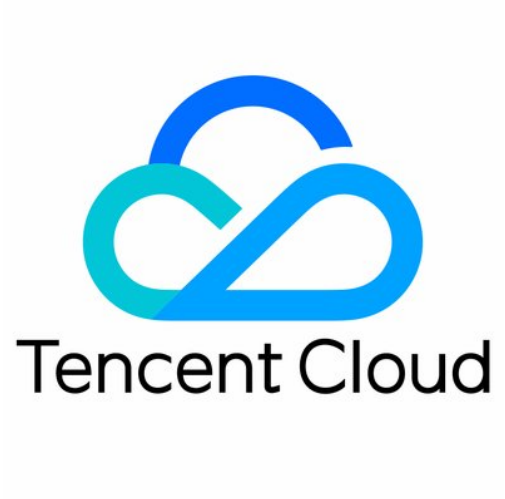
TencentDB for MariaDB

Best Practice

Product Documentation



Contents

Best Practice
Programming and Usage Specification

Last updated：2024-01-11 15:21:58

1. Introduction

1.1. Purpose

This document describes main technical standards and guidelines for each phase throughout the lifecycle of TencentDB for MariaDB databases and applications from planning and design, installation and deployment, to operation management and maintenance, so as to facilitate unified construction and management of TencentDB for MariaDB application projects and improve their normalization, performance, and maintainability.

1.2. Target audience

This document is aimed at designers, developers, administrators, and OPS engineers of TencentDB for MariaDB projects.

2. Design Specifications

2.1. Principles of database design

TencentDB for MariaDB is oriented to OLTP application scenarios rather than complex OLAP scenarios.
Use de-normalized design:
The Third Normal Form (3NF) does not have to be met forcibly, and use of foreign keys should be minimized.
Foreign keys can be used to protect the referential integrity and implemented on the business side.
Appropriate redundancy can reduce multi-table join queries, which is more adaptive to the horizontal scaling of the MPP architecture.
Perform optimization directly based on I/O and queries.
Take into full account business logic and data separation. The database should be used only as a persistent storage system of relational data that guarantees the ACID characteristics. Use of custom functions, stored procedures, triggers, and views should be avoided whenever possible.
Take into full account the overall database security design, database management, and user permission control.
Take into full account the access frequency and performance requirements of specific data objects and plan well for database performance based on the requirements of servers and storage.
Take into full account the data growth model to decide whether to use the "distributed (horizontal sharding)" mode.
Take into full account the business data security level to develop appropriate backup and restoration policies.

2.2. Specifications for designing database models

2.2.1. Basic specifications

Use the InnoDB storage engine only and avoid the MyISAM engine. Compared with MyISAM, InnoDB has the following advantages:

Complete ACID support.

Automatic check of and recovery from crashes.

Row-level lock for guaranteed high concurrence performance.

Better use of multi-core CPU performance.

Built-in buffer pool for better memory use.

Unified character set for all tables (UTF-8 or UTF8MB4 is recommended).

No storage of large-sized data entries such as images and binary files in the database.

Determination of single-table scale, number of rows, and size in advance.

Configuration of upper limit for total length of row fields. You should reasonably set `innodb\\_page\\_size` and use the COMPACT row format as much as possible to avoid row overflow

2.2.2. Specifications for naming database objects

The specifications for naming database objects are aimed at database designers, administrators, and developers. They are not applicable to built-in system tables of MariaDB.

Database objects should be named according to the following specifications:

Use meaningful English words and avoid abbreviations or acronyms that are not widely used.

If multiple names such as table name and field name are included when objects such as indexes and constraints are named, abbreviations or acronyms created in a unified manner in the same length can be used.

Only use combinations of lowercase letters, digits, and underscores (_).

Keep name length below 32 characters.

Do not use SQL keywords.

An object name must contain at least the object type, parent object name, and object name.

The naming conventions for different database objects are listed below:

Database Object	Format	Example	Description
Database (SCHEMA)	db_<database name>	db_user	It indicates that all data in this database is user-related information
Table	tbl_<table name>	tbl_employees	It indicates a table that stores employee information
Primary key	pk_<table name>_<field name>[_field name]	pk_emlo_id_name	It indicates that the `id` and `name` fields of the `tbl\\_employees` table

			comprise the primary key together. However, in TencentDB for MariaDB, this primary key name is meaningless and will be ignored and then changed to `PRIMARY` by the system
Unique index	uidx_<table name>_<field name>[_field name]_<number>	uidx_empl_name_age_1	It indicates the first unique index created by the `name` and `age` fields of the `tbl\\_employees` table
Normal index	idx_<table name abbreviation>_<column name abbreviation>[_column name abbreviation]_<number>	idx_empl_name_age_1	It indicates the first normal index created by 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>[_associated filed name]	fk_empl_user_uid_id	It indicates that the foreign key of the `uid` field in the `tbl\\_employees` table constrains the `id` field in the `tbl\\_user` table
View	v_<view name>	v_female	-
Stored procedure	sp_<stored procedure name>	sp_add_empl	-
Custom function	f_<function name>	f_count_empl	-
Trigger	trg_<trigger name>	trg_update_empl	-
Temporary table	tmp_<table name>	tmp_latecomer	-

2.2.3. Instance configuration

A TencentDB for MariaDB instance needs to be initialized after being successfully applied for. You need to specify two important options during initialization:

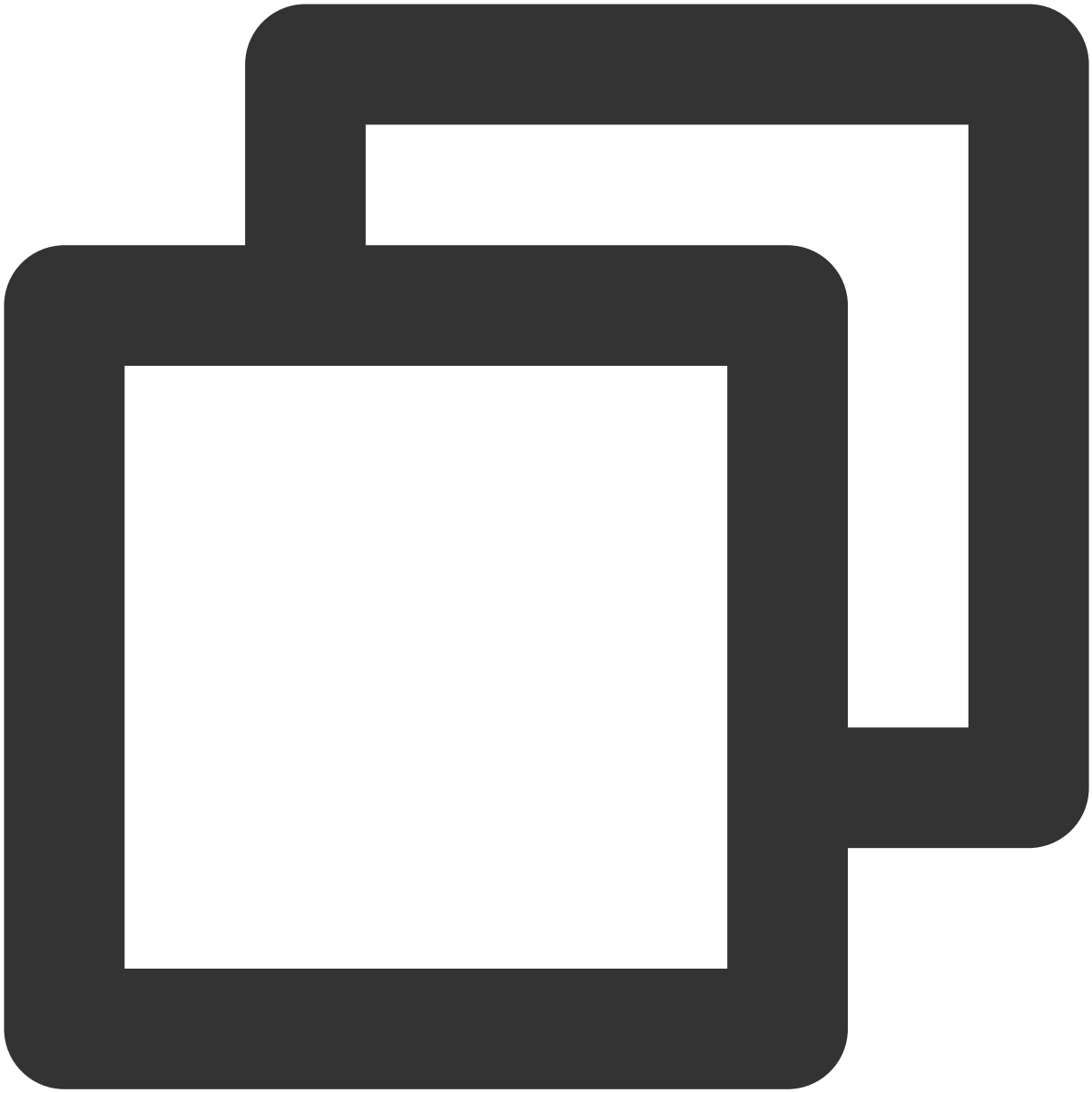
Character set: default character set of tables.

`innodb_page_size` : size of a page stored in an InnoDB table file. Page is the minimum storage unit in the InnoDB engine, and the default size is 16 KB in native MySQL. Consider the following when configuring this value:

Size of a single data row in the primary table in the instance.
Configuration of `ROW_FORMAT` in the InnoDB table.
Whether SSD is used.
After initialization, you can set the following parameters:
Based on your business needs, you can use SQL MODE to set the `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` flag to enable the "STRICT" mode, so that writes of data that does not meet the field type definition will be strictly rejected. Below are the differences between the two flags:
`STRICT_ALL_TABLES` : for a storage engine that does not support transactions, when multiple rows of data are updated at the same time, if a row of data that does not meet the definition is not the first row, rows above it will be updated, while it and rows below it will not be updated.
`STRICT_TRANS_TABLES` : for a storage engine that does not support transactions, if a row of data that does not meet the definition is the first row, the request will be rejected; otherwise, the data will be updated as in normal mode.
TencentDB for MariaDB enables automatic commitment of transactions (AUTOCOMMIT) by default, and you are recommended to keep this feature enabled.
The default transaction isolation level in TencentDB for MariaDB is `REPEATABLE-READ` . Please decide whether to adjust it to `READ-COMMITED` (default level in Oracle) based on your actual needs.

2.2.4. Database

The concept of database in TencentDB for MariaDB is different from that in Oracle. In TencentDB for MariaDB, a database can be viewed as a schema, which is a folder similar to a table and used to facilitate category management by DBAs. When creating a database, besides naming it, you are also strongly recommended to specify the default character set as shown below:



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

2.2.5. Table

2.2.5.1. Table parameter settings

The default value of `ROW_FORMAT` of an InnoDB table is `COMPACT`. Please select an appropriate value for this parameter based on your needs:
COMPACT: one row of data is stored in one data page if there is no row overflow generated, which has the highest data read efficiency.

DYNAMIC: one row of data is stored in at least two pages. The data page only stores pointers of the overflow page where data is stored. The index storage in this structure is most concentrated with the highest index access efficiency.
REDUNDANT: this is a legacy format and should not be used.
COMPRESSED: the data is compressed before being stored. This option has the highest space utilization; however, it will increase CPU and memory usage and system response time and reduce the throughput. Please use this option with caution.
You are recommended to specify the default character set of tables.
Physical parameters such as file storage path cannot be specified.

2.2.5.2. Primary foreign key and constraint design

A primary key must be specified for an InnoDB table.
You are not recommended to use a meaningful field as the primary key. If you have to do so, please make sure that this field is practically unique and does not change (such as bank transaction serial number) so as to avoid modification of the primary key.
Avoid using foreign keys, especially in the "distributed (horizontal sharding)" mode. You are not recommended to use foreign keys in consideration of automatic database sharding and table splitting.
The shorter the length of the field constituting the primary key, the higher the efficiency. An integer-type field is most appropriate to server as the primary key.
If multiple unique keys exist, you are recommended to use the most commonly used one as the primary key.
You are recommended to use an auto-increment field as the primary key.
You are recommended to add the `NOT NULL` constraint and default values to field attributes as much as possible, as the use of `NULL` values will cause the following problems:
Definition is unclear, which cannot be applied to many operators and increases complexity. For example, `a!=5` cannot match the row where `a` is `NULL`.
Query optimization is hard to be implemented.
Compound indexes with `NULL` will not take effect.
Do not use the `CHECK` constraint. Although TencentDB for MariaDB will not report an error but ignore it. You can use the `ENUM` type instead.

2.2.5.3. Field type design

For the integer type, please select an appropriate type based on the actual storage value range.
For the unsigned integer type, please add the `UNSIGNED` keyword.
For the string type, please select the char type or varchar type based on the actual storage value.
Reduce the length of string type as much as possible based on the actual storage value.
Avoid using TEXT/BLOB types as much as possible and do not store data such as images and binary files in the database.
Use precise DECIMAL as much as possible for non-integers and avoid using floating points.
Recommended value ranges and storage requirements of main data types are as listed below:

Type	Value Range	Storage Requirement
TINYINT[(M)]	-128–127 or 0–255	1 byte
SMALLINT[(M)]	-32768–32767 or 0–65535	2 bytes
INT[(M)]	-2147483648–214748364 or 0–4294967295	4 bytes
BIGINT[(M)]	-9223372036854775808–9223372036854775807 or 0–18446744073709551615	8 bytes
DECIMAL[(M[,D])]	The maximum number of digits of an integer (M) is 65, and the maximum number of decimal places (D) is 30	Variable
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	Integer times of M subject to the configured character set
VARCHAR[(M)]	0<M<65532/N	The value of N is subject to the configured character set
TEXT[(M)]	0<M<65535/N	The value of N is subject to the configured character set

2.2.6. Index

Do not index columns with low selectivity, such as gender.

In a composite index, put commonly used fields and fields with high selectivity at the beginning.

Index fields that often need to be sorted, and keep the field order the same as the most commonly used order.

You are recommended to use prefix index for long string-type fields.

Merge indexes reasonably to avoid redundancy. For example, you should remove (a) when merging (a,b) and (a).

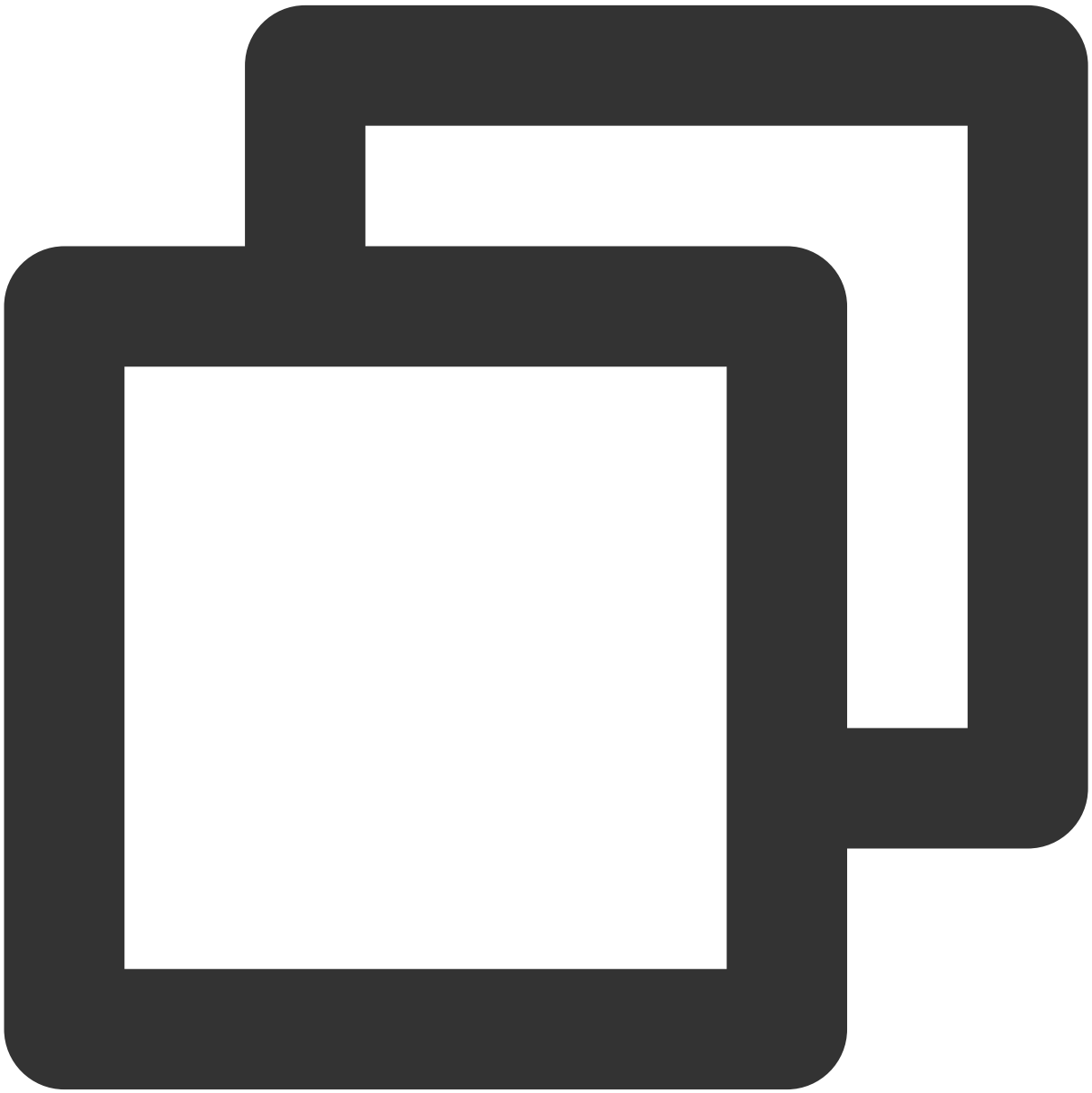
Keep the number of indexes in one table below 5.

Keep the number of fields in one index below 5.

The InnoDB engine of TencentDB for MariaDB supports full-text index which is only available for English currently.

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 `post` table
```

```
*/
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 8000
10 rows in set (0.02 sec)
```

The result above is very clear: you should avoid directly using the `LIMIT m,n` pagination mode.

2.3. Database security design

Principles of database user security design.

Perform database user authorization based on the principle of least privilege.

Classify database users into administration, application, maintenance, and backup accounts.

Specifications for database user permission design.

Do not grant users SUPER permissions, except core maintenance personnel of TencentDB for MariaDB.

Do not use weak passwords.

As TencentDB for MariaDB supports field-level permissions, you should control the permission objects based on the principle of least privilege.

Keep the user permission settings the same in the development, testing, and production environments.

Strictly forbid storing any plaintext passwords in the database.

3. Development Specifications

3.1. Specifications for SQL coding

Keep each line below 80 characters in length and add a line break with indentation whenever the upper limit is reached.

Indent code with two spaces.

Capitalize keywords such as SELECT.

Capitalize constants such as NULL.

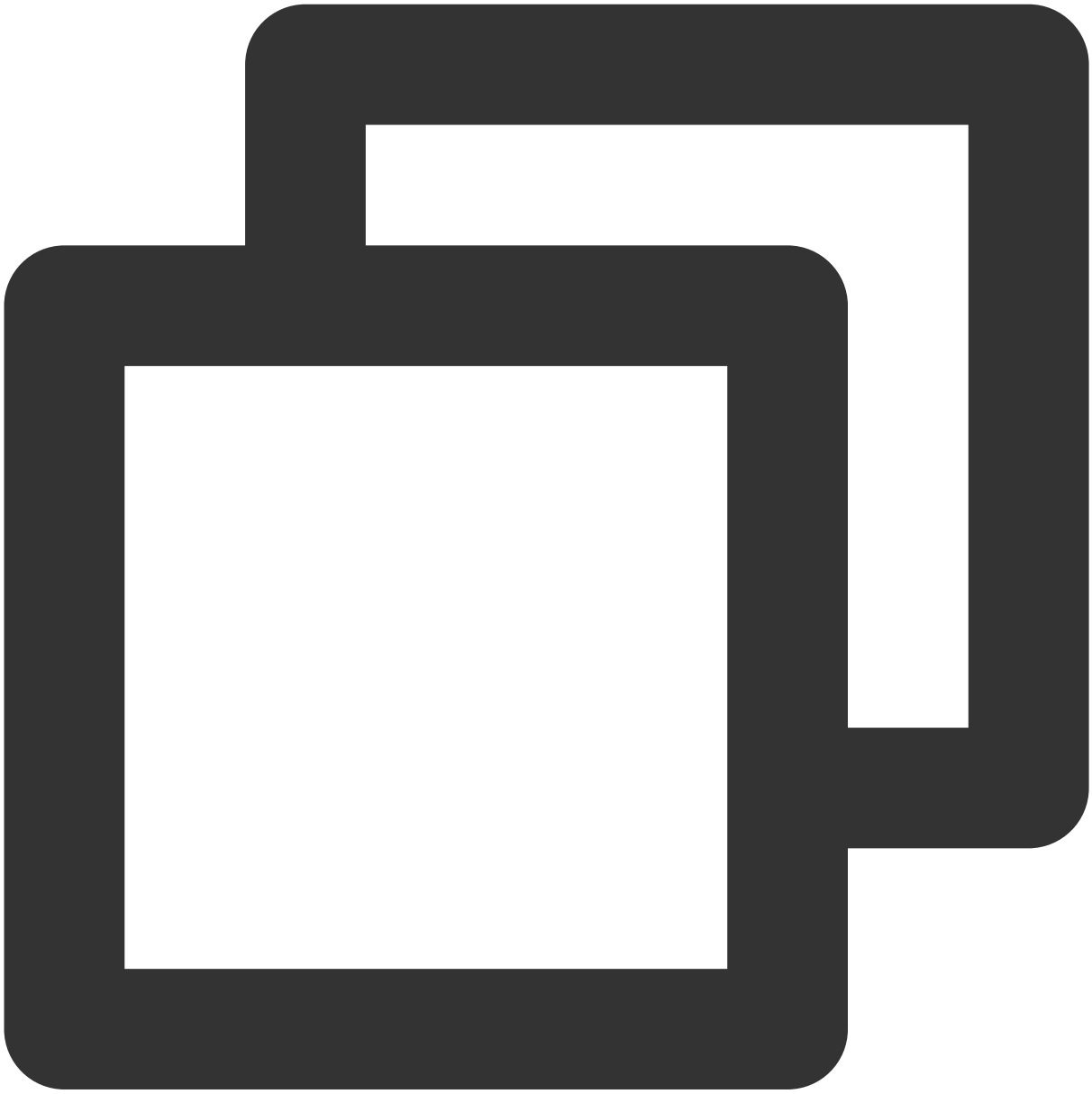
Use comments reasonably.

Add detailed comments for table usage before the table creation statement.

Use a `COMMENT` clause to add a comment after each field.

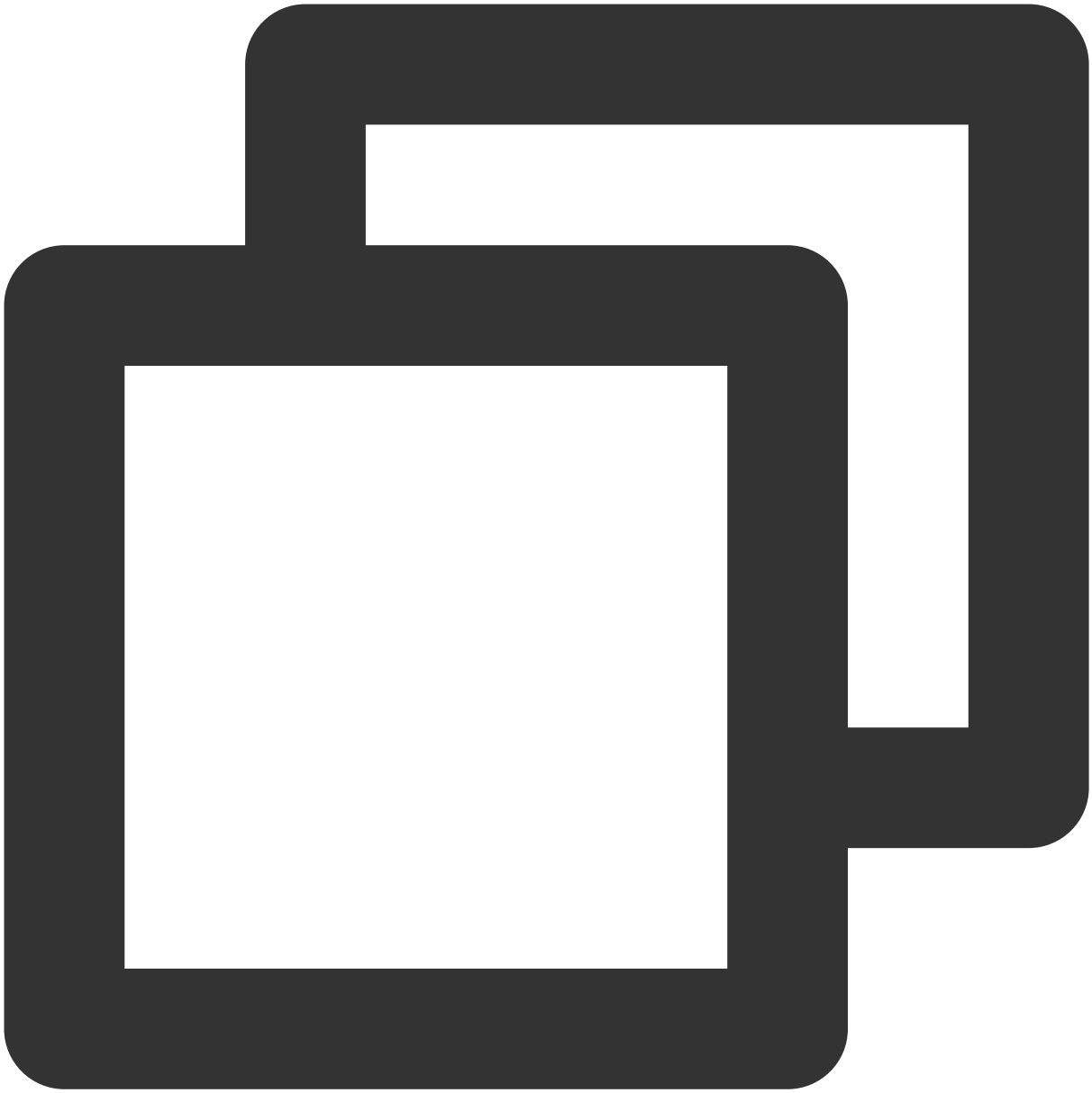
Use a `COMMENT` clause to add a one-sentence comment on table after the end of the table creation statement.

Remove redundant scaling; for example,



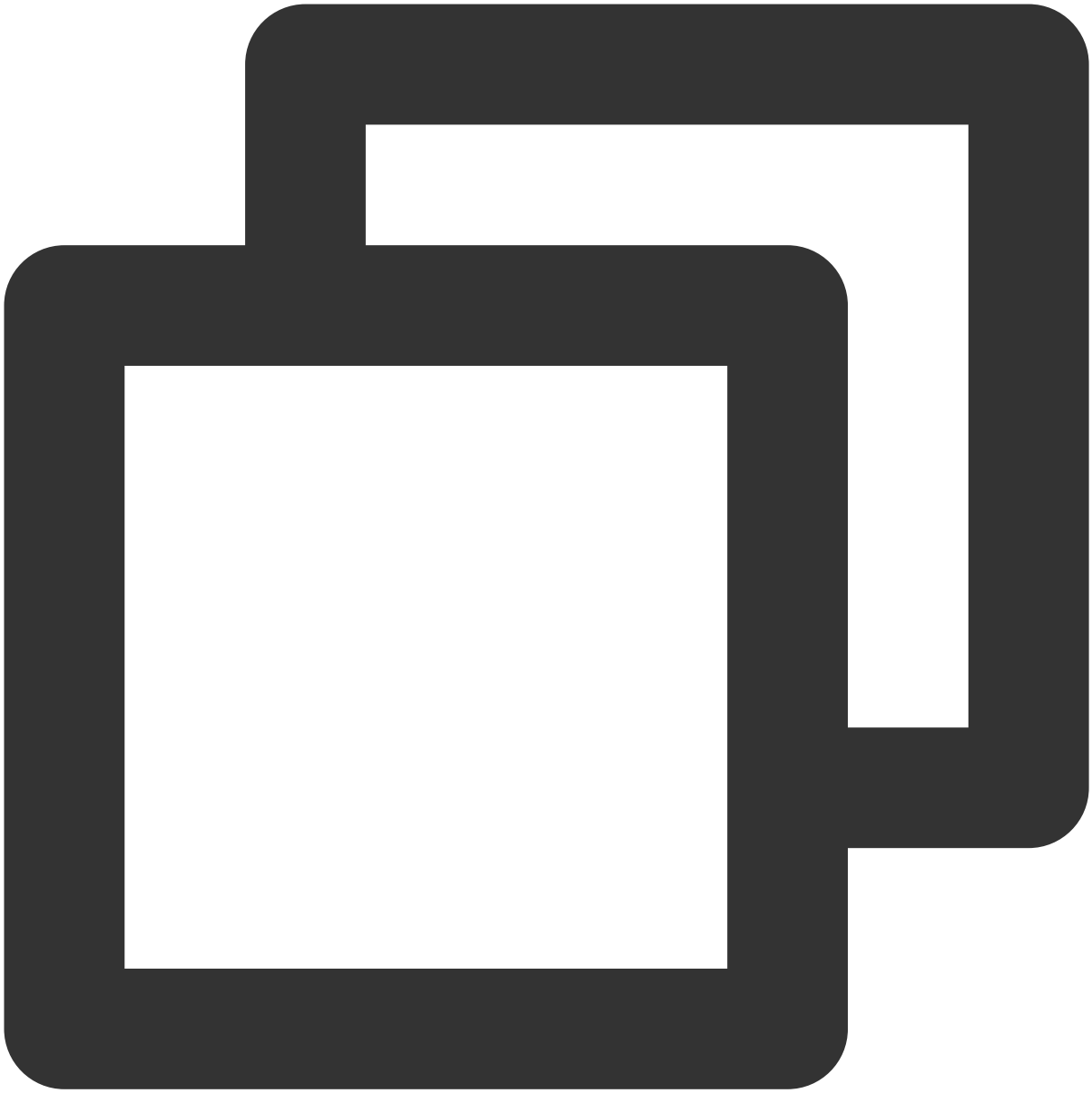
```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
```

The code above should be optimized to:



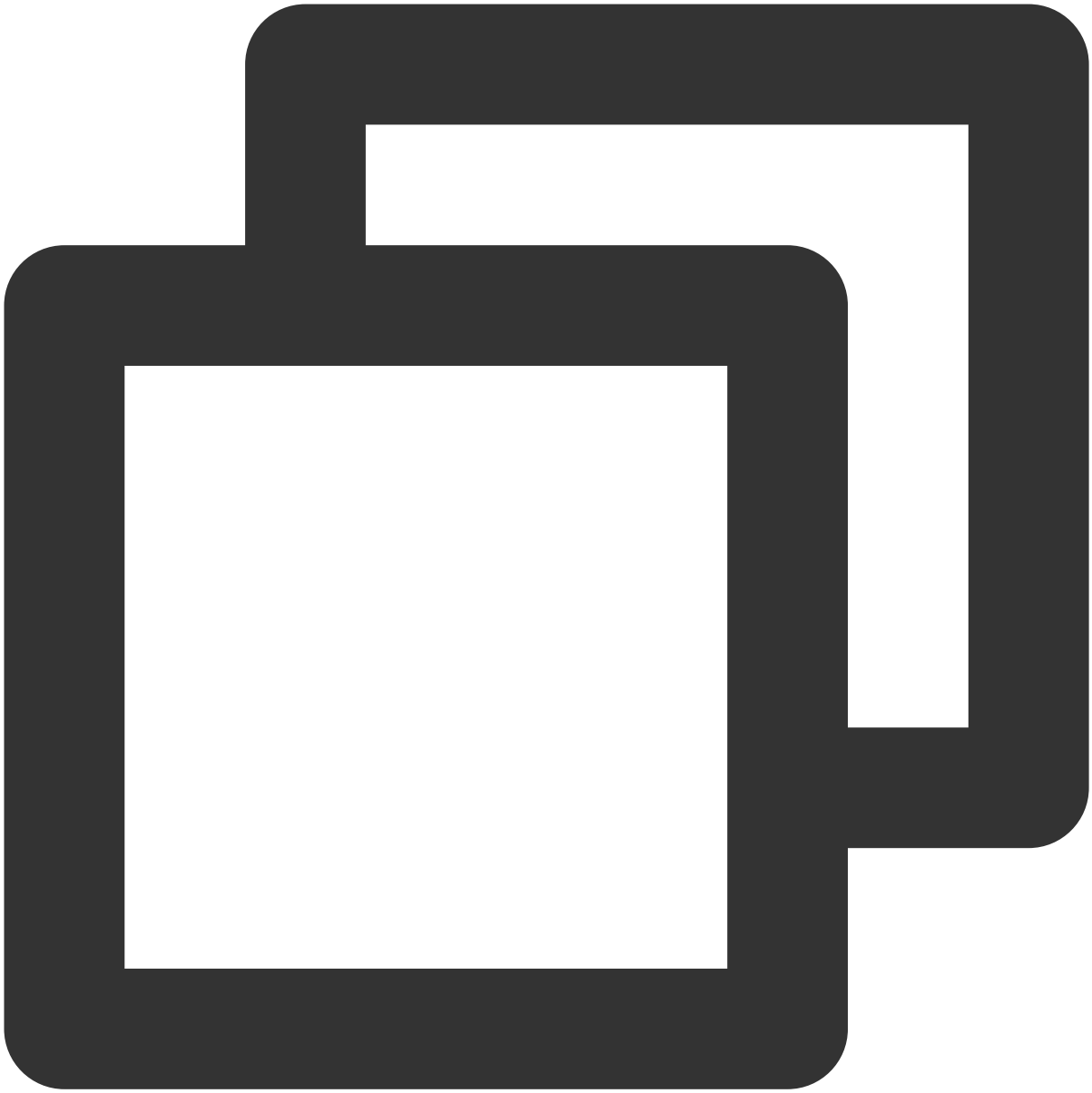
```
(a AND b AND c) OR (a AND b AND c AND d)
```

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. Specifications for SQL statement

3.2.1. Indexing and partitioning

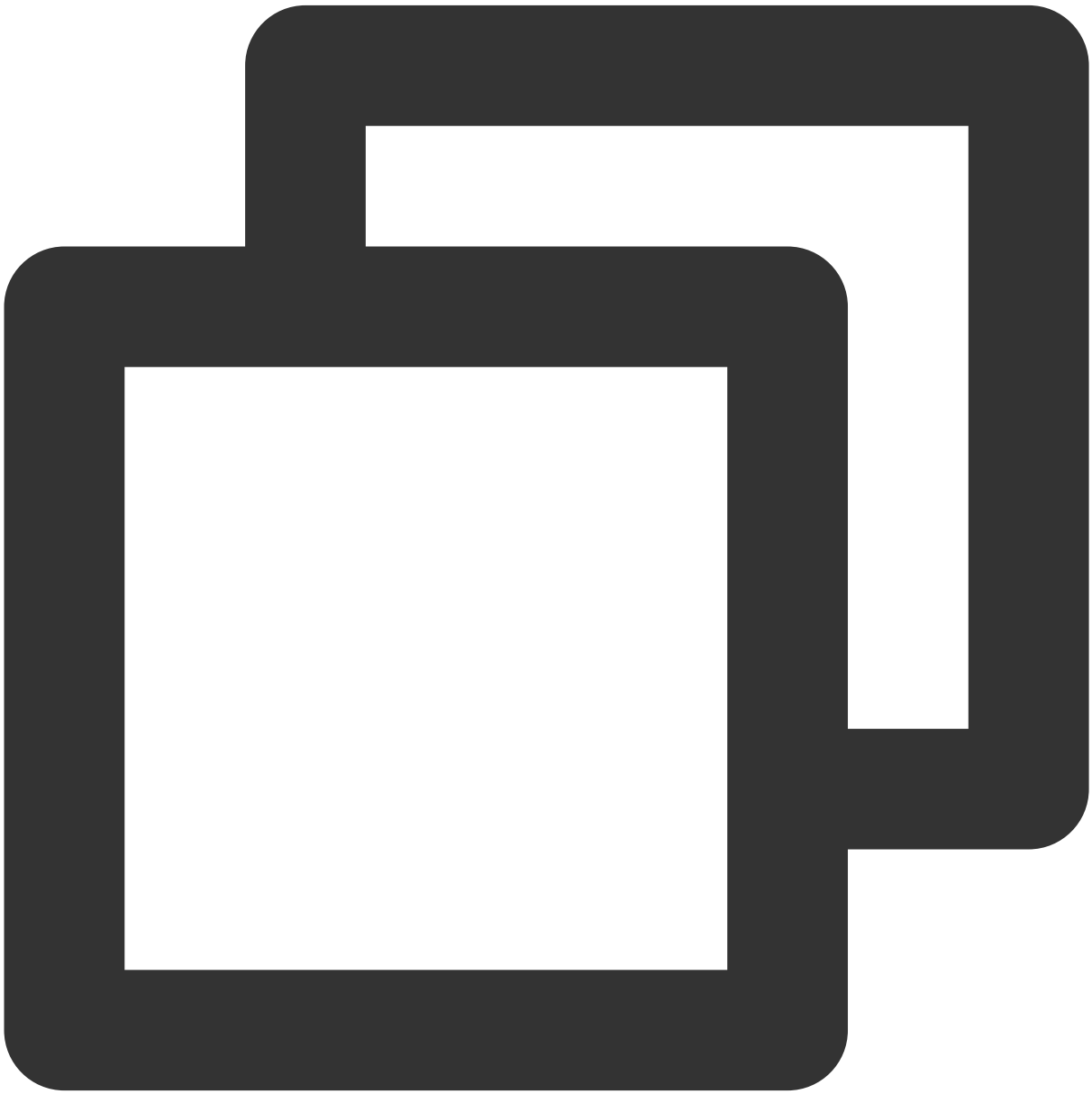
Choose `USE INDEX` or `IGNORE INDEX` reasonably.

Index query conditions as much as possible.

Pay attention to field type and avoid type conversion, as that will increase CPU usage or even invalidate indexes if conversion fails.
For composite indexes, the query conditions must include all prefix fields to take effect. For example, for index (a,b,c), the query condition must be `a` , `a, b` , or `a, b, c` for the index to take effect.

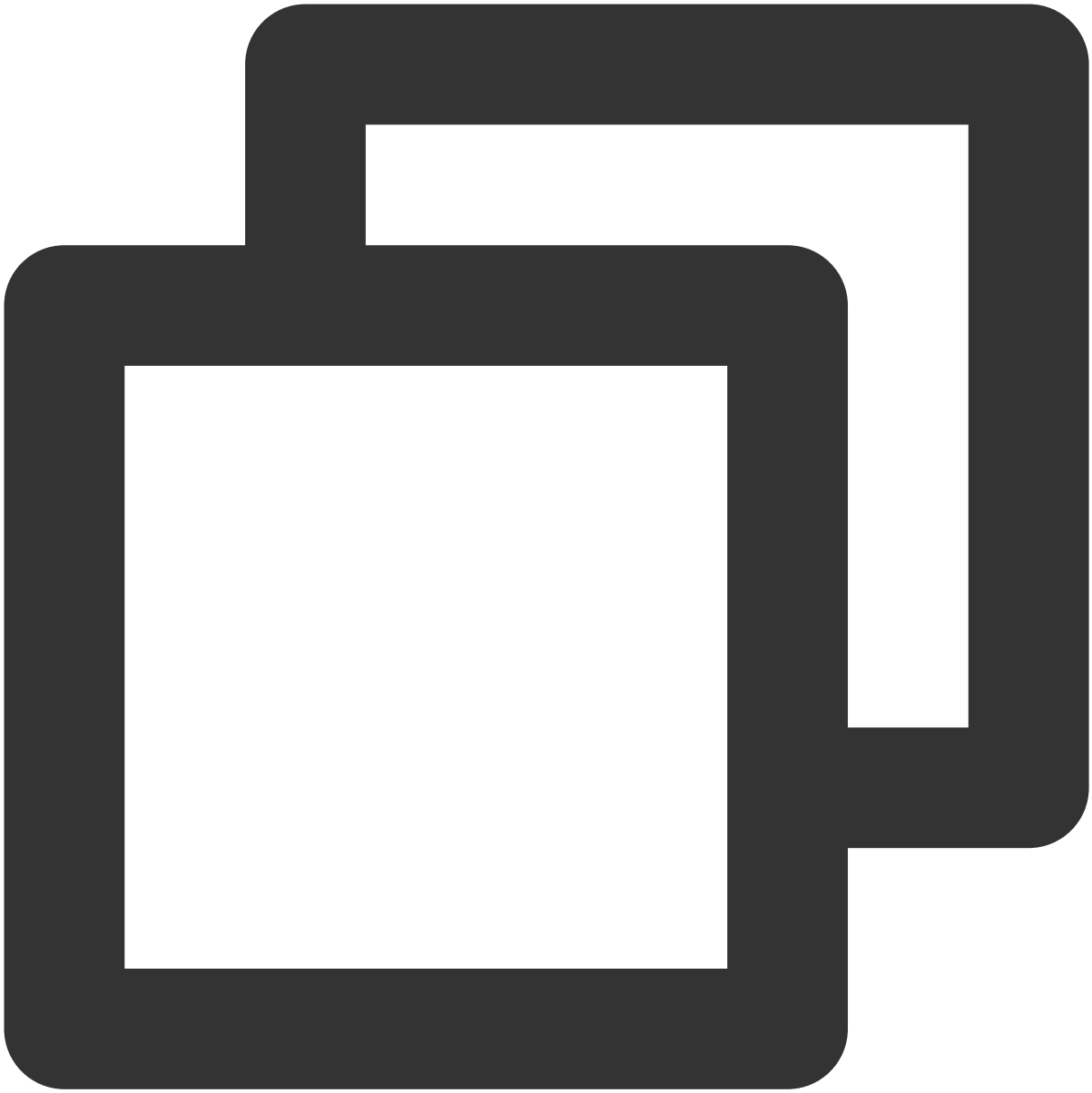
3.2.2. SELECT column and WHERE condition

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



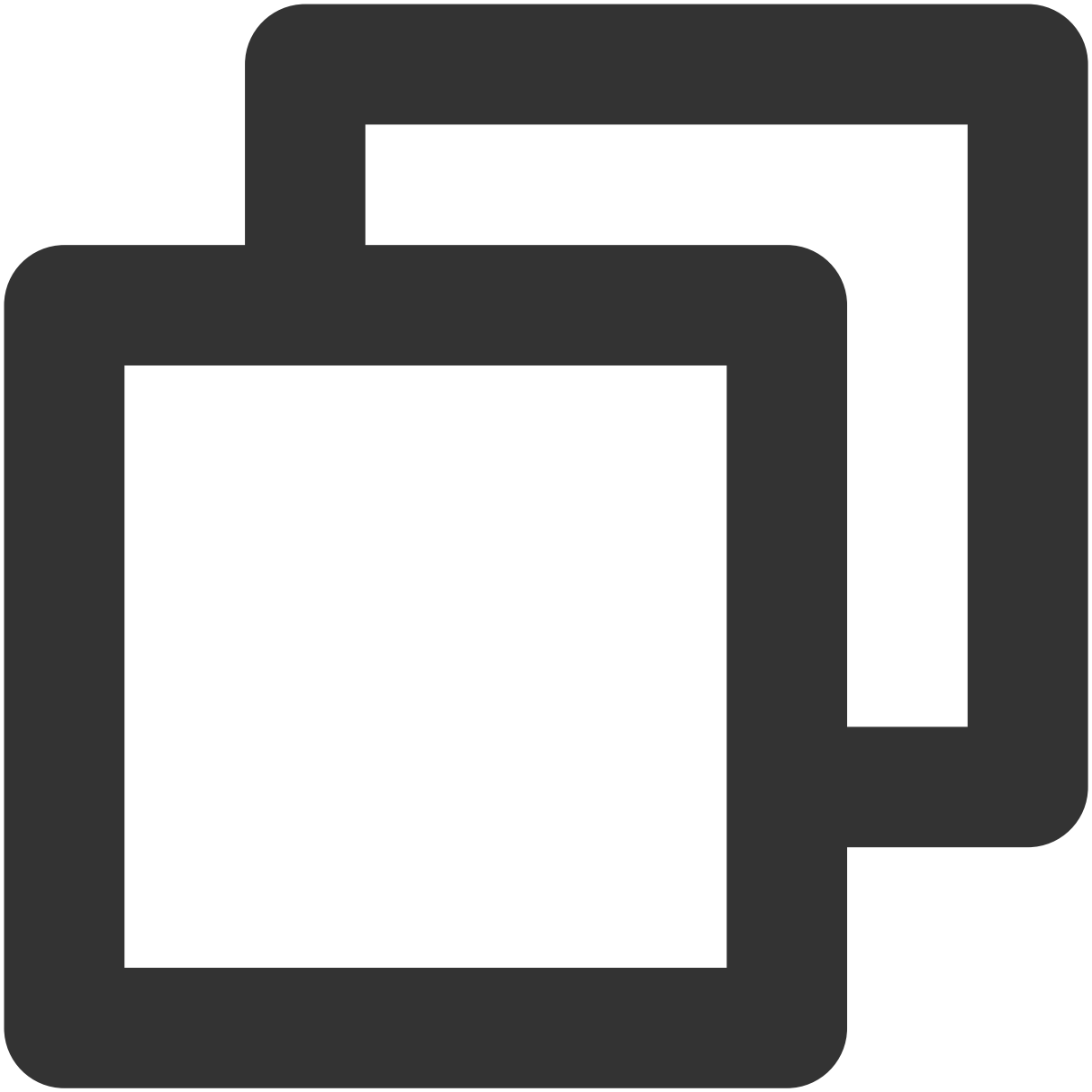
```
SELECT a FROM tbl WHERE id*10=100;
```

Avoid using `SELECT \\*` directly as much as possible; instead, list all fields that need to be queried.
Use `UNION ALL` instead of `UNION` as much as possible, as `UNION` will cause deduplication and sorting.
Principle of using the `WHERE` clause: use index as much as possible and keep it as simple as possible to match fewer rows.
You should use more equality operators in the `WHERE` clause but fewer non-equality operators which will usually invalidate indexes (MySQL does not support range index)
Even though there is an index, the number of rows matched by the `WHERE` clause should not exceed 30% of rows in the table; otherwise, the efficiency will still be low. In the InnoDB storage engine, the index may be directly aborted, and full-table scan will be used.
In the `LIKE` clause, `%` should not be the first character and needs to be placed near the end. You can consider using full-text index for more complex requirements.
When there are more than 3 "OR" conditions:
Use `UNION ALL` instead if the fields are different.
Use `IN` instead if the fields are identical.
Use the `WHERE` clause instead of `HAVING` clause as much as possible; 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;
```

There should be no more three combinations of `ORDER BY` and `GROUP BY` in a table; otherwise, the table should be optimized at the business logic level or split into multiple tables.

If sorting is not needed, write the `GROUP BY` clause as `GROUP BY NULL` .

Use the primary key for the `WHERE` clause as much as possible.

Avoid using full-table scan similar to `COUNT(\\*)` in an InnoDB table as much as possible; instead, consider storing this count value in another table during table design.

Avoid using subqueries as much as possible.

3.2.3. DML statements

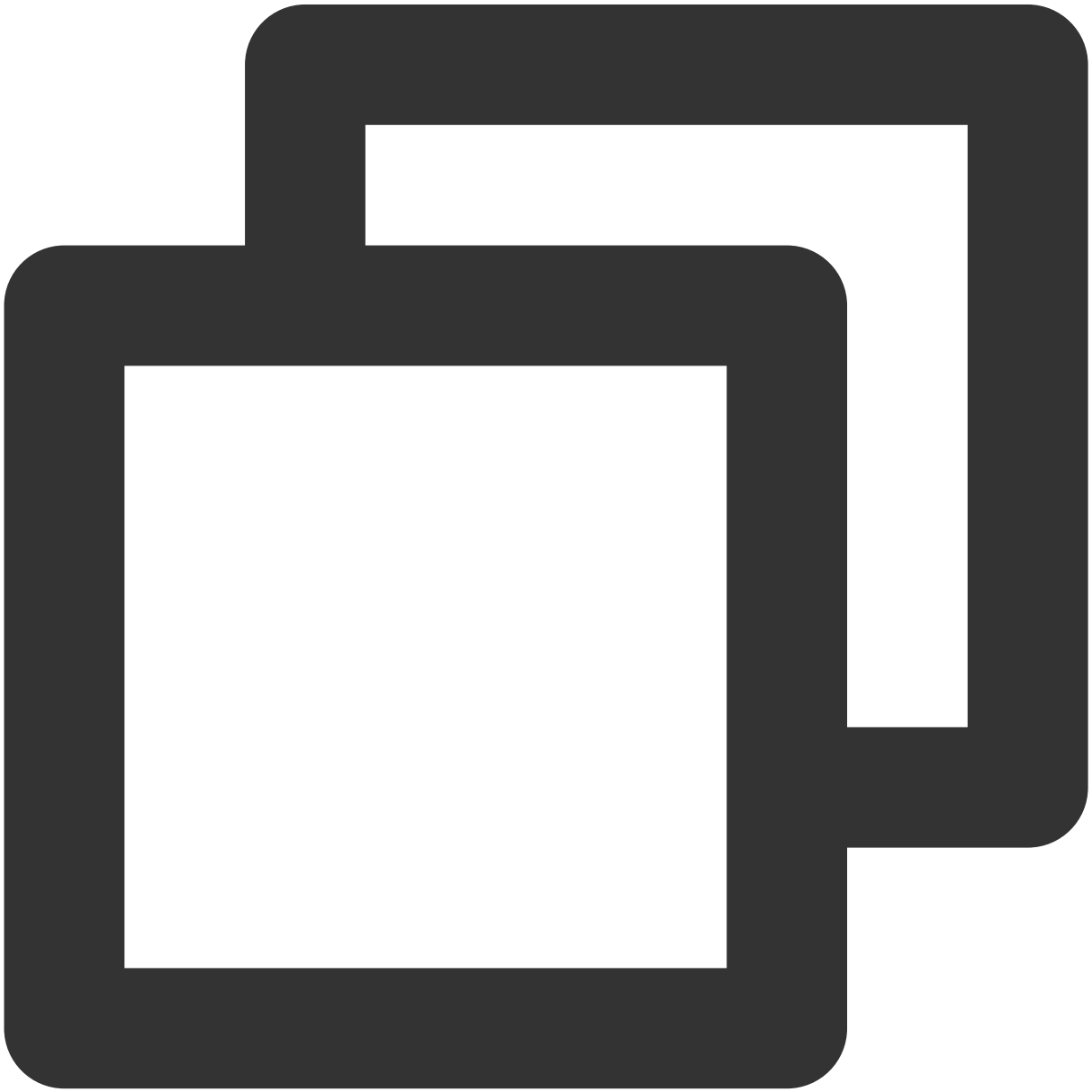
Add the `WHERE` clause to statements such as `UPDATE` and `DELETE` and use the index field and primary key as much as possible.

Use `INSERT ... ON DUPLICATE KEY update (INSERT IGNORE)` to avoid unnecessary queries.

Do not use functions with uncertain values in `INSERT` , `UPDATE` , and `DELETE` statements, such as `RAND()` and `NOW()` .

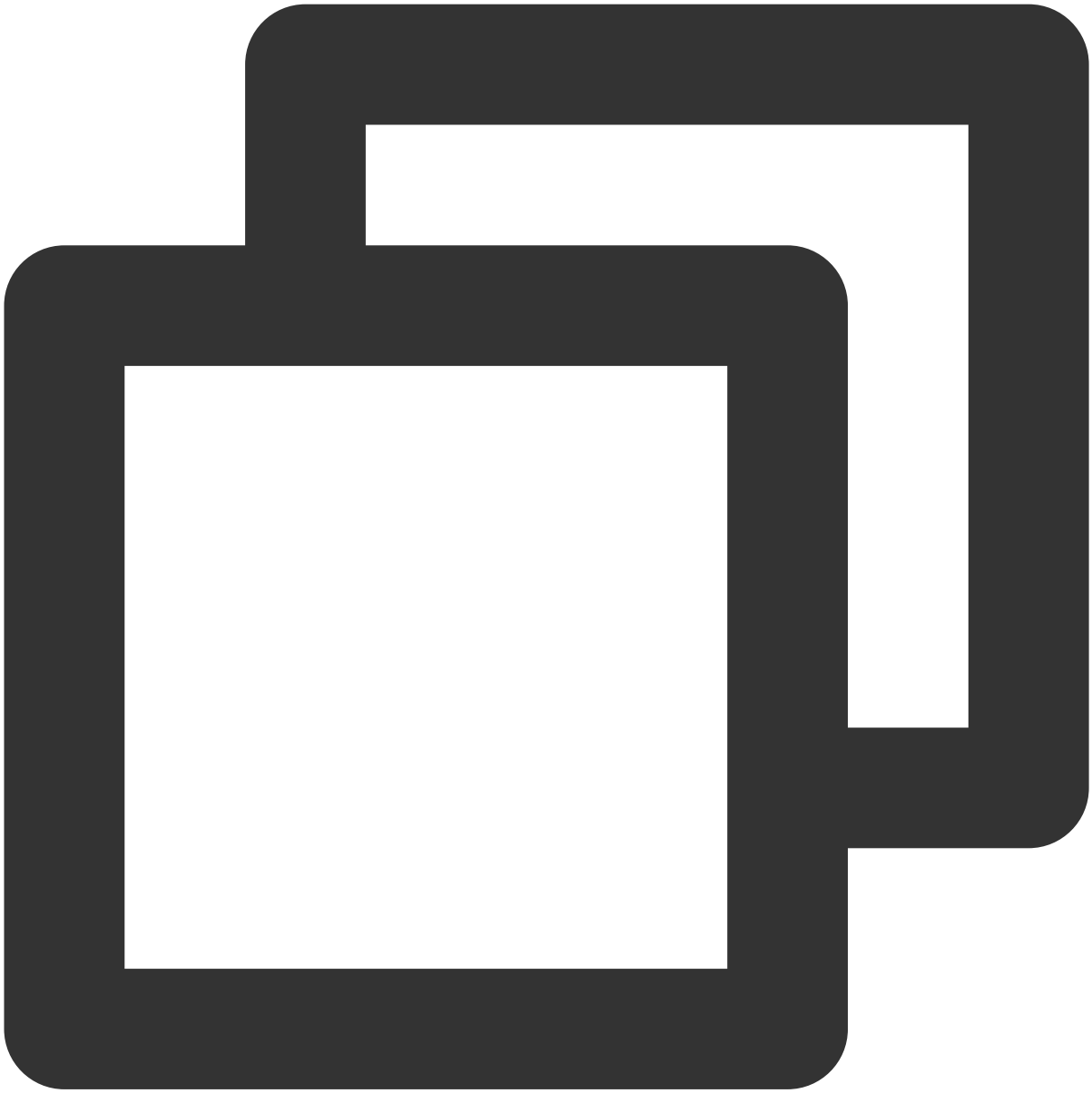
Merge multiple `INSERT` statements into one statement for batch commitment, and do not include more than 500 rows of data at a time.

Do not write `&quot;,&quot;` as `AND` in `UPDATE` statements, as that is very dangerous; for example:



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

If the code above is written as:



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

Then, `fid+1000 AND gid=gid+1000` will be assigned as the value of `fid` with no warning displayed.

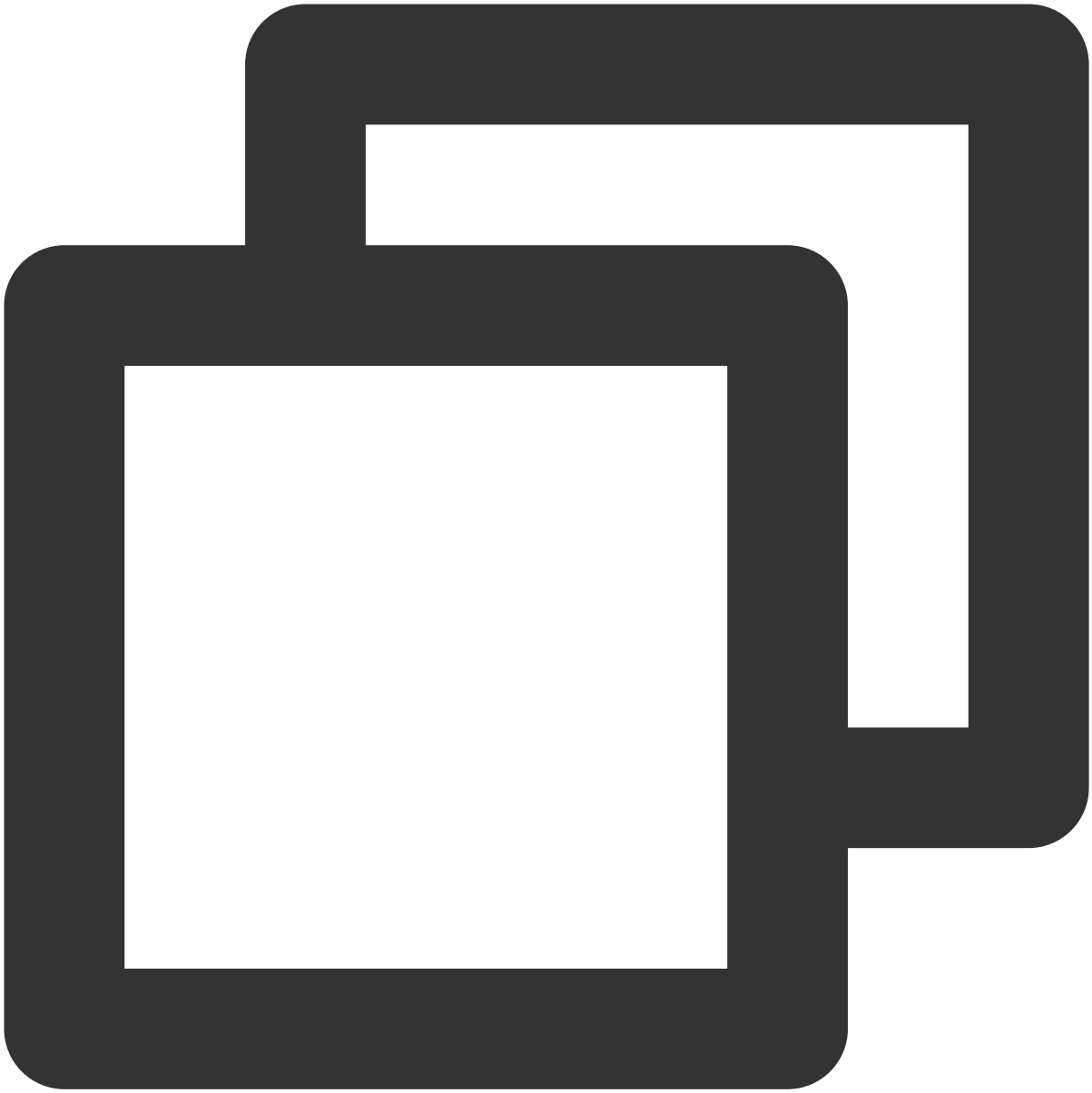
Use `TRUNCATE` instead of `DELETE` to drop a table.

3.2.4. Multi-table JOIN

When a multi-table join is performed, tables should be sorted based on their number of rows returned by the `WHERE` clause condition, with the table containing the fewest rows being at the leftmost.
Avoid JOIN between big tables. You are recommended to use JOIN only when the number of data records in the table is below 100,000.

3.2.5. Miscellaneous

Use the `PREPARE` statement reasonably, which can improve performance and prevent SQL injections.
Split a complicated SQL statement into multiple statements to avoid large transactions.
For DDL operations on a table, perform them in one SQL statement as much as possible; for example, you can use the following statement to add field `b` to table `t` and index the existing field `a` :



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

4. Specifications for Database Management

4.1. Basic requirements

The installer that comes with TencentDB for MariaDB will check the system environment before the installation. Check items include:

Check Item	Expected Result
Operating system version	Linux
File handler	Over 100,000
Time sync	Correct NTP configuration
User	Check whether the user specified in the installer exists
Installation directory	Check whether the installation directory exists and can be written to
Data directory	Check whether the specified data directory exists and can be written to

4.2. Selection between "regular (non-sharding)" and "distributed (horizontal sharding)" schemes

If the following conditions are met, you can consider using the "distributed (horizontal sharding)" scheme:
The data in the table will keep increasing and exceed the maximum storage capacity of one single server in the foreseeable future.
The read/write volume of the table is very high and exceeds the maximum throughput of one single server.
You have taken into full account various restrictions on table design and use that may be caused by database sharding and table splitting.
For more information on the use limits of the "distributed (horizontal sharding)" edition, please see TDSQL Development Guide.

4.3. Master/slave configuration and disaster recovery

Currently, TencentDB has the following capabilities:
TencentDB for MariaDB supports configuring one master and multiple slaves and can automatically elect a slave as master when the master fails.
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).
TencentDB for MariaDB supports two master/slave sync modes (strong sync replication and async replication):
Strong sync replication ensures that there will be no data loss if the master fails; therefore, you are recommended to configure at least two slaves, because if there is only one slave, when the master fails, the remaining slave can only provide read-only services in order to prevent data loss.
The performance of strong sync replication may be lower than that of async replication, and the performance loss is subject to the network quality between the master and slave. Generally, you are recommended to use strong sync replication only in the same data center or across data centers in the same city.
TencentDB for MariaDB will support configuring two logical instances as the master and slave, which is generally used for remote disaster recovery.

4.4. Backup and restoration

Currently, TencentDB has the following capabilities:
An HDFS cluster can be configured for a TencentDB for MariaDB cluster to store cold backups and binlog backups.
TencentDB for MariaDB supports configuring periodic cold backup to HDFS.
Binlogs are backed up to HDFS in real time.
TencentDB for MariaDB supports rollback to a specified time point from cold backups and binlog backups.

4.5. Users and permissions

TencentDB for MariaDB users can be classified into cluster administrators, database instance administrators, and database users:
A cluster administrator mainly uses the OPS management platform to complete the following tasks:
Manage all resource pools of the entire cluster and activate/deactivate and assign resources reasonably.
Assign database instances to instance administrators.
Help instance administrators manage instances.
Manage cold backup and binlog backup data of all instances.
Monitor cluster and instance metrics to ensure normal operations of the cluster and instances.
A database instance administrator mainly uses the database management platform to complete the following tasks:
Submit applications for creating, scaling, and deleting instances.
Maintain custom parameters of instances.
Manage database instance accounts and passwords.
Monitor instance metrics and perform database performance management.
Perform database performance optimization analysis based on information such as slow query analysis, error logs, and slow query logs.
A database user connects to databases through an SQL client to commit SQL statements and access data.

4.6. Log management

Logs of a TencentDB for MariaDB cluster mainly include:
Operations logs of each component in the cluster.
Gateway logs of the instance.
Database operations logs, slow query logs, and error logs of the instance.
Database binlog files of the instance.
Various log backups stored in HDFS.
You should configure reasonable deletion policies for all logs based on the actual needs.

5. Suggestions on Performance Optimization

5.1. Principles of performance optimization

Database performance optimization usually covers two aspects: methodology and characteristics/tools. The methodology involves 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 optimization: you should write efficient SQL statements, optimize relevant database objects, and develop the optimal execution plan by fully leveraging optimizers.

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, sort them by importance in performance optimization, and implement them in sequence. Do not implement all optimization measures at a time; instead, you must try and compare them one by one.
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 tips on optimization

View the `long_query` and `long_query_rate` metrics of the instance to analyze the frequency and pattern of slow query occurrences.
Use the slow query analysis tool to analyze SQL statements starting from the most frequent and slowest one and optimize them by adding indexes.
View the `mem_hit_rate` and `mem_available` metrics of the instance to analyze whether the buffer pool of InnoDB is sufficient and whether the memory reaches a bottleneck.
View `cpu_usage_rate` with slow query analysis to check whether the CPU utilization is reasonable and whether the CPU performance reaches a bottleneck.
Adjust the `innodb_page_size` parameter and compare performance test results to find the most appropriate configuration.
Use `EXPLAIN` to query and analyze frequently used statements to find potential design problems.

Design appropriate use cases based on business scenarios to test the performance of instances with different specifications.

Programming and Usage Specification of Distributed Version

Last updated：2024-01-11 15:21:58

For more information, please see TDSQL Development Manual.

Using Hotspot Update for Flash Sales

Last updated：2024-01-11 15:21:58

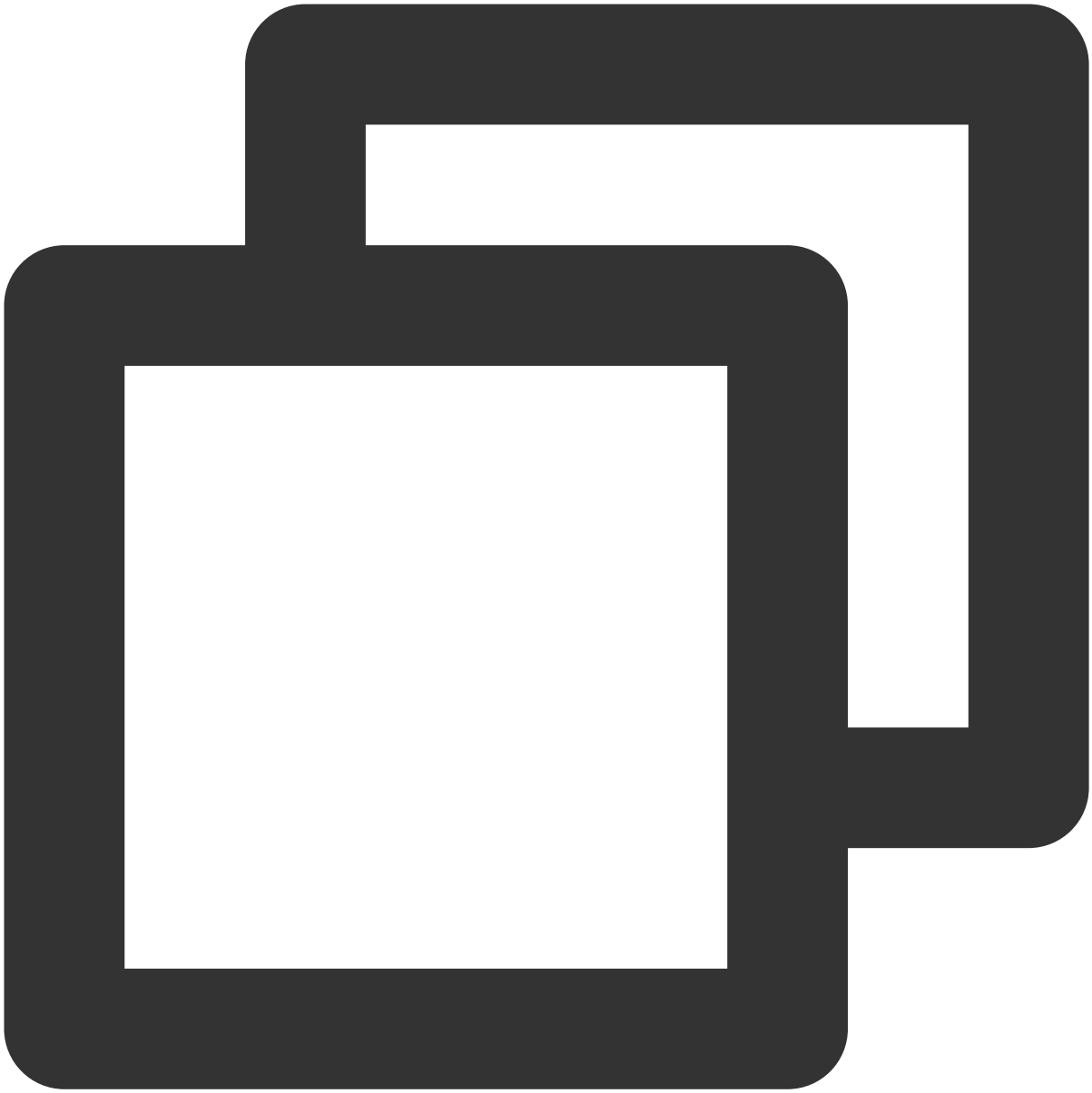
1. Introduction

1.1. Background

In scenario such as flash sales, a large number of users send access requests for a large number of items in a very short period of time. In a MySQL database, an item is stored in one data row; therefore, many threads will compete for the InnoDB row lock, and the higher the concurrence, the more the waiting threads. In this case, TPS will decrease, RT will increase, and the throughput of the database will be severely affected. This document describes the special optimization made by TencentDB for MariaDB for such scenarios: hotspot update technology.

1.2. How to use

Hotspot update: The statement as shown below is used to frequently update a data object.
Currently, this feature is only supported in Percona 5.7.17, which can be purchased on the TencentDB for MariaDB purchase page.



```
UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 88 TARGET_AFFECT_ROW 1 table_
```

2. 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_FAIL] [QUEUE_ON_PK 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 | DELAYED | HIGH_PRIORITY]
        [COMMIT_ON_SUCCESS] [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 the standard syntax, see the official documentation: UPDATE Syntax and INSERT Syntax.
5. Hotspot update is implemented on the object with the expr value specified by QUEUE_ON_PK. Generally, the expr value is a positive integer.
6. Parameter description:

`COMMIT_ON_SUCCESS` : Immediately commits a successful update, which is applicable to a single statement as a transaction.

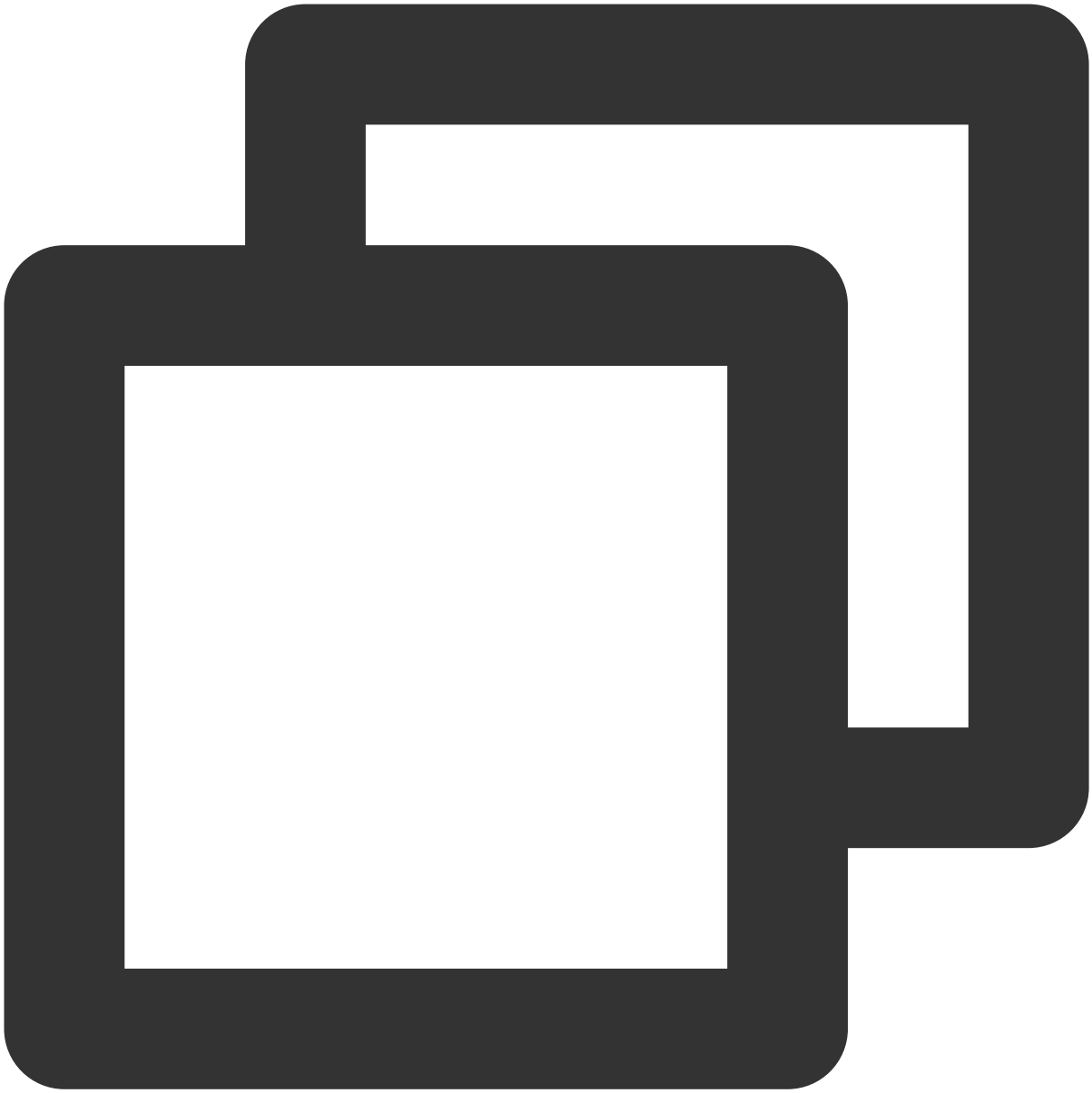
`ROLLBACK_ON_FAIL` : Immediately performs rollback if update fails, which is applicable to a single statement as a transaction.

`QUEUE_ON_PK expr` : Specifies the hotspot update object and locks/unlocks the updated object. The total number of updated objects cannot exceed `hot_commodity_query_size` , i.e., the number of `exprs` with different values cannot exceed `hot_commodity_query_size` . The value of `expr` is arbitrary, but it is recommended to keep it the same as the primary key (a different value is also acceptable though).

`TARGET_AFFECT_ROW expr` : Specifies the data row affected by hotspot update, where `expr` is a positive integer ([1, MAX] with MAX being the maximum 8-digit number). Generally, `expr` is 1, indicating that only one row will be affected.

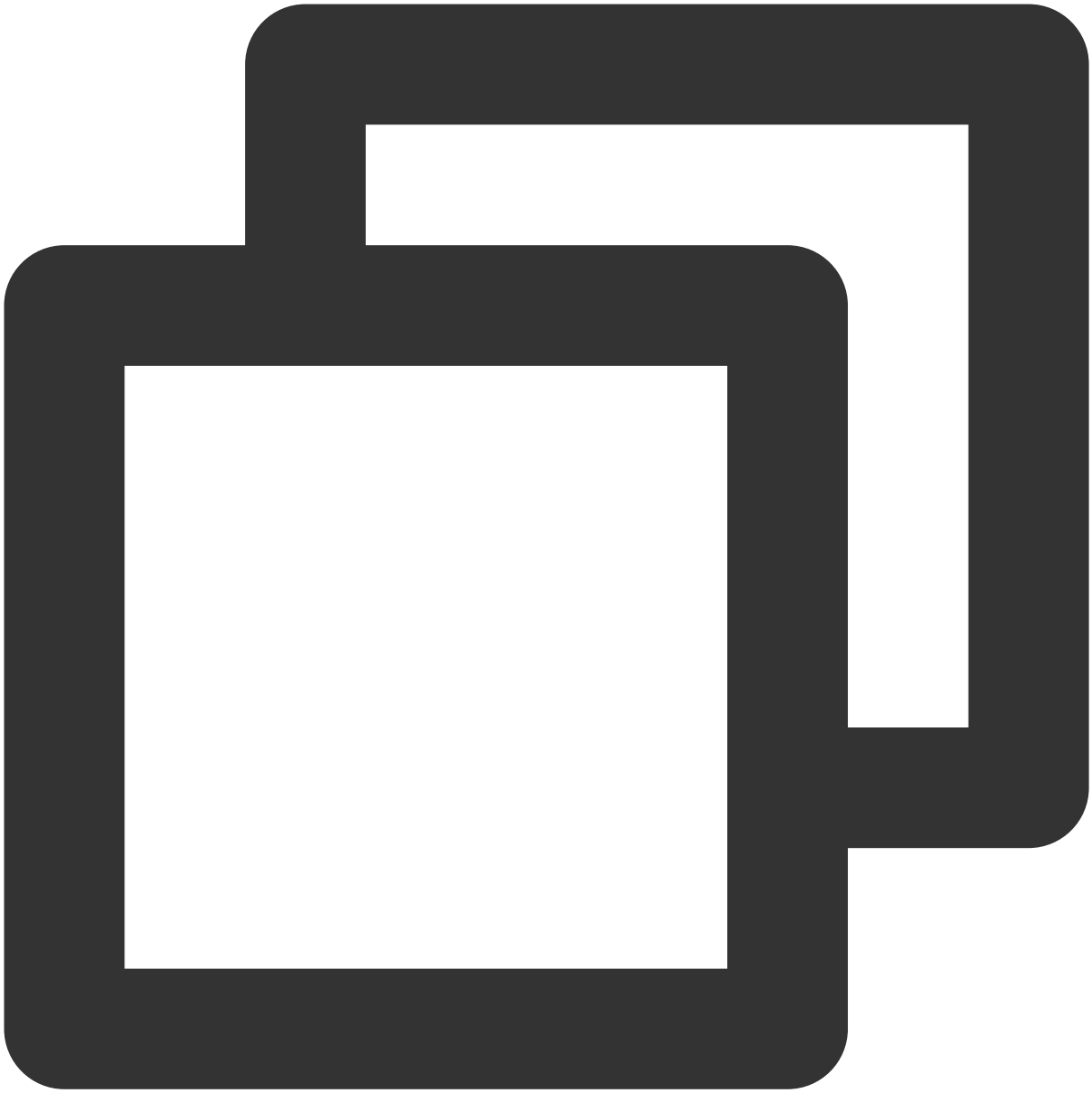
2.4. Suggestions

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_
```

2.5. Sample code

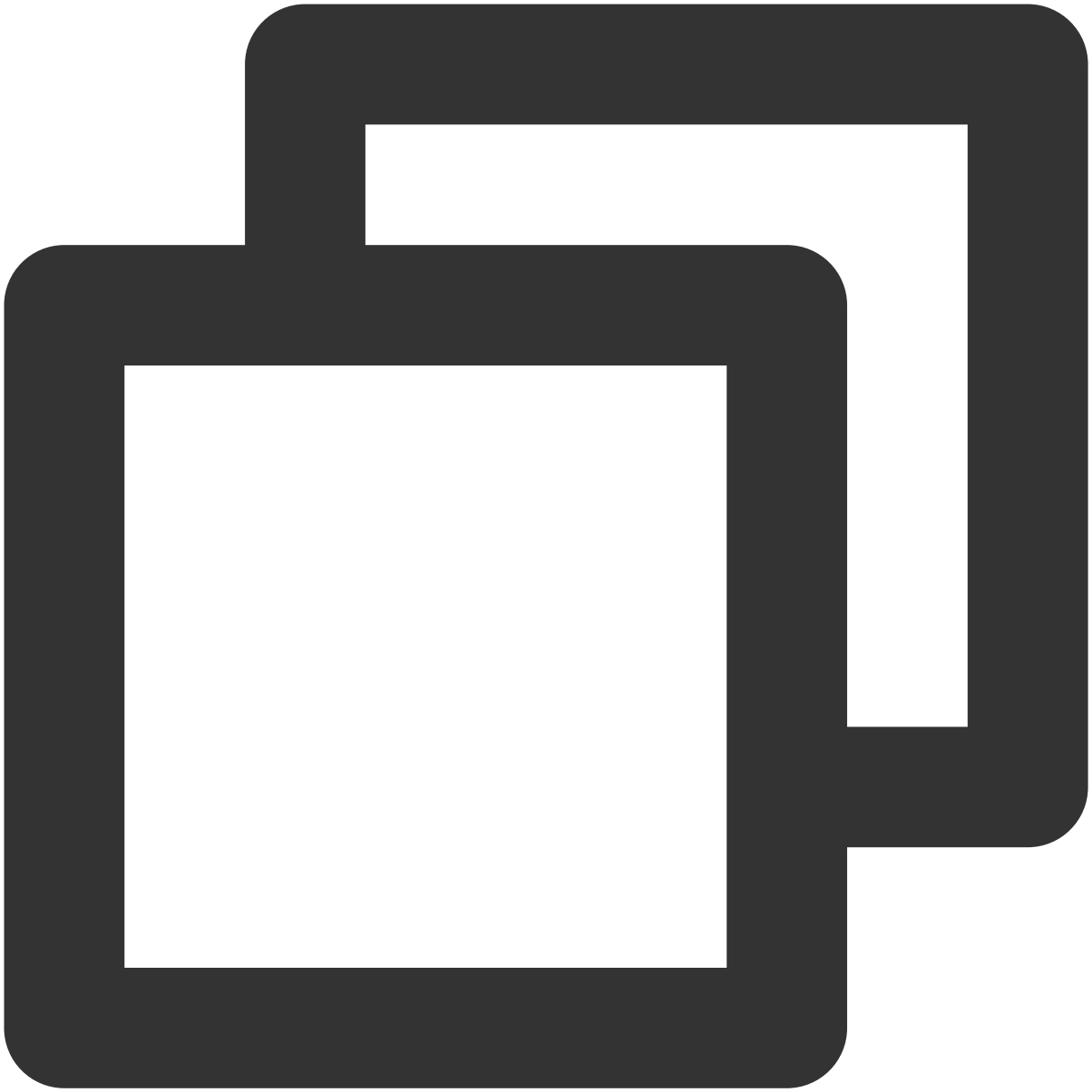


```
CREATE DATABASE hc_db;

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

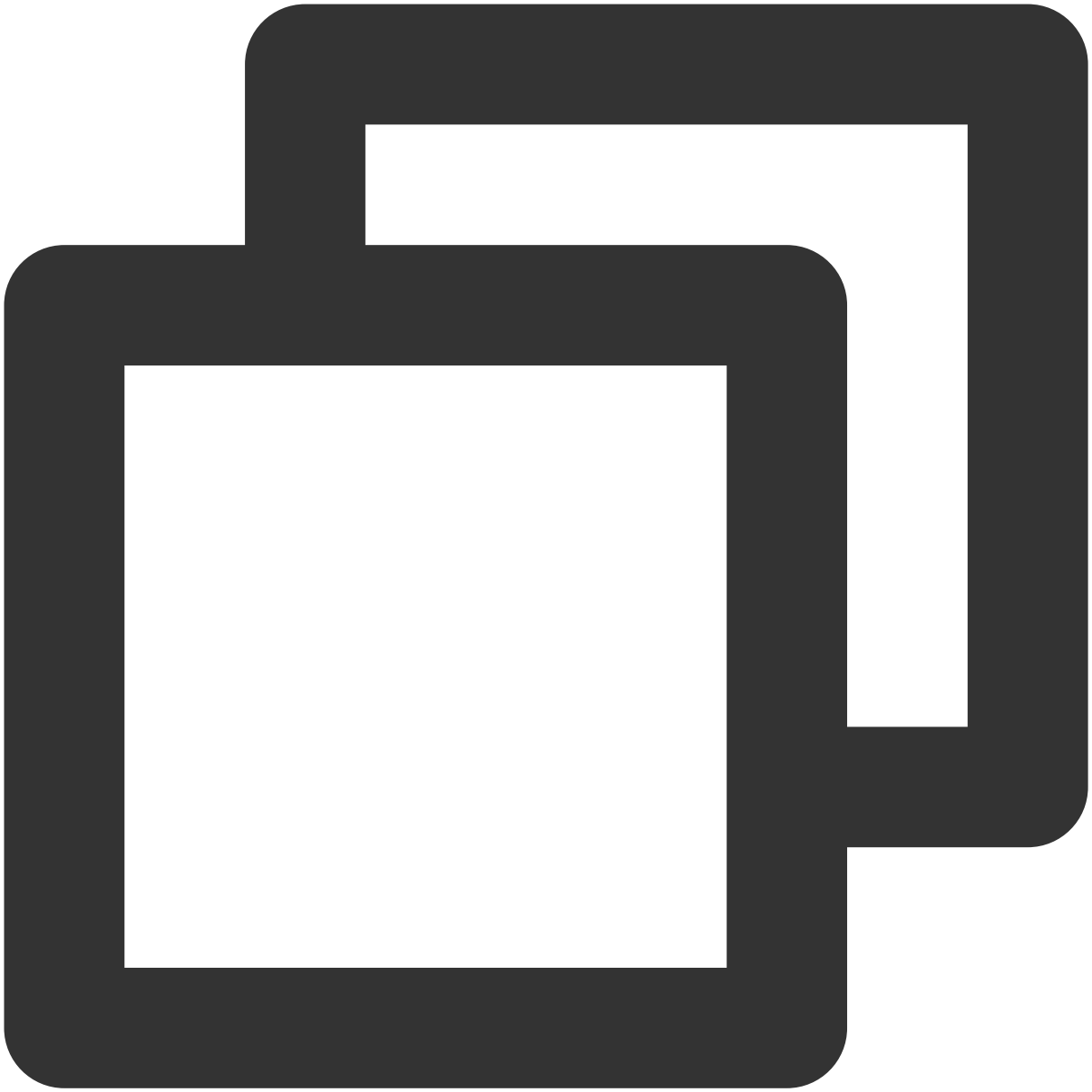


```
INSERT COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 1 INTO hc_tbl VALUES(1, 1, 1)

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
```

2.5.2. Sample code of UPDATE



```
UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 1 TARGET_AFFECT_ROW 1 hc_tbl

`expr` in `QUEUE_ON_PK expr` is not necessarily the same as that in the WHERE claus

UPDATE COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL QUEUE_ON_PK 2 TARGET_AFFECT_ROW 1 hc_tbl
```

3. Newly Added Parameter Description

Parameter Name	Feature	Type
`hot_commodity_enable`	Enables/Disables hotspot update	Boolean
`hot_commodity_trace`	Enables/Disables trace	Boolean
`hot_commodity_query_size`	Controls the number of hotspot objects that can be updated/inserted	Numeric
`hot_commodity_query_size_modify_enable`	Controls whether `hot_commodity_query_size` can be modified	Boolean

Note:

When the MySQL server is started, if the `hot_commodity_enable` parameter is disabled, you need to enable it and restart the server before you can initialize the global data table. However, if `hot_commodity_query_size` is 0, even if the `hot_commodity_enable` parameter is enabled, hotspot update is still unavailable. Therefore, it is also necessary to set the following:

`hot_commodity_enable` =ON

`hot_commodity_query_size` =10000; it is a value greater than 0 that preferably falls between 10,000 and 20,000, depending on the hardware environment and application pressure.

Start the server