云数据库 MariaDB

最佳实践

产品文档

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。

文档目录

最佳实践
编程与使用规范

最近更新时间：2021-08-23 11:43:45

1 引言

1.1 文档目的

本文档旨在明确 MariaDB 数据库技术与应用的规划与设计、安装与部署、运行管理与维护等全生命周期各阶段的主要技术标准和指导原则，便于 MariaDB 数据库应用项目的统一建设和管理，增加 MariaDB 数据库技术应用的规范性、性能保障和可维护性。

1.2 预期读者

使用 MariaDB 的项目相关数据库设计、开发管理和运行维护人员。

2 设计规范

2.1 数据库设计原则

- MariaDB 面向的是 OLTP 的应用场景，并不适用于大多数复杂的 OLAP。
- 反范式设计：
  不必强制满足第三范式，尽量少使用外键。
  - 外键用来保护参照完整性，可在业务端实现。
  - 适度的冗余设计，减少多表join查询，更适应 MPP 架构的横向扩展能力。
  - 直接基于 I/O 和查询进行优化。
- 充分考虑业务逻辑和数据分离，数据库只作为一个保证 ACID 特性的关系数据的持久化存储系统，尽量避免使用自定义函数、存储过程、触发器和视图。
- 充分考虑数据库整体安全设计，数据库管理和使用人员权限分离。
- 充分考虑具体数据对象的访问频度及性能需求，结合主机、存储等需求，做好数据库性能设计。
- 充分考虑数据增长模型，决策是否采用“分布式（水平拆分）”模式。
- 充分考虑业务数据安全等级，设计合适的备份和恢复策略。

2.2 数据库模型设计规范

2.2.1 基础规范

- 只使用 InnoDB 存储引擎，避免使用 MyISAM 引擎，二者对比，InnoDB 具有如下特性：
  - 完整的 ACID 支持。
  - 崩溃自检恢复。
  - 行级锁，高并发的保证。
  - 更能发挥多核 CPU 的性能。
  - 自带缓存池，更好的利用内存。
- 所有表使用统一的字符集，建议使用 UTF8 或 UTF8MB4 字符集。
- 不在数据库中存储图片、二进制文件等大数据。
- 提前规划好单表规模、行数和大小。
- 控制单行字段总长度，合理设置 innodb_page_size，尽量采用 COMPACT 行格式，避免行溢出。

2.2.2 数据库对象命名规范

数据库对象命名规范的适用范围为使用 MariaDB 的数据库设计、管理、开发人员，对于 MariaDB 产品自带的系统库表等对象不在本规范约束范围内。

数据库对象命名时整体遵循如下规范：

- 对象命名要使用富有意义英文词汇，除约定俗成外，避免使用缩写。
- 在索引约束等对象命名时会同时包含表名、字段名等多个名字，此时可以使用缩写，缩写规则和字符数要统一。
- 只使用小写字母、数字和下划线的组合。
- 名字长度不要超过32个字符。
- 不要使用 SQL 关键字。
- 对象名字至少包括：对象类型、父对象名、对象名。

下表列出了不同数据库对象的命名规则：

数据库对象	格式	样例	说明
数据库（SCHEMA）	db_<数据库名>	db_user	表示这个数据库下都是用户相关的数据
表	tbl_<表名>	tbl_employees	表示存储雇员信息的表
主键	pk_<表名>_<字段名>[_字段名]	pk_emlo_id_name	表示 tbl_employees 表的 id 和 name 字段共同组成主键。但是在 MariaDB，这里的主键名其实没有意义，系统会忽略这个名字，然后统一命名为：PRIMARY
唯一索引	uidx_<表名>_<字段名>[_字段名]_<编号>	uidx_empl_name_age_1	表示 tbl_employees 表的 name 和 age 字段创建的第一个唯一索引
普通索引	idx_<表名缩写>_<列名缩写>[_列名缩写]_<编号>	idx_empl_name_age_1	表示 tbl_employees 表的 name和 age 字段创建的第一个普通索引
外键	fk_<表名>_<关联表名>_<字段名>[_字段名] _<关联字段名>[_关联字段名]	fk_empl_user_uid_id	表示 tbl_employees 表的 uid 字段外键约束到 tbl_user 的 id 字段
视图	v_<视图名>	v_female	-
存储过程	sp_<存储过程名>	sp_add_empl	-
自定义函数	f_<函数名>	f_count_empl	-
触发器	trg_<触发器名>	trg_update_empl	-
临时表	tmp_<表名>	tmp_latecomer	-

2.2.3 实例配置

MariaDB 实例在申请后需要初始化，初始化过程中有2个重要指标需要指定：

- 字符集：指定库表默认的字符集。
- innodb_page_size：InnoDB 表文件存储中页的大小。页是 InnoDB 引擎在文件中存储的最小单位，原生 MySQl 的默认值为 16K，该值的配置从以下几点考虑：

- 实例中主要表的单行数据大小。
- InnoDB 表的ROW_FORMAT配置。
- 是否使用 SSD 盘。

在初始化后，还有些参数可以设置：

- 根据业务需要，可通过 SQL MODE 设置STRICT_ALL_TABLES或STRICT_TRANS_TABLES标志位开启 STRICT 模式，严格拒绝不符合字段类型定义的数据写入，两个标志位的区别：
  - STRICT_ALL_TABLES：对于不支持事务的存储引擎，如果同时更新多行数据，如果不符合定义的数据不是第一行，则会导致该行前面的行数据更新，该行及后续行数据不更新的情况。
  - STRICT_TRANS_TABLES：对于不支持事务的存储引擎，如果不符合定义的数据是第一行，则拒绝请求，否则按照普通模式运行。
- MariaDB 默认会启动事务自动提交（AUTOCOMMIT），也建议保持开启。
- MariaDB 默认的事务隔离级别是 REPEATABLE-READ，请根据需要决定是否调整为 READ-COMMITED（ORACLE 的默认级别）。

2.2.4 数据库

MariaDB 中数据库的概念和 ORACLE 不同，可以认为是一个 SCHEMA，类似于一个表的文件夹，方便 DBA 的分类管理。创建数据库时，除了指定数据库名，强烈建议明确指定数据库默认的字符集参数，例如：

```
CREATE DATABASE `db_user` CHARSET ='utf8' COLLATE='utf8_bin'
```

2.2.5 表

2.2.5.1 表参数设置

- InnoDB 表的默认ROW_FORMAT为 COMPACT，请根据情况指定合适的值：
  - COMPACT：在不产生行溢出的情况下，一行数据存储在一个数据页中，数据读取效率最好。
  - DYNAMIC：至少存储在2个页中，数据页只存储溢出页的指针，数据存在溢出页中，这种结构的索引存储最集中，索引访问效率较高。
  - REDUNDANT：老的格式，不要使用。
  - COMPRESSED：会对数据进行压缩后存储，最高的空间利用率，不过会增加 CPU 和内存消耗，增加系统响应时间，降低吞吐量，谨慎使用。
- 建议明确指定表默认的字符集参数。
- 文件存放路径等物理相关的参数是不允许指定的。

2.2.5.2 主外键、约束设计

- InnoDB 表必须指定主键。
- 建议不使用具有实际意义的字段做主键，如果一定要使用，确保该字段具有现实世界的唯一且不变性（例如：银行交易流水号），以尽量避免主键修改。
- 减少外键的使用，特别是在“分布式（水平拆分）”模式下，考虑到数据库的自动分库分表，建议不使用外键。
- 组成主键的字段总长度越短，效率越高。整数类型的字段做主键最合适。
- 如果存在多个唯一键，考虑最常用的唯一键作为主键。
- 建议使用自增字段作为主键。
- 字段属性尽量加上 NOT NULL 约束以及默认值，使用NULL值会导致如下副作用：
  - 含义不明，对很多运算符不管用，增加复杂度。例如 a!=5 是无法匹配到 a 为 NULL 的行。
  - 很难进行查询优化。

- 含 NULL 的复合索引失效。
- 不要使用 CHECK 约束，MariaDB 虽然不报错，但会忽略，可以使用 ENUM 类型代替。

2.2.5.3 字段类型设计

- 整数类型请根据实际存储值的范围选择合适的类型。
- 无符号整数类型请添加 UNSIGNED 关键字。
- 字符串类型请根据实际存储值的情况确定选择定长类型还是变长类型。
- 根据实际存储值的情况，尽量减少字符串类型的长度。
- 尽量避免使用 TEXT/BLOB 类型，不要在数据库中存储图片、二进制文件等数据。
- 非整数尽量使用精确的 DECIMAL，避免使用浮点数。

下表是推荐使用的主要数据类型取值范围和存储需求：

类型	值范围	存储需求
TINYINT[(M)]	-128到127 或 0到255	1个字节
SMALLINT[(M)]	-32768到32767 或 0到65535	2个字节
INT[(M)]	-2147483648到214748364或 0到4294967295	4个字节
BIGINT[(M)]	-9223372036854775808到92233720368547758O7或 0到18446744073709551615	8个字节
DECIMAL[(M[,D])]	整数最大位数（M）为65，小数位数最大（D）为30	变长
DATE	'1000-01-01' 到 '9999-12-31'	3个字节
TIME[(<microsecond precision>)]	'-838:59:59.999999'到'838:59:59.999999'	5个字节
DATETIME[(microsecond precision)]	'1000-01-01 00:00:00.000000'到'9999-12-31 23:59:59.999999'	8个字节
TIMESTAMP[(<microsecond precision)]	'1970-01-01 00:00:01' (UTC)到'2038-01-19 05:14:07' (UTC)	4个字节
CHAR[(M)]	0<M<=255	M 的整数倍，和字符集设置有关
VARCHAR[(M)]	0<M<65532/N	N 的取值和实际字符集设置有关
TEXT[(M)]	0<M<65535/N	N 的取值和实际字符集设置有关

2.2.6 索引

- 低选择性的列不加索引，如：性别。
- 在组合索引中，常用的字段放在前面，选择性高的字段放在前面。
- 需要经常排序的字段可加到索引中，字段顺序和最常用的排序一致。
- 对于较长的字符串类型字段，建议使用前缀索引。
- 合理合并索引，避免冗余，例如 (a,b) 和 (a)，应该去掉 (a)。

- 单张表的索引控制在5个以内。
- 单个索引的字段不要超过5个。
- MariaDB 的 InnoDB 引擎支持全文索引，但目前限于英文。

2.2.7 分页设计

分页是应用中最常见的访问模型，我们用下面几种分页方式的实际测试情况来看如何设计合理的分页模型：

```
/*
* id 是表 post 的主键
*/
MySQL> SELECT sql_no_cache * FROM post LIMIT 20000,10;
10 row in set (0.13 sec)

MySQL> SELECT sql_no_cache * FROM post LIMIT 80000,10;
10 rows in set (0.58 sec)

MySQL> SELECT sql_no_cache id FROM post LIMIT 80000,10;
10 rows in set (0.02 sec)

MySQL> SELECT sql_no_cache * FROM post WHERE id>=323423 LIMIT 10;
10 rows in set (0.01 sec)

MySQL> SELECT * FROM post WHERE id >= ( SELECT sql_no_cache id FROM post LIMIT 80000,1 ) LIMIT 10 ;
10 rows in set (0.02 sec)
```

上面的结果很明显，要尽量避免直接使用 LIMIT m,n 这种分页方式。

2.3.数据库安全设计

- 数据库用户安全设计原则。
  - 数据库用户权限授权按照最小分配原则。
  - 数据库用户要分为管理、应用、维护、备份四类用户。
- 数据库用户权限设计规范。
  - 除 MariaDB 和核心维护人员外，其他用户不能拥有 SUPER 权限账号。
  - 避免使用简单密码。
  - MariaDB 的权限支持到字段级别，权限的主体对象要按最小原则控制。
  - 开发、测试和生产环境中用户权限设置要保持一致。
- 严格禁止在数据库中存储任何形式的密码明文。

3 开发规范

3.1 SQL 编码规范

- 每行不要超过80个字符，超过就换行缩进。
- 使用两个空格来缩进代码。
- 关键词要大写，例如：SELECT。

- 常数符号要大写，例如：NULL。
- 合理使用注释。
  - 建表语句前要对表的用途进行详细注释。
  - 每个字段后使用 COMMNET 子句添加字段的注释。
  - 建表语句最后面使用 COMMENT 子句添加对表的一句话注释。
- 去掉多余的扩容，例如：

```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
```

- 应该优化为：

```
(a AND b AND c) OR (a AND b AND c AND d)
```

- 去掉重叠的条件，例如：

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
```

- 应该优化为：

```
B=5 OR B=6
```

3.2 SQL 语句规范

3.2.1 索引和分区

- 合理使用 USE INDEX 和 IGNORE INDEX 进行索引的选择。
- 查询条件尽量使用索引。
- 注意字段类型，避免类型转换。类型转换除了会增加 CPU 消耗，如果转换失败，还会导致索引失效。
- 对于复合索引，查询条件必须包含所有前缀字段才管用，例如索引 (a,b,c)，查询条件必须是 a 或者 a、b 或者 a、b、c 才能使用到索引。

3.2.2 SELECT 列和 WHERE 条件

- 尽量不要让数据库做算术运算，交给应用层来做，例如：

```
SELECT a FROM tbl WHERE id*10=100;
```

- 尽量不要直接 SELECT *，直接列出需要查询的字段。
- 尽量使用 UNION ALL，而不是 UNION。UNION 会做去重和排序。
- WHERE 子句使用的原则：尽量使用索引，尽量简单，尽量匹配更少的行。
- WHERE 子句中多使用等值操作符，少使用非等值操作符，非等值操作符通常会导致索引失效（MySQL 不支持范围索引）。
- 就算有索引，WHERE 子句匹配的行数不要超过表的30%，否则效率还是很低，InnoDB 引擎还很有可能直接放弃使用索引，采用全表扫描。
- LIKE 子句的条件中，%不要是第一个字符，尽量靠后。更复杂的需求考虑使用全文索引。
- OR 条件大于3个：
  - 不同字段的，使用 UNION ALL 代替。

- 相同字段的，用 IN 代替。

- 尽量使用 WHERE 子句代替 HAVING 子句，例如：

```
SELECT id,COUNT(*) FROM tbl GROUP BY id HAVING age>=30;
```

- 应该替换为：

```
SELECT id,COUNT(*) FROM tbl WHERE age>30 GROUP BY id;
```

- 一个表的 ORDER BY 和 GROUP BY 的组合都不应该超过3种，否则从业务逻辑考虑进行优化或者分成多张表。
- 如果不需要排序，GROUP BY 子句写成：GROUP BY NULL。
- WHERE 子句尽量使用主键。
- InnoDB 表尽量避免使用类似 COUNT(*) 的全表扫描查询，从设计上考虑另外用一张表存这个计数值。
- 尽量避免使用子查询。

3.2.3 DML 语句

- UPDATE、DELETE 等语句尽量带上 WHERE 子句，且使用索引字段，最好使用主键。
- 使用 INSERT ... ON DUPLICATE KEY update (INSERT IGNORE) 来避免不必要的查询。
- INSERT、UPDATE、DELETE 语句中不要使用不确定值的函数，例如：RAND() 和 NOW()。
- 多条 INSERT 语句要合并成一条批量提交，一次不要超过500行数据。
- 禁止在 UPDATE 语句中，将 "," 写成 AND，非常危险，例如：

```
UPDATE tbl SET fid=fid+1000, gid=gid+1000 WHERE id > 2;
```

如果写成

```
UPDATE tbl SET fid=fid+1000 AND gid=gid+1000 WHERE id > 2;
```

此时fid+1000 AND gid=gid+1000将作为值赋给 fid，且没有 Warning。

- 删除一张表用 TRUNCATE，而不是 DELETE。

3.2.4 多表 JOIN

- 多表 JOIN 时，各表的顺序按照各表在 WHERE 子句条件下返回的行数从小到大排列，行数最小的在最左边。
- 避免大表间的 JOIN，一般表的记录数不超过10W条的情况下，才建议使用 JOIN。

3.2.5 其它

- 合理使用 PREPARE Statement，提供性能还能防 SQL 注入。
- 拆分负载 SQL 为多条 SQL，避免大事务。
- 如果要对表进行 DDL 操作，尽量将对一张表的 DDL 在一条 SQL 语句完成，例如：要给表 t 增加一个字段 b，然后给已有字段 a 建立索引，可以用下面一条语句完成：

```
ALTER TABLE t ADD COLUMN b VARCHAR(10), ADD INDEX idx_a(a);
```

4 数据库管理规范

4.1 基本要求

MariaDB 自带的安装程序会在安装前进行系统环境检查，包括以下项目：

检查项	期望结果
操作系统版本	Linux 系列
文件句柄	大于100000
时间同步	NTP 配置正确
用户	检查安装指定的用户是否存在
安装目录	检查安装目录是否存在且有写权限
数据目录	检查指定的数据目录是否存在且有写权限

4.2 “常规（不分片）”与“分布式（水平拆分）”的选择

当满足以下条件时，可以考虑使用“分布式（水平拆分）”：

- 表中数据会持续增长，在可预见的时间内，超过单机最大存储量。
- 表的读写量很大，超过了单机最大吞吐量。
- 充分考虑了分库分表会导致的在表设计和使用上的各种限制。

关于“分布式（水平拆分）”版本在使用上的限制，请参考 分布式版本开发指南。

4.3 主从配置与容灾

目前云数据库已支持下列能力：

- MariaDB 支持一主多从配置，且能在主机宕机时自动选举一台从机为主机。
- 从机个数可以自定义，且多个从机提供负载均衡的只读能力，所以增加从机可以扩容读能力（从机的数据有延迟）。
- MariaDB 支持强同步和异步两种主从同步方式：
  - 强同步能保证主机宕机不丢数据，因此建议至少配置2个从机。因为如果只有1个从机，当主机宕机，为保证数据不丢失，剩下1个从机只能提供只读服务。
  - 强同步性能会低于异步，损失情况视主从机之间的网络质量而定。一般建议只在同机房，或者同城多个机房之间使用强同步。
- MariaDB 还支持将2个逻辑实例设置为主从，一般用于异地数据容灾。

4.4 备份与恢复

目前云数据库已支持下列能力：

- 为 MariaDB 集群配置一个 HDFS 集群，用来存储冷备和 BINLOG 备份。
- MariaDB 支持配置定期冷备到 HDFS。
- BINLOG 会实时备份到 HDFS。
- 利用冷备和 BINLOG 备份，MariaDB 支持指定时间点的回档。

4.5 用户和权限

MariaDB 的使用者应该分为集群管理者、数据库实例管理者和数据库使用者：

- 集群管理者主要通过运维管理平台完成以下工作：
  - 管理整个集群的资源池，负责资源的上下架和合理的分配。
  - 为实例管理者分配数据库实例。
  - 协助实例管理者进行实例管理。
  - 负责管理所有实例的冷备和 BINLOG 备份数据。
  - 关注集群和实例的运行指标，保证集群和各实例的正常运行。
- 数据库实例管理者主要通过数据库管理平台完成以下工作：
  - 提交新建、扩容、删除实例的申请。
  - 负责维护实例的个性化参数。
  - 负责管理数据库实例的账号和密码。
  - 关注实例的运行指标，进行数据库性能管理。
  - 通过查看慢查询分析、错误日志、慢查询日志等信息，进行数据库的性能优化分析。
- 数据库使用者通过 SQL 客户端连接数据库，提交 SQL 语句，存取数据。

4.6 日志管理

MariaDB 集群的日志主要包括：

- 集群中各个组件的运行日志。
- 实例的网关日志。
- 实例的数据库运行日志、慢查询日志、错误日志。
- 实例的数据库 BINLOG 文件。
- HDFS 中存储的各种日志备份。

所有日志都根据需要配置合理的清理策略。

5 性能优化建议

5.1 性能优化原则

数据库性能优化通常涵盖方法论体系、特性及工具2个方面。方法论体系包含3部分内容：

- 性能规划：深入了解应用与数据库的交互特征，确立良好的设计、开发、测试迭代过程，上线前消除模型上的性能瓶颈。
- 实例调优：建立性能基准，对比调节数据库、操作系统、存储、网络等的配置，主动监控、消除瓶颈。
- SQL 调优：书写高效 SQL，优化相关数据库对象，充分借助优化器，确定最佳执行计划。

5.2 性能优化步骤

- 首先执行下面的初始检查：
  - 获取直接用户的使用反馈，确定性能目标和范围。
  - 获取性能表现好与坏时的操作系统、数据库、应用统计信息。
  - 对数据库做一次全面健康检查。
- 定期收集数据库统计信息。
- 根据收集的信息，以及对应用特性的了解，构建性能概念模型，明确性能瓶颈所在，以及导致性能的根本原因。
- 提出一系列针对的优化措施，并根据它们对性能改善的重要程度排序，然后逐一加以实施。不要一次执行所有的优化措施，必须逐条尝试，逐步对比。
- 通过获取直接用户的反馈验证调节是否已经产生预期的效果，否则，需要重新提炼性能概念模型，直到对应用特性了解进一步准确。
- 重复上述，直到性能达到目标或由于客观约束无法进一步优化。

5.3 常用优化技巧

- 查看实例long_query和long_query_rate指标，分析慢查询出现频率及规律。
- 使用慢查询分析工具，从出现频率最高、耗时最长的 SQL 语句开始分析，通过优化 SQL 语句，添加索引等方式解决。
- 查看实例mem_hit_rate和mem_available指标，分析 innodb 的缓存池是否足够，内存是否是瓶颈。
- 查看cpu_usage_rate，结合慢查询分析 CPU 消耗是否合理，CPU 是否是瓶颈。
- 调整innodb_page_size参数，对比性能测试，找到最合适的配置。
- 对常用语句使用 EXPLAIN 进行查询分析，找出潜在的设计问题。
- 根据业务场景设计合适的用例对不同规格实例进行性能测试。

分布式版本编程与使用规范

最近更新时间：2021-08-23 11:30:15

相关材料详见 分布式数据库开发手册。

利用热点更新技术应对秒杀场景

最近更新时间：2021-10-26 19:30:50

1 引言

1.1 背景

在“秒杀”和“限时抢购”等这样的场景下，大量用户在极短时间内请求大量商品。而体现在 MySQL 数据库中，同一商品在数据库里肯定是一行存储，所以会有大量的线程来竞争 InnoDB 行锁，当并发度越高时等待的线程也会越多，TPS 会下降 RT 会上升，数据库的吞吐量会严重受到影响。本文档描述 MariaDB 解决“秒杀”和“限时抢购”所做的特殊优化——热点更新技术。

1.2 使用简介

热点更新：采用如下示例语句对某个数据对象频繁进行更新。
目前仅支持 Percona 5.7.17 版本，可在 MariaDB 购买页 购买。

```
UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 88 TARGET_AFFECT_ROW 1 table_name SET k=k+1 WHERE id=88
```

2 UPDATE 和 INSERT 语法变化

UPDATE 和 INSERT 的 SQL 语句可以增加新关键字，以表达热点更新的功能，红色为新增内容。

2.1 UPDATE 语法

```
UPDATE [LOW_PRIORITY]

[COMMIT_ON_SUCCESS] [ROLLBACK_ON_FAIL] [QUEUE_ON_PK expr1] [TARGET_AFFECT_ROW expr2]

[IGNORE] table_reference

SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...

[WHERE where_condition]

[ORDER BY ...]

[LIMIT row_count]
```

2.2 INSERT 语法

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]

[COMMIT_ON_SUCCESS] [ROLLBACK_ON_FAIL] [QUEUE_ON_PK expr]
```

```
[IGNORE]

[INTO] tbl_name

[PARTITION (partition_name,...)]

[(col_name,...)]

{VALUES | VALUE} ({expr | DEFAULT},...),(...),...

[ ON DUPLICATE KEY UPDATE

col_name=expr

[, col_name=expr] ... ]
```

2.3 说明

1. UPDATE 只支持单对象更新，即支持 "single-table-syntax"，不支持 "multiple-table-syntax"。
2. 只支持单机场景，XA 场景之后的迭代版本由 proxy 实现。
3. INSERT 的三种语法都支持，这里只列举一种。
4. 标准语法参考官方标准：UPDATE Syntax、INSERT Syntax。
5. 对QUEUE_ON_PK指定的 expr 的值的对象，实施热点更新功能，通常 expr 的值是一个正整数值。
6. 参数含义：
   - COMMIT_ON_SUCCESS：更新操作成功后，立即提交。适合单语句作为一个事务。
   - ROLLBACK_ON_FAIL：更新操作失败后吗，立即回滚。适合单语句作为一个事务。
   - QUEUE_ON_PK expr：指定热点更新对象，对被更新的对象封锁和解锁。被更新的对象总数不超过hot_commodity_query_size，即，具有不同值的 expr 的个数不超过hot_commodity_query_size。expr 取值自由，但建议与主键保持一致，也可以不一致。
   - TARGET_AFFECT_ROW expr：指定热点更新影响的数据行。expr 是一个正整数（[1, MAX], MAX 是 8 位正数的最大值）。通常 expr 为 1，表示只有一行受到影响。

2.4 建议

使用时，只在单语句事务中增加全部新增的参数使用，并且建议蓝色字体值匹配（可以不匹配）。

```
UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 88 TARGET_AFFECT_ROW 1 table_name SET k=
k+1 WHERE id=88
```

2.5 示例

```
CREATE DATABASE hc_db;

CREATE TABLE hc_tbl(a INT PRIMARY KEY, b INT, c INT);
```

```
CREATE TABLE hc_tbl_2(a INT PRIMARY KEY, b INT, c INT);
```

2.5.1 INSERT 示例

```
INSERT COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 1 INTO hc_tbl VALUES(1, 1, 1);

INSERT COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 1 INTO hc_tbl SET a= 2;

INSERT COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 1 INTO hc_tbl_2 SELECT * FROM hc_tbl;
```

2.5.2 UPDATE 示例

```
UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 1 TARGET_AFFECT_ROW 1 hc_tbl SET b= b+1
WHERE a = 1;

QUEUE_ON_PK expr 中 expr 不一定和 WHERE clause 中的值一致

UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 2 TARGET_AFFECT_ROW 1 hc_tbl SET b= b+1
WHERE a = 1;
```

3 新增参数说明

参数名	功能	类型	默认值	其他
hot_commodity_enable	控制热点更新功能的开闭	布尔型	true 打开热点更新功能	运行中关闭此参数，新的事务不再使用热点更新。最好是系统启动前就设置好，而不是运行时改变
hot_commodity_trace	控制跟踪功能的开闭	布尔型	false 关闭跟踪功能	打开时，跟踪信息会输出到标准输出
hot_commodity_query_size	控制允许对多少个热点更新对象进行更新/插入操作	数值型	10000	起到限流的作用
hot_commodity_query_size_modify_enable	控制能否修改 hot_commodity_query_size	布尔型	false 不允许修改 hot_commodity_query_size	方便在单元测试中改 hot_commodity_query_size

注意：如果 MyQL server 启动的时候，参数 hot_commodity_enable 是关闭的，则需要设置其为打开，重新启动 server，才能初始化全局的数据对象表。但如果 hot_commodity_query_size 值为 0，即使打开了 hot_commodity_enable ，也不能使用热点更新。所以热点更新功能需要同时设置：

- hot_commodity_enable=ON
- hot_commodity_query_size=10000 为一个大于 0 的数值，建议控制在 10000、20000 左右，需要根据硬件环境和应用压力等实际情况测试确定其适合的值。
- 启动 server。