log-error-services=...` 选择的日志接收器中没有提供日志解析器	–
4002	`@@global.log_error_services` 中选择的日志接收器不支持写入性能模式	–
4003	`@@global.log_error_services` 列出了多个日志过滤器服务	–

4004	无法打开文件进行错误记录	无法打开文件 `/var/log/mysql/error.log` 进行错误记录
4005	用户在特定上下文中被引用为定义者账户	用户 `admin@localhost` 在 存储过程 中被引用为定义者账户
4006	对用户执行操作失败，因为它在特定上下文中被引用为定义者账户	对 `john@localhost` 执行操作 `DROP USER` 失败，因为它在视图中被引用为定义者账户
4007	正则表达式中的十进制数太大	–
4008	变量具有非整数基础类型	变量 `"max_connections"` 具有非整数基础类型
4009	不支持对 ACL 表进行可序列化隔离/ `SELECT FOR SHARE` 读取	–
4010	该语句不安全，因为它更新了依赖于 ACL 表读取操作的表	–
4011	`STOP REPLICA` 命令执行未完成：Replica Monitor 线程已收到停止信号	–
4012	Replica Monitor 线程启动失败	–
4013	无法启动复制，因为此服务器使用 `@@GLOBAL.GTID_MODE <> ON`	无法启动 `channel1` 复制，因为此服务器使用 `@@GLOBAL.GTID_MODE = OFF`
4014	`ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 和 `MASTER_AUTO_POSITION = 1` 不能一起使用	–
4015	无法执行 `CHANGE MASTER TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`	无法执行 `CHANGE MASTER TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = LOCAL`，因为 `@@GLOBAL.GTID_MODE = OFF`
4016	`sql_replica_skip_counter` 的值仅对使用 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS <> OFF` 运行的通道生效	–
4017	使用 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 会给复制拓扑带来限	–

	制	
4018	不能在配置了 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 的通道上使用 `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS`	-
4019	当复制通道配置了 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 时，不能使用 `START REPLICA` 的特定子句	-
4020	复制配置了 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`，其中 UUID 值等于 `group_replication_group_name`	复制 `channel2` 配置了 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = uuid123`，其中 UUID 值等于 `group_replication_group_name`
4021	无法执行 `CHANGE MASTER TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`，因为 UUID 值等于 `group_replication_group_name`	-
4022	`group_replication_recovery` 通道仍在运行，可能正在等待数据库/表锁	-
4023	`AUTOEXTEND_SIZE` 应为特定值的倍数	`AUTOEXTEND_SIZE` 应为 64M 的倍数
4024	InnoDB: 通用表空间不允许使用特定选项	InnoDB: 通用表空间不允许使用 `COMPRESSED`
4025	`AUTOEXTEND_SIZE` 值应在特定范围内	`AUTOEXTEND_SIZE` 值应在 1M 和 1024M 之间
4026	`AUTOEXTEND_SIZE` 子句对特定表空间无效	`AUTOEXTEND_SIZE` 子句对 `TEMPORARY` 表空间无效
4027	用户账户被直接或间接授予角色，此 GRANT 操作将创建一个循环	用户账户 `user1@localhost` 被直接或间接授予角色 `role1`
4028	表必须至少有一个可见列	-
4029	压缩失败	压缩失败，错误如下：数据损坏

4030	参数 `network_namespace` 保留供将来使用	–
4031	客户端因不活动被服务器断开连接	–
4032	从特定类型到特定类型的转换无效	从 `POINT` 到 `POLYGON` 的转换无效
4033	从特定类型到特定类型的转换无效，多边形环的方向错误	从 `LINESTRING` 到 `POLYGON` 的转换无效
4034	函数参数包含具有不同 SRID 的几何对象	函数 `ST_Intersects` 的参数包含具有不同 SRID 的几何对象：4326 和 3857
4035	密钥环重新加载失败	–
4036	表空间对于 SDI 操作无效	表空间 `system_tablespace` 对于 SDI 操作无效
4037	为通道设置 `SOURCE_COMPRESSION_ALGORITHMS` 选项的字符串长度过长	为通道 `channel3` 设置 `SOURCE_COMPRESSION_ALGORITHMS` 选项的字符串长度 1000 过长
4038	通道启用了已弃用的 TLS 版本	通道 `channel4` 启用了已弃用的 TLS 版本 `TLSv1.0`
4039	无法执行 `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`	无法执行 `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = uuid456`，因为 UUID 值等于 `group_replication_view_change_uuid`
4040	复制配置了 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`，其中 UUID 值等于 `group_replication_view_change_uuid`	复制 `channel5` 配置了 `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = uuid456`，其中 UUID 值等于 `group_replication_view_change_uuid`
4041	在复制操作的服务器检查期间无法检索 `group_replication_view_change_uuid` 变量	–
4042	`audit_log_max_size` 和 `audit_log_prune_seconds` 都设置为非零	–

4043	`audit_log_rotate_on_size` 对于提供的 `audit_log_max_size` 值来说不够精细	`audit_log_rotate_on_size` 对于提供的 `audit_log_max_size` 值来说不够精细。应至少小10倍
4044	创建/更改用户失败，配置的用户域作为身份验证字符串为空	-
4045	当特定插件已安装时，无法安装另一个插件	当 `keyring_file` 插件已安装时，无法安装 `keyring_okv` 插件
4046	错误记录组件刷新失败	2 个错误记录组件刷新失败
4047	当 `GTID_MODE = ON` 且 `SOURCE_AUTO_POSITION=1` 时，为通道使用了 `SQL_AFTER_MTS_GAPS` 直到子句	-
4048	`INSERT` 或 `UPDATE` 语句中的赋值目标无效	`INSERT` 或 `UPDATE` 语句 `UPDATE users SET invalid_col=?` 中的赋值目标无效
4049	此操作不能在组复制次要成员上执行	-
4050	无法将通道的 `SOURCE_CONNECTION_AUTO_FAILOVER` 值传播到组复制成员	无法将通道 `channel6` 的 `SOURCE_CONNECTION_AUTO_FAILOVER` 值传播到组复制成员
4051	`audit_log_format_unix_timestamp` 仅当 `audit_log_format = JSON` 时适用	-

7000-7030

错误码	错误描述	示例
7000	无法执行查询，因为您有一个冲突的备份锁	-
7001	压缩列不允许出现在键列表中	压缩列 `compressed_data` 不允许出现在键列表中
7002	无法以压缩格式定义列	无法以压缩格式定义列 `text_column`

7003	压缩字典已存在	压缩字典 `my_dictionary` 已存在
7004	压缩字典不存在	压缩字典 `missing_dict` 不存在
7005	压缩字典名称过长	压缩字典名称 `very_long_dictionary_name` 过长（最大长度 = 64）
7006	压缩字典的数据过长	压缩字典 `large_dict` 的数据过长（最大长度 = 65536）
7007	压缩字典正在使用中	压缩字典 `active_dict` 正在使用中
7008	如果不停止服务器并重新创建重做日志，则无法切换重做日志加密模式	如果不停止服务器并重新创建重做日志，则无法切换重做日志加密模式。当前模式为 `ENCRYPTED`，请求为 `UNENCRYPTED`
7009	重做日志密钥生成失败	–
7010	加载重做密钥版本失败	加载重做密钥版本2失败
7011	无法获取新生成的重做密钥	–
7012	无法解析系统密钥	无法解析系统密钥：格式错误
7013	如果密钥环未加载，则重做日志无法加密	–
7014	如果密钥环未加载，则撤销日志无法加密	–
7015	表已加密但解密失败，可能是密钥环密钥不正确	表 `users` 已加密但解密失败。似乎从密钥环获取的加密密钥不正确。您使用的是正确的密钥环吗?
7016	表空间已被加密线程加密，不能从加密线程中排除	表空间 `secure_tablespace` 已完全或部分被加密线程加密，因此不能从加密线程中排除。请先完成加密/解密操作，然后再试。
7017	表空间已被加密线程加密，无法显式解密	表空间 `encrypted_tablespace` 已被加密线程加密，无法显式解密。请使用 `default-table-encry`

		`ption=ONLINE_FROM_KEYRING_TO_UNENCRYPTED` 来解密该表空间。
7018	表空间已被加密线程加密，无法 ALTER 为 Master Key 加密	表空间 `db_tablespace` 已被加密线程加密，无法 ALTER 为 Master Key 加密。要使用 Master Key 加密此表空间，请先使用加密线程解密此表空间。
7019	当 `default_table_encryption` 设置为 ONLINE_TO_KEYRING 时，不能将数据库默认加密设置为 Y（Master Key 加密）。如果打算重新加密为 KEYRING 加密，则创建新的 Master Key 加密表没有意义。	-
7020	当 `default_table_encryption` 设置为 ONLINE_TO_KEYRING 时，不能将数据库默认加密设置为 N（显式请求数据库中的表不加密）。如果要在创建表时排除表被加密线程加密，您仍然可以这样做。为此，您需要在创建表时显式指定 `ENCRYPTION='N'` 。	-
7021	表空间的 ENCRYPTION 无法更改，因为它当前正被加密线程加密/解密	表空间 `active_tablespace` 的 ENCRYPTION 无法更改，因为它当前正被加密线程加密/解密。请稍后重试。
7022	加密选项不被允许	加密选项不被允许。
7023	SEQUENCE_TABLE() 生成的记录数不能超过限制	SEQUENCE_TABLE() 生成的记录数不能超过 1000000（请求了 2000000）。请尝试将 `@@tf_sequence_table_max_upper_bound` 增加到更大的值。
7024	InnoDB 系统密钥无法轮换，因为它不存在	InnoDB 系统密钥5无法轮换，因为它不存在
7025	InnoDB 系统密钥的值不能大于 4294967294	-
7026	无法生成系统密钥的新版本	无法生成系统密钥3的新版本。密钥环报告错误。请检查密钥环是否可用。
7027	要更改 default_table_encryption，请禁用加密线程。请将	-

	innodb_encryption_threads 的数量设置为 0。	
7028	请求的加密密钥 ID 值超过了允许的最大值	请求的加密密钥 ID 值 10000000000 超过了允许的最大值 4294967294
7029	函数中的包装 UDF 异常	函数 `custom_function` 中的包装 UDF 异常；内存分配失败
7030	启用特定选项时不支持多表删除	启用 `foreign_key_checks` 选项时不支持多表删除

8000-8019

错误码	错误描述	示例
8000	批量加载期间键顺序错乱	-
8001	批量加载行与现有行重叠	-
8002	当 `binlog_format != ROW` 时，无法在主库上执行更新	-
8003	MyRocks 仅支持 `READ COMMITTED` 和 `REPEATABLE READ` 隔离级别。请修改当前隔离级别。	MyRocks 仅支持 `READ COMMITTED` 和 `REPEATABLE READ` 隔离级别。请修改当前隔离级别 `READ UNCOMMITTED` 。
8004	在 MyRocks 中禁用唯一性检查时，不允许执行带有更新或替换键的子句的 INSERT、UPDATE、LOAD 语句（例如 INSERT ON DUPLICATE KEY UPDATE、REPLACE）。	在 MyRocks 中禁用唯一性检查时，不允许执行带有更新或替换键的子句的 INSERT、UPDATE、LOAD 语句（例如 INSERT ON DUPLICATE KEY UPDATE、REPLACE）。查询：`INSERT INTO users VALUES (1,'john') ON DUPLICATE KEY UPDATE name='john'`
8005	当使用 `START TRANSACTION WITH CONSISTENT [ROCKSDB] SNAPSHOT` 启动事务时，无法执行更新	-
8006	此事务已回滚且无法提交。唯一支持的操作是回滚，因此所有挂起的更改将被丢弃。请重新启动另	-

	一个事务。	
8007	如果修改了行，MyRocks 目前不支持 `ROLLBACK TO SAVEPOINT`	–
8008	对于 RocksDB 存储引擎中的 `START TRANSACTION WITH CONSISTENT SNAPSHOT`，仅支持 `REPEATABLE READ` 隔离级别	–
8009	字符串索引列使用了不支持的排序规则	字符串索引列 `users.name` 使用了不支持的排序规则。请使用二进制排序规则 (`utf8_bin`)
8010	表不存在，但 MyRocks 内部存在元数据信息。这是数据不一致的迹象。	表 `temp_table` 不存在，但 MyRocks 内部存在元数据信息。这是数据不一致的迹象。
8011	MyRocks 在修改期间创建新的键定义失败	–
8012	MyRocks 在修改期间填充二级索引失败	–
8013	列族的标志与现有标志不同。请分配一个新的 CF 标志，或者不要更改现有的 CF 标志。	列族 (`cf1`) 的标志 (2) 与现有标志 (1) 不同
8014	当表具有隐藏主键时，TTL 支持当前被禁用	–
8015	MyRocks 中的 TTL 列必须是无符号非空 64 位整数，存在于表中，并且具有相应的 ttl 持续时间。–	MyRocks 中的 TTL 列 (`expire_time`) 必须是无符号非空 64 位整数，存在于表中，并且具有相应的 ttl 持续时间。
8016	MyRocks 中的 TTL 持续时间必须是无符号非空 64 位整数	MyRocks 中的 TTL 持续时间 (`ttl_duration`) 必须是无符号非空 64 位整数
8017	每个索引的列族选项已被弃用	–
8018	列族正在被删除	列族 (`old_cf`) 正在被删除
8019	无法删除列族，因为它不存在或正在被使用	无法删除列族 (`active_cf`)，因为它不存在或正在被使用

8500-8600

错误码	错误描述	示例
8500	TDSQL一般错误信息	连接错误，错误代码：`CONNECTION_FAILED(1001)`，错误信息：无法连接到服务器
8501	TDSQL 仅支持 TDStore(RocksDB) 存储引擎	-
8502	由于写围栏错误导致查询失败	由于写围栏错误导致查询失败。表(`db1.users`)，tindex_id：`123`，发送的写围栏：`5`，当前写围栏：`6`
8503	正常的 DDL 进程被 DDL 恢复线程中断，请稍后重试。	-
8504	更新最新表版本失败	更新最新表(`db1.orders`)版本失败，本地 sv：`100`，全局 sv：`101`
8505	表版本太旧	表(`db1.products`)-tindex_id(`456`) 版本太旧：`50`，当前版本：`60`
8506	其他 SQLEngine 或 DDL 恢复线程中正在进行冲突的 DDL 作业	其他 SQLEngine 或 DDL 恢复线程中正在进行冲突的 DDL 作业（id 123456）：数据库是 `db1`，表是 `users`。请等待并稍后重试。
8507	创建表失败，元数据丢失	创建表 `db1.temp_table` 失败，因为在创建表时元数据丢失。请等待并稍后重试。
8508	注册自动递增 IndexId 重复	注册自动递增 IndexId 100重复
8509	IndexId 已耗尽	-
8510	SQLEngine 系统错误	SQLEngine 系统错误，内存分配失败
8511	自动递增 IndexId 未注册	自动递增 IndexId 200未注册

8512	TDStore 无法设置迭代器对象	TDStore 无法设置迭代器对象。错误代码：`500`，错误信息：迭代器初始化失败
8513	无法从 TDStore 分配自动递增值	无法从 tdstore 分配自动递增值，请求 ID：`1000`，错误代码：`600`，错误信息：分配失败
8514	未定义自增字段但给出了自增值	–
8515	TDStore 找不到事务上下文。请查看错误日志获取更多信息。	TDStore 找不到事务上下文。请查看错误日志获取更多信息。错误代码：`700`，错误信息：事务未找到
8516	TDStore 找不到事务上下文，可能由于主节点转移导致。请查看错误日志获取更多信息。	TDStore 找不到事务上下文，可能由于主节点转移导致。请查看错误日志获取更多信息。错误代码：`800`，错误信息：主节点切换
8517	初始化全局区域缓存失败。请查看错误日志获取更多信息。	–
8518	用户没有权限设置变量	用户 `'john'@'localhost'` 没有权限设置变量 `max_connections`
8519	函数 STORAGE_FORMAT 接受 uint32_t 类型的数值 – 例如，STORAGE_FORMAT(10001)。	–
8520	事务提交失败并被 tdstore 中止	事务提交失败并被 tdstore 中止。错误代码：`900`，错误信息：冲突检测，协调者：`node1`
8521	事务必须被中止。请查看错误日志获取更多信息	–
8522	集群未引导。请检查错误日志或元集群	–
8523	初始化事务保活线程失败	–
8524	RocksDB 引擎和其他引擎不能相互转换	–
8525	不允许对系统权限表执行原始 DML 操作	不允许对系统权限表 `mysql.user` 执行原始 DML 操作
8526	不可用的表，请重新启动事务。	不可用的表 `db1.temp_table`。表正在重建

8527	关联表错误	关联表错误（错误代码：`1000`，错误信息：关联失败）
8528	在线更改表失败，如果更改期间不接受写入，请将变量 'tdsql_use_online_copy_ddl' 设置为 'false'。	在线更改表 `mydb.users` 失败，原因：'锁等待超时'，如果更改期间不接受写入，请将变量 'tdsql_use_online_copy_ddl' 设置为 'false'
8529	本地缓存中的持久化变量已过时，设置持久化选项失败，请重试。	_
8530	不允许对指定变量使用 SET PERSIST_ONLY 语法，因为它不是只读变量	不允许对变量 `max_connections` 使用 SET PERSIST_ONLY 语法，因为它不是只读变量
8531	当前不允许启动新的 DDL 作业	当前不允许启动新的 DDL 作业，因为系统维护中
8532	DDL 作业包含过多信息，请拆分此作业	DDL 作业（id 123456）包含过多信息，请拆分此作业。
8533	LOCK/UNLOCK 选项仅用于兼容性	_
8534	原生存储过程参数不匹配	原生存储过程 `calculate_stats` 的第2个参数不匹配
8535	无法添加固定执行计划规则	无法添加固定执行计划规则：规则冲突
8536	无法删除固定执行计划规则	无法删除固定执行计划规则：规则不存在
8537	无法启用或禁用固定执行计划规则	无法启用或禁用固定执行计划规则：规则无效
8538	规则表中的固定执行计划规则无效	规则表中的固定执行计划规则 123456 无效：格式错误
8539	刷新固定执行计划规则时出错	刷新固定执行计划规则时出错：内存不足
8540	语句应用规则失败	语句应用规则 123456 失败：解析错误
8541	为其他语句刷新规则时出错	为其他语句刷新规则时出错：网络错误

8542	从 MC 获取对象锁等待超时	从 MC 获取对象锁 `table_lock` 等待超时
8543	路由刷新失败	–
8544	显示路由失败	–
8545	创建数据对象失败	创建数据对象失败。权限不足
8546	并行查询工作线程启动失败	并行查询工作线程启动失败（错误号：100 – 资源不足）
8547	并行度无效	–
8548	此语句不支持 PARALLEL 提示	–
8549	RPC 应用流失败	RPC 应用流失败。连接超时
8550	获取或更改分布策略失败	–
8551	强制优化 INSERT/REPLACE/LOAD 为 BatchPut	强制优化 INSERT/REPLACE/LOAD 为 BatchPut。数据量过大
8552	并行查询运算符发送或接收行失败，因为行太大	–
8553	事件服务无法添加或移除事件	事件服务无法添加或移除事件。（错误号：200 – 事件已存在）
8554	并行查询运算符发送或接收行失败	并行查询运算符发送或接收行失败。（错误号：300 – 传输错误）
8555	并行查询运算符读取行失败，因为遇到了未知的帧类型	–
8556	检查事务存活失败	–
85	extract_int_from_str 使用错误	extract_int_from_str 使用错误。格式无效

57		
8558	由于日志服务限制，目前不允许操作包含多表	由于日志服务限制，目前不允许 DELETE 操作包含多表
8559	并行查询远程工作线程错误	并行查询远程工作线程错误。节点故障
8560	并行查询远程执行器错误	并行查询远程执行器错误。执行超时
8561	过时读取仅支持非锁定的只读自动提交事务	过时读取仅支持非锁定的只读自动提交事务。事务模式不匹配
8562	过时读取获取快照失败	过时读取获取快照失败。快照不可用
8563	尚未支持在 LOCK TABLE 语法之后删除表	–
8564	更新批量加载模式设置失败	更新批量加载模式设置失败。配置冲突
8565	更新 tdstore 选项失败	更新 tdstore 选项失败。权限不足
8566	更新 tdstore 选项警告	更新 tdstore 选项警告。性能影响
8567	超节点 ID 已被节点缓存	–
8568	加锁失败	–
8569	无法在数据库中创建/移动表	无法在数据库中创建/移动表。数据库(`db1`)和表(`db2.users`)位于不同的数据空间中
8570	TDSQL 标识类型不匹配	TDSQL 标识类型不匹配，需要整数但得到字符串
8571	参数数量不正确	参数 `table_name` 的数量不正确，需要 `1` 但得到 `0`
8572	参数数量不正确	参数 `columns` 的数量不正确，需要多于 `0` 但得到 `0`

8573	未知参数	未知参数：`invalid_param`
8574	提交作业失败	提交作业失败：`job_123`，错误信息：资源不足
8575	清除 Raft 日志失败	清除 Raft 日志失败：`raft_group_1`
8576	并行工作线程之一报告错误	–
8577	创建分区策略失败	创建分区策略失败。策略冲突
8578	绑定分区策略失败	绑定分区策略失败。表不存在
8579	语句大纲规则已应用，可能已更改查询计划	语句大纲规则123456已应用，它可能已更改查询计划
8580	刷新内存表失败	刷新内存表失败：内存不足
8581	在线更改表失败，当 `tdsql_ddl_fillback_mode='BulkLoadLock'` 时，不支持 `ALGORITHM=COPY`，请将变量 `'tdsql_ddl_fillback_mode'` 设置为其他值	在线更改表 `db1.users` 失败，当 `tdsql_ddl_fillback_mode='BulkLoadLock'` 时，不支持 `ALGORITHM=COPY`，请将变量 `'tdsql_ddl_fillback_mode'` 设置为其他值
8582	不允许更改某些集群全局变量	不允许更改某些集群全局变量：`cluster_mode`
8583	并行工作线程之间的交换记录格式不匹配	–
8584	在回收站中未找到相应记录	–
8585	超出回收站大小限制	超出回收站大小限制1000000
8586	不能将回收站中的表与其他表一起删除	–
8587	回收站中发生错误	回收站中发生错误：存储空间不足

8588	读取回收站信息失败	-
8589	回收站写入有关已删除表的记录失败	-
8590	未找到数据库	未找到 `temp_db` 数据库
8591	未找到 I_S 视图	未找到 I_S 视图 `information_schema.users`
8592	丢失 I_S 视图	丢失了 5 个 I_S 视图
8593	转储 I_S 视图失败	在 `/tmp/views.sql` 中转储 I_S 视图失败
8594	无法创建带有分区的同步/广播表	-
8595	过时读取意外错误	过时读取意外错误。内部错误
8596	未知的分区策略	未知的分区策略 `invalid_policy`
8597	删除分区策略失败	删除分区策略 `old_policy` 失败
8598	分区策略已存在	分区策略 `existing_policy` 已存在
8599	自增值超过最大值	-
8600	路由不存在。可能存在并发的 DDL 销毁数据对象	-

8601-8613

错误码	错误描述	示例
8601	强制关闭线程。应关闭用户连接	强制关闭线程12345。应关闭用户连接。会话超时

8602	代理错误	代理错误。连接池耗尽
8603	无法升级	系统维护中，现在无法升级
8604	流式加载数据失败	流式加载数据失败，数据格式错误，错误代码：1001
8605	事务写入大小超过限制，修改 `tdsql_max_memory_per_transaction_bytes` 以增加限制	-
8606	Ingestbehind 子工作线程失败	Ingestbehind 子工作线程失败，错误地址：`node1:8080`，子工作线程 ID：5678，错误代码：2001，错误信息：内存不足
8607	无效的代理节点 ID	无效的代理节点 ID，可用节点 ID：`node1,node2,node3`
8608	更改分区策略失败	更改分区策略失败。策略冲突
8609	当前查询禁用了固定执行计划规则	当前查询禁用了固定执行计划规则：`SELECT * FROM users WHERE id=1`
8610	表的模式状态	表 `users` 的模式状态是 `MODIFYING`
8611	表的键的模式状态	表 `orders` 的键 `pk_order` 的模式状态是 `INVALID`
8612	在 CDC 节点中不支持执行 set persist	-
8613	刷新数据字典缓存对象失败	刷新数据字典缓存对象失败：缓存损坏