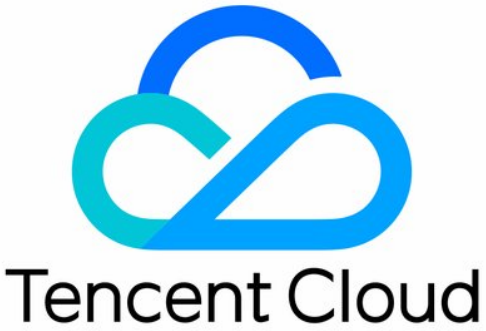
TencentDB for MariaDB Best Practice



Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Best Practice Programming and Usage Specification

Last updated: 2023-08-31 09:35:40

1. Preamble

1.1 Purpose

This document aims to clarify the main technical standards and guiding principles for the planning and design, installation and deployment, operation management, and maintenance of MariaDB database technology and applications throughout their entire lifecycle. This facilitates the unified construction and management of MariaDB database application projects, enhancing the standardization, performance assurance, and maintainability of MariaDB database technology applications.

1.2 Intended Audience

Database designers, developers, managers, and operation and maintenance personnel involved in projects using MariaDB.

2. Design Specifications

2.1 Database Design Principles

- MariaDB is designed for OLTP application scenarios and is not suitable for most complex OLAP tasks.
- Denormalized Design:
  There is no need to strictly adhere to the third normal form; try to minimize the use of foreign keys.
  - Foreign keys are used to maintain referential integrity and can be implemented at the application level.
  - Moderate redundancy in design reduces multi-table join queries, better adapting to the horizontal scalability of the MPP architecture.
  - Directly optimize based on I/O and queries.
- Fully consider the separation of business logic and data, with the database serving solely as a persistent storage system for relational data that ensures ACID properties. Avoid using custom functions, stored procedures, triggers, and views as much as possible.

- Thoroughly consider the overall security design of the database, with a separation of privileges for database management and user personnel.
- Take into account the access frequency and performance requirements of specific data objects, and combine them with the needs of hosts and storage to ensure effective database performance design.
- Take into account the data growth model to determine whether to adopt a "distributed (horizontal partitioning)" mode.
- Thoroughly consider the security level of business data and design appropriate backup and recovery strategies.

2.2 Database Model Design Specifications

2.2.1 Basic Standards

- Use only the InnoDB storage engine, avoiding the MyISAM engine. In comparison, InnoDB has the following features:
  - Full ACID compliance.
  - Crash self-diagnosis and recovery.
  - Row-level locking, ensuring high concurrency.
  - Better utilization of multi-core CPU performance.
  - Features an integrated cache pool for better memory utilization.
- All tables should use a consistent character set, preferably UTF8 or UTF8MB4.
- Avoid storing large data such as images and binary files in the database.
- Plan the scale, number of rows, and size of individual tables in advance.
- Control the total length of fields in a single row, set the innodb_page_size reasonably, use the COMPACT row format whenever possible, and avoid row overflow.

2.2.2 Database Object Naming Conventions

The scope of the database object naming conventions applies to database designers, managers, and developers using MariaDB. This convention does not apply to system libraries, tables, and other objects that come with the MariaDB product.
Database objects should be named according to the following specifications:

- Object names should consist of meaningful English words, and abbreviations should be avoided unless they are widely accepted.
- When naming objects such as index constraints, multiple names like table names and field names may be included. In this case, abbreviations can be used, and the abbreviation rules and character count should be consistent.
- Use only lowercase letters, digits, and underscores.

- The name should not exceed 32 characters in length.
- Avoid using SQL keywords.
- The object name must include at least: object type, parent object name, and object name.

The naming conventions for different database objects are listed below:

Database Objects	Format	Example	Note
Database Schema	db_<database_name>	db_user	This indicates that the data under this database is all user-related.
Table	tbl_${Table name}	tbl_employees	Table representing employee information storage
Primary Key	pk_<Table name>_<Field name>[_Field name]	pk_emlo_id_name	The id and name fields of the tbl_employees table together form the primary key. However, in MariaDB, the primary key name is actually insignificant, as the system will ignore this name and uniformly rename it as: PRIMARY.
Unique Index	uidx_<table_name>_<field_name>[_field_name]_<number>	uidx_empl_name_age_1	The first unique index created for the name and age fields of the tbl_employees table.
Regular Index	idx_<TableAbbreviation>_<ColumnNameAbbreviation>[_ColumnNameAbbreviation]_<Number>	idx_empl_name_age_1	The first regular index created for the name and age fields of the tbl_employees table.
Foreign Key	fk_<table_name>_<associated_table_name>_<field_name>[_field_name]_<associated_field_name>	fk_empl_user_uid_id	Indicates that the uid field in the tbl_employees table has a foreign key constraint to the id field in the tbl_user table.

	[_associated_field_name]		
View	v_<view_name>	v_female	–
Stored procedure	sp_<Stored Procedure Name>	sp_add_empl	–
Custom Functions	f_<FunctionName>	f_count_empl	–
Trigger	trg_<TriggerName>	trg_update_empl	–
Temp Table	tmp_${Table name}	tmp_latecomer	–

2.2.3 Instance Configuration

After applying for a MariaDB instance, it needs to be initialized. During the initialization process, two important parameters must be specified:

- Character Set: Specifies the default character set for the database and tables.
- `innodb_page_size` : The size of pages in InnoDB table file storage. Pages are the smallest storage units in the InnoDB engine within files, with the default value in native MySQL being 16K. Consider the following factors when configuring this value:
  - The size of a single row of data in the primary table of an instance.
  - InnoDB table's `ROW_FORMAT` configuration.
  - Whether to use an SSD disk.

After initialization, you can set the following parameters:

- Based on business requirements, you can enable STRICT mode by setting the `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` flag in SQL MODE. This mode strictly rejects data that does not conform to field type definitions. The differences between the two flags are:
  - `STRICT_ALL_TABLES` : For storage engines that do not support transactions, when updating multiple rows simultaneously, if the data that does not meet the definition is not the first row, it will result in the update of the rows before that row, while the current row and subsequent rows will not be updated.

- `STRICT_TRANS_TABLES`: For storage engines that do not support transactions, reject the request if the non-compliant data is the first row; otherwise, operate in normal mode.
- MariaDB enables autocommit by default, and it is recommended to keep it enabled.
- MariaDB's default transaction isolation level is REPEATABLE-READ. Please decide whether to adjust it to READ-COMMITTED (Oracle's default level) based on your requirements.

2.2.4 Database

The concept of a database in MariaDB differs from that in ORACLE, as it can be considered a SCHEMA, similar to a folder for tables, which facilitates classification management for DBAs. When creating a database, in addition to specifying the database name, it is strongly recommended to explicitly define the default character set parameter, for example:

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

2.2.5 Tables

2.2.5.1 Table Parameter Settings

- The default `ROW_FORMAT` for InnoDB tables is COMPACT. Please specify an appropriate value according to your needs:
  - COMPACT: When there is no row overflow, a single row of data is stored within one data page, resulting in the highest data reading efficiency.
  - DYNAMIC: Data is stored in at least two pages, with data pages only storing pointers to overflow pages where the actual data resides. This index structure has the most concentrated storage and offers higher index access efficiency.
  - REDUNDANT: Obsolete format, do not use.
  - COMPRESSED: Stores data after compression, achieving the highest space utilization. However, it increases CPU and memory consumption, extends system response time, and reduces throughput. Use with caution.
- It is recommended to explicitly specify the default character set parameter for tables.
- Specifying parameters related to physical aspects, such as file storage paths, is not allowed.

2.2.5.2 Primary and Foreign Key, Constraint Design

- InnoDB tables must specify a primary key.
- It is recommended not to use fields with practical significance as primary keys. If it is necessary to use such a field, ensure that it possesses real-world uniqueness and

immutability (e.g., bank transaction serial numbers) to minimize the need for primary key modifications.

- Minimize the use of foreign keys, especially in the "distributed (horizontal partitioning)" mode. Considering the automatic database sharding and partitioning, it is recommended not to use foreign keys.
- The shorter the total length of fields composing the primary key, the higher the efficiency. Integer type fields are most suitable for primary keys.
- If multiple unique keys exist, consider using the most frequently used unique key as the primary key.
- It is recommended to use an auto-increment field as the primary key.
- Field attributes should preferably include the NOT NULL constraint and default values, as using NULL values may lead to the following side effects:
  - The meaning is unclear, and many operators are ineffective, increasing complexity. For example, a!=5 cannot match rows where a is NULL.
  - Query optimization can be challenging.
  - Compound indexes containing NULL values are ineffective.
- Avoid using CHECK constraints, as MariaDB will not report an error but will ignore them. Instead, consider using ENUM type as a substitute.

2.2.5.3 Field Type Design

- Please choose an appropriate integer type based on the actual range of values to be stored.
- Please add the UNSIGNED keyword for unsigned integer types.
- Choose between fixed-length and variable-length string types based on the actual storage requirements of the values.
- Based on the actual stored values, try to minimize the length of string types.
- Avoid using TEXT/BLOB types as much as possible, and refrain from storing images, binary files, and other data in the database.
- For non-integers, use precise DECIMAL values whenever possible, and avoid using floating-point numbers.

Recommended value ranges and storage requirements of main data types are as listed below:

Local Disk Types	Value Range	Storage Requirements
TINYINT[(M)]	-128 to 127 or 0 to 255	1 byte
SMALLINT[(M)]	-32768 to 32767 or 0 to 65535	2 bytes

INT[(M)]	-2147483648 to 214748364 or 0 to 4294967295	4 bytes
BIGINT[(M)]	-9223372036854775808 to 9223372036854775807 or 0 to 18446744073709551615	8 bytes
DECIMAL[(M[,D])]	The maximum number of integer digits (M) is 65, and the maximum number of decimal places (D) is 30.	Variable length
DATE	'1000-01-01' to '9999-12-31'	3 bytes
TIME[(<microsecond precision>)]	'-838:59:59.999999' to '838:59:59.999999'	5 bytes
DATETIME[(microsecond precision)]	'1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'	8 bytes
TIMESTAMP[(<microsecond precision)]	'1970-01-01 00:00:01' (UTC) to '2038-01-19 05:14:07' (UTC)	4 bytes
CHAR[(M)]	0<M<=255	An integer multiple of M, related to character set settings.
VARCHAR[(M)]	0<M<65532/N	The value of N is related to the actual character set settings.
TEXT[(M)]	0<M<65535/N	The value of N is related to the actual character set settings.

2.2.6 Indexes

- Avoid indexing columns with low selectivity, such as: gender.
- In composite indexes, frequently used fields should be placed at the beginning, and fields with high selectivity should be prioritized.

- Fields that require frequent sorting can be added to the index, with the field order consistent with the most commonly used sorting method.
- For longer string-type fields, it is recommended to use prefix indexing.
- Rationally merge indexes to avoid redundancy, for example, if there are indexes (a, b) and (a), the index (a) should be removed.
- Keep the number of indexes for a single table within 5.
- Avoid using more than 5 fields in a single index.
- MariaDB's InnoDB engine supports full-text indexing, but currently, it is limited to English.

2.2.7 Pagination Design

Pagination is the most common access model in applications. The following shows how to design an appropriate pagination model based on test results of several pagination modes:

```
/*
 ID is the primary key of the table 'post'
*/
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)
```

It is evident that one should avoid using the pagination method LIMIT m,n directly whenever possible.

2.3. Database security design

- Database User Security Design Principles.
  - Database user permissions should be granted according to the principle of least privilege.

  - Database users should be divided into four categories: administration, application, maintenance, and backup.
- Database User Permission Design Standards.
  - Apart from MariaDB and core maintenance personnel, other users should not have SUPER privilege accounts.
  - Avoid using simple passwords.
  - MariaDB supports field-level permissions, and the principal objects of permissions should be controlled according to the principle of least privilege.
  - User permission settings should be consistent across development, testing, and production environments.
- Storing plaintext passwords in any form within the database is strictly prohibited.

3. Development Standards

3.1 SQL Coding Standards

- Ensure each line does not exceed 80 characters; if it does, break the line and indent.
- Use two spaces for code indentation.
- Keywords should be capitalized, for example: SELECT.
- Constant symbols should be in uppercase, for example: NULL.
- Make judicious use of annotations.
  - Provide detailed annotations on the purpose of the table before the table creation statement.
  - Use the COMMENT clause after each field to add annotations for the field.
  - Add a brief comment about the table using the COMMENT clause at the end of the table creation statement.
- Remove redundant scaling; for example,

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

- The code above should be optimized to:

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

- Remove repeated conditions; for example:

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

- The code above should be optimized to:

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

3.2 SQL Statement Standards

3.2.1 Indexes and Partitions

- Make judicious use of USE INDEX and IGNORE INDEX for index selection.
- Optimize query conditions by utilizing indices whenever possible.
- Pay attention to field types to avoid type conversions. Type conversions not only increase CPU consumption, but also may cause index failure if the conversion is unsuccessful.
- For compound indexes, the query conditions must include all prefix fields to be effective. For example, with an index (a, b, c), the query conditions must be either a, a and b, or a, b, and c to utilize the index.

3.2.2 SELECT Columns and WHERE Conditions

- Avoid using databases for arithmetic operations as much as possible, which should be conducted at the application layer; for example:

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

- Avoid using SELECT * directly; instead, list the specific fields to be queried.
- Prefer using UNION ALL instead of UNION, as UNION performs deduplication and sorting.
- Principles for using the WHERE clause: maximize the use of indexes, keep it simple, and aim to match fewer rows.
- In the WHERE clause, use more equality operators and fewer non-equality operators, as non-equality operators often cause index inefficiency (MySQL does not support range indices).
- Even with an index, if the number of rows matched by the WHERE clause exceeds 30% of the table, the efficiency may still be low. The InnoDB engine might even abandon using the index and opt for a full-table scan instead.
- In the LIKE clause conditions, avoid using % as the first character and place it further back. For more complex requirements, consider using full-text indexing.
- OR condition with more than three elements:
  - For different fields, use UNION ALL as a substitute.
  - For the same field, use IN instead.
- Prefer using the WHERE clause over the HAVING clause, for example:

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

- The code above should be replaced with:

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

- The combination of ORDER BY and GROUP BY in a single table should not exceed three types; otherwise, consider optimizing from a business logic perspective or dividing the data into multiple tables.
- If sorting is not required, the GROUP BY clause should be written as: GROUP BY NULL.
- Use primary keys in the WHERE clause whenever possible.
- For InnoDB tables, try to avoid using full-table scan queries like COUNT(*); consider designing an alternative table to store the count value.
- Try to minimize the use of subqueries.

3.2.3 DML Statements

- It is recommended to include a WHERE clause in UPDATE and DELETE statements, utilizing indexed fields, preferably the primary key.
- Use INSERT ... ON DUPLICATE KEY UPDATE (INSERT IGNORE) to avoid unnecessary queries.
- Avoid using functions with indeterminate values in INSERT, UPDATE, and DELETE statements, such as RAND() and NOW().
- Combine multiple INSERT statements into a single batch submission, not exceeding 500 rows of data at a time.
- It is strictly prohibited to replace "," with "AND" in an UPDATE statement, as it is extremely dangerous. For example:

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

In case of writing

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

At this point, `fid+1000 AND gid=gid+1000` will be assigned as the value for fid without any warning.

- Use TRUNCATE to delete a table, rather than DELETE.

3.2.4 Multi-table JOIN

- When performing a multi-table JOIN, arrange the tables in ascending order based on the number of rows returned under the WHERE clause conditions, with the table having the fewest rows on the left.
- Avoid using JOIN between large tables; it is recommended to use JOIN only when the record count of a table does not exceed 100,000 rows.

3.2.5 Others

- Utilize PREPARE Statement judiciously to enhance performance and prevent SQL injection.
- Divide load SQL into multiple SQL statements to avoid large transactions.
- To perform DDL operations on a table, try to complete the DDL for a single table in one SQL statement. For example, to add a field 'b' to table 't' and create an index for the existing field 'a', you can use the following statement:

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

4. Database Management Standards

4.1 Basic Requirements

The built-in installer for MariaDB performs a system environment check before installation, which includes the following items:

Check Items	Expected Results
OS Version	Linux Series
File Handle	Greater than 100,000
Time Synchronization	NTP configuration is valid.
User	Check if the specified user for installation exists.
Installation Directory	Check whether the installation directory exists and has write permissions.
Catalog	Check whether the specified data directory exists and has write permission.

4.2 Choosing Between "Conventional (Non-sharded)" and "Distributed (Horizontally Split)"

If the following conditions are met, you can consider using the "distributed (horizontal sharding)" scheme:

- The data in the table will continue to grow and, within a foreseeable time frame, exceed the maximum storage capacity of a single machine.
- The read and write volume of the table is substantial, exceeding the maximum throughput of a single machine.
- Fully considers the various limitations in table design and usage caused by database sharding and table partitioning.

For limitations on using the "Distributed (Horizontal Split)" version, please refer to Distributed Version Development Guide.

4.3 Master-Slave Configuration and Disaster Recovery

Currently, TencentDB has the following capabilities:

- MariaDB supports a primary/replica configuration and can automatically elect a new primary server from the replicas in case of primary server failure.
- The number of slaves can be customized. As multiple slaves can provide load-balanced read-only performance, more slaves can be added to expand the read capacity (there will be a delay in the slave data).
- MariaDB supports two types of master-slave synchronization: strong sync and asynchronous.
  - Strong synchronization ensures that no data is lost when the primary server fails; therefore, it is recommended to configure at least two replica servers. If there is only one replica server, when the primary server fails, the remaining replica server can only provide read-only services to ensure no data loss.
  - Strong synchronization performance is lower than asynchronous, and the extent of loss depends on the network quality between the primary and secondary nodes. It is generally recommended to use strong synchronization only within the same data center or across multiple data centers within the same city.
- MariaDB also supports configuring two logical instances as master and replica, typically used for cross-regional data disaster recovery.

4.4 Backup and restore

Currently, TencentDB has the following capabilities:

- Configure an HDFS cluster for a MariaDB cluster to store cold backups and BINLOG backups.
- MariaDB supports scheduling periodic cold backups to HDFS.
- Binlog is backed up to HDFS in real-time.

- Utilizing cold backup and BINLOG backup, MariaDB supports rollback to a specified point in time.

4.5 Users and Permissions

MariaDB users can be categorized into cluster administrators, database instance administrators, and database users:

- A cluster administrator mainly uses the OPS management platform to complete the following tasks:
  - Managing the entire cluster's resource pool, responsible for the onboarding and offboarding of resources and their reasonable allocation.
  - Assign database instances to instance administrators.
  - Assisting instance administrators in managing instances.
  - Responsible for managing the cold backup and BINLOG backup data of all instances.
  - Monitor the operational metrics of clusters and instances to ensure the proper functioning of both the cluster and individual instances.
- A database instance administrator mainly uses the database management platform to complete the following tasks:
  - Submit applications for creating, expanding, or deleting instances.
  - Responsible for maintaining the personalized parameters of instances.
  - Responsible for managing the account and password of the database instance.
  - Monitor the operational metrics of instances for effective database performance management.
  - By examining slow query analysis, error logs, slow query logs, and other information, perform database performance optimization analysis.
- Database users connect to the database through an SQL client, submit SQL statements, and access data.

4.6 Log Management

The main logs for MariaDB clusters include:

- Runtime logs of various components within the cluster.
- Gateway logs of the instance.
- Instance's database operation logs, slow query logs, and error logs.
- Instance's database BINLOG files.
- Various log backups stored in HDFS.

All logs are configured with appropriate cleanup strategies as needed.

5. Performance Optimization Suggestions

5.1 Performance Optimization Principles

Database performance optimization typically encompasses two aspects: methodology systems and features and tools. The methodology system consists of three parts:

- Performance planning: you should deeply understand the characteristics of interactions between application and database and establish good design, development, testing, and iteration procedures to eliminate performance bottlenecks in the model before service launch.
- Instance optimization: you should establish performance benchmarks and adjust the configuration of components such as database, operating system, storage, and network accordingly to actively monitor and eliminate bottlenecks.
- SQL Tuning: Write efficient SQL statements, optimize related database objects, fully utilize the optimizer, and determine the best execution plan.

5.2 Performance Optimization Steps

- First, run the following initial checks:
  - Collect feedback from direct users to determine the performance goal and range.
  - Collect performance statistics of operating systems, databases, and applications in good and bad cases.
  - Perform an overall database health check.
- Routinely collect database statistics.
- Construct a performance model based on the collected information and your understanding of application characteristics to identify performance bottlenecks and root causes.
- Propose a series of targeted optimization measures, and rank them according to their importance in improving performance. Implement them one by one, rather than executing all optimization measures at once. It is essential to try each measure individually and make gradual comparisons.
- Check whether the adjustment produces expected effect based on the feedback collected from direct users; and if not, summarize the problems and establish a new performance model until you have more accurate understanding of application characteristics.
- Repeat the above steps until the performance reaches the goal or cannot be further optimized due to objective limits.

5.3 Common Optimization Techniques

- View the `long_query` and `long_query_rate` metrics of the instance to analyze the frequency and patterns of slow queries.

- Utilize slow query analysis tools to start analyzing SQL statements with the highest frequency and longest duration. Resolve issues by optimizing SQL statements and adding indexes, among other methods.
- Examine the `mem_hit_rate` and `mem_available` metrics to analyze whether the InnoDB cache pool is sufficient and if memory is a bottleneck.
- Examine `cpu_usage_rate` and determine whether CPU consumption is reasonable in conjunction with slow query analysis, and if the CPU is a bottleneck.
- Adjust the `innodb_page_size` parameter, conduct performance comparisons, and identify the most suitable configuration.
- Use EXPLAIN to analyze common queries and identify potential design issues.
- Design appropriate use cases based on business scenarios to conduct performance tests on instances with different specifications.

Programming and Usage Specification of Distributed Version

Last updated: 2023-08-31 09:36:05

For detailed information, see Distributed Database Development Manual.

Using Hotspot Update for Flash Sales

Last updated: 2025-07-16 18:10:51

1. Preamble

1.1. Background

In scenarios such as "flash sales" and "limited-time promotions," a large number of users request a massive amount of products within a very short period. In the context of a MySQL database, each product is undoubtedly stored as a single row, leading to numerous threads competing for InnoDB row locks. As concurrency increases, the number of waiting threads also grows, causing a decline in TPS and an increase in RT, which severely impacts the database's throughput. This document describes the special optimization implemented by MariaDB to address the challenges posed by "flash sales" and "limited-time promotions" - the hotspot update technology.

1.2. How to use

Hotspot Update: Use the following example statement to frequently update a specific data object. Currently, only Percona 5.7.17 is supported, which can be purchased on the MariaDB Purchase Page.

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

Change of UPDATE and INSERT Syntax

You can add new keywords to the SQL statement UPDATE/INSERT/SELECT to indicate hotspot update. The words in red are newly added.

2.1. UPDATE syntax

```
UPDATE [LOW_PRIORITY]
        [COMMIT_ON_SUCCESS] [ROLLBACK_ON_FALL] [QUEUE_ONPK expr1]
      [TARGET_AFFECT_ROW expr2]
        [IGNORE] table_reference
SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
[WHERE where_condition]
```

2.2. INSERT syntax

```
INSERT [LOW_PRIORITY | DLAYED | HIGH_PRIORITY]
        [COMMIT_ON_SUCESS] [ROLLBACK_ON_FAIL] [QUEUE_ON_PK expr]
        [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name,...)]
```

2.3. Description

1. `UPDATE` only supports updating a single object, i.e., single-table-syntax but not multiple-table-syntax.
2. Only single-server scenarios are supported. Iterative versions in the XA scenario are implemented by proxy.
3. Three types of `INSERT` syntax are supported, and only one is described below.
4. For standard syntax, refer to the official standards: UPDATE Syntax, INSERT Syntax.
5. For objects with the value of expr specified by `QUEUE_ON_PK`, implement the hotspot update feature. Typically, the value of expr is a positive integer.
6. Parameter description:
   - `COMMIT_ON_SUCCESS`: Commit immediately after a successful update operation. Suitable for single statements as a transaction.
   - `ROLLBACK_ON_FAIL`: Immediately roll back after the update operation fails. Suitable for single statements as a transaction.
   - `QUEUE_ON_PK expr`: Specifies the hotspot update object, and locks and unlocks the updated object. The total number of updated objects does not exceed `hot_commodity_query_size`, meaning the number of different values for expr does not exceed `hot_commodity_query_size`. The value of expr is flexible, but it is recommended to be consistent with the primary key, although it can also be different.
   - `TARGET_AFFECT_ROW expr`: Specifies the data rows affected by the hotspot update. `expr` is a positive integer ([1, MAX], where MAX is the maximum value of an 8-bit positive number). Typically, `expr` is set to 1, indicating that only one row is affected.

2.4. Suggestions

When you use hotspot update, we recommend that you add all newly added parameters only to single-statement transactions.

```
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. Sample code

```
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. Sample code of INSERT

```
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. Sample code of UPDATE

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

expr in QUEUE_ON_PK expr is not necessarily the same as that in the
WHERE clause.

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

Newly Added Parameter Description

Parameter	SDK	Local Disk Types	Default value	Others

`hot_commodity_enable`	Enables/disables hotspot update	Boolean	true Enable the hotspot update feature	To disable this parameter during runtime, new transactions will no longer use the hotspot update feature. It is recommended to set this parameter before system startup rather than changing it during runtime.
`hot_commodity_trace`	Enables/disables tracing	Boolean	false Disable tracking feature	When it is enabled, trace information will be exported to standard output
`hot_commodity_query_size`	Controls the number of hotspot objects that can be updated/inserted	Value	10000	Used for throttling
`hot_commodity_query_size_modify_enable`	Control whether to modify `hot_commodity_query_size`	Boolean	False: Modifying `hot_commodity_query_size` is not allowed.	Easily modify `hot_commodity_query_size` in unit tests.

Note

If the MySQL server is started with the `hot_commodity_enable` parameter disabled, it needs to be enabled and the server restarted to initialize the global data object table. However, if the `hot_commodity_query_size` value is 0, the hotspot update cannot be used even if `hot_commodity_enable` is enabled. Therefore, to use the hotspot update feature, both parameters must be set:

- `hot_commodity_enable` =ON
- `hot_commodity_query_size` =10000 is a value greater than 0, recommended to be around 10,000 or 20,000. The suitable value should be determined through testing based on the hardware environment and application pressure.

- Start the server