ClickHouse统一网关设计与实践之缓存加速服务

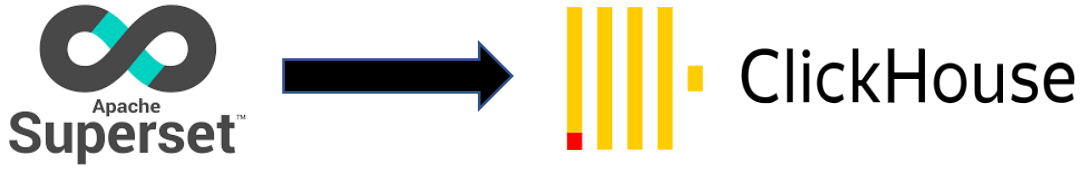
腾讯音乐 麦嘉铭

2021-06-26

摘要

1. 问题背景

1. 问题背景



问题的表象

- 经常看的报表加载速度慢，影响用户体验

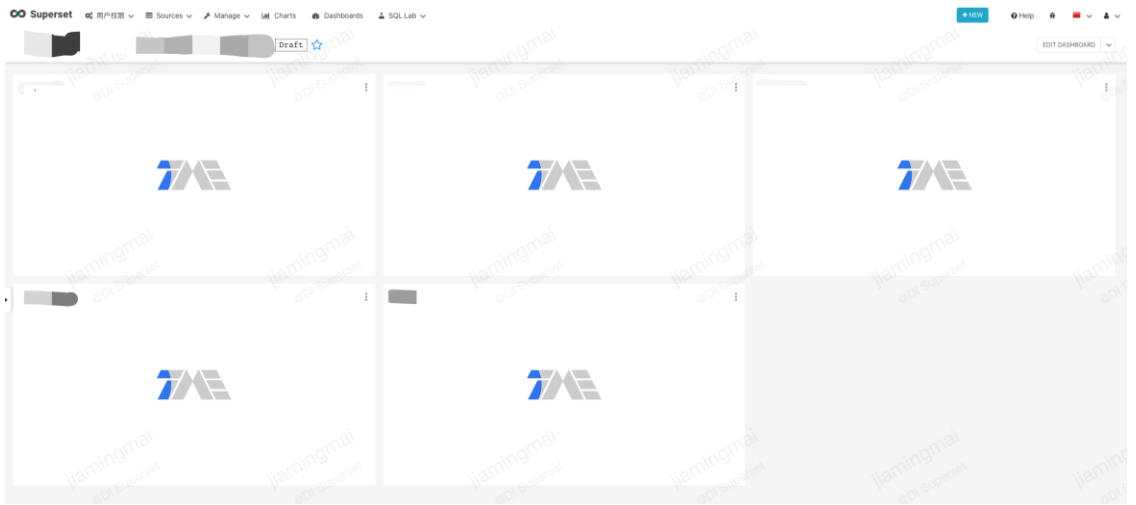


图1 打开superset页面时需要等待查询

从审计日志观察分析出来的特征

- 用户经常需要反复查看一些固定的报表
- 不同用户经常同时访问相同的报表页面
- 查询结果都相对较小

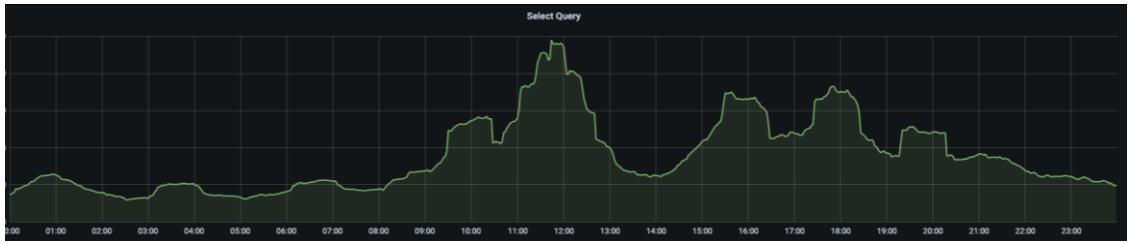
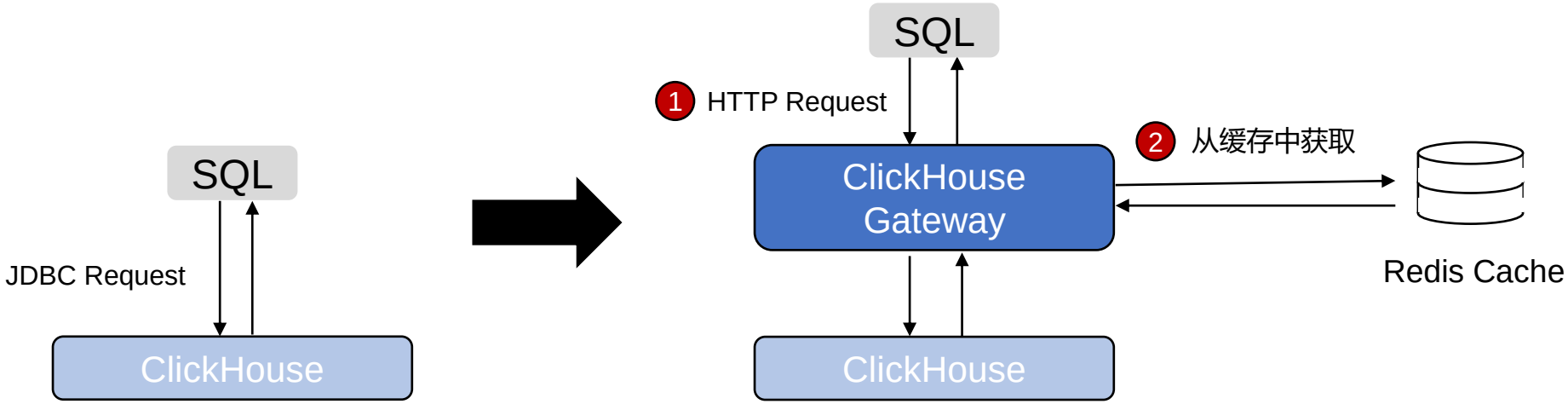


图2 一天内 query 数量分布

2. 技术解决方案：ClickHouse缓存加速服务

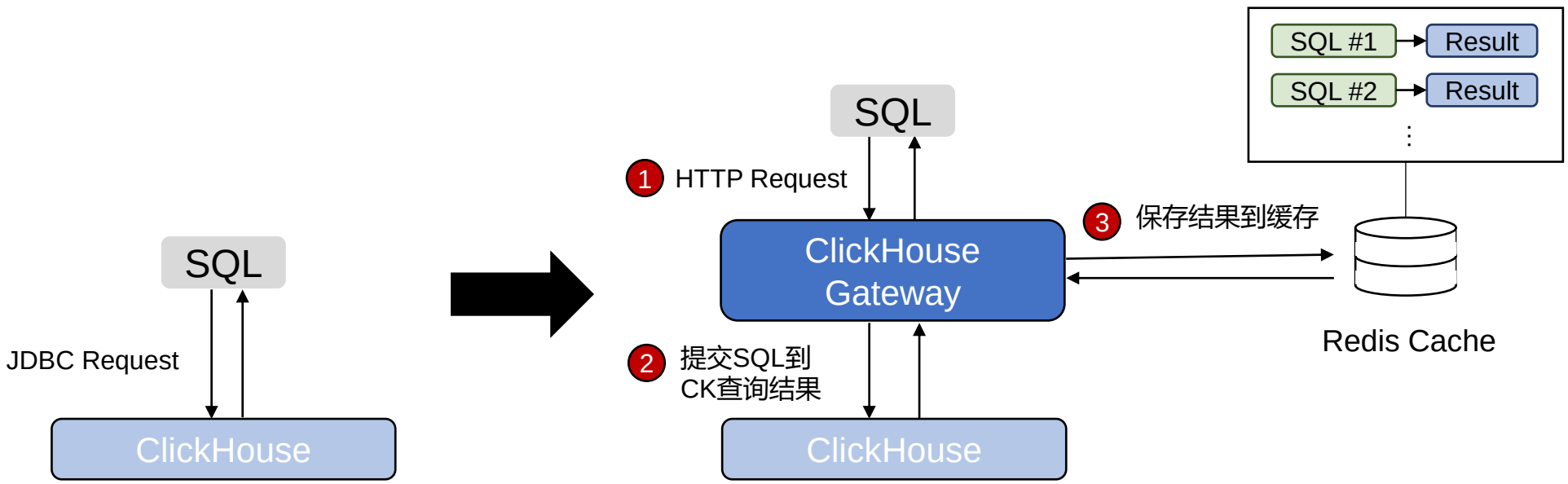
2.1 整体思路

通过缓存SQL查询结果进行加速



2.1 整体思路

从缓存中获取结果并返回



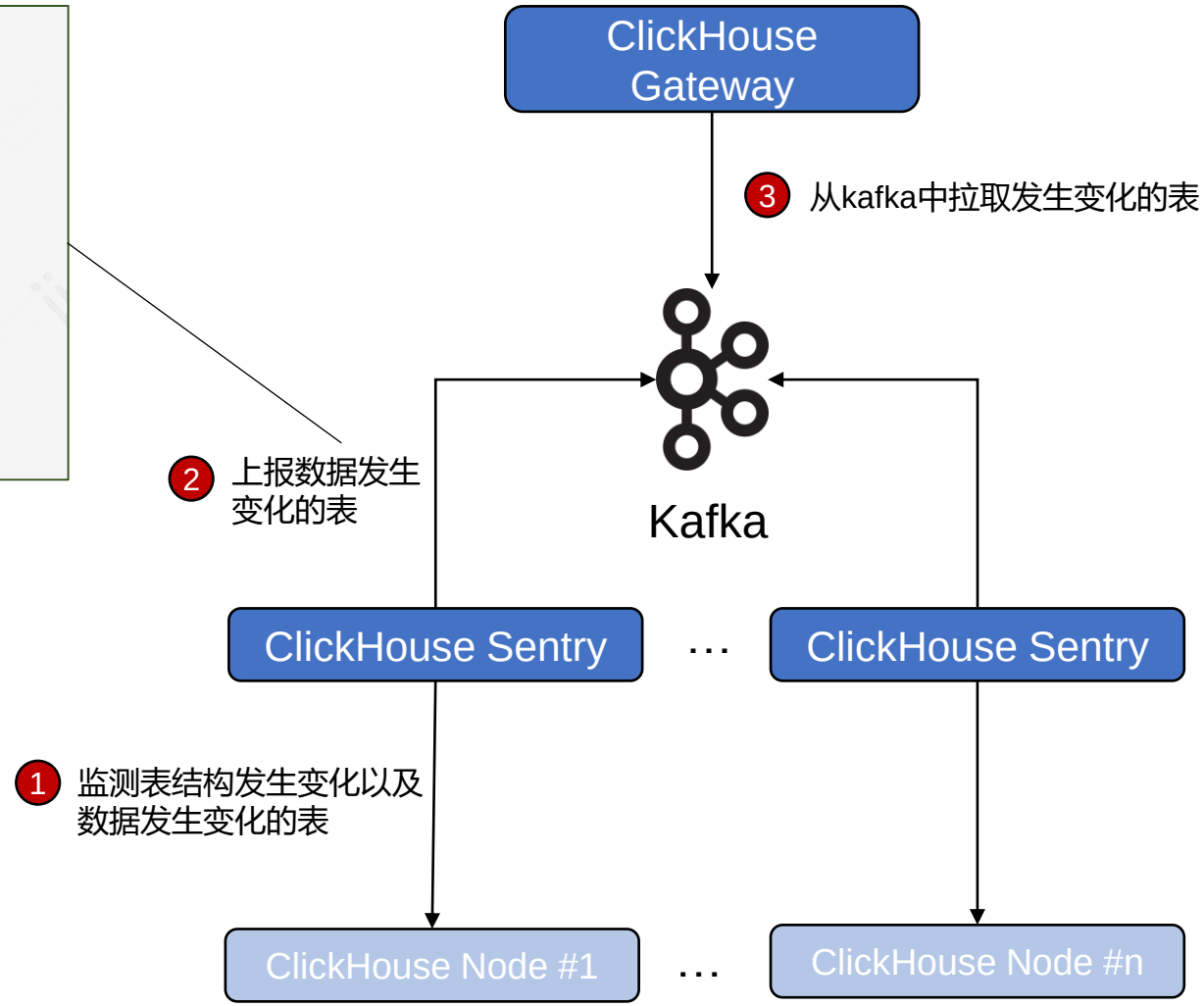
2.2 缓存失效机制

想彻底实现缓存要解决两个关键问题

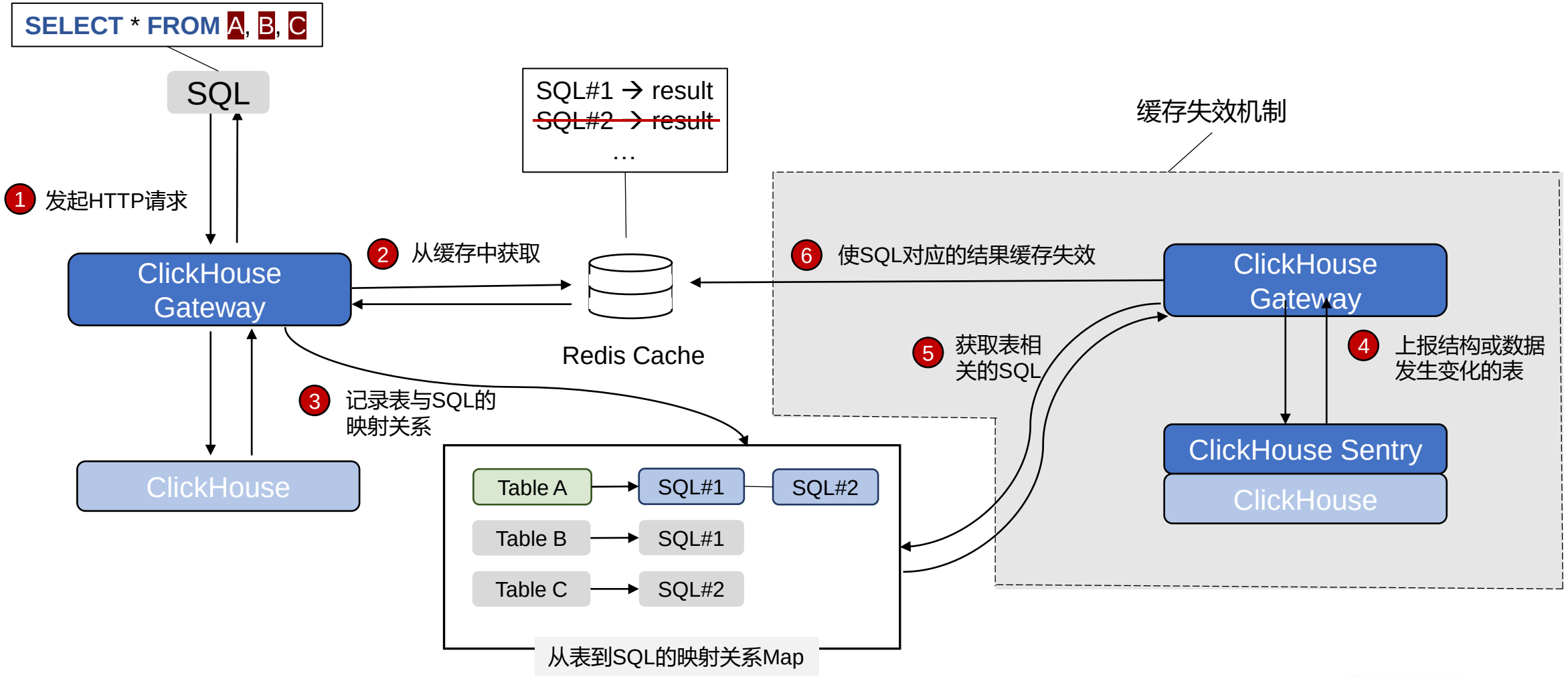
- ClickHouse Gateway 要感知到 ClickHouse 中哪些表发生了变化
- 要知道缓存中的每条 SQL所涉及到的表， 构建出从表到 SQL 的映射关系

2.2 缓存失效机制：表变化上报

```sql
-- 每隔60秒发起此 SQL 感知发生变化的表
SELECT
    database,
    table,
    max(modification_time)
FROM
    system.parts
GROUP BY
    database,
    table
HAVING
    max(modification_time) > now() - 60
```



2.2 缓存失效机制：整体流程概览



2.2 缓存失效机制：根据SQL解析出相关的表

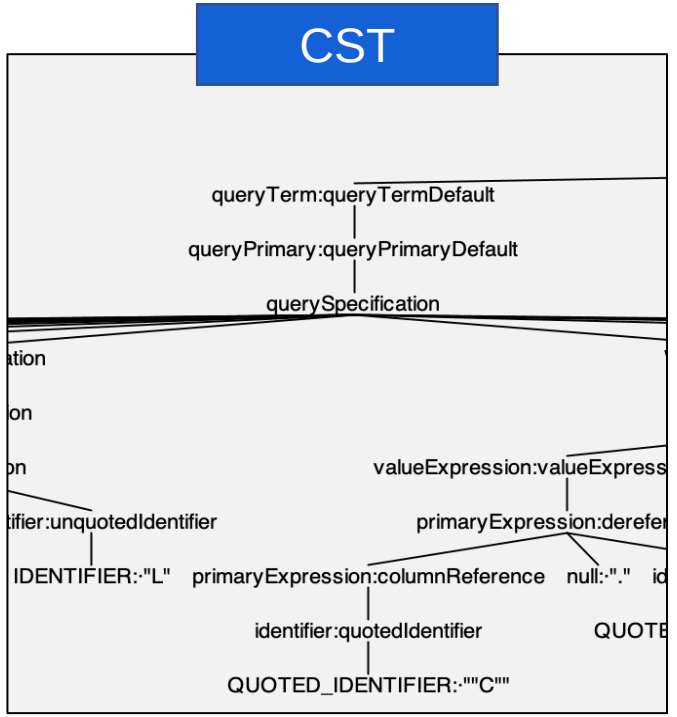
```
SELECT
    c.custkey, sum(l.price)
FROM
    customer c, orders o, lineitem l
WHERE
    c.custkey = o.custkey
    AND l.orderkey = o.orderkey
GROUP BY
    c.custkey
ORDER BY
    sum(l.price) DESC;
```

SqlBase.g4

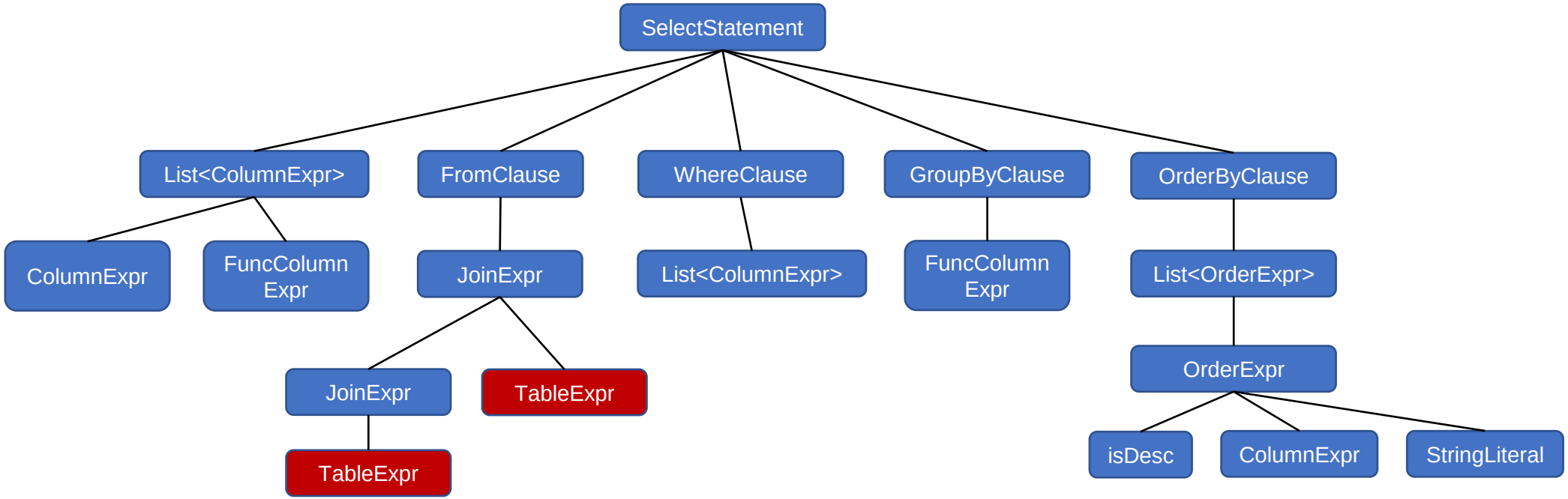
```
grammar SqlBase;

tokens...

singleStatement
    : statement EOF
    ;

standaloneExpression
    : expression EOF
    ;

standalonePathSpecification
    : pathSpecification EOF
    ;

standaloneType
    : type EOF
    ;

statement
    : query                                                          #statementDefault
    | USE schema=identifier                                          #use
    | USE catalog=identifier '.' schema=identifier                   #use
    | CREATE SCHEMA (IF NOT EXISTS)? qualifiedName
        (AUTHORIZATION principal)?
        (WITH properties)?                                           #createSchema
    | DROP SCHEMA (IF EXISTS)? qualifiedName (CASCADE | RESTRICT)?   #dropSchema
    | ALTER SCHEMA qualifiedName RENAME TO identifier                #renameSchema
    | ALTER SCHEMA qualifiedName SET AUTHORIZATION principal         #setSchemaAuthorization
    | CREATE TABLE (IF NOT EXISTS)? qualifiedName columnAliases?
        (COMMENT string)?
        (WITH properties)? AS (query | '('query')')
        (WITH (NO)? DATA)?                                           #createTableAsSelect
    | CREATE TABLE (IF NOT EXISTS)? qualifiedName
```

ANTLR

CST



2.2 缓存失效机制：根据SQL解析出相关的表



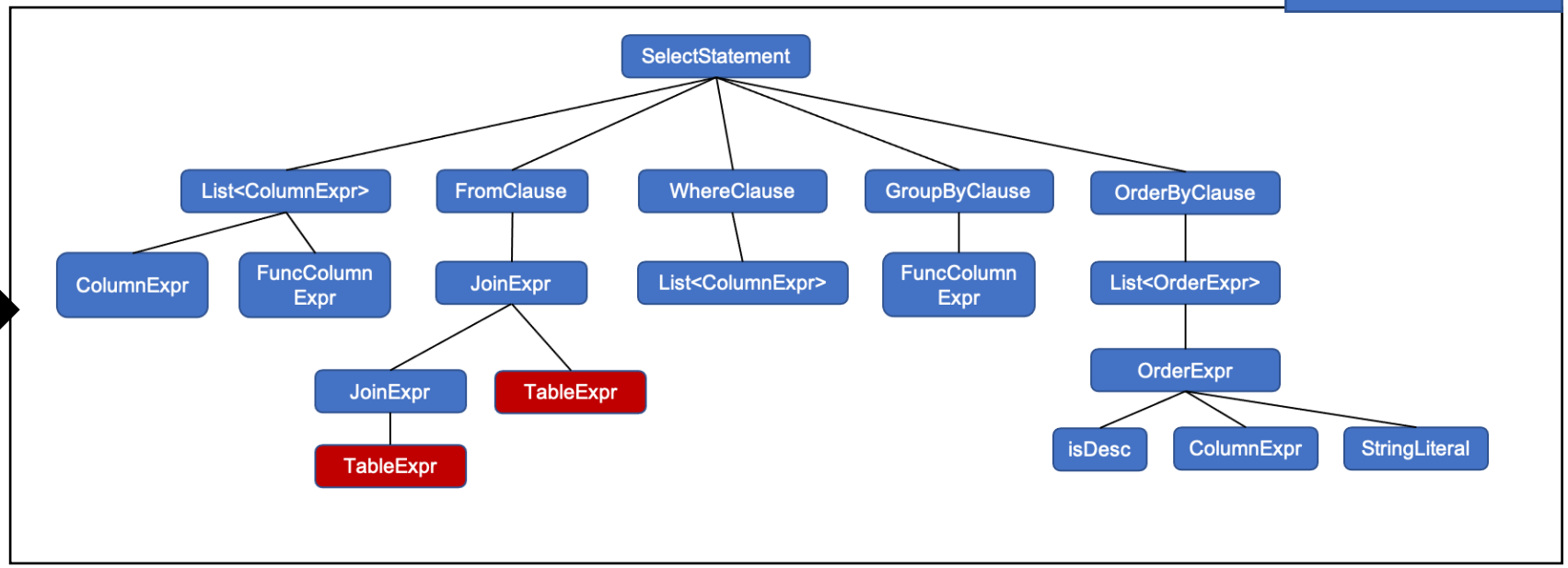
关于CST和AST的区别可参阅：https://eli.thegreenplace.net/2009/02/16/abstract-vs-concrete-syntax-trees/

2.2 缓存失效机制：根据SQL解析出相关的表

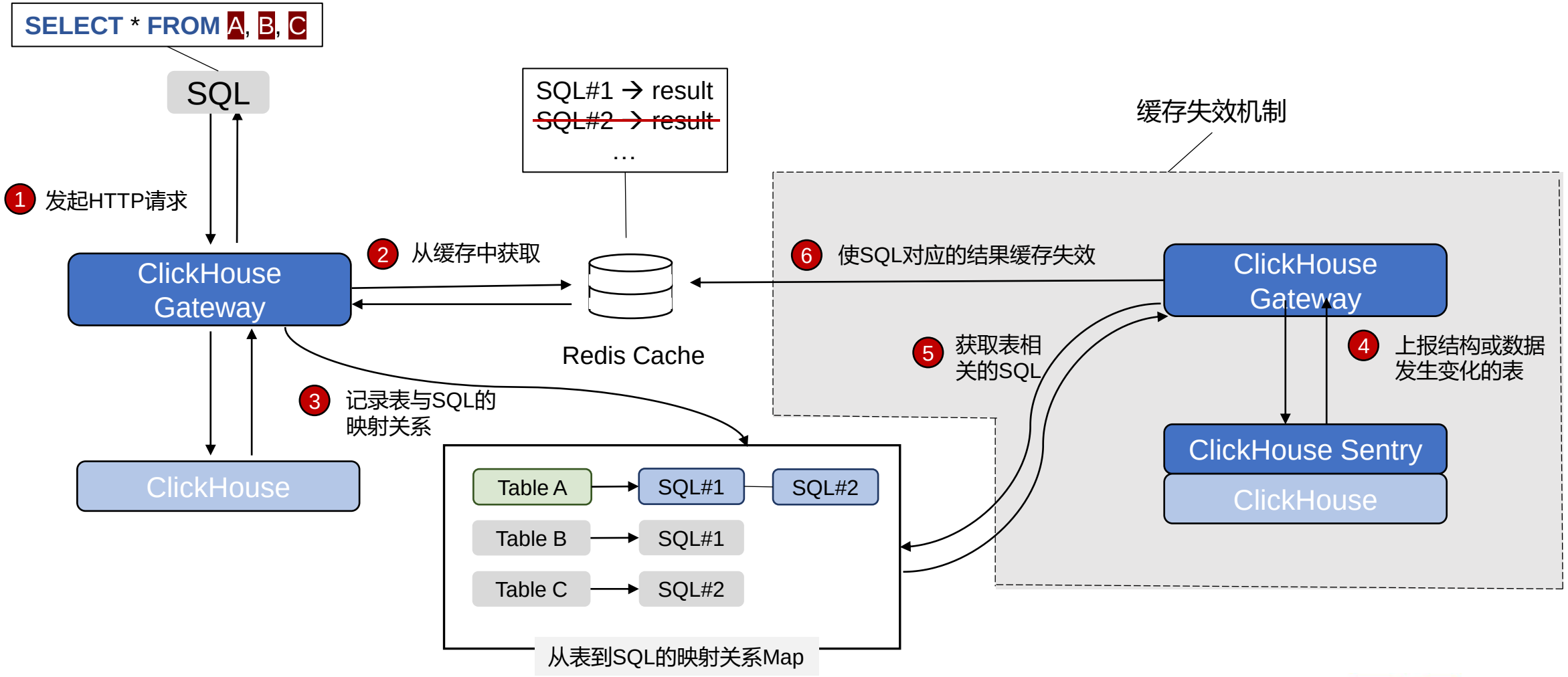
SQL

```
SELECT
    c.custkey, sum(l.price)
FROM
    customer c, orders o, lineitem l
WHERE
    c.custkey = o.custkey
    AND l.orderkey = o.orderkey
GROUP BY
    c.custkey
ORDER BY
    sum(l.price) DESC;
```

AST

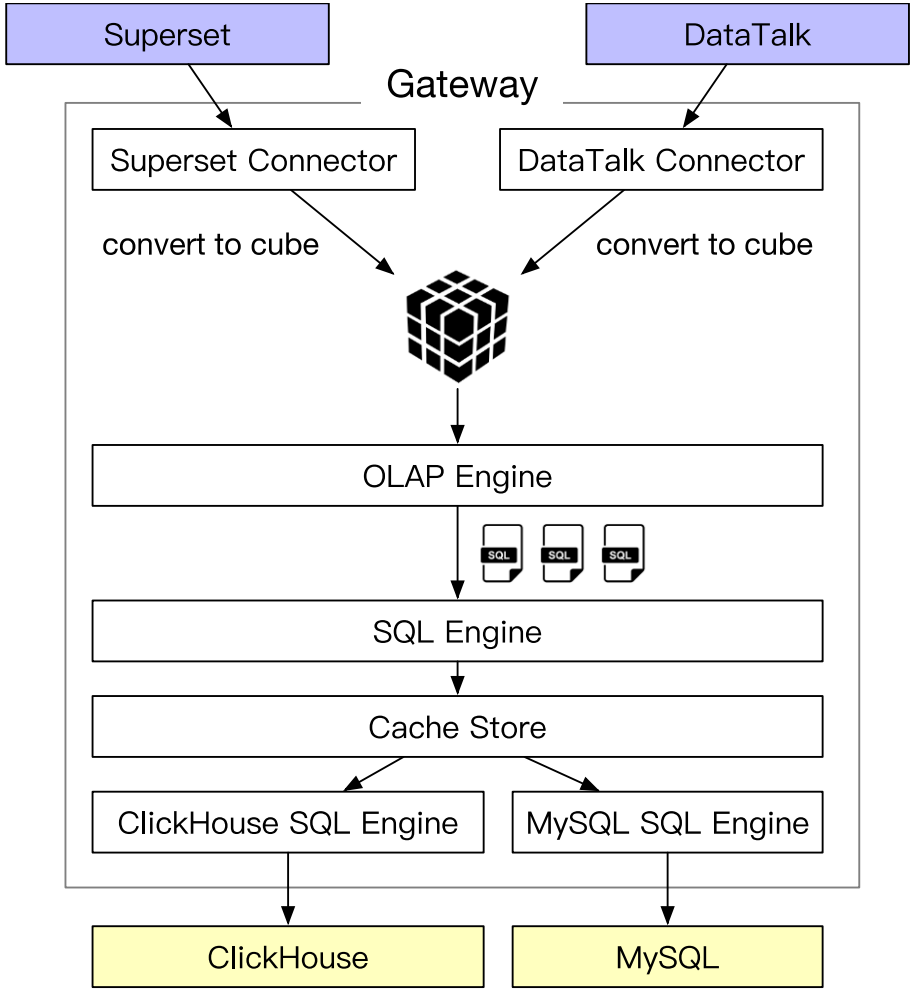


2.2 缓存失效机制：回顾整体流程



3. 基于Cube的缓存机制

3. 基于Cube的缓存机制：理论推导



3. 基于Cube的缓存机制：理论推导

S_t

```
SELECT
    dimension1,
    dimension2,
    time_dimension,
    SUM(measure1),
    AVG(measure2)
FROM
    (${数据源—自定义SQL}) T1
WHERE
    time_dimension >= '${时间戳t}'
GROUP BY
    dimension1,
    dimension2,
    time_dimension
HAVING
    ...
ORDER BY
    ...
LIMIT
    ${数字n}
```

$S_{t1,t2}$

```
SELECT
    dimension1,
    dimension2,
    time_dimension,
    SUM(measure1),
    AVG(measure2)
FROM
    (${数据源—自定义SQL}) T1
WHERE
    time_dimension >= '${时间戳t1}'
AND time_dimension <= '${时间戳t2}'
GROUP BY
    dimension1,
    dimension2,
    time_dimension
HAVING
    ...
ORDER BY
    ...
LIMIT
    ${数字n}
```

R_t

```
SELECT
    dimension1,
    dimension2,
    time_dimension,
    SUM(measure1),
    AVG(measure2)
FROM
    (${数据源—自定义SQL}) T1
WHERE
    time_dimension = '${时间戳t}'
GROUP BY
    dimension1,
    dimension2,
    time_dimension
HAVING
    ...
ORDER BY
    ...
LIMIT
    ${数字n}
```

以下递推关系式得到满足：

$$S_t = R_t \oplus S_{t-1}$$

$$S_{t1,t2} = R_{t2} \oplus S_{t1,t2-1}$$

$$S_{t1-1,t2} \subseteq S_{t1,t2}$$

$$\Downarrow$$

$$S_{t1-1,t2} \subseteq S_{t1,t2} = R_{t2} \oplus S_{t1,t2-1}$$

需要注意：

1）ORDER BY和LIMIT需要在合并后再进行计算。

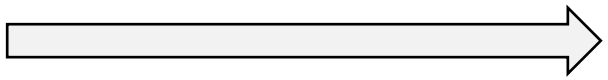
2）实际情况中，当天的数据可能还在不断地入库，也就是说当 t=TODAY 时，查询结果Rt很有可能一直在变，因此不能直接缓存当天的结果，要求出Rt和Rt-1再和St-2合并（因为前面描述的递推关系式满足**传递性**，所以可以这样操作）。

3）对于结果随自然时间推移发生变化的UDF，则不适用（例如度量中存在 today() – measure1）

3. 基于Cube的缓存机制：案例分析

date	type	sales	profit
2021-03-20	software	1000	600
2021-03-20	hardware	500	250
2021-03-21	software	1500	800
2021-03-21	hardware	750	500

```
SELECT
    date,
    SUM(sales),
    AVG(profit)
FROM
    datasource
WHERE
    date >= '2021-03-20'
GROUP BY
    date
```



S_{t-1}

date	sum(sales)	avg(profit)
2021-03-20	1500	425
2021-03-21	2250	650
2021-03-22	750	300

假设在2021-03-22的时候执行过上述的SQL，得到结果St-1

3. 基于Cube的缓存机制：案例分析

S_t

```
SELECT
    date,
    SUM(sales),
    AVG(profit)
FROM
    datasource
WHERE
    date >= '2021-03-23'
GROUP BY
    date
```

R_{t-1}, R_t

```
SELECT
    date,
    SUM(sales),
    AVG(profit)
FROM
    datasource
WHERE
    date = '2021-03-22’
    and date = '2021-03-23'
GROUP BY
    date
```

S_{t-1}

date	sum(sales)	avg(profit)
2021-03-20	1500	425
2021-03-21	2250	650
2021-03-22	750	300

S_t

date	sum(sales)	avg(profit)
2021-03-20	1500	425
2021-03-21	2250	650
2021-03-22	2000	600
2021-03-23	600	800

3. 基于Cube的缓存机制：合并算子的实现

设 t4 时刻进行查询，查询的时间范围为[t1, t4]，查询结果保存的缓存结果集为 St1,t4，到了 t5 时刻，又继续进行一次查询，查询范围为的[t2,t5]，这时先从 ck 中查询时间范围为 [t4, t5] 的结果，结果集为 St4,t5。此时合并操作的不能直接裁剪掉St1,t4最后一条记录。正确做法：

date	sum(sales)	avg(profit)
2021-03-21	2250	650
2021-03-20	1500	425
2021-03-22	750	300
2021-03-23	600	500

步骤1：从缓存结果中取出St1,t4，并且去掉 date < t2 以及 date = t4 的记录，得到St2,t3

date	sum(sales)	avg(profit)
2021-03-23	2000	600
2021-03-24	600	800

步骤2：从ck中查询[t2, t5]的结果，得到St2,t5

date	sum(sales)	avg(profit)
2021-03-21	2250	650
2021-03-22	750	300
2021-03-23	2000	600
2021-03-24	600	800

步骤3：合并 St2,t3 和 St4,t5，得到St2,t5

date	sum(sales)	avg(profit)
2021-03-21	2250	650
2021-03-23	2000	600
2021-03-22	750	300
2021-03-24	600	800

步骤4：对 St2,t5 执行 ORDER BY 算子和 LIMIT 算子，得到最终结果

4. 缓存加速在线上的效果表现

4. 缓存加速在线上的效果表现

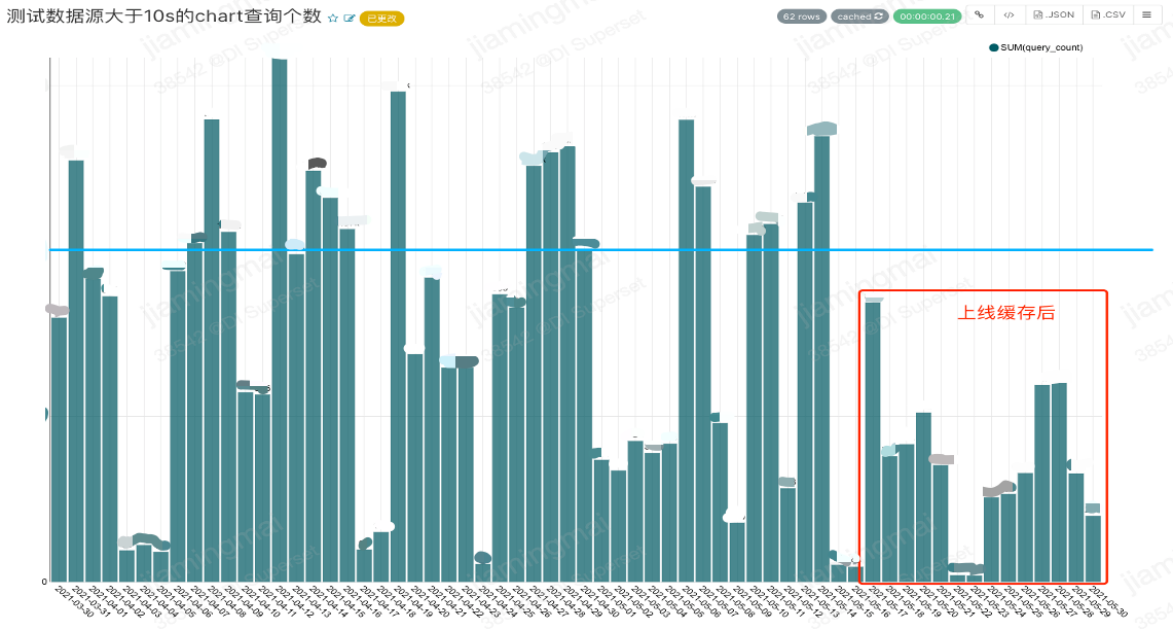


图1 查询超过10秒的数量明显减少

现阶段的表现

- 查询超过10秒的数量明显减少
- 缓存命中率：约 **25%**

后续的工作

- 通过预计算提高命中率
- 将ClickHouse查询转换为Spark计算任务减少预计算成本

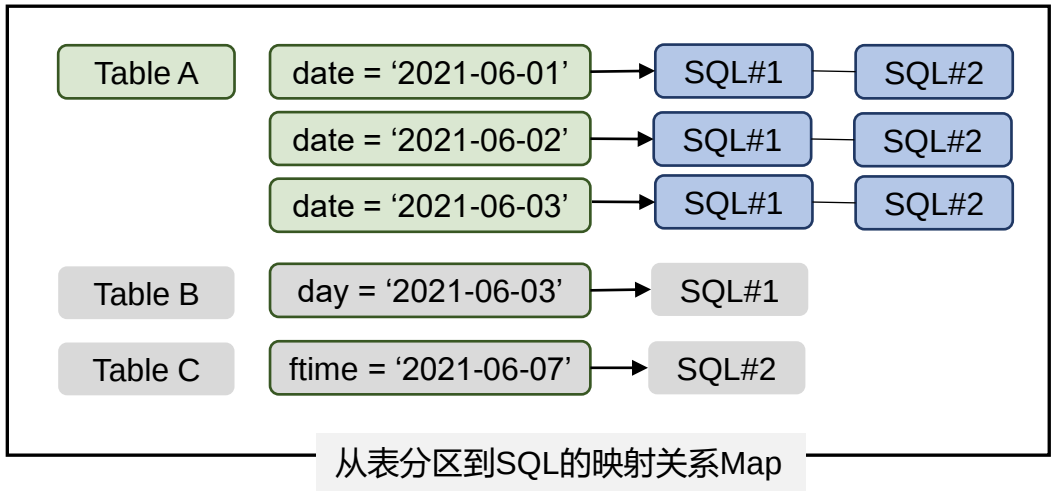
4. 缓存加速在线上的效果表现

报表中能分析到所有 query 所涉及到的表的分布情况

5. 后续的工作

5. 后续的工作

- 基于统一网关实现流控和资源隔离
- 实现分区粒度的缓存失效机制，提高缓存命中率
- 分析历史query的分区热点，实现冷热存储分离



5. 后续的工作

通过转换离线任务进行预计算，降本提效

Dashboard

Pre-computing by scheduled task

create AST

AST

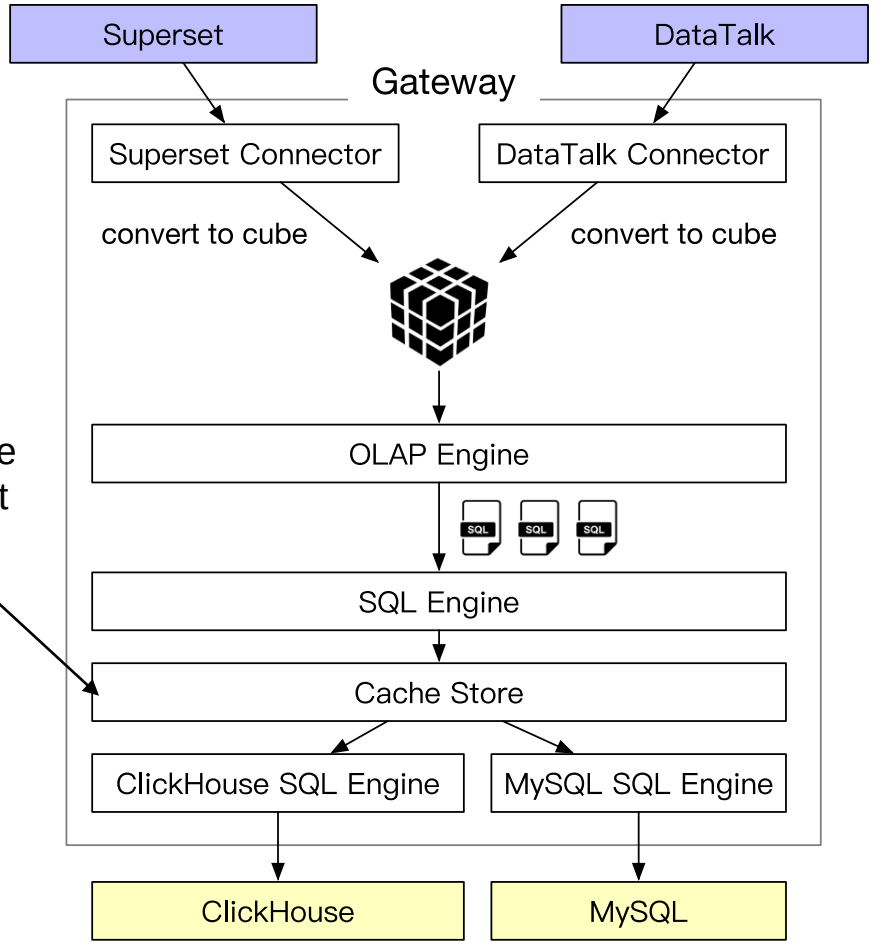
convert to Spark SQL

Submit & Compute

cache result

ClickHouse

Spark SQL



Thanks & More Heroes

数据平台工程师

高级数据分析师

数据科学家

数据后台工程师

创 造 音 乐 无 限 可 能

CREATING ENDLESS
OPPORTUNITIES WITH MUSIC

腾讯音乐娱乐集团
TENCENT MUSIC ENTERTAINMENT

THANKS