腾讯云数据仓库 TCHouse-D

实践教程

产品文档

腾讯云

文档目录

实践教程
基本功能使用

最近更新时间：2024-07-31 09:16:31

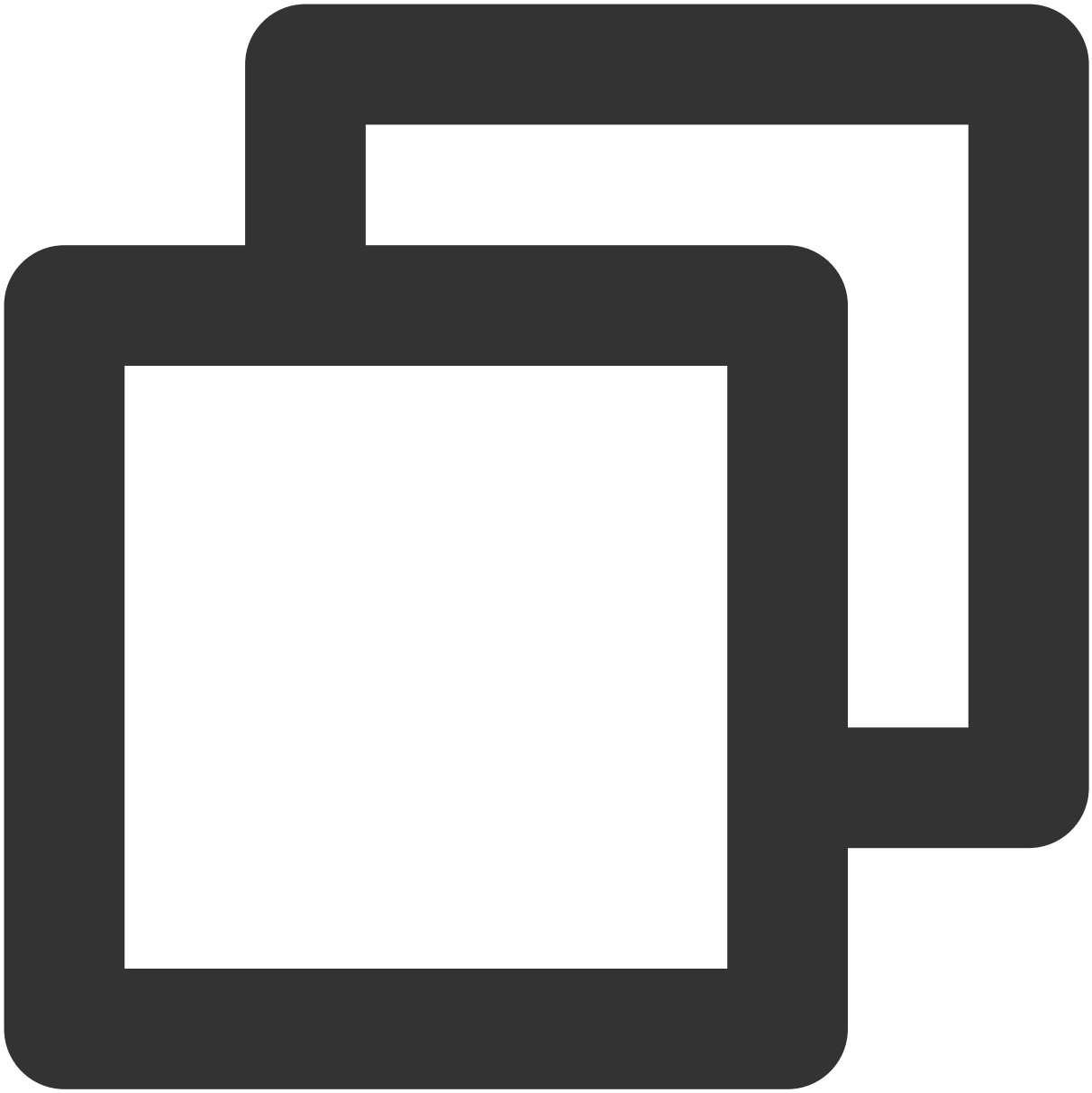
建表

数据模型选择

Doris 数据模型上目前分为三类: AGGREGATE KEY，UNIQUE KEY，DUPLICATE KEY。三种模型中数据都是按 KEY 进行排序。

1. AGGREGATE KEY

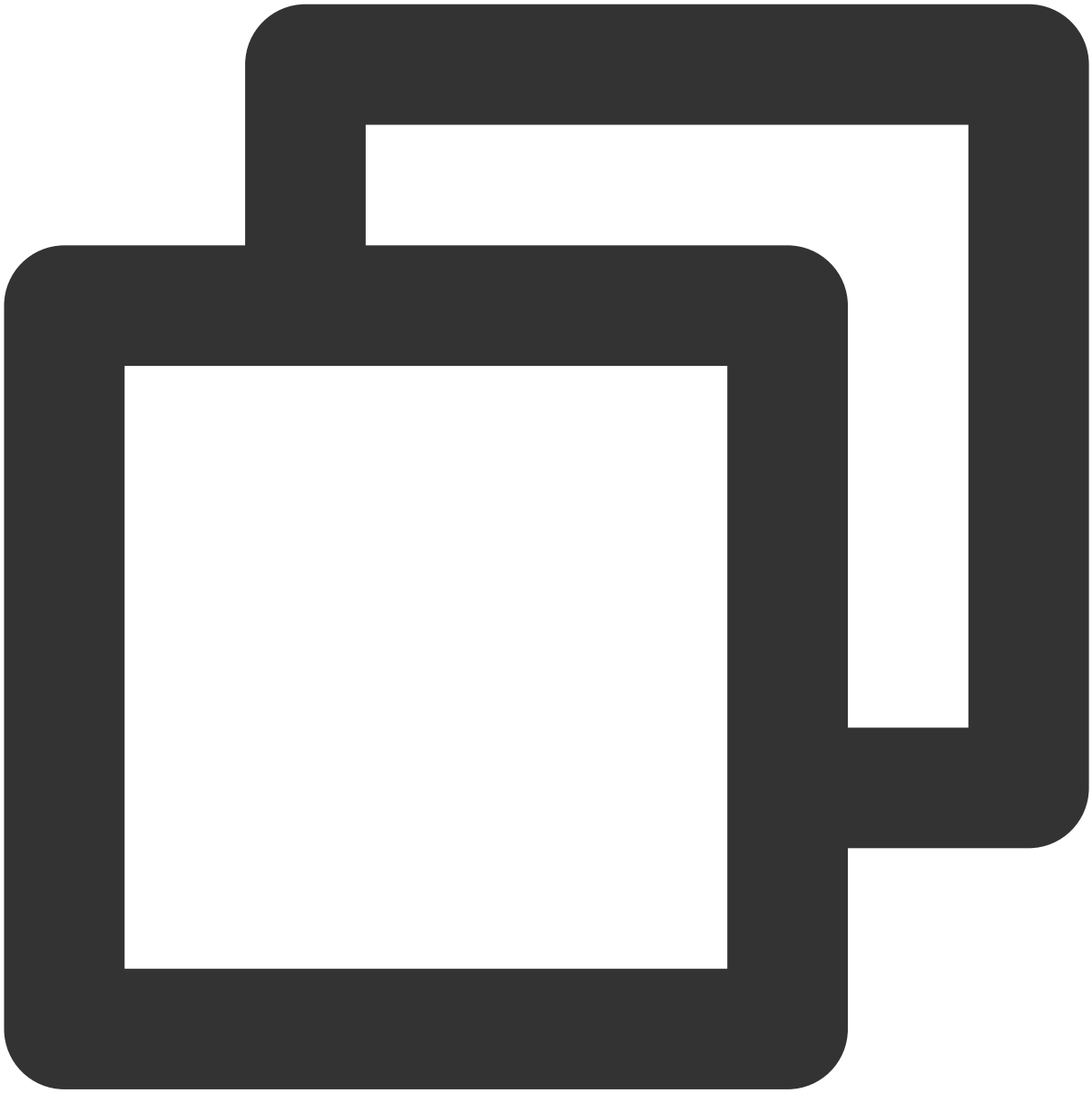
AGGREGATE KEY 相同时，新旧记录进行聚合，目前支持的聚合函数有 SUM，MIN，MAX，REPLACE。AGGREGATE KEY 模型可以提前聚合数据, 适合报表和多维分析业务。



```
CREATE TABLE site_visit
(
 siteid      INT,
 city        SMALLINT,
 username    VARCHAR(32),
 pv BIGINT   SUM DEFAULT '0'
)
AGGREGATE KEY(siteid, city, username)
DISTRIBUTED BY HASH(siteid) BUCKETS 10;
```

2. UNIQUE KEY

UNIQUE KEY 相同时，新记录覆盖旧记录。目前 UNIQUE KEY 实现上和 AGGREGATE KEY 的 REPLACE 聚合方法一样，二者本质上相同。适用于有更新需求的分析业务。

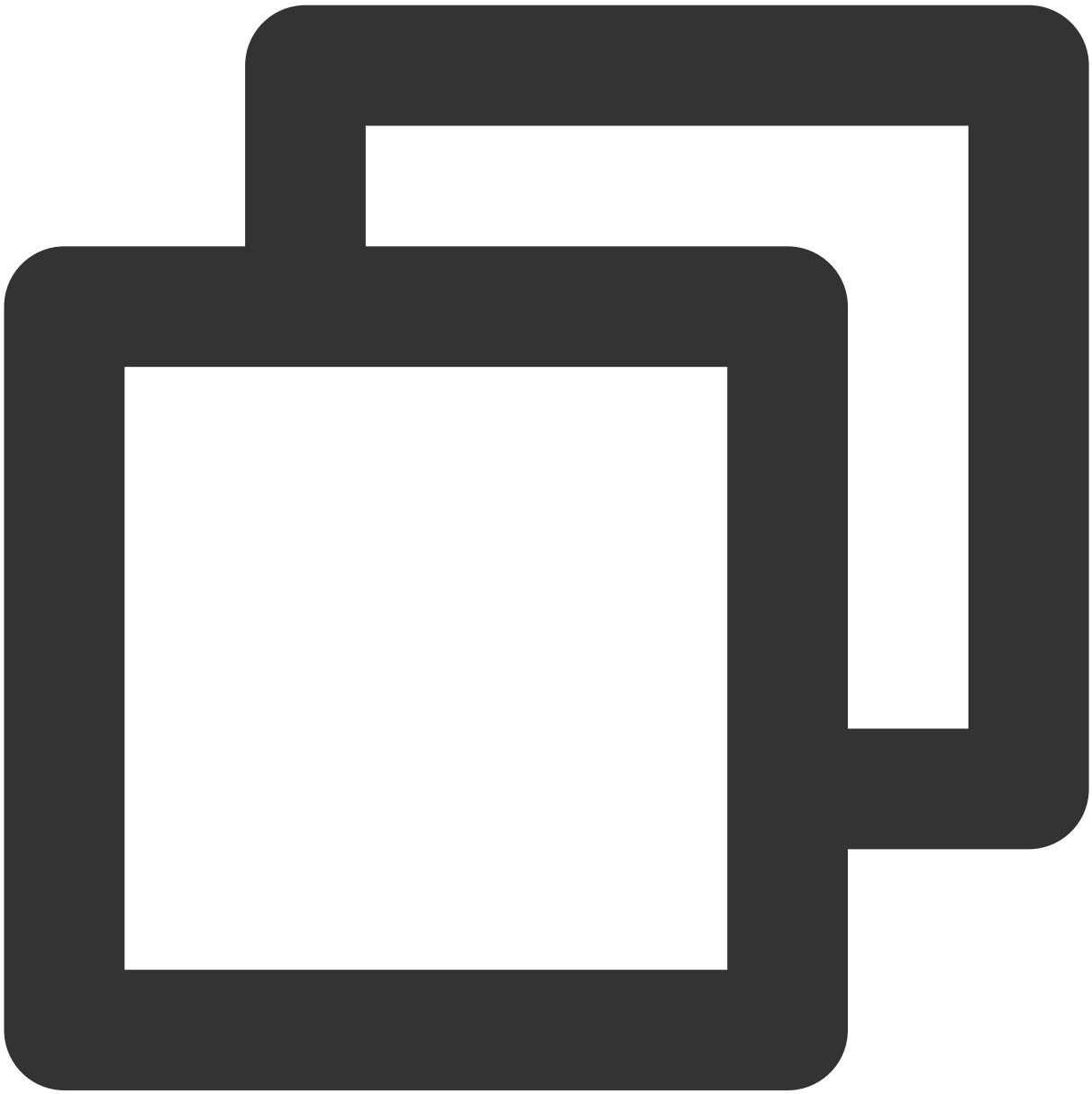


```
CREATE TABLE sales_order
(
     orderid     BIGINT,
     status      TINYINT,
     username    VARCHAR(32),
     amount      BIGINT DEFAULT '0'
)
```

```
UNIQUE KEY(orderid)
DISTRIBUTED BY HASH(orderid) BUCKETS 10;
```

3. DUPLICATE KEY

只指定排序列，相同的行不会合并。适用于数据无需提前聚合的分析业务。



```
CREATE TABLE session_data
(
     visitorid   SMALLINT,
     sessionid   BIGINT,
     visittime   DATETIME,
```

```
    city         CHAR(20),
    province     CHAR(20),
    ip           varchar(32),
    brower       CHAR(20),
    url          VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid)
DISTRIBUTED BY HASH(sessionid, visitorid) BUCKETS 10;
```

大宽表与 Star Schema

业务方建表时，为了和前端业务适配，往往不对维度信息和指标信息加以区分，而将 Schema 定义成大宽表。对于 Doris 而言，这类大宽表往往性能不尽如人意。
Schema 中字段数比较多，聚合模型中可能 key 列比较多，导入过程中需要排序的列会增加。
维度信息更新会反应到整张表中，而更新的频率直接影响查询的效率。
使用过程中，建议用户尽量使用 Star Schema 区分维度表和指标表。频繁更新的维度表也可以放在 MySQL 外部表中。而如果只有少量更新, 可以直接放在 Doris 中。在 Doris 中存储维度表时，可对维度表设置更多的副本，提升 Join 的性能。

分区和分桶

Doris 支持两级分区存储，第一层为分区(partition)，目前支持 RANGE 分区和 LIST 分区两种类型，第二层为 HASH 分桶(bucket)。
1. 分区(partition)
分区用于将数据划分成不同区间，逻辑上可以理解为将原始表划分成了多个子表。可以方便的按分区对数据进行管理，例如，删除数据时，更加迅速。
1.1 RANGE 分区。
业务上，多数用户会选择采用按时间进行partition。
1.2 LIST 分区。
业务上，用户可以选择城市或者其他枚举值进行 partition。
2. HASH 分桶(bucket)。
根据 hash 值将数据划分成不同的 bucket。
建议采用区分度大的列做分桶， 避免出现数据倾斜。
为方便数据恢复，建议单个 bucket 的 size 不要太大, 保持在 10GB 以内，所以建表或增加 partition 时请合理考虑 bucket 数目，其中不同 partition 可指定不同的 buckets 数。

稀疏索引和 Bloom Filter

Doris 对数据进行有序存储，在数据有序的基础上为其建立稀疏索引，索引粒度为 block(1024行)。
稀疏索引选取 schema 中固定长度的前缀作为索引内容, 目前 Doris 选取 36 个字节的前缀作为索引。
建表时建议将查询中常见的过滤字段放在 Schema 的前面，区分度越大，频次越高的查询字段越往前放。

这其中有一个特殊的地方，就是 varchar 类型的字段。varchar 类型字段只能作为稀疏索引的最后一个字段。索引会在 varchar 处截断，因此 varchar 如果出现在前面，可能索引的长度可能不足 36 个字节。具体可以参阅 数据模型、ROLLUP 及前缀索引。

除稀疏索引之外，Doris 还提供 bloomfilter 索引， bloomfilter 索引对区分度比较大的列过滤效果明显。 如果考虑到 varchar 不能放在稀疏索引中，可以建立 bloomfilter 索引。

物化视图(rollup)

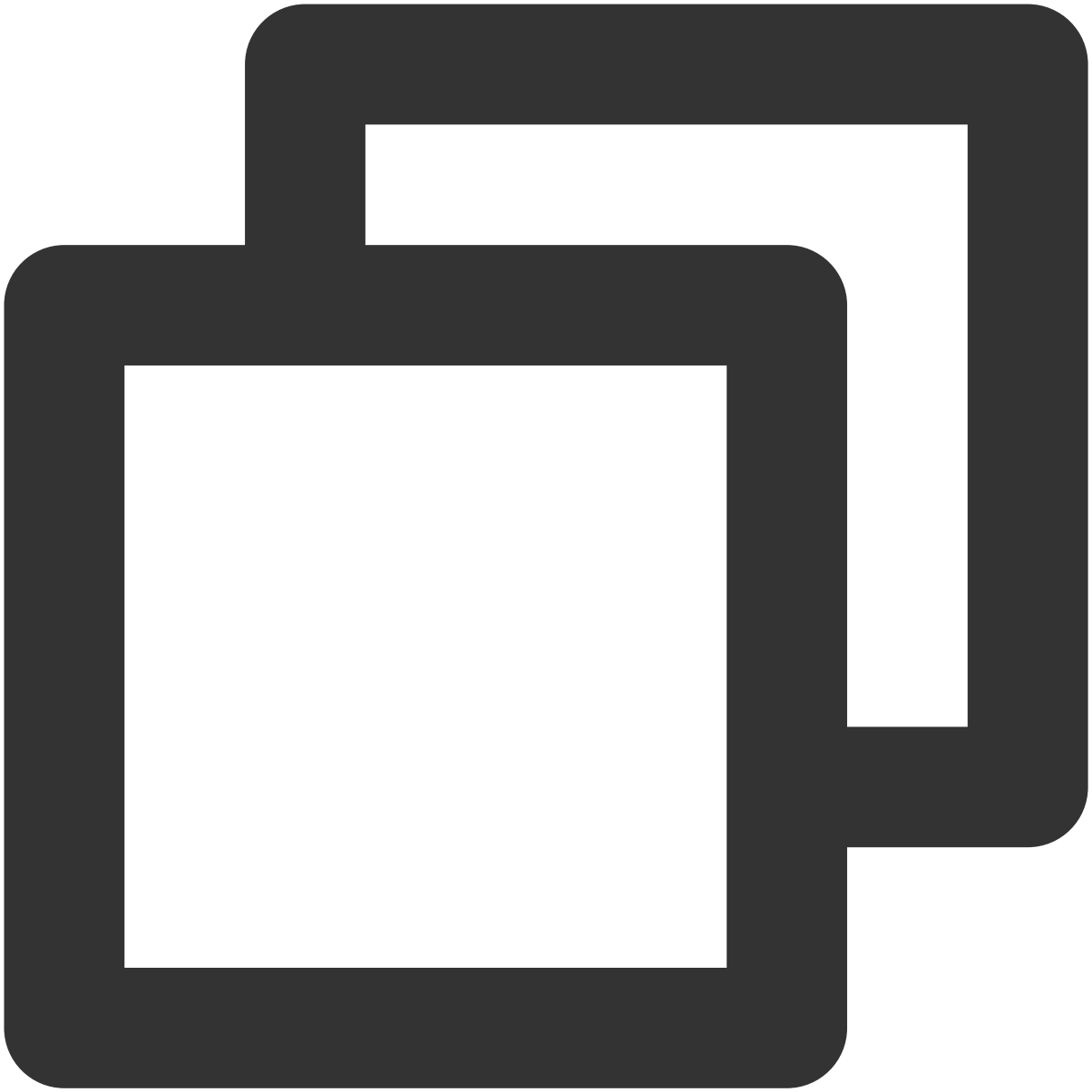
Rollup 本质上可以理解为原始表(Base Table)的一个物化索引。建立 Rollup 时可只选取 Base Table 中的部分列作为 Schema。Schema 中的字段顺序也可与 Base Table 不同。

下列情形可以考虑建立 Rollup：

1. Base Table 中数据聚合度不高。

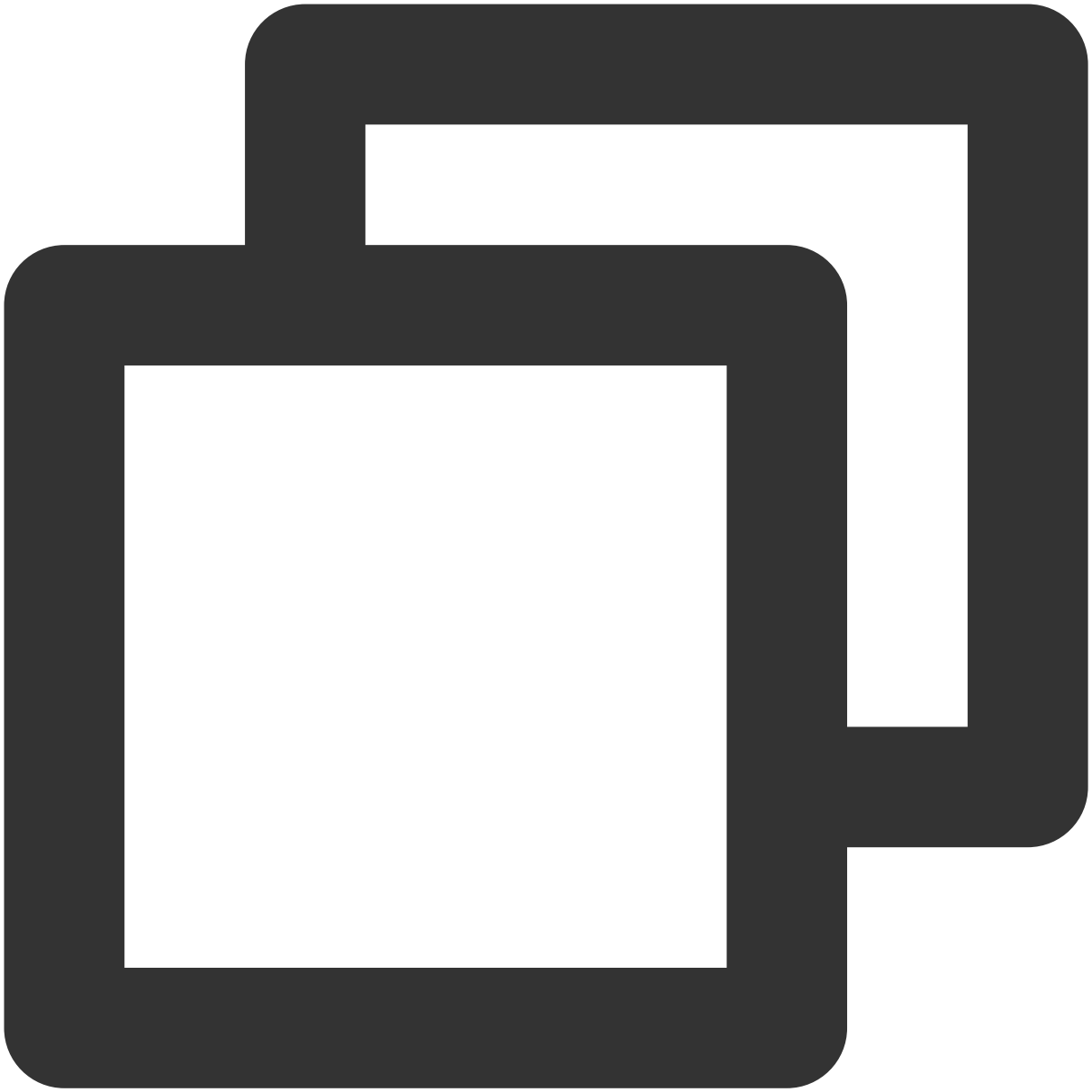
这一般是因 Base Table 有区分度比较大的字段而导致。此时可以考虑选取部分列，建立 Rollup。

如对于 `site_visit` 表：



```
site_visit(siteid, city, username, pv)
```

siteid 可能导致数据聚合度不高，如果业务方经常根据城市统计pv需求，可以建立一个只有 city, pv 的 Rollup：

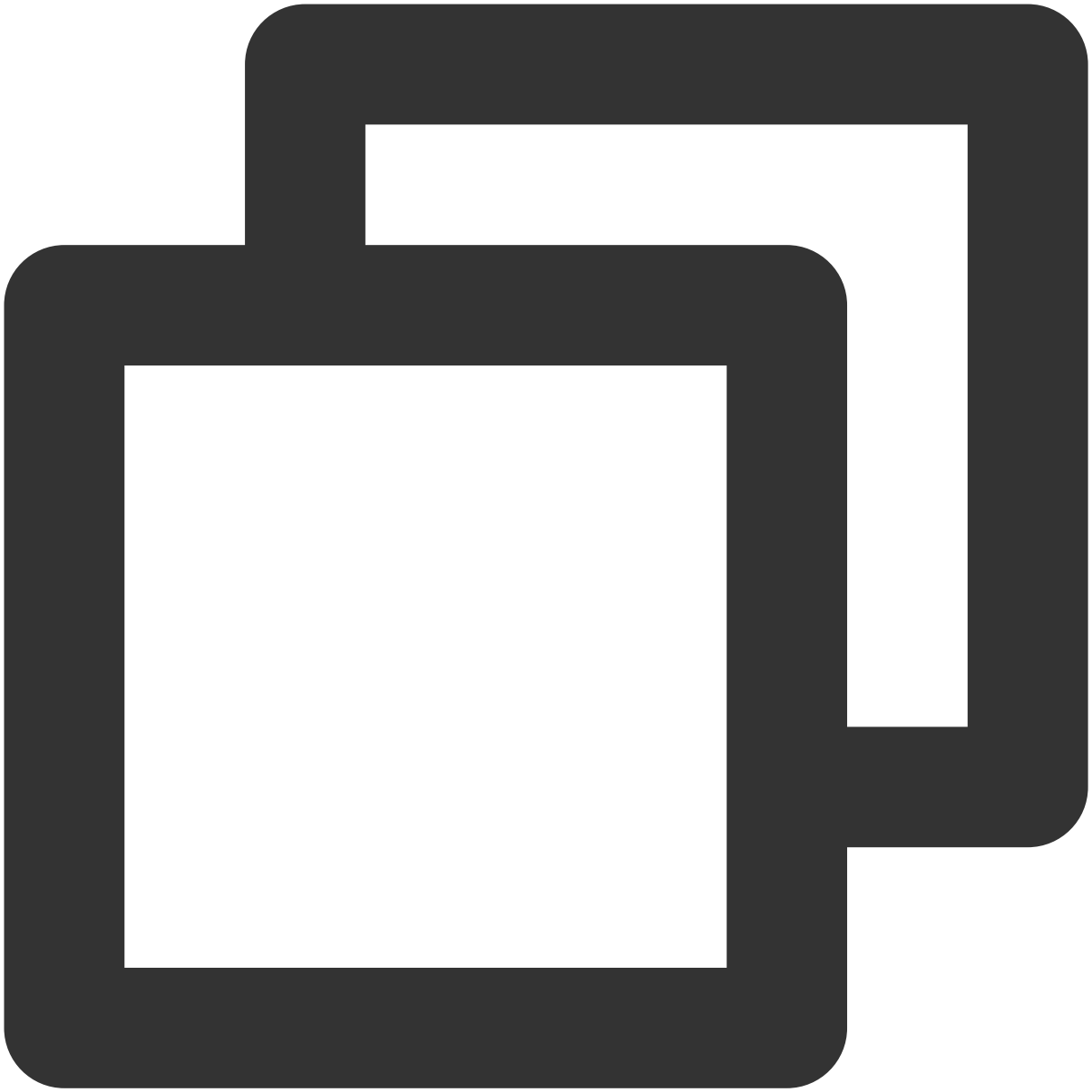


```
ALTER TABLE site_visit ADD ROLLUP rollup_city(city, pv);
```

2. Base Table 中的前缀索引无法命中。

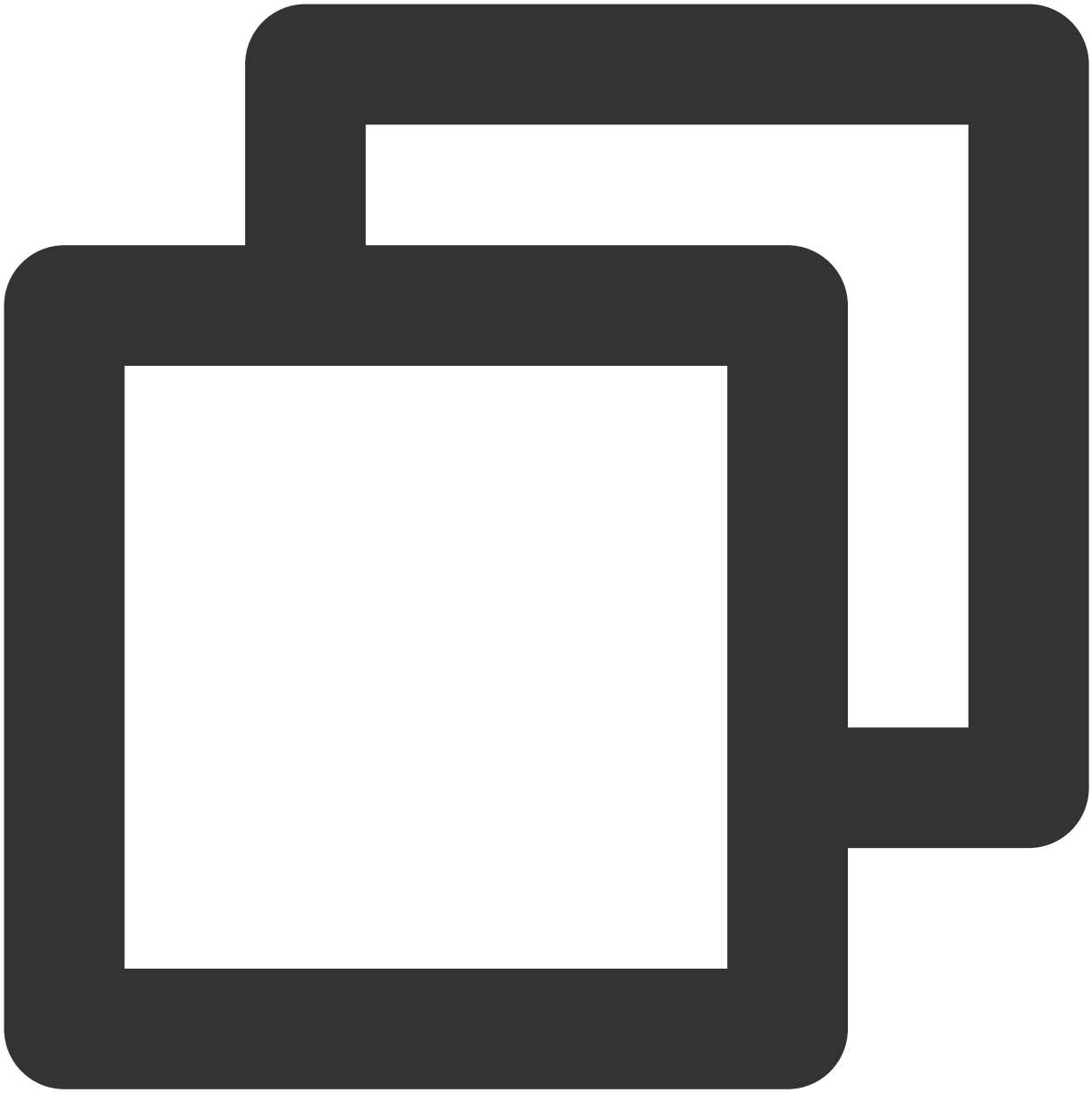
这一般是 Base Table 的建表方式无法覆盖所有的查询模式。此时可以考虑调整列顺序，建立 Rollup。

如对于 session_data 表：



```
session_data(visitorid, sessionid, visittime, city, province, ip, brower, url)
```

如果除了通过 visitorid 分析访问情况外，还有通过 brower， province 分析的情形，可以单独建立 Rollup。



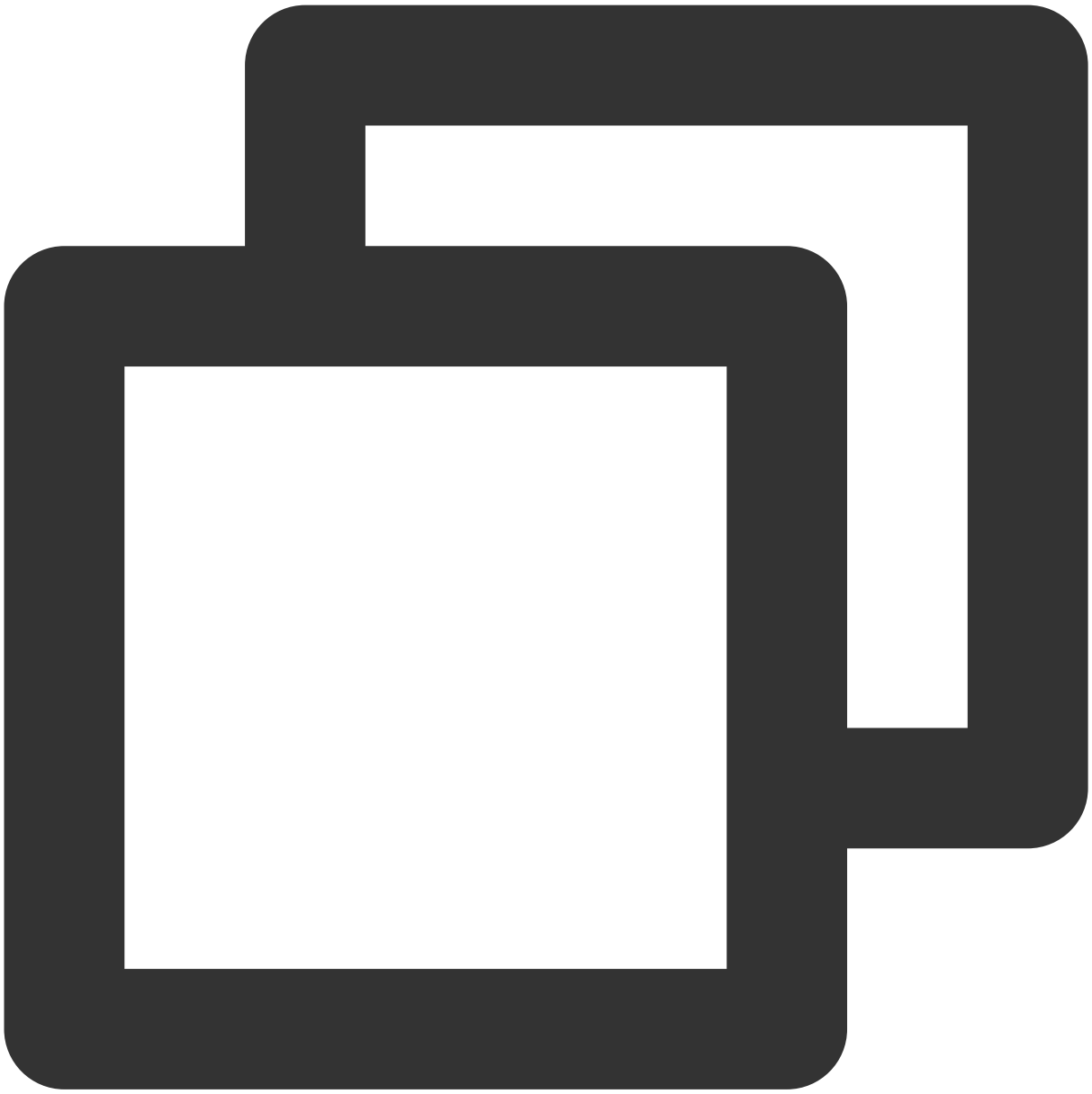
```
ALTER TABLE session_data ADD ROLLUP rollup_brower(brower,province,ip,url) DUPLICATE
```

Schema Change

Doris 中目前进行 Schema Change 的方式有三种：Sorted Schema Change，Direct Schema Change, Linked Schema Change。

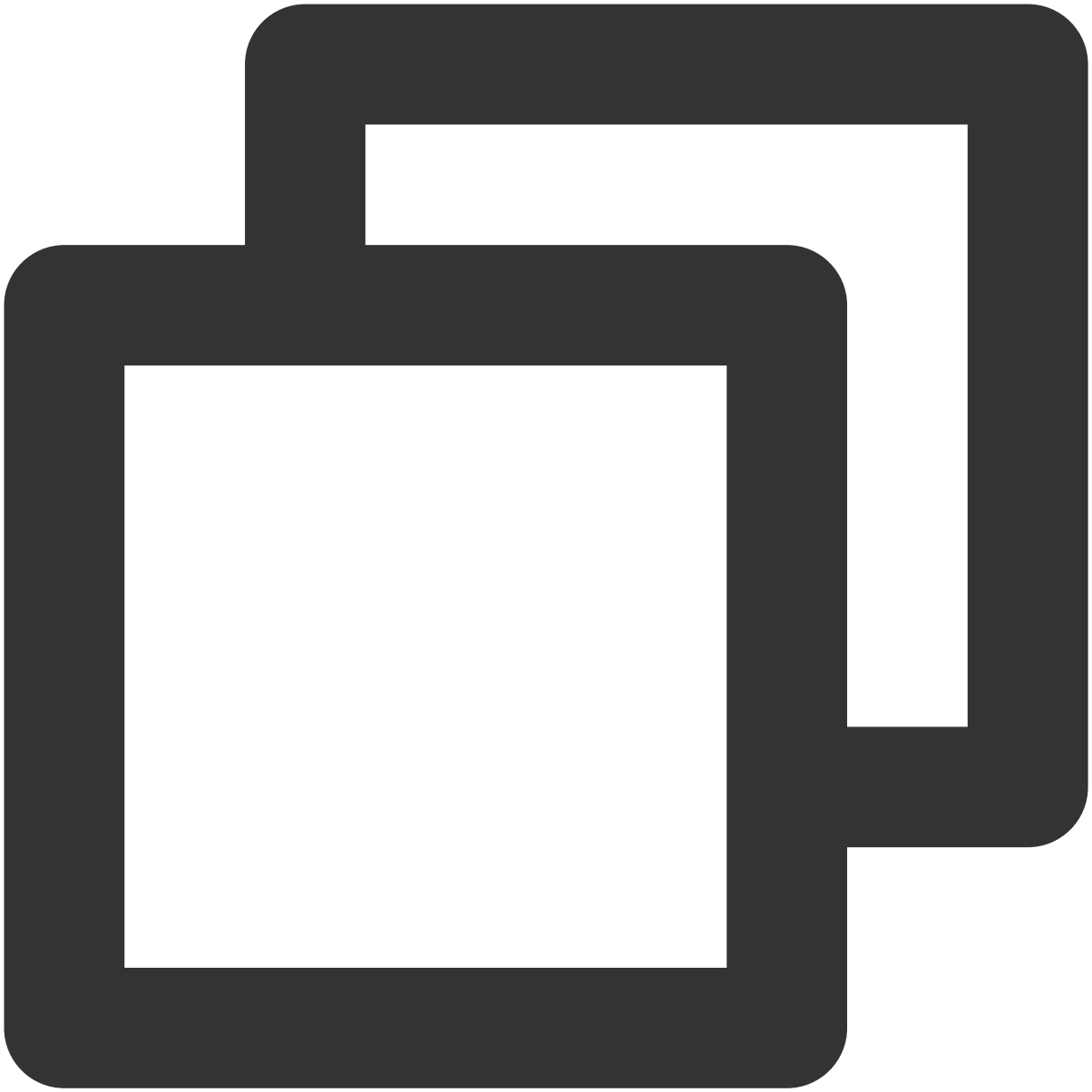
1. Sorted Schema Change

改变了列的排序方式，需对数据进行重新排序。例如删除排序列中的一列, 字段重排序。



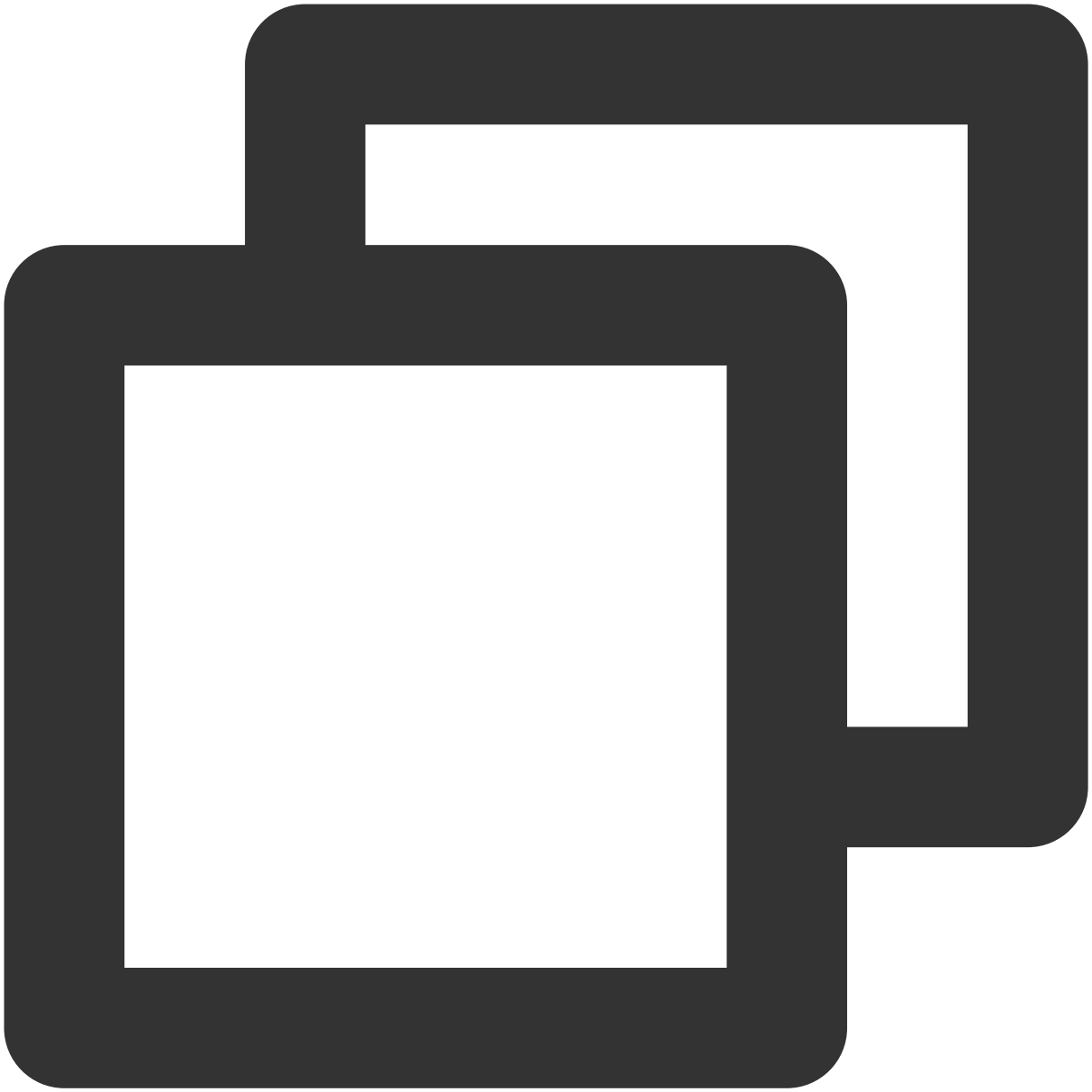
```
ALTER TABLE site_visit DROP COLUMN city;
```

2. Direct Schema Change: 无需重新排序，但是需要对数据做一次转换。例如修改列的类型，在稀疏索引中加一列等。



```
ALTER TABLE site_visit MODIFY COLUMN username varchar(64);
```

3. Linked Schema Change： 无需转换数据，直接完成。例如加列操作。



```
ALTER TABLE site_visit ADD COLUMN click bigint SUM default '0';
```

建表时建议考虑好 Schema，这样在进行 Schema Change 时可以加快速度。

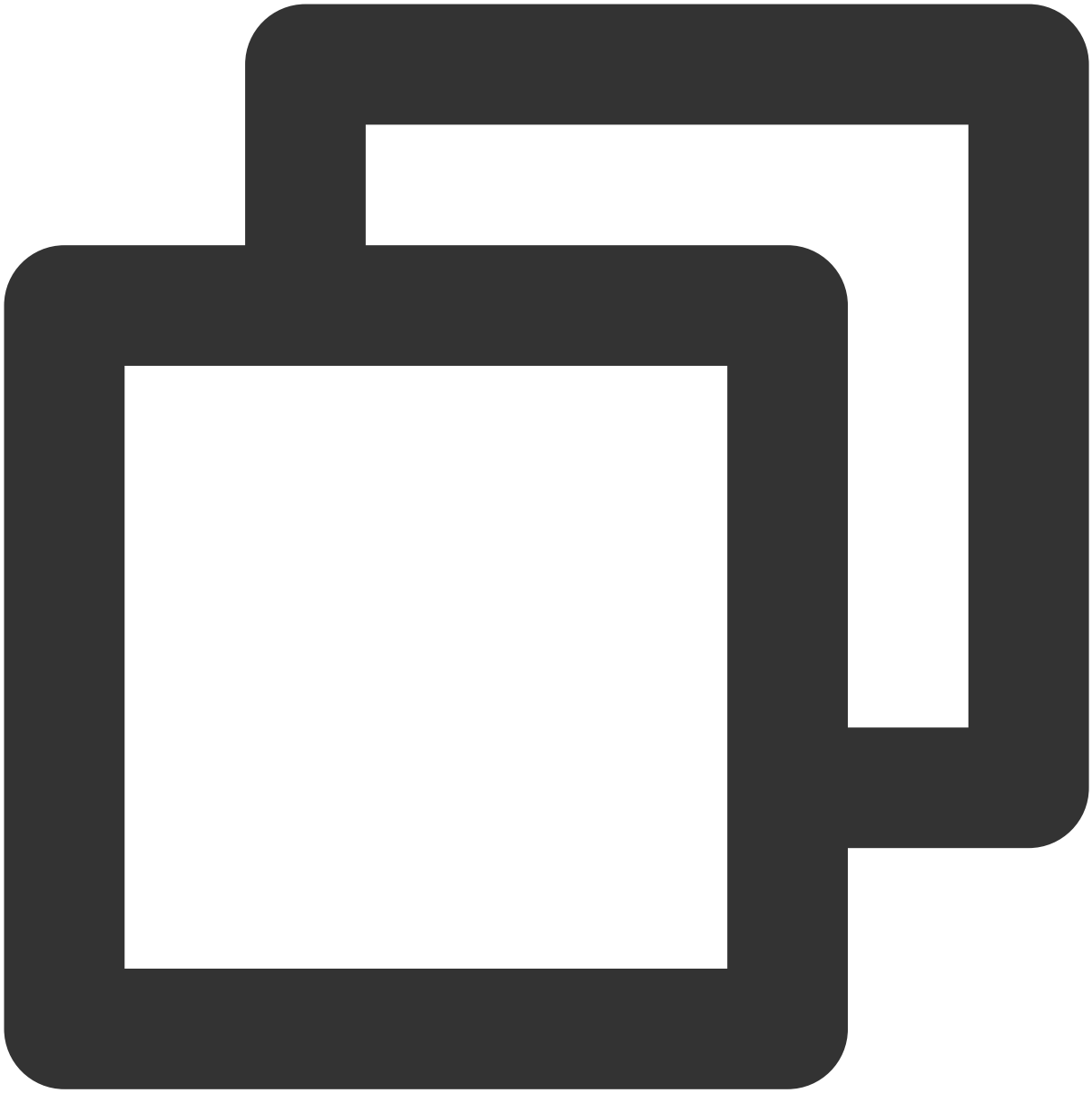
高级特性使用

最近更新时间：2024-07-31 09:16:48

本文档将对一些高级特性进行介绍。

表结构变更

使用 ALTER TABLE 命令可以修改表的 Schema，包括如下修改：
增加列。
删除列。
修改列类型。
改变列顺序。
以下举例说明。
原表 table1 的 Schema 如下:



```
+----------+-------------+------+-------+---------+-------+
| Field    | Type        | Null | Key   | Default | Extra |
+----------+-------------+------+-------+---------+-------+
| siteid   | int(11)     | No   | true  | 10      |       |
| citycode | smallint(6) | No   | true  | N/A     |       |
| username | varchar(32) | No   | true  |         |       |
| pv       | bigint(20)  | No   | false | 0       | SUM   |
+----------+-------------+------+-------+---------+-------+
```

我们新增一列 uv，类型为 BIGINT，聚合类型为 SUM，默认值为 0:

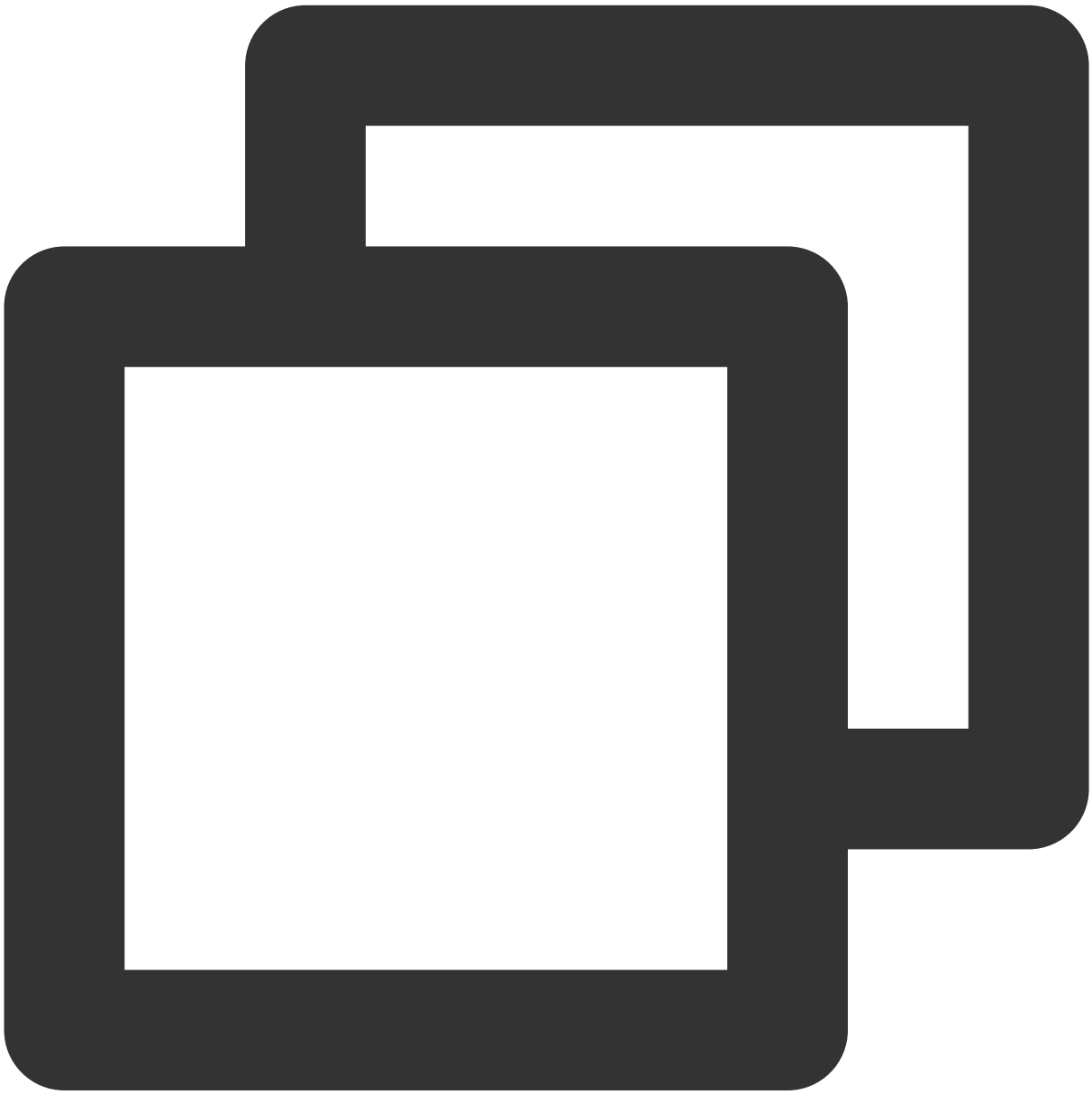
```
ALTER TABLE table1 ADD COLUMN uv BIGINT SUM DEFAULT '0' after pv;
```

提交成功后，可以通过以下命令查看作业进度:

```
SHOW ALTER TABLE COLUMN;
```

当作业状态为 FINISHED，则表示作业完成。新的 Schema 已生效。

ALTER TABLE 完成之后, 可以通过 `DESC TABLE` 查看最新的 Schema。



```
mysql> DESC table1;
+----------+-------------+------+-------+---------+-------+
| Field    | Type        | Null | Key   | Default | Extra |
```

```
+----------+-------------+------+-------+---------+-------+
| siteid   | int(11)     | No   | true  | 10      |       |
| citycode | smallint(6) | No   | true  | N/A     |       |
| username | varchar(32) | No   | true  |         |       |
| pv       | bigint(20)  | No   | false | 0       | SUM   |
| uv       | bigint(20)  | No   | false | 0       | SUM   |
+----------+-------------+------+-------+---------+-------+
5 rows in set (0.00 sec)
```

可以使用以下命令取消当前正在执行的作业: `CANCEL ALTER TABLE COLUMN FROM table1`

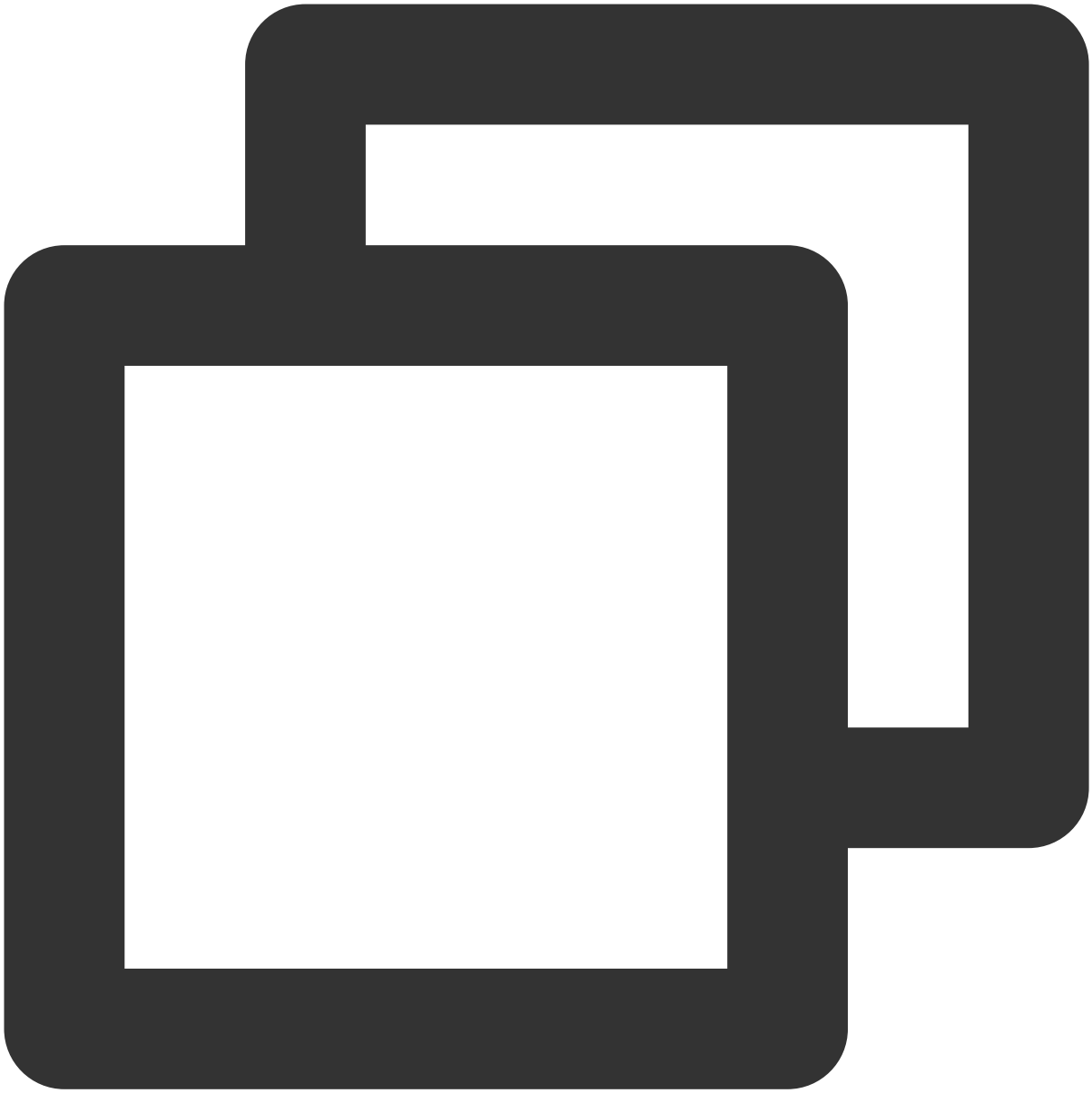
更多帮助，可以参阅 `HELP ALTER TABLE` 。

Rollup

Rollup 可以理解为 Table 的一个物化索引结构。**物化** 是因为其数据在物理上独立存储，而 **索引** 的意思是，Rollup可以调整列顺序以增加前缀索引的命中率，也可以减少key列以增加数据的聚合度。

以下举例说明。

原表 table1的 Schema如下:



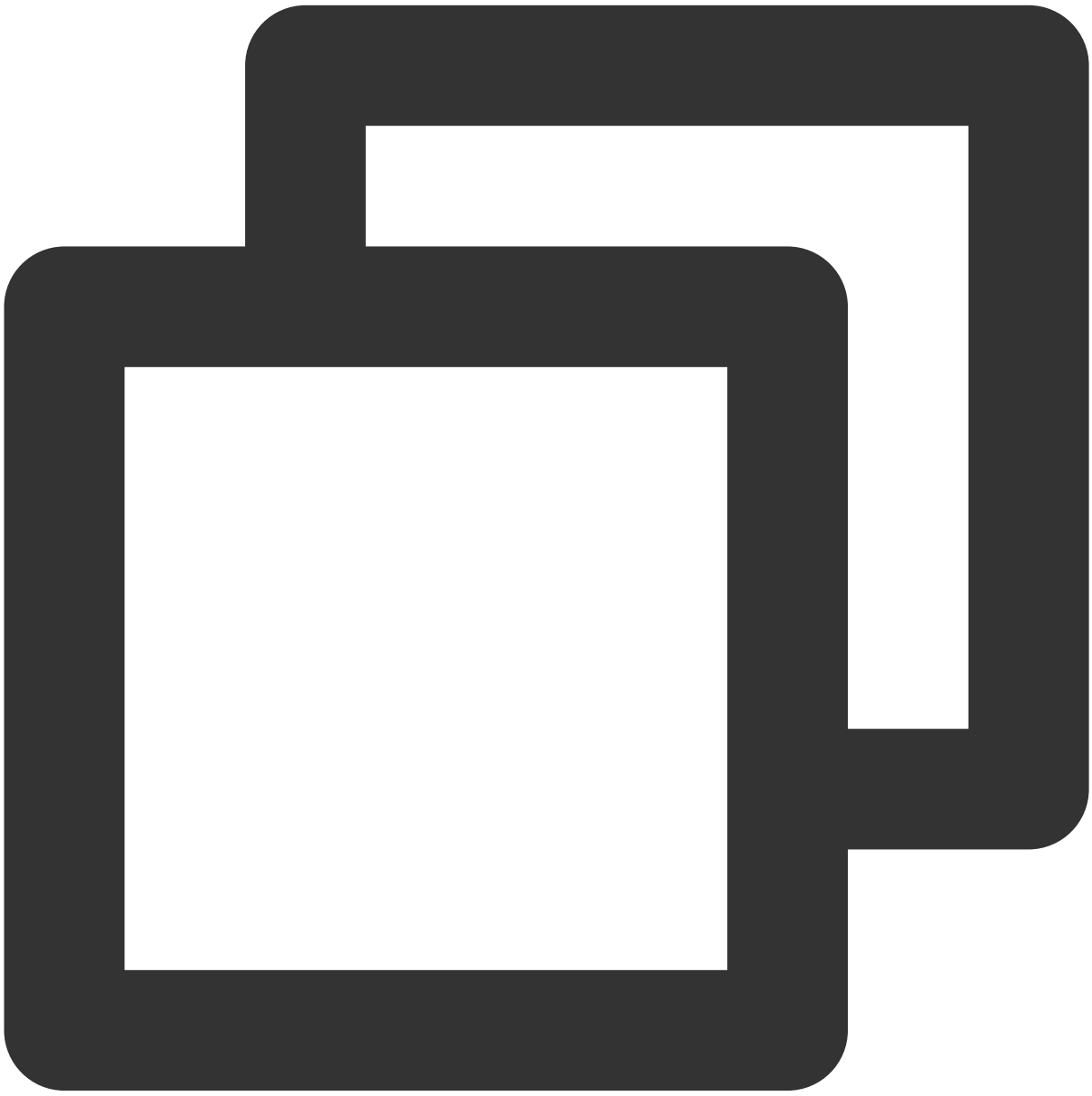
```
+----------+-------------+------+-------+---------+-------+
| Field    | Type        | Null | Key   | Default | Extra |
+----------+-------------+------+-------+---------+-------+
| siteid   | int(11)     | No   | true  | 10      |       |
| citycode | smallint(6) | No   | true  | N/A     |       |
| username | varchar(32) | No   | true  |         |       |
| pv       | bigint(20)  | No   | false | 0       | SUM   |
| uv       | bigint(20)  | No   | false | 0       | SUM   |
+----------+-------------+------+-------+---------+-------+
```

对于 table1 明细数据是 siteid, citycode, username 三者构成一组 key，从而对 pv 字段进行聚合；如果业务方经常有看城市 pv 总量的需求，可以建立一个只有 citycode, pv 的rollup。

```
ALTER TABLE table1 ADD ROLLUP rollup_city(citycode, pv);
```

提交成功后，可以通过以下命令查看作业进度：`SHOW ALTER TABLE ROLLUP;`，当作业状态为 FINISHED，则表示作业完成。

Rollup 建立完成之后可以使用 `DESC table1 ALL` 查看表的 Rollup 信息。



```
mysql> desc table1 all;
+-------------+----------+-------------+------+-------+--------+-------+
| IndexName   | Field    | Type        | Null | Key   | Default | Extra |
```

```
+-------------+----------+-------------+------+-------+---------+-------+
| table1      | siteid   | int(11)     | No   | true  | 10      |       |
|             | citycode | smallint(6) | No   | true  | N/A     |       |
|             | username | varchar(32) | No   | true  |         |       |
|             | pv       | bigint(20)  | No   | false | 0       | SUM   |
|             | uv       | bigint(20)  | No   | false | 0       | SUM   |
|             |          |             |      |       |         |       |
| rollup_city | citycode | smallint(6) | No   | true  | N/A     |       |
|             | pv       | bigint(20)  | No   | false | 0       | SUM   |
+-------------+----------+-------------+------+-------+---------+-------+
8 rows in set (0.01 sec)
```

可以使用以下命令取消当前正在执行的作业： `CANCEL ALTER TABLE ROLLUP FROM table1;` 。

Rollup 建立之后，查询不需要指定 Rollup 进行查询。还是指定原有表进行查询即可。程序会自动判断是否应该使用 Rollup。是否命中 Rollup可以通过 `EXPLAIN your_sql;` 命令进行查看。

更多帮助，可以参阅 `HELP ALTER TABLE` 。

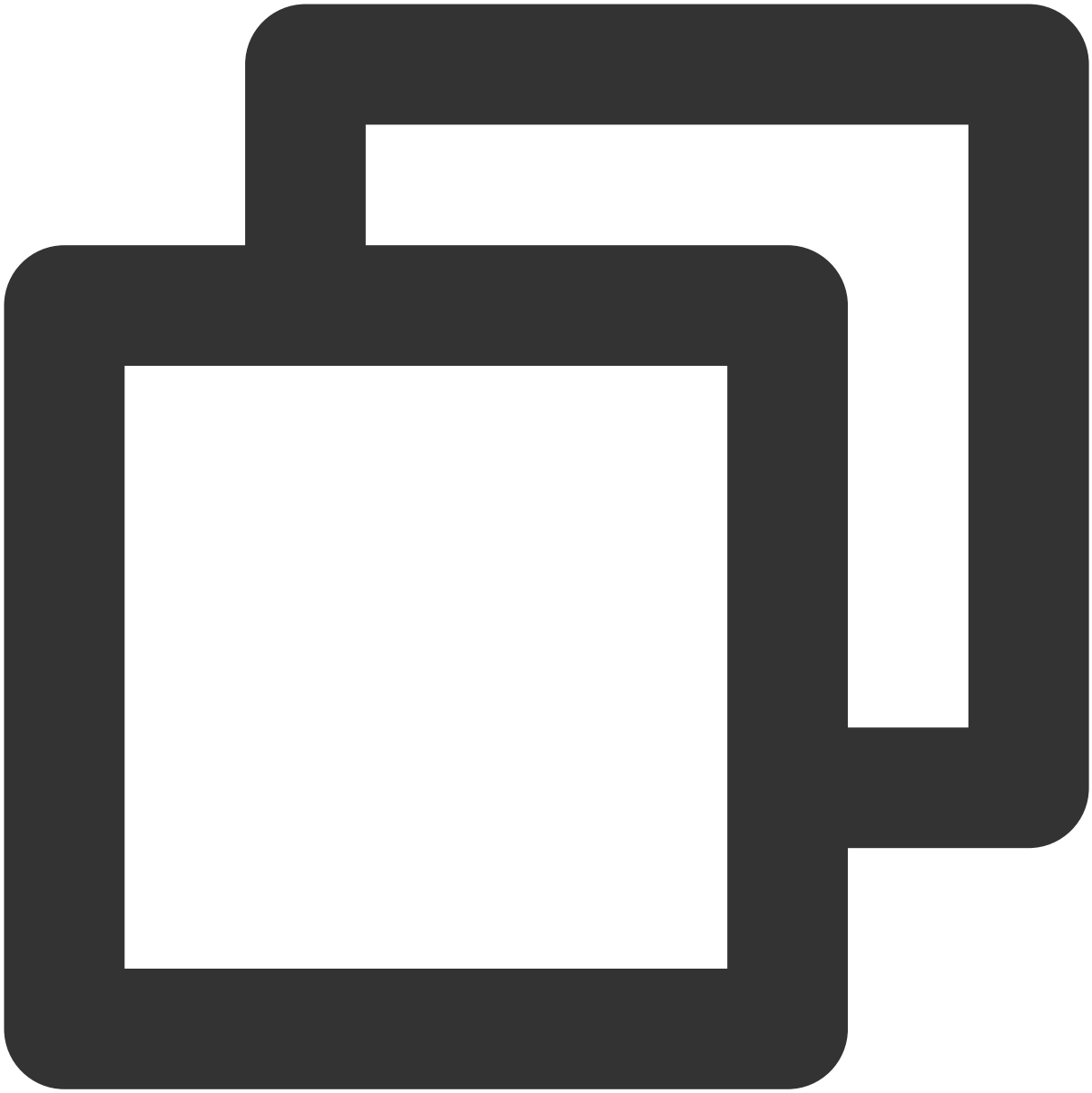
数据表的查询

内存限制

为了防止用户的一个查询可能因为消耗内存过大。查询进行了内存控制，一个查询任务，在单个 BE 节点上默认使用不超过2GB 内存。

用户在使用时，如果发现报 `Memory limit exceeded` 错误，一般是超过内存限制了。遇到内存超限时，用户应该尽量通过优化自己的 sql 语句来解决。如果确切发现2GB 内存不能满足，可以手动设置内存参数。

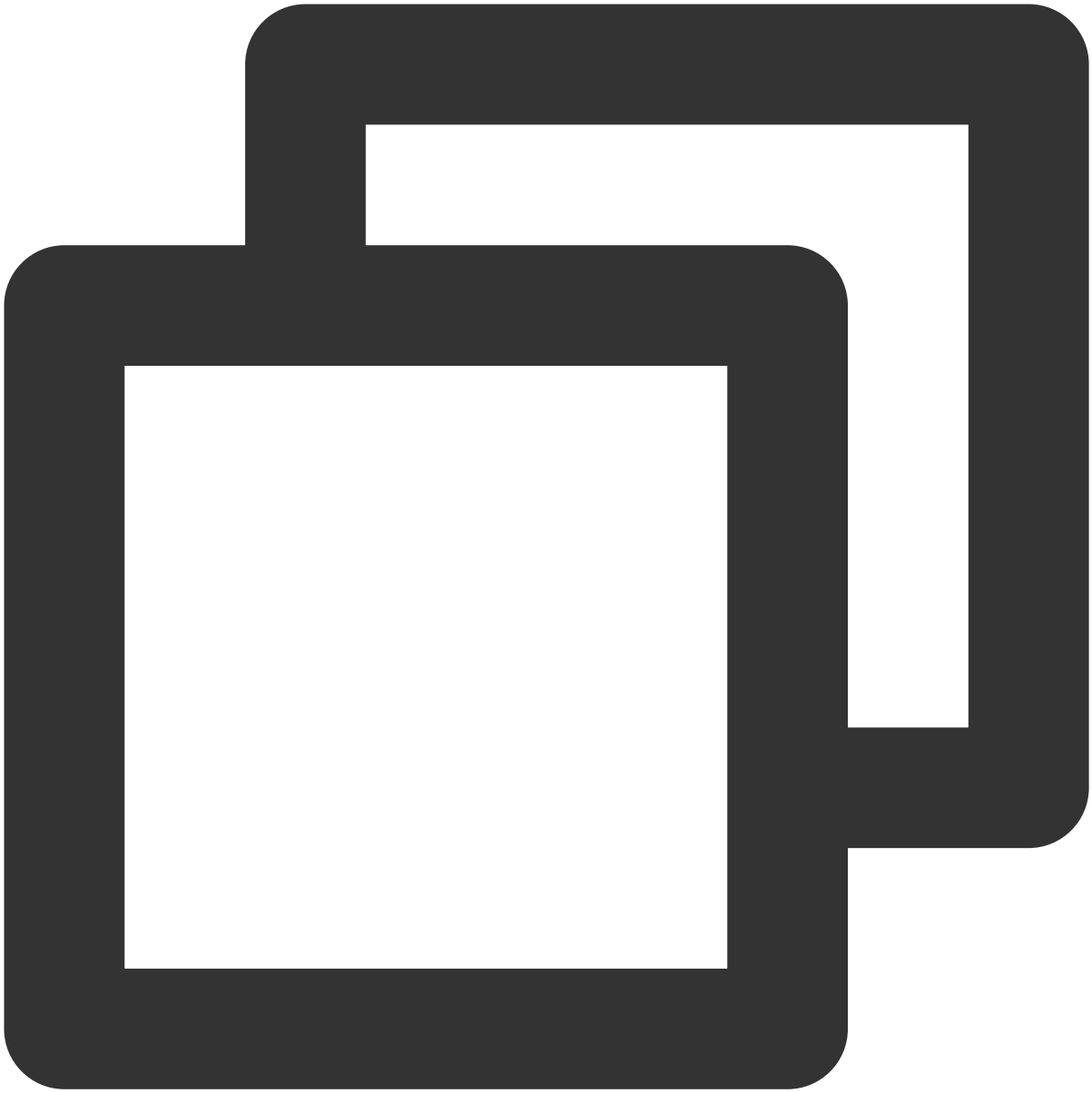
显示查询内存限制:



```
mysql> SHOW VARIABLES LIKE "%mem_limit%";
+---------------+------------+
| Variable_name | Value      |
+---------------+------------+
| exec_mem_limit| 2147483648 |
+---------------+------------+
1 row in set (0.00 sec)
```

`exec_mem_limit` 的单位是 byte，可以通过 `SET` 命令改变 `exec_mem_limit` 的值。如改为 8GB。

```
SET exec_mem_limit = 8589934592;
```



```
mysql> SHOW VARIABLES LIKE "%mem_limit%";
+---------------+------------+
| Variable_name | Value      |
+---------------+------------+
| exec_mem_limit| 8589934592 |
+---------------+------------+
1 row in set (0.00 sec)
```

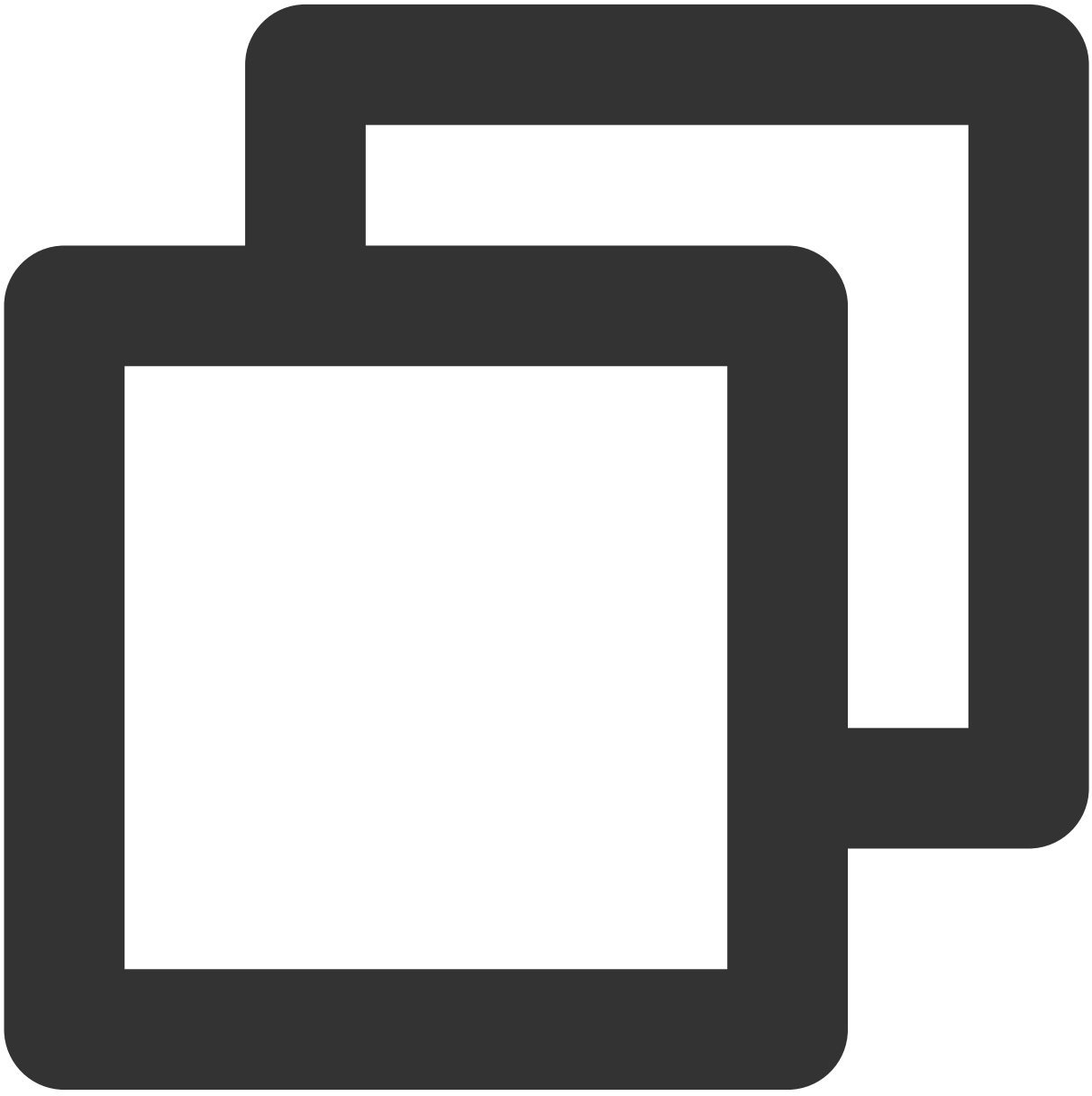
说明

以上该修改为 session 级别，仅在当前连接 session 内有效。断开重连则会变回默认值。

如果需要修改全局变量，可以这样设置：`SET GLOBAL exec_mem_limit = 8589934592;` 。设置完成后，断开 session 重新登录，参数将永久生效。

查询超时

当前默认查询时间设置为最长为 300 秒，如果一个查询在 300 秒内没有完成，则查询会被 Doris 系统 cancel 掉。用户可以通过这个参数来定制自己应用的超时时间，实现类似 wait(timeout) 的阻塞方式。
查看当前超时设置:



```
mysql> SHOW VARIABLES LIKE "%query_timeout%";
+---------------+-------+
```

```
| Variable_name | Value |
+---------------+-------+
| QUERY_TIMEOUT | 300   |
+---------------+-------+
1 row in set (0.00 sec)
```

修改超时时间到1分钟:

```
SET query_timeout = 60;
```

说明

当前超时的检查间隔为 5 秒，所以小于 5 秒的超时不会太准确。

以上修改同样为 session 级别。可以通过 `SET GLOBAL` 修改全局有效。

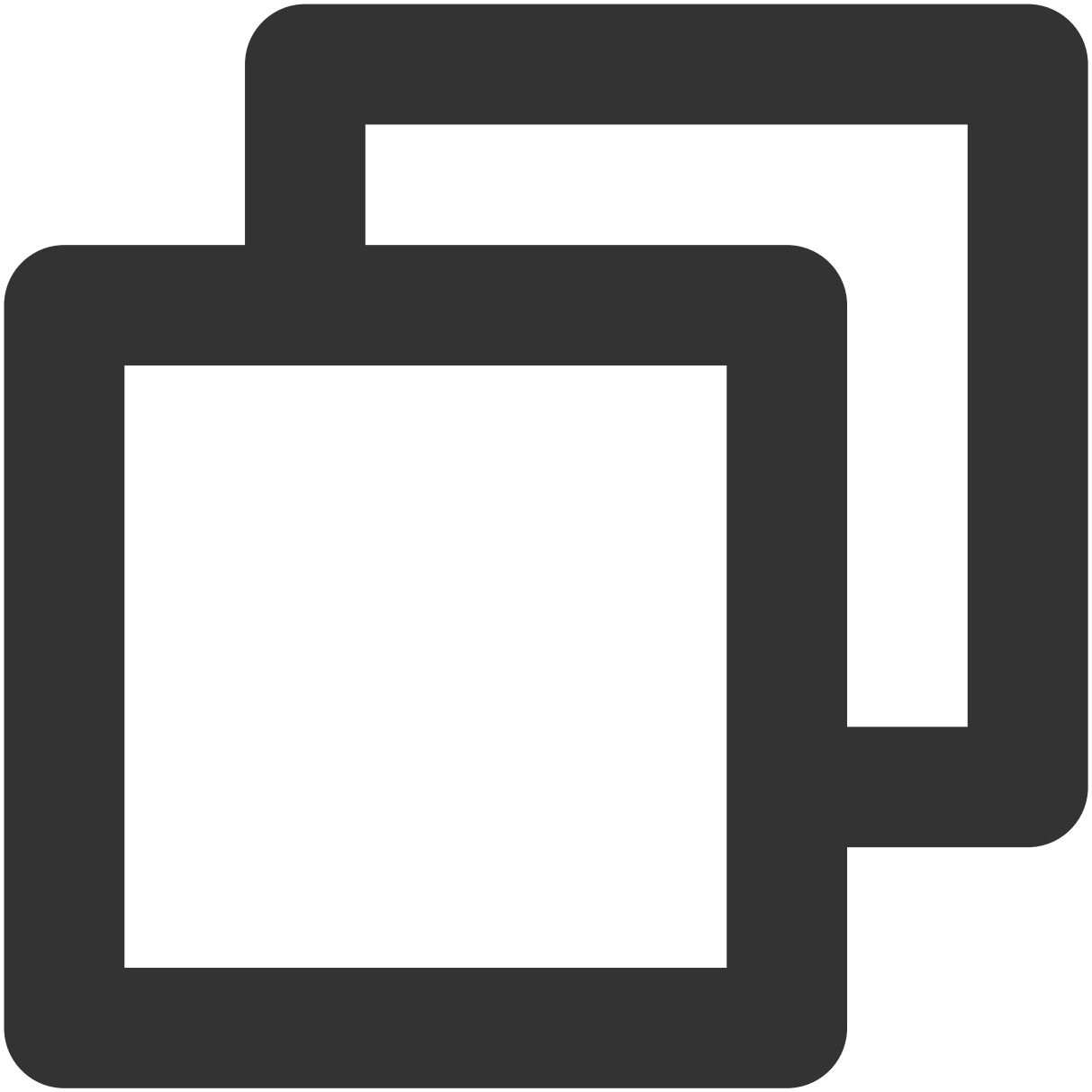
Broadcast/Shuffle Join

系统默认实现 Join 的方式，是将小表进行条件过滤后，将其广播到大表所在的各个节点上，形成一个内存 Hash 表，然后流式读出大表的数据进行Hash Join。但是如果当小表过滤后的数据量无法放入内存的话，此时 Join 将无法完成，通常的报错应该是首先造成内存超限。

如果遇到上述情况，建议显式指定 Shuffle Join，也被称作 Partitioned Join。即将小表和大表都按照 Join 的 key 进行 Hash，然后进行分布式的 Join。这个对内存的消耗就会分摊到集群的所有计算节点上。

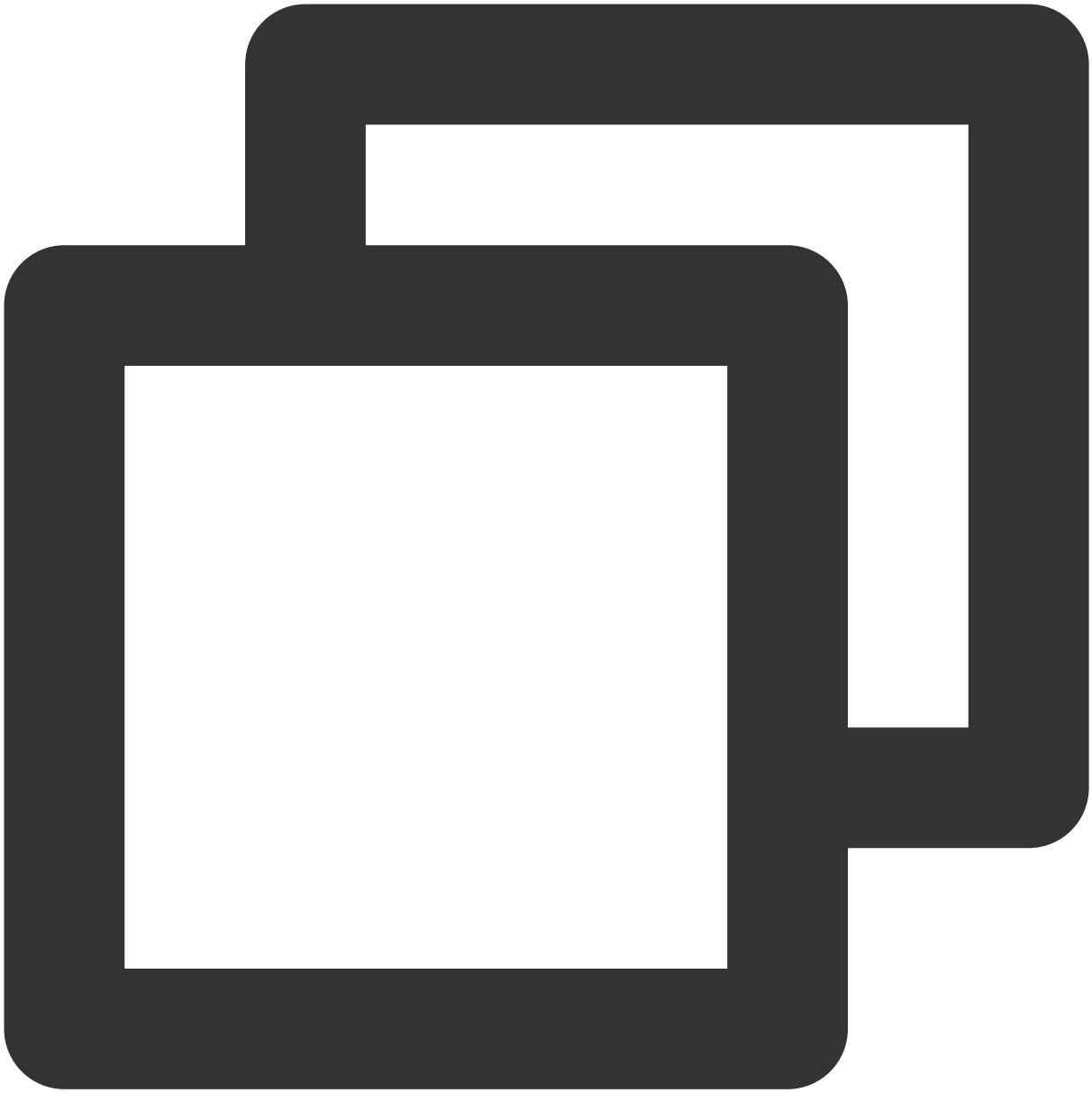
Doris会自动尝试进行 Broadcast Join，如果预估小表过大则会自动切换至 Shuffle Join。注意，如果此时显式指定了 Broadcast Join 也会自动切换至 Shuffle Join。

使用 Broadcast Join（默认）：



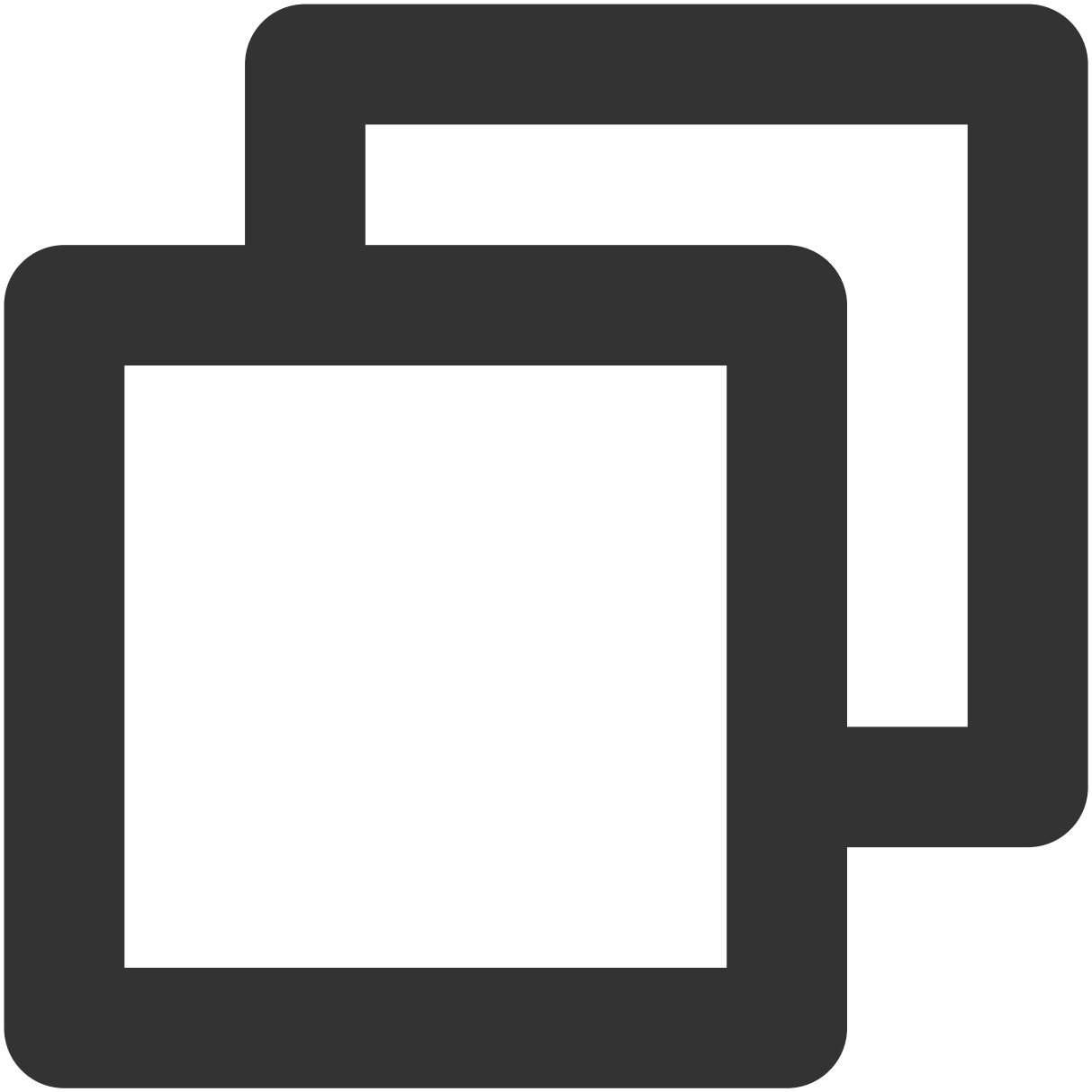
```
mysql> select sum(table1.pv) from table1 join table2 where table1.siteid = 2;
+--------------------+
| sum(`table1`.`pv`) |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.20 sec)
```

使用 Broadcast Join（显式指定）：



```
mysql> select sum(table1.pv) from table1 join [broadcast] table2 where table1.sitei
+--------------------+
| sum(`table1`.`pv`) |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.20 sec)
```

使用 Shuffle Join:



```
mysql> select sum(table1.pv) from table1 join [shuffle] table2 where table1.siteid
+--------------------+
| sum(`table1`.`pv`) |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.15 sec)
```

查询重试和高可用

当部署多个 FE 节点时，用户可以在多个 FE 之上部署负载均衡层来实现 Doris 的高可用。

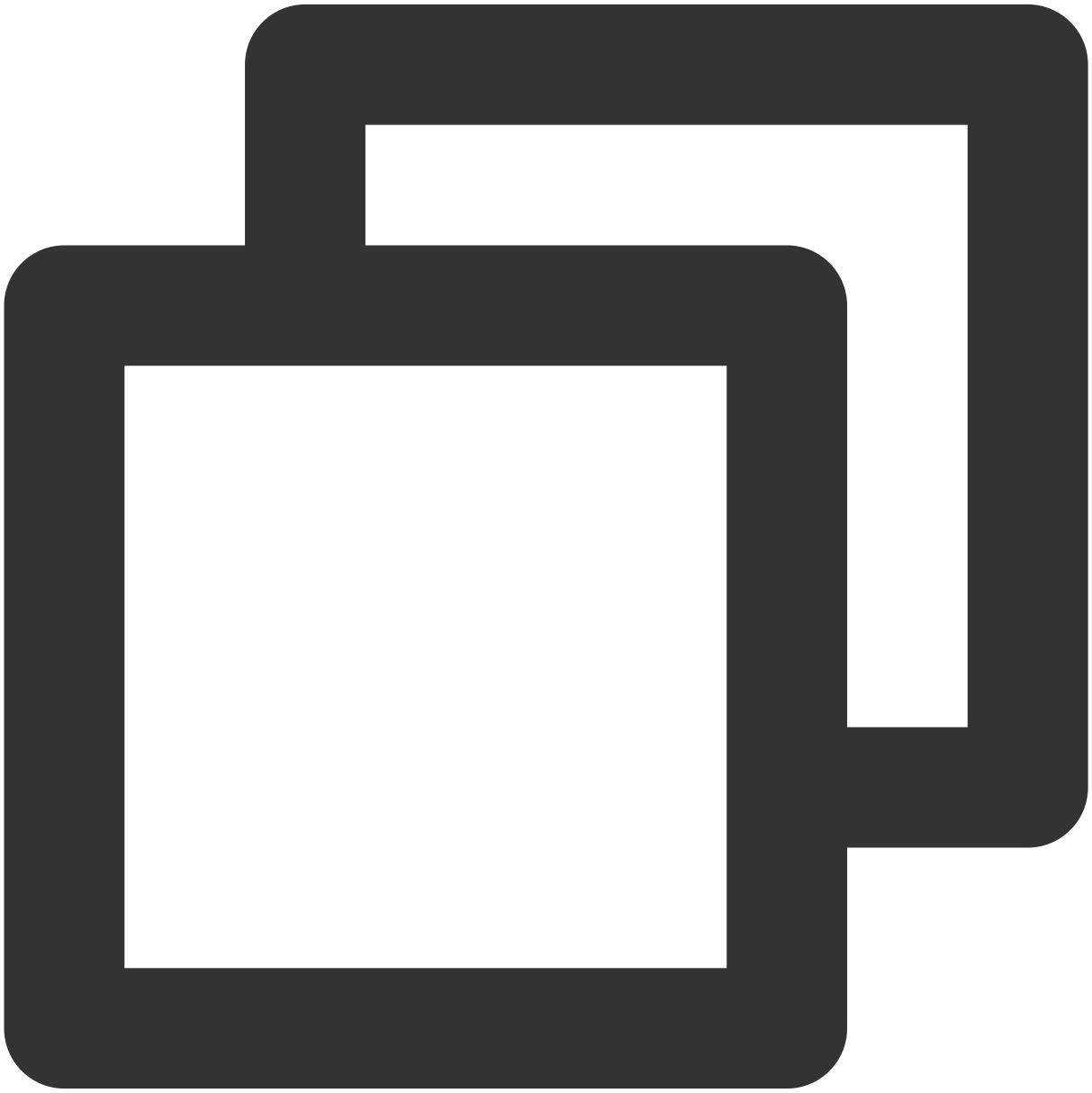
以下提供一些高可用的方案：

第一种

自己在应用层代码进行重试和负载均衡。例如发现一个连接挂掉，就自动在其他连接上进行重试。应用层代码重试需要应用自己配置多个doris前端节点地址。

第二种

如果使用 mysql jdbc connector 来连接Doris，可以使用 jdbc 的自动重试机制：



```
jdbc:mysql://[host:port],[host:port].../[database][?propertyName1][=propertyValue1]
```

第三种

应用可以连接到和应用部署到同一机器上的 MySQL Proxy，通过配置 MySQL Proxy 的 Failover 和 Load Balance 功能来达到目的。

`http://dev.mysql.com/doc/refman/5.6/en/mysql-proxy-using.html`

资源规格选型及调优建议

最近更新时间：2024-08-02 12:44:32

本文将为您介绍如何选择腾讯云数据仓库 TCHouse-D 的实例规格，并会给出资源不足时的调优建议。

注意：

多种不同类型的业务建议配置资源隔离策略或拆分集群，如：实时报表业务一个集群、实时风控业务一个集群。

一个业务同时支持多个 ToB 租户时，建议视情况进行资源隔离或集群拆分，减轻互相影响，如：同时为200个租户提供 SaaS 服务，拆分成4个集群，每个集群支持50个租户。

资源规格及适配场景

购买 Doris 集群时，需要选择 FE 节点、BE 节点的计算资源规格和存储资源规格，并选择是否开启高可用。

资源规格及建议场景

机型类型	计算节点规格	建议存储类型	建议场景
标准型	4核16G	高性能云硬盘 SSD 云硬盘 增强型 SSD 云硬盘	仅限于 **POC 功能测试或个人学习使用**，主要用于体验测试产品能力。
	8核32G	高性能云硬盘 SSD 云硬盘 增强型 SSD 云硬盘	推荐用于**测试环境**，可支持中等数据规模、较复杂的数据分析
	16核64G	高性能云硬盘 SSD云硬盘 增强型 SSD 云硬盘	推荐用于**生产环境**，可支持较大规模、较复杂场景的数据分析，及高并发场景
	32核及以上	高性能云硬盘 SSD 云硬盘 增强型 SSD 云硬盘	生产环境推荐配置，可支持大量高复杂度数据分析，高并发等场景

高可用及节点数量建议

场景	高可用选择	建议最小 FE 节点数	建议最小 BE 节点数

POC 功能测试	非高可用	1个	3个
生产场景（查询高可用）	读高可用	最少3个FE节点	最少3个 BE 节点，按需扩缩容
生产场景（查询/写入高可以用）	读写高可用	最少5个FE节点	最少3个 BE 节点，按需扩缩容
跨AZ高可用场景	读写高可用+3AZ部署	最少5个FE节点	最少3个 BE 节点，以3为阶梯扩缩容

资源规格选择举例

注意：

以下内容仅供参考，不同业务场景下性能可能会有较大的差异。

1. **场景一：产品功能验证，进行简单数据分析**

FE：不开启高可用，单节点4核16G

BE：3节点，每个节点4核16G

2. **场景二：中小规模数据简单查询，如百GB数据量级，1000QPS以下**

FE：不开启高可用，单节点8核32G

BE：3节点，每个节点8核32G

3. **场景三：生产场景，TB级数据量，涉及多表关联、GROUP BY 等复杂查询**

FE：开启高可用，3节点，每个节点16核64G

BE：3节点，每个节点16核64G

4. **场景四：生产业务，TB级数据量，涉及复杂查询，涉及大量高并发点查**

FE：开启高可用，3节点，每个节点16核64G

BE：6节点，每个节点16核64G

资源监控及调优建议

大批量数据导入、数据查询、并发查询、多表关联 join 等操作都会导致 CPU、内存的大量占用，若CPU/内存使用率持续超过85%会导致集群不稳定，建议优化业务或变配。

资源使用监控

可在**集群管理 > 集群监控**中查看 BE、FE 各节点的 CPU、内存使用情况，如下图所示。

集群监控 > BE 指标

集群监控 > FE 指标

资源扩容建议

FE 和 BE 的 CPU、内存使用率持续超过85%时，就需要考虑进行资源升配或扩容。

说明：

导致 FE 和 BE 的 CPU、内存大量占用的主要原因如下：

FE CPU 大量占用：多并发查询、大量复杂查询。

FE 内存大量占用：元数据过多（分区不合理等）、频繁进行表删除等。

BE CPU 大量占用：大量数据导入、大量复杂查询（如聚合查询）等。

BE 内存大量占用：大量数据导入、大量复杂查询（如聚合查询）等。

常见场景	资源耗用表现	使用率持续超过85%时调优建议
过多数据持续导入	FE 和 BE 的 CPU、内存都会被大量占用	如果是 FE 瓶颈：建议纵向升配 如果是 BE 瓶颈：建议纵向升配
点查较多/高并发	FE 和 BE 的 CPU 都会被大量占用	如果是 FE 瓶颈：建议纵向升配 如果是 BE 瓶颈：建议纵向升配
元数据频繁变更删除	FE 内存大量占用	建议 FE 纵向升配，增加内存
多表关联/聚合查询较多	BE 的 CPU、内存会大量占用	优先建议 BE 横向扩容，也可纵向升配
数据多并发度写入	BE 的 CPU、内存会大量占用	优先建议 BE 横向扩容，也可纵向升配

集群扩缩容注意事项

操作类型	注意事项
水平扩容	水平扩容过程中，系统读写仍可进行，但是可能出现一些抖动，执行操作大约需要5 - 15分钟，请选择在非业务高峰期进行。 在数据存储量及查询量均相对增长时，优先选择水平扩容。
水平缩容	只能每次选择一类节点进行缩容操作，如仅缩容 FE 或 仅缩容 BE。 FE 缩容：可一次性缩容多个。 BE 缩容：一次性缩容多个 BE 节点有可能导致数据丢失或时间过长，建议逐个缩容。 缩容过程中，系统读写仍可进行，但是可能出现一些抖动。
垂直升配/降配	垂直变配系统不可读、不可写。 计算规格支持升配、降配；存储规格仅支持升配。 变配操作结果对集群所有节点均生效。

业务调优建议

调优类型	调优说明

使用建议	如果经常对某列进行点查，且列的基数较高，建议在此列创建 bloom filter 索引。 如果经常对某表进行模式固定的聚合查询，建议在此表创建物化视图。 建议结合业务场景合理分区分桶，避免分区分桶过多占用FE内存。 普通数据探查的 sql，如果不需要全部数据，建议加上limit返回条数限制，也可加速查询。 导入数据建议用 CSV，避免 Json 数据格式。
尽量避免	避免 select * 查询； 避免全局开profile（会带来较多资源开销，建议针对需要的 SQL 开 profile） 建表时：避免开启 merge_on_write（此功能暂不成熟） 建表时：避免开启 auto bucket（此功能暂不成熟） 建表时：避免开启动态 Schema 表（此功能暂不成熟） 避免多个大表 Join，涉及多个大表关联时： 可转为大表两两 join，并使用 Colocation Join。 或使用预聚合表、索引等进行查询加速。
参数调优	一条 SQL 涉多并发时，建议调大 `parallel_fragment_exec_instance_num` 参数，此参数默认值200，可按倍数调大（如400、800），建议控制在2000以内。 建议控制 compaction 速度，若监控指标 `base_compaction_score` 超过200且持续上升的话（具体可在“集群监控-BE指标-BE”页查看），可以将 compaction_task_num_per_disk 参数配置调大（系统默认2，可调大至4或更多）。

命名规范及库表限制

最近更新时间：2024-07-31 09:17:18

本文将为您介绍腾讯云数据仓库 TCHouse-D 的命名规范，及一些库表创建/变更的限制。

命名规范

数据库

名称：表名以字母或下划线开头，可包含字母、数字以及下划线，长度为1到64个字符。
描述：最多不超过2048个字符。
限制：同一个数据链接下，不允许有相同的数据库名称。

数据表/视图

名称：表名以字母或下划线开头，可包含字母、数字以及下划线，长度为1到64个字符。
描述：最多不超过2048个字符。
限制：同一个数据库下，不允许有相同的数据表名称。

属性列

名称：列名以字母或下划线开头，可包含字母、数字以及下划线，长度为1到256个字符。
描述：最多不超过256个字符。
限制：同一个数据表下，不允许有相同的数据列名称。

分区

分区字段名称：长度为1到256个字符。
使用建议：

1. 单表数据量在2亿条以下时，为了方便，可选择不设置分区，而是直接使用分桶；
2. 分区字段具有连续的数据范围（如日期、ID）时，建议选择 Range 分区；
3. 数据具有离散的取值（如国家、地区、状态等）时，建议选择 LIST 分区。

分桶

分桶字段名称：长度为1到256个字符
使用建议：

1. 分桶 key 可以一个或者多个，多个保证数据分布更均衡，单个容易匹配命中。
2. 分桶列的选择：一般选择区分度/基数较高、hash均匀的字段，避免数据倾斜；且应该是经常使用的字段，提高查询效率；
3. 创建的分桶数不宜过多或者过少，建议每个分桶最好保持在 1-10G 之间。

使用限制

库/表/列/分区的数量限制

限制项	数量限制
可创建 catalog 数量	20
每个集群的数据库数量	10,000
每个用户的数据库数量	1,000
每个集群的数据表数量	100,000
每个用户的数据表数量	10,000
每个数据库的数据表数量	4,096
每个表的字段数量	2,048
每个表的分区数	10,000
每个集群的视图数	10,000
视图最大嵌套层级	10
每个集群的物化视图数	10,000

查询限制

限制项	限制说明	补充
每天的查询数据量	无限制	-
单个 SQL 查询的表数量	32	-
单个 SQL 查询超时时间	900秒	-
每个外部表的最大文件数	不限制	-

DML/DML 限制

限制项	限制说明	补充
每个集群同时在进行的 Schema 变更	20	-

每个表同时在进行的 Schema 变更	1	-
每个表的数据写入并发(每秒)	10	不建议高频写入，建议适量攒批：每个表一分钟总导入次数不得超过20次
每个表的同一个 key 的数据写入并发(每秒)	1	同一张表不能同时更新同一个 key，否则可能导致数据不准确

表设计与数据导入

最近更新时间：2024-07-31 09:17:35

如何选择数据模型？

对于本身已完成聚合的数据，选择 Aggregate/Duplicate 模型均可。
对于清洗已完成的明细数据，若需要聚合后查询统计值，选择 Aggregate 模型。
对于清洗已完成的明细数据，若需要做明细级的查询，选择 Duplicate 模型（没有预聚合，聚合查询效率较低，可以通过物化视图，提升聚合查询效率）。
如果有**唯一的 Key**，需要按 Key 去重，选择 **Unique 模型**。

如何设置分区？

分区字段的值分布尽量分散，建议选择日期或 ID 等作为分区字段。

如何设置副本？

副本数量可以设为3（也可以设为2，但不建议设为1，原因是后续滚动升级会出现数据不可用状态）。

如何设置分桶？

采用一些 Hash 均匀的 Key 作为分桶 Key，避免数据倾斜。
创建的分桶数不宜过多或者过少，建议每个分桶最好保持在1 - 10G之间，对于小表，一般几个分桶已经足够。
分桶 Key 可以一个或者多个，多个保证数据分布更均衡，单个容易匹配命中（单个分桶一般选择区分度较高的 Key）。

如何选择字段格式？

表的每一个字段优先考虑使用**整型**的类型，而不是 string 等类型，能够极大地促进查询和版本合并效率。
对于浮点数使用 decimal，而不是 double，float。

数据导入注意事项

实时导入建议采用 Stream load，离线导入建议采用 Broker load，导入的基本原则就是批量导入，**减少并发，尽可能一次性导入尽量多的数据**，减少合并的成本，也尽量避免影响读的效率。（例如一分钟总导入次数不得超过20次，考虑各种并发在内，高频导入目前不适合）。**小文件太多可严重影响查询效率**。
对于作为 Hash key 的字段在数据导入的时候一定要注意 **NULL 值的过滤处理**，避免出现数据倾斜。出现数据倾斜将导致扩容机器**无法缓解集群压力**，另外容易导致**集群不稳定**。
数据导入是要**指定要导入的分区**，否则大表导入数据容易导致失败。
导入数据建议用 CSV，避免 JSON 数据格式。

Schema Change 注意事项

为了集群稳定性考量和业务的实际需求，我们建议创建表之前做好字段类型的评估，业务修改表的 Schema，只建议 **Add Column** 操作，我们保障在尽可能短的时间内完成新列的增加。

数据清理注意事项

如果要清空分区的数据，建议优先考虑 **truncate** 操作（等同于 drop 分区，然后 create 分区的操作）而不是 delete 操作，delete 操作对查询性能有较大影响。

其他

对于不熟悉的操作，优先考虑在命令行键入 help keyword 寻求帮助（例如 help stream load 可以了解如何进行数据实时导入），或者 提交工单 以获得更深入的帮助来解决您的问题。

查询调优

最近更新时间：2024-07-31 09:17:49

基础查询调优

查询分区表时，一定要带上**分区字段**，Explain + SQL 可以协助用户分析查询了几个分区和 tablet 的数据。
查询 SQL 条件最好能命中分区 Key 和分桶 Key。
查询 SQL 条件最好能命中前缀索引。
由于 Doris 是列存数据库，当查询的字段足够多的时候，可能性能还不如行式存储，建议查询时尽可能选择具体的字段代替 *，在查询的最后方加上 **limit number** 的限制。
执行 Select 操作的时候尽可能**避免写成 function(column)= "xxxx"**的形式，这样将导致无法发挥 Doris 系统谓词下推的优势，左侧应为列名，右侧应该为可以计算展平的常数值。
查询尽可能避免使用 or，union all 的情况，在大多数场景下，考虑使用 **in 代替 or**。
普通数据探查的 SQL，如果不需要全部数据，建议加上 limit 返回条数限制，也可加速查询。

Join 优化

Shuffle 方式优化：效率为 **Colocate join** > **Bucket Shuffle** > **Shuffle** > **BroadCast**，具体参见 Bucket Shuffle Join。
RuntimeFilter：join 查询中，存在除了关联条件之外，右边有其他过滤条件。

使用 Rollup

查询无法覆盖基表前缀索引，通过 Rollup 调整 Key 顺序形成前缀索引。
对 Aggregate 表进行 Key 筛选聚合

使用物化视图

如果经常对某表进行模式固定的聚合查询，建议在此表创建物化视图；
Rollup 支持的场景都能用；
对 Duplicate 表形成额外聚合。
详情请参见 物化视图。

索引优化

Bitmap 索引：取值基数比较小的列[100-100000]，查询条件命中列。
BloomFliter 索引：如果经常对某列进行精确点查，且列的基数较高，建议在此列创建 Bloom filter 索引。

使用 Cache

PageCache：此配置默认开启。
SqlCache：此配置默认关闭。并发高，查询结果集较小时效果好。

建议规避的用法

最近更新时间：2024-07-31 09:18:04

建议避免的场景

避免在生产集群大规模周期性调度离线/批 ETL 作业（insert into select / create table as select），尤其在同一个集群中同时运行离线、在线业务，离线作业会占用较大资源从而影响在线业务的稳定性与性能。

说明：

建议离线/在线业务通过不同的集群隔离，或提前通过 Spark 完成离线处理后，再将数据写入 Doris。

避免逐条 insert into：Doris 每个 insert into 都是一个事务，逐条写入可能导致并发超过事务上限。

说明：

建议进行攒批，如每个 insert into 几十或上百条数据，以降低写入压力。

1.2内核版本：尽量避免使用复杂数据类型（例如 MAP、ARRAY、STRUCT 等）。

1.2内核版本：对复杂数据类型的支持不够完善，部分写入和查询可能会报错。

建议避免的查询

尽量避免在多列且数据规模较大的表上进行 select * 查询。

避免全局开 profile（这会带来较大的资源开销，因此建议仅对需要的 SQL 语句开启 profile）。

尽量避免多个大表 Join。

说明：

涉及多个大表关联时，建议可转为大表两两 join 并使用 Colocation Join，或使用预聚合表、索引等进行查询加速。

建议避免的功能

1.2内核版本：尽量避免开启 merge_on_write（此功能暂不成熟）。

1.2内核版本：尽量避免开启 Light scheme change（此功能暂不成熟）。