腾讯云 | 云+社区 intel

腾讯云 ClickHouse 性能调优及实践

pengjian.uestc@gmail.com

目录

一、 腾讯云ClickHouse在QQ音乐实践

二、 常见ClickHouse实时分析场景

三、 腾讯云ClickHouse性能调优

四、 腾讯云ClickHouse现状与规划

ClickHouse简介

1 完备的DBMS功能

- DDL(数据定义语言)
- DML(数据操作语言)
- 权限控制
- 数据备份与恢复
- 分布式管理

2 列式存储与数据压缩

- 极大地节约了IO带宽
- 压缩比高（支持LZ4, ZSTD)

3 向量化执行引擎

- 分布式计算
- 多核并行计算
- 向量化执行与SIMD
- 动态代码生成

4 SQL支持

- 对用户友好的SQL语法
- 内置功能齐全的分析统计函数
- 丰富的数据结构支持，字典、json, array, bitmap等

5 数据存储

- 自管理数据存储，不依赖其他组件
- 主键索引/二级索引
- 数据集分片(sharding)
- 数据分区(partition)
- 数据容灾
- TTL支持

6 性能卓越

- 无论是查询还是写入，性能极其卓越

一、 QQ音乐PB级数据实时分析带来的挑战

- Hive离线分析仅能满足T+1定时报表单一场景
- Hive无法满足QQ音乐多角色（产品、运营人员等）对实时交互分析的诉求

时效性 低

- 基于Hive跑离线任务需要**数小时**
- 分析结果价值随着时间推移而迅速降低

易用性 低

- 数据分析需求来源于 产品、运营、市场 多方人员
- 由于分析门槛高，**产品、运营、市场 人员无法自主分析**

流程效率 低

- 分析需求均需由分析师完成，需排期、沟通、建模、分析、可视化等流程
- 整套流程常需几周时间， 分析结果不及时

一、 QQ音乐PB级数据实时分析带来的挑战

ClickHouse集群现状

- **集群规模**

近万核，**PB级存储**、十万亿级别记录量。每天过千亿数据落地入库保存（实时流水、离线中间表等约700张表）。

- **性能指标**

查询千亿、万亿流水的请求可在**数秒内**完成。

ClickHouse

Superset

业务价值

- **实时性**

复杂交互分析秒级完成（如分版本、分平台DAU，营收及其他多口径业务指标）。

- **易用性**

利用Superset可自主DIY各类报表，当前SuperSet过万图表中，**超一半由产品、研发、运营、研究员、财务等非数据同学创建**。

目录

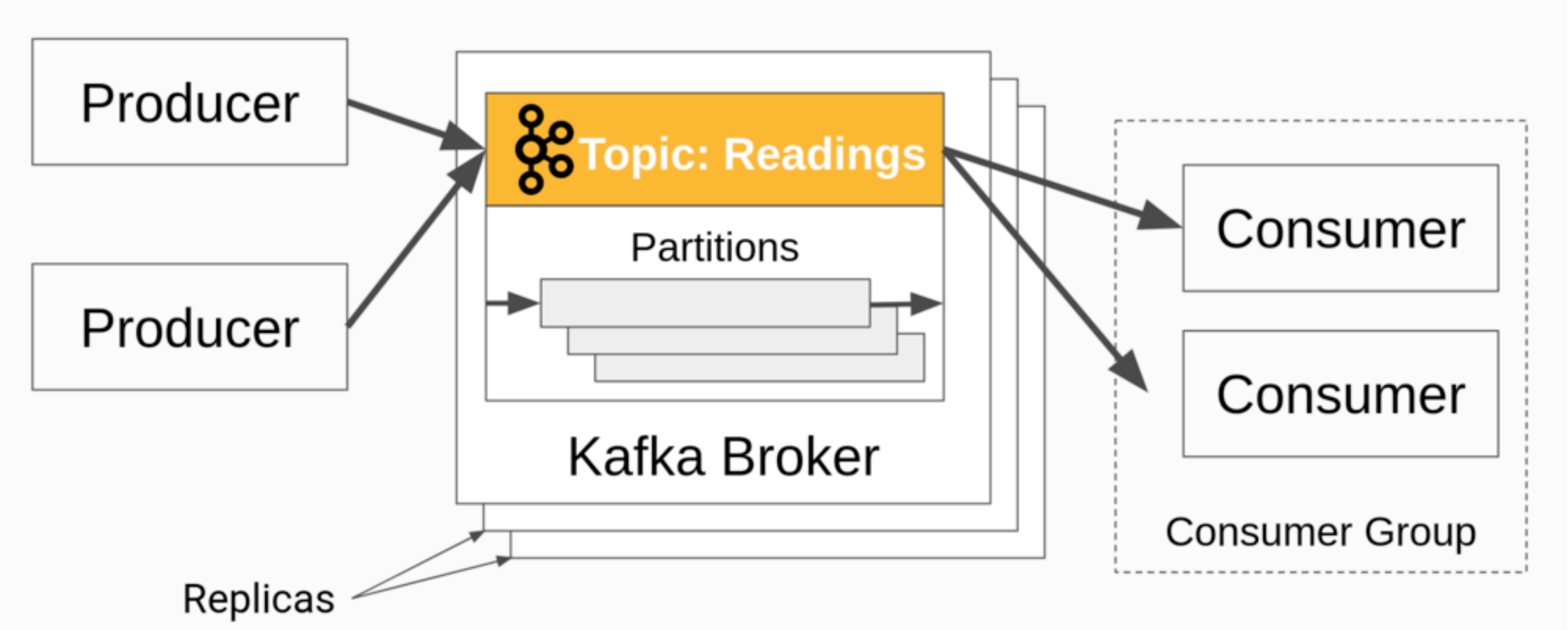
一、 腾讯云ClickHouse在QQ音乐实践

二、 常见ClickHouse实时分析场景

三、 腾讯云ClickHouse性能调优

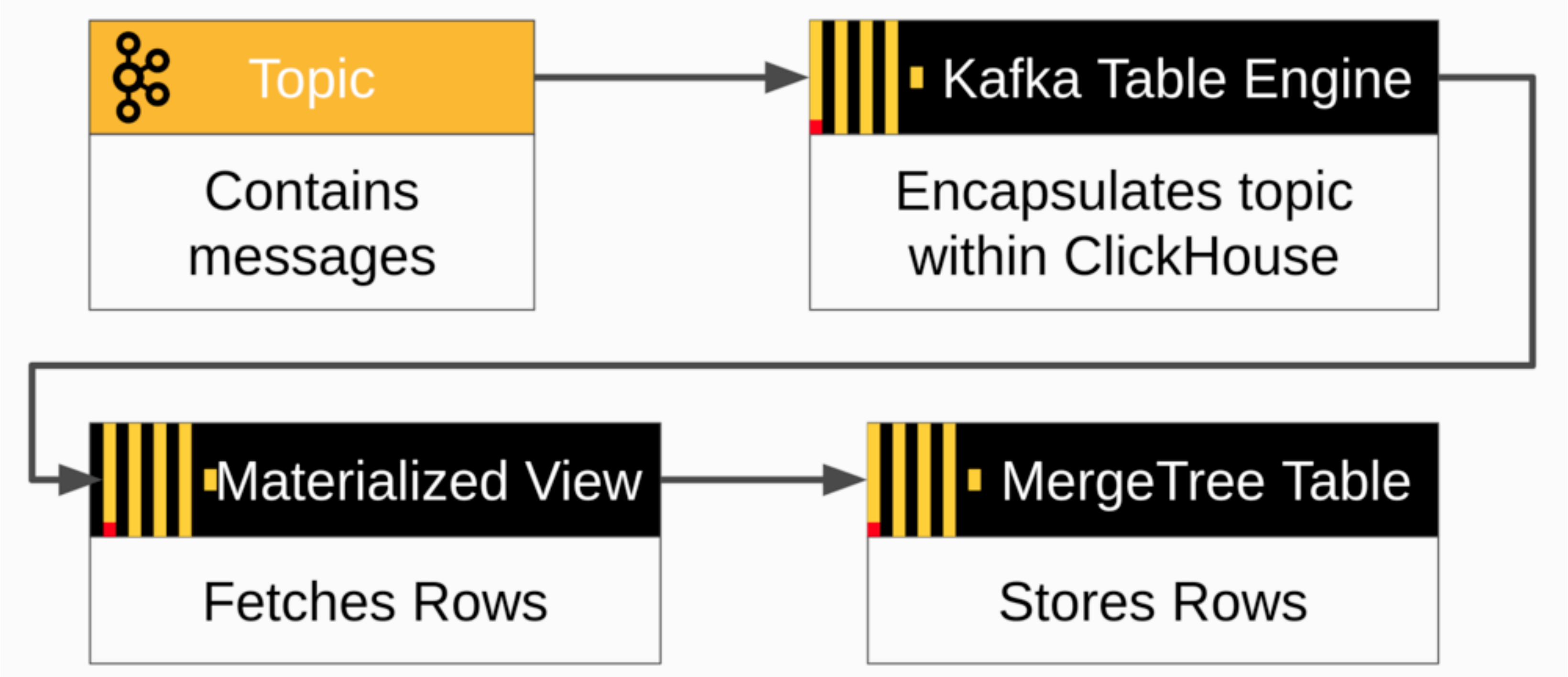
四、 腾讯云ClickHouse现状与规划

场景1： 物化视图应用：数据摄入



三、 常见ClickHouse实时分析场景举例

场景1： 物化视图应用：数据摄入



Topic

Contains messages

Kafka Table Engine

Encapsulates topic within ClickHouse

Materialized View

Fetches Rows

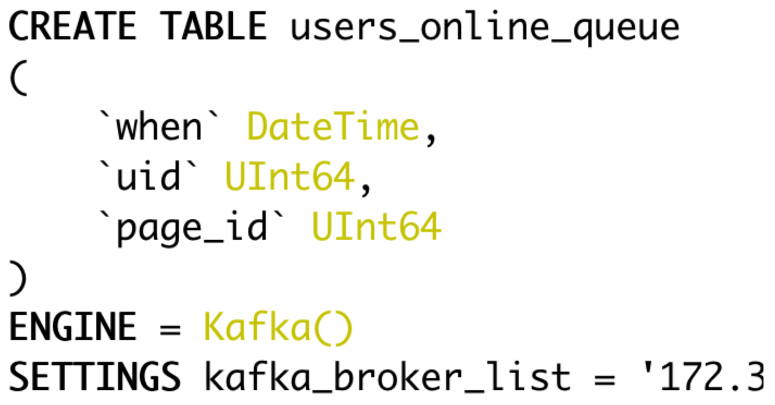
MergeTree Table

Stores Rows

三、 常见ClickHouse实时分析场景举例

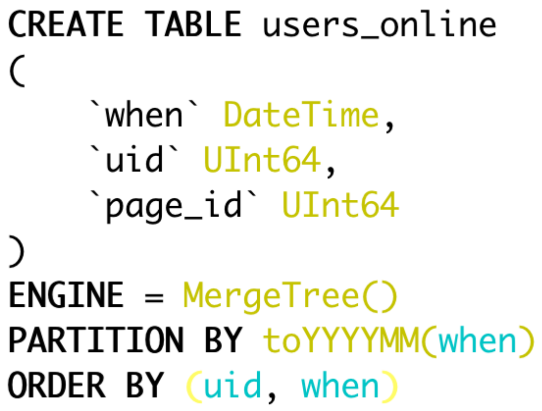
场景1： 物化视图应用：数据摄入

1 创建Kafka表引擎，监听数据



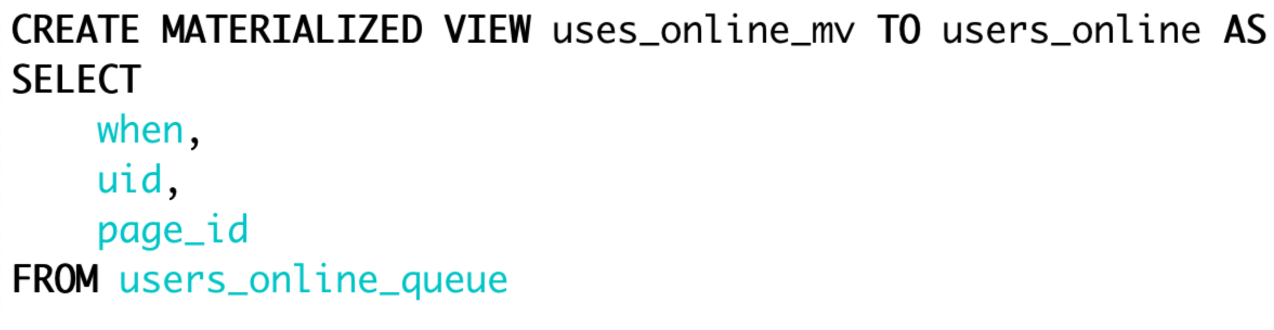
```
CREATE TABLE users_online_queue
(
    `when` DateTime,
    `uid` UInt64,
    `page_id` UInt64
)
ENGINE = Kafka()
SETTINGS kafka_broker_list = '172.3
```

2 创建目标表，存储数据



```
CREATE TABLE users_online
(
    `when` DateTime,
    `uid` UInt64,
    `page_id` UInt64
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(when)
ORDER BY (uid, when)
```

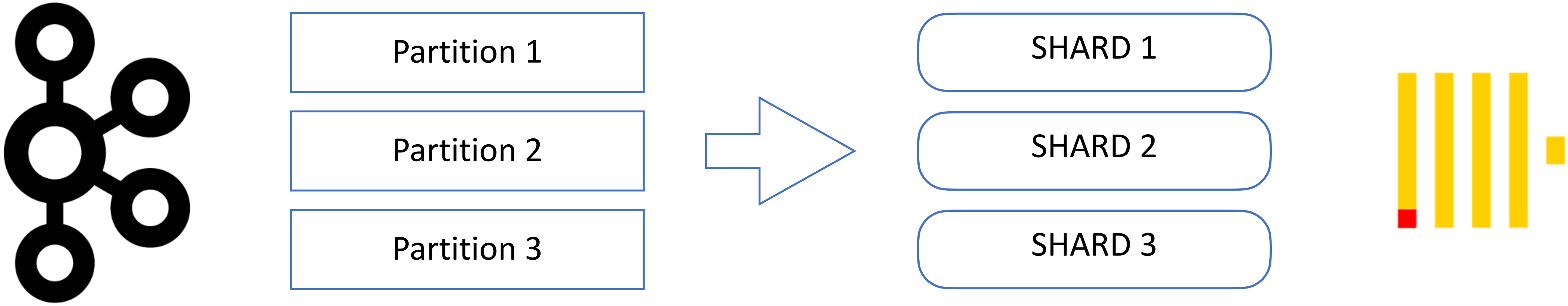
3 创建物化视图，摄取数据



```
CREATE MATERIALIZED VIEW uses_online_mv TO users_online AS
SELECT
    when,
    uid,
    page_id
FROM users_online_queue
```

三、 常见ClickHouse实时分析场景举例

场景1： 物化视图应用：数据摄入



Partition 1

Partition 2

Partition 3

SHARD 1

SHARD 2

SHARD 3

场景2：ClickHouse在实时更新场景中的应用

假设某APP 需要在线向用户下发通知，如果用户读取消息后，状态回传后台，并标记已读取。

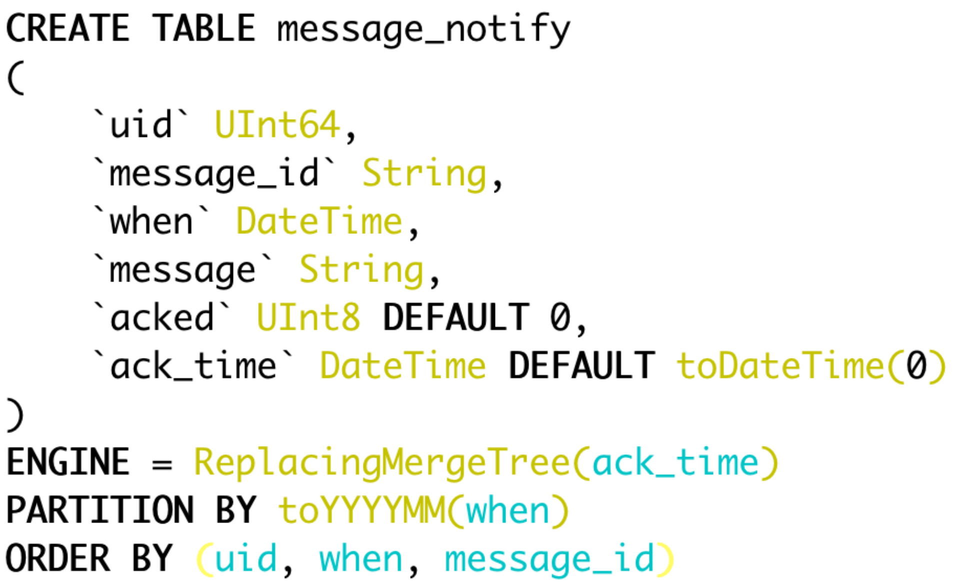
数据实时更新！

查询某用户未读取信息

三、 常见ClickHouse实时分析场景举例

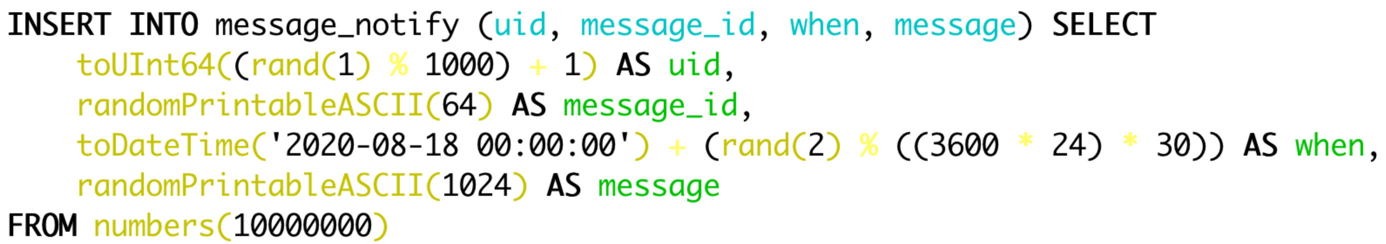
场景2：ClickHouse在实时更新场景中的应用 方案1：ReplacingMergeTree

1 创建明细表

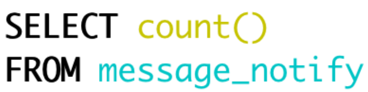


```
CREATE TABLE message_notify
(
    `uid` UInt64,
    `message_id` String,
    `when` DateTime,
    `message` String,
    `acked` UInt8 DEFAULT 0,
    `ack_time` DateTime DEFAULT toDateTime(0)
)
ENGINE = ReplacingMergeTree(ack_time)
PARTITION BY toYYYYMM(when)
ORDER BY (uid, when, message_id)
```

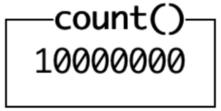
2 插入模拟数据



```
INSERT INTO message_notify (uid, message_id, when, message) SELECT
    toUInt64((rand(1) % 1000) + 1) AS uid,
    randomPrintableASCII(64) AS message_id,
    toDateTime('2020-08-18 00:00:00') + (rand(2) % ((3600 * 24) * 30)) AS when,
    randomPrintableASCII(1024) AS message
FROM numbers(10000000)
```



```
SELECT count()
FROM message_notify
```

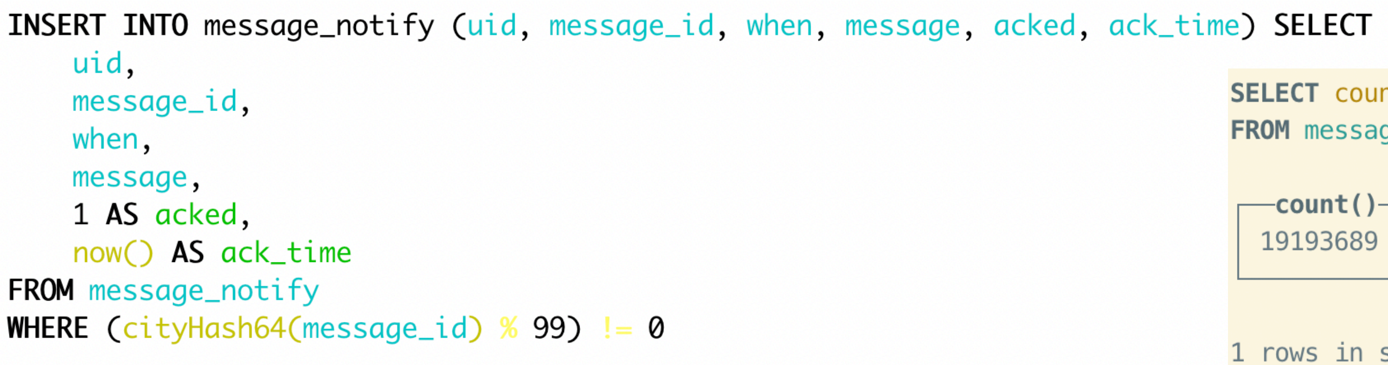


```
┌─count()─┐
│ 10000000 │
└──────────┘
```

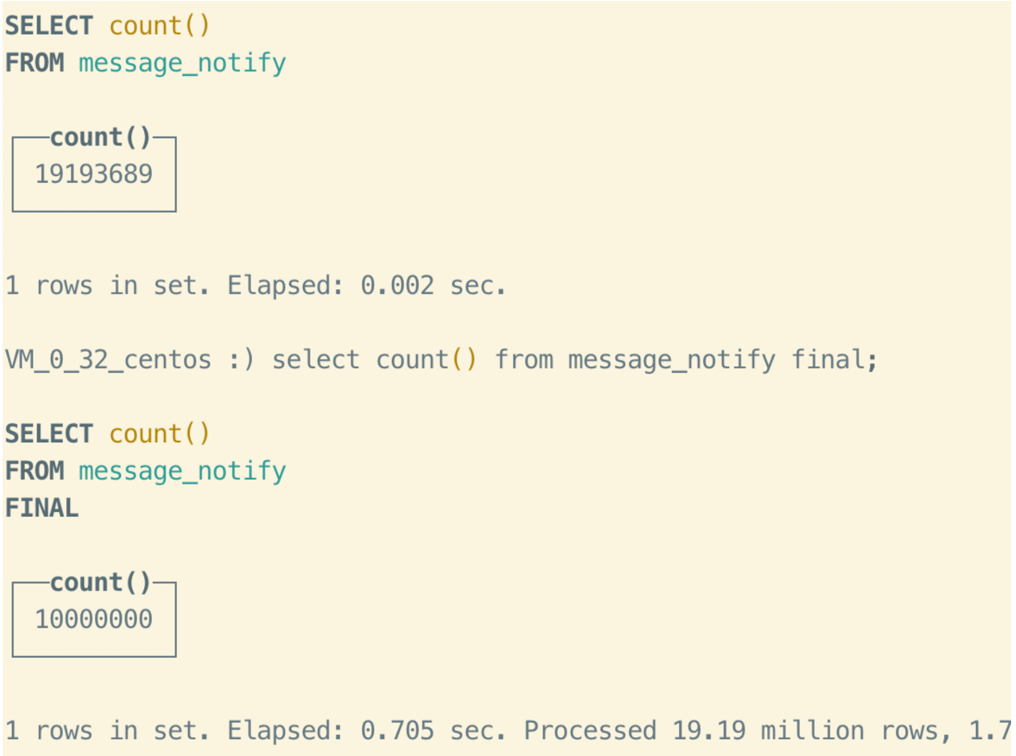
场景2：ClickHouse在实时更新场景中的应用 方案1：ReplacingMergeTree

3 插入模拟数据，模拟用户APP应答

4 查看数据



```
INSERT INTO message_notify (uid, message_id, when, message, acked, ack_time) SELECT
    uid,
    message_id,
    when,
    message,
    1 AS acked,
    now() AS ack_time
FROM message_notify
WHERE (cityHash64(message_id) % 99) != 0
```

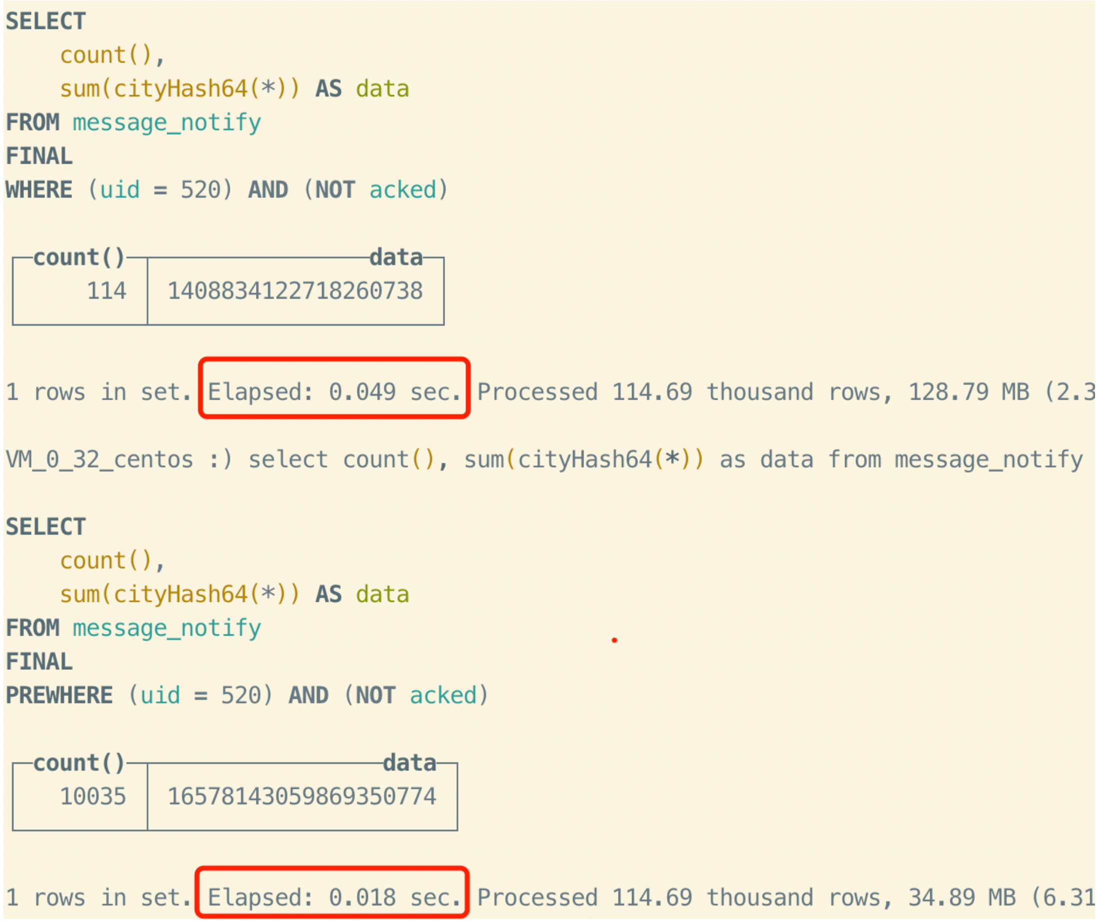


```
SELECT count()
FROM message_notify

┌─count()─┐
│ 19193689 │
└──────────┘

1 rows in set. Elapsed: 0.002 sec.

VM_0_32_centos :) select count() from message_notify final;

SELECT count()
FROM message_notify
FINAL

┌─count()─┐
│ 10000000 │
└──────────┘

1 rows in set. Elapsed: 0.705 sec. Processed 19.19 million rows, 1.7
```

三、 常见ClickHouse实时分析场景举例

场景2：ClickHouse在实时更新场景中的应用 方案1：ReplacingMergeTree

5 查看用户(uid=520) 未读数据

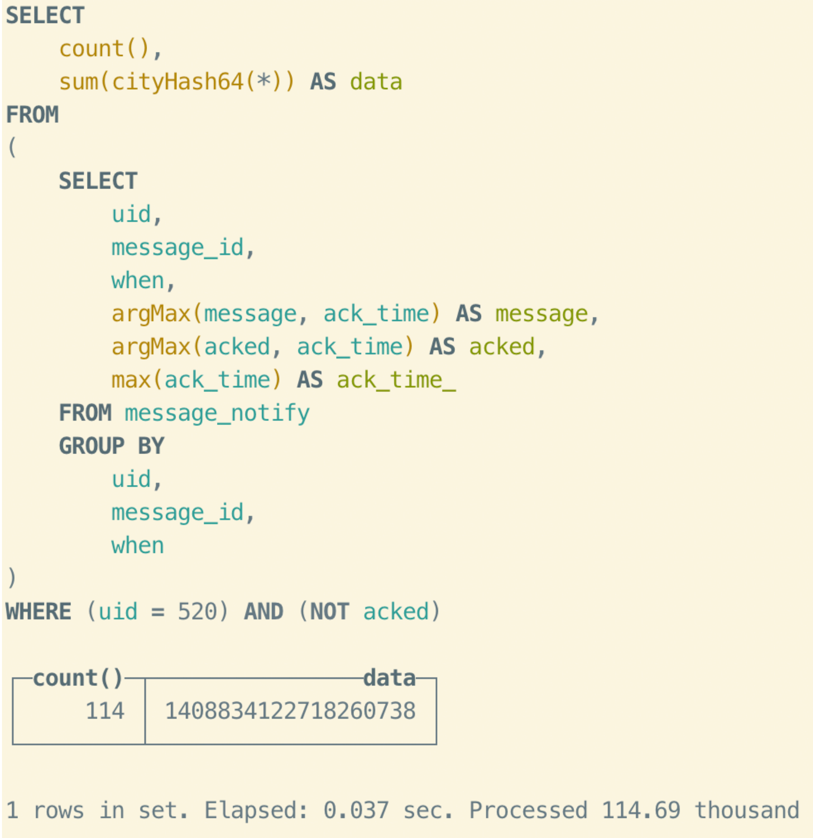


```
SELECT
    count(),
    sum(cityHash64(*)) AS data
FROM message_notify
FINAL
WHERE (uid = 520) AND (NOT acked)

┌─count()─┬─────────────────data─┐
│     114 │ 1408834122718260738 │
└─────────┴─────────────────────┘

1 rows in set. Elapsed: 0.049 sec. Processed 114.69 thousand rows, 128.79 MB (2.3

VM_0_32_centos :) select count(), sum(cityHash64(*)) as data from message_notify

SELECT
    count(),
    sum(cityHash64(*)) AS data
FROM message_notify
FINAL
PREWHERE (uid = 520) AND (NOT acked)

┌─count()─┬──────────────────data─┐
│   10035 │ 16578143059869350774 │
└─────────┴──────────────────────┘

1 rows in set. Elapsed: 0.018 sec. Processed 114.69 thousand rows, 34.89 MB (6.31
```

三、 常见ClickHouse实时分析场景举例

场景2：ClickHouse在实时更新场景中的应用 方案2：Aggregate Functions

1 查看用户(uid=520) 未读数据

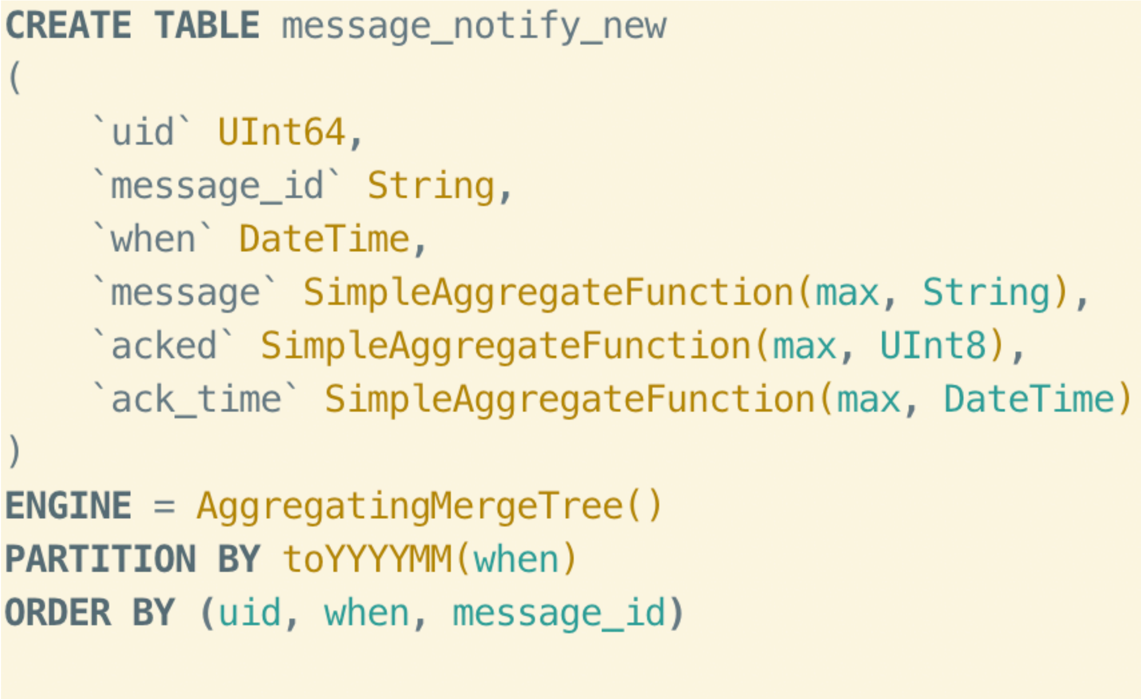


```
SELECT
    count(),
    sum(cityHash64(*)) AS data
FROM
(
    SELECT
        uid,
        message_id,
        when,
        argMax(message, ack_time) AS message,
        argMax(acked, ack_time) AS acked,
        max(ack_time) AS ack_time_
    FROM message_notify
    GROUP BY
        uid,
        message_id,
        when
)
WHERE (uid = 520) AND (NOT acked)

┌─count()─┬────────────────data─┐
│     114 │ 1408834122718260738 │
└─────────┴─────────────────────┘

1 rows in set. Elapsed: 0.037 sec. Processed 114.69 thousand
```

三、 常见ClickHouse实时分析场景举例

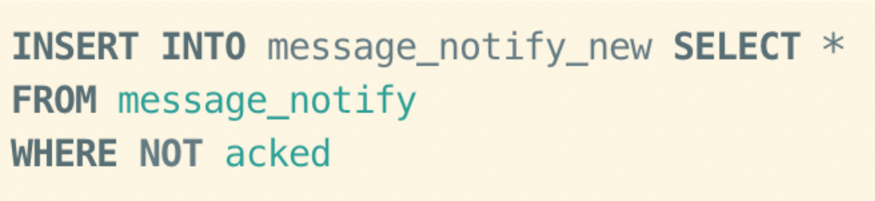
场景2：ClickHouse在实时更新场景中的应用 方案3：AggregatingMergeTree

1 聚合表引擎

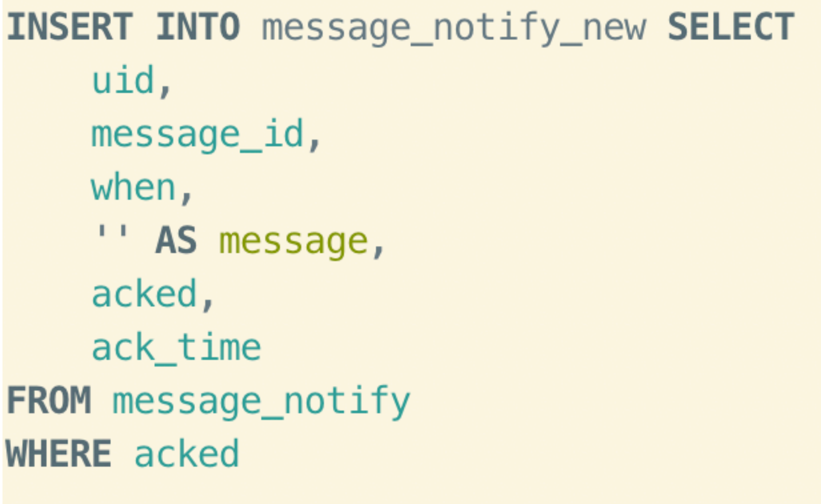


```
CREATE TABLE message_notify_new
(
    `uid` UInt64,
    `message_id` String,
    `when` DateTime,
    `message` SimpleAggregateFunction(max, String),
    `acked` SimpleAggregateFunction(max, UInt8),
    `ack_time` SimpleAggregateFunction(max, DateTime)
)
ENGINE = AggregatingMergeTree()
PARTITION BY toYYYYMM(when)
ORDER BY (uid, when, message_id)
```

2 准备数据



```
INSERT INTO message_notify_new SELECT *
FROM message_notify
WHERE NOT acked
```

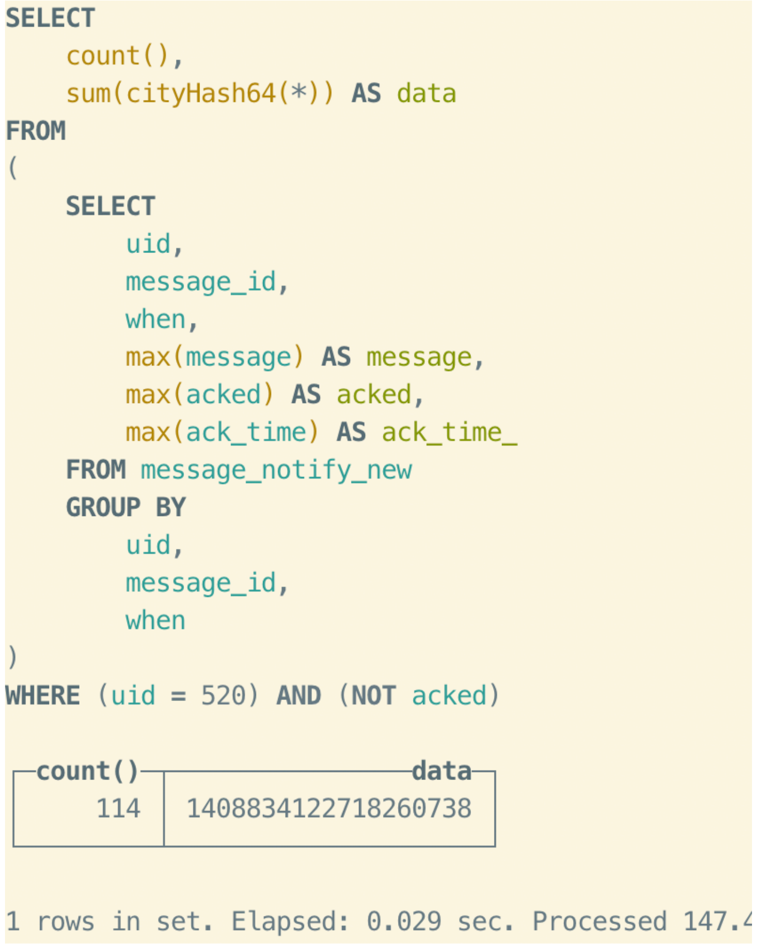


```
INSERT INTO message_notify_new SELECT
    uid,
    message_id,
    when,
    '' AS message,
    acked,
    ack_time
FROM message_notify
WHERE acked
```

三、 常见ClickHouse实时分析场景举例

场景2：ClickHouse在实时更新场景中的应用 方案3：AggregatingMergeTree

3 查看用户(uid=520) 未读数据

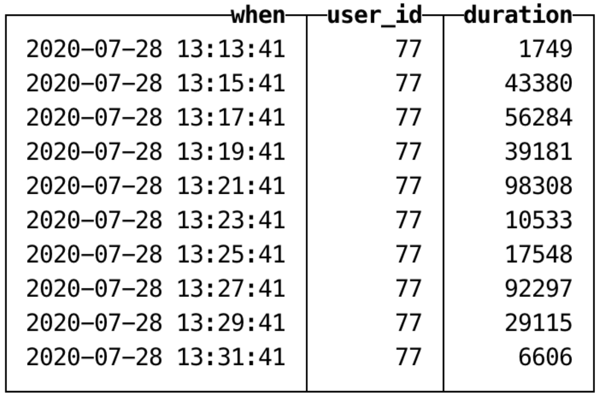


```
SELECT
    count(),
    sum(cityHash64(*)) AS data
FROM
(
    SELECT
        uid,
        message_id,
        when,
        max(message) AS message,
        max(acked) AS acked,
        max(ack_time) AS ack_time_
    FROM message_notify_new
    GROUP BY
        uid,
        message_id,
        when
)
WHERE (uid = 520) AND (NOT acked)

┌─count()─┬─────────────────data─┐
│     114 │ 1408834122718260738 │
└─────────┴──────────────────────┘

1 rows in set. Elapsed: 0.029 sec. Processed 147.4
```

三、 常见ClickHouse实时分析场景举例

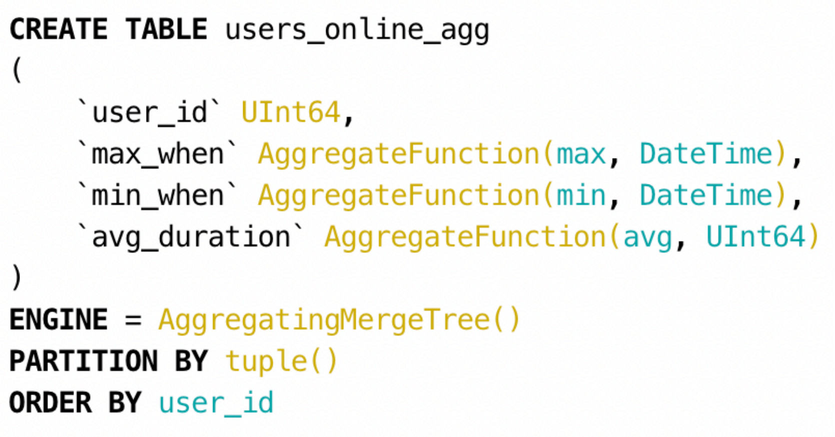
场景3： AggregatingMergeTree 引擎 + 物化视图 应用

1 明细表users_online



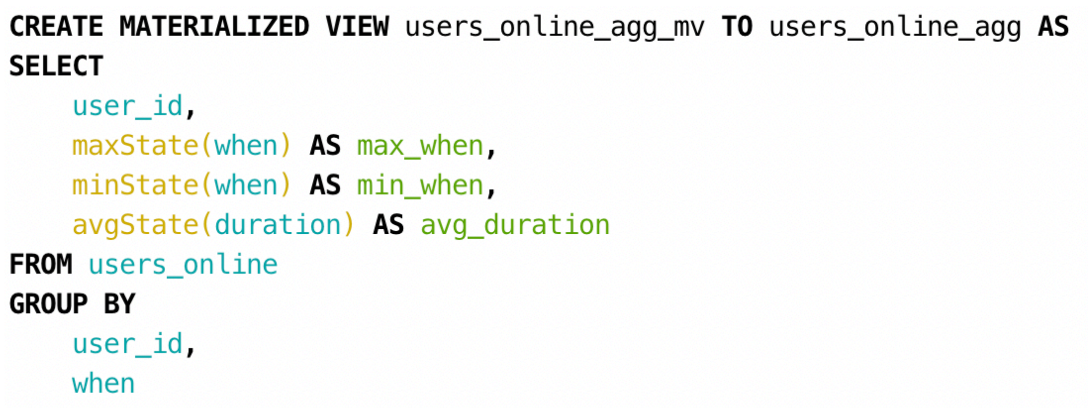
```
┌────────────────when─┬─user_id─┬─duration─┐
│ 2020-07-28 13:13:41 │      77 │     1749 │
│ 2020-07-28 13:15:41 │      77 │    43380 │
│ 2020-07-28 13:17:41 │      77 │    56284 │
│ 2020-07-28 13:19:41 │      77 │    39181 │
│ 2020-07-28 13:21:41 │      77 │    98308 │
│ 2020-07-28 13:23:41 │      77 │    10533 │
│ 2020-07-28 13:25:41 │      77 │    17548 │
│ 2020-07-28 13:27:41 │      77 │    92297 │
│ 2020-07-28 13:29:41 │      77 │    29115 │
│ 2020-07-28 13:31:41 │      77 │     6606 │
└─────────────────────┴─────────┴──────────┘
```

2 创建聚合表



```
CREATE TABLE users_online_agg
(
    `user_id` UInt64,
    `max_when` AggregateFunction(max, DateTime),
    `min_when` AggregateFunction(min, DateTime),
    `avg_duration` AggregateFunction(avg, UInt64)
)
ENGINE = AggregatingMergeTree()
PARTITION BY tuple()
ORDER BY user_id
```

3 创建物化视图

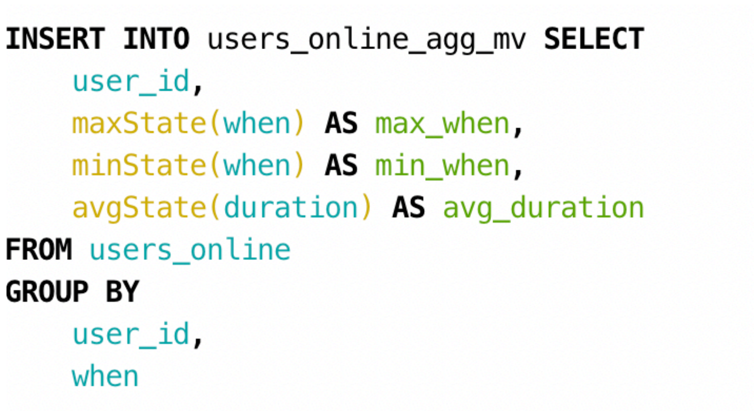


```
CREATE MATERIALIZED VIEW users_online_agg_mv TO users_online_agg AS
SELECT
    user_id,
    maxState(when) AS max_when,
    minState(when) AS min_when,
    avgState(duration) AS avg_duration
FROM users_online
GROUP BY
    user_id,
    when
```

三、 常见ClickHouse实时分析场景举例

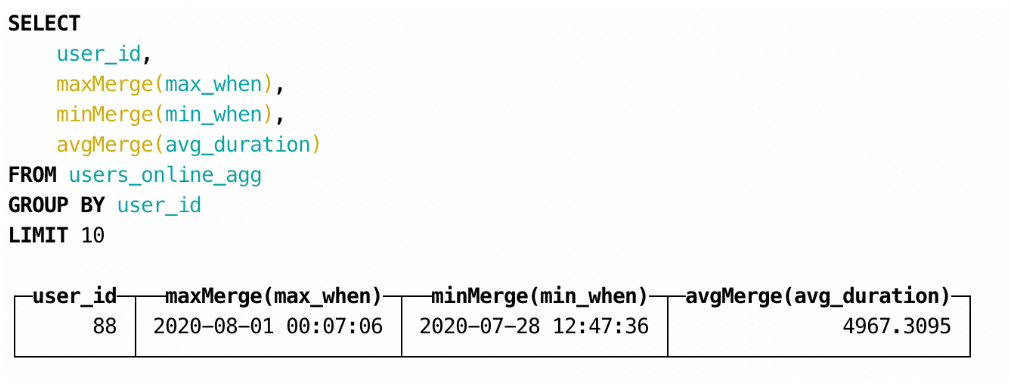
场景3： AggregatingMergeTree 引擎 + 物化视图 应用

4 导入存量数据



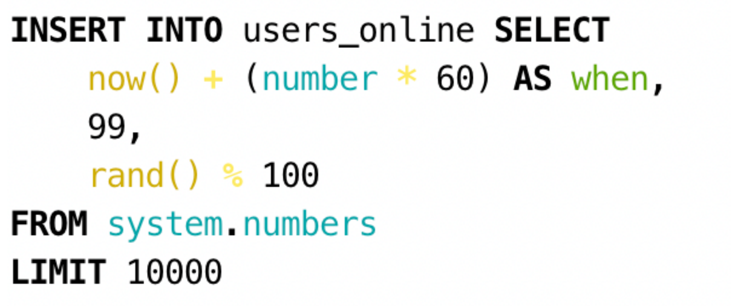
```
INSERT INTO users_online_agg_mv SELECT
    user_id,
    maxState(when) AS max_when,
    minState(when) AS min_when,
    avgState(duration) AS avg_duration
FROM users_online
GROUP BY
    user_id,
    when
```

5 查询聚合表中数据

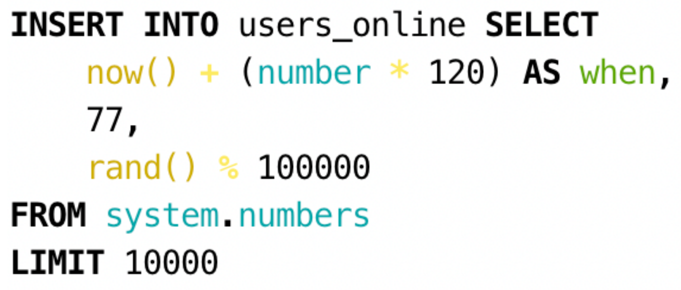


```
SELECT
    user_id,
    maxMerge(max_when),
    minMerge(min_when),
    avgMerge(avg_duration)
FROM users_online_agg
GROUP BY user_id
LIMIT 10

┌─user_id─┬───maxMerge(max_when)─┬───minMerge(min_when)─┬─avgMerge(avg_duration)─┐
│      88 │ 2020-08-01 00:07:06 │ 2020-07-28 12:47:36 │              4967.3095 │
└─────────┴─────────────────────┴─────────────────────┴────────────────────────┘
```

6 导入增量数据



```
INSERT INTO users_online SELECT
    now() + (number * 60) AS when,
    99,
    rand() % 100
FROM system.numbers
LIMIT 10000
```

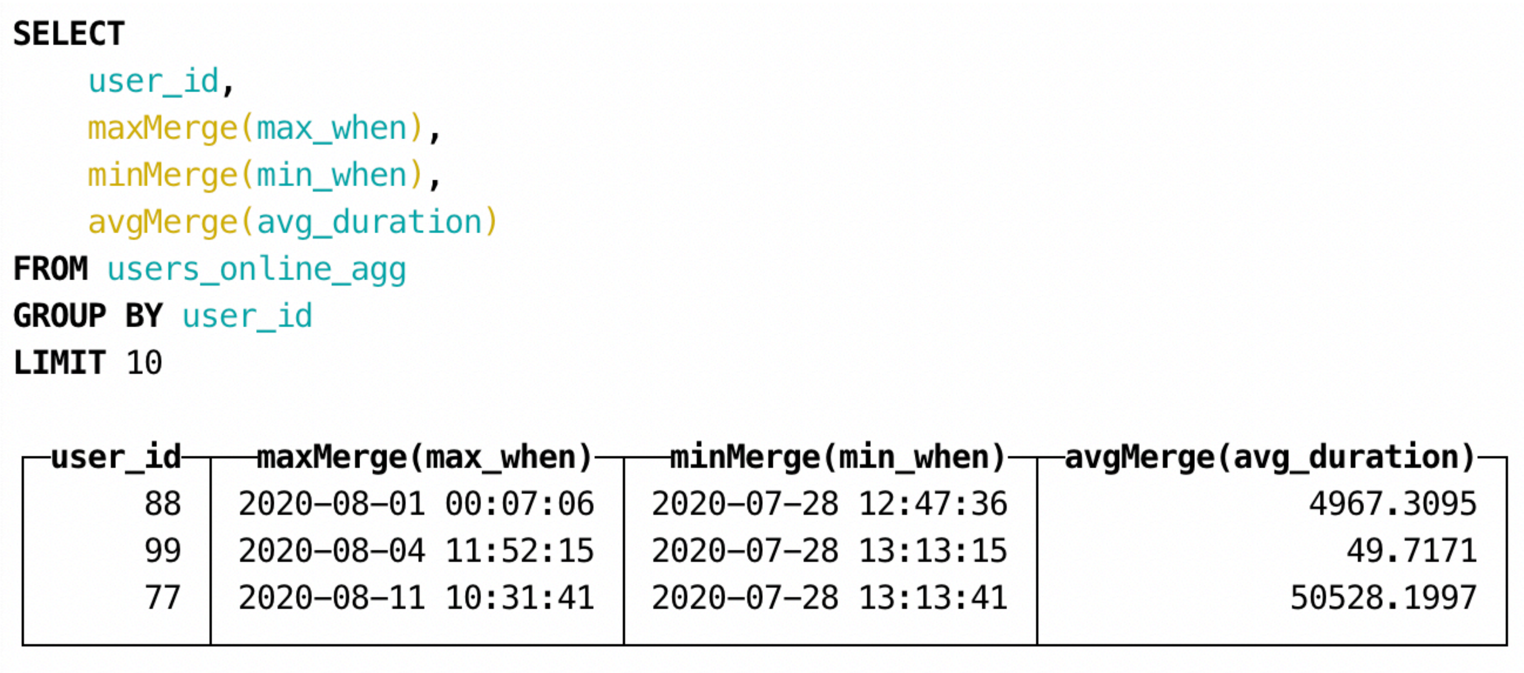


```
INSERT INTO users_online SELECT
    now() + (number * 120) AS when,
    77,
    rand() % 100000
FROM system.numbers
LIMIT 10000
```

三、 常见ClickHouse实时分析场景举例

场景3： AggregatingMergeTree 引擎 + 物化视图 应用

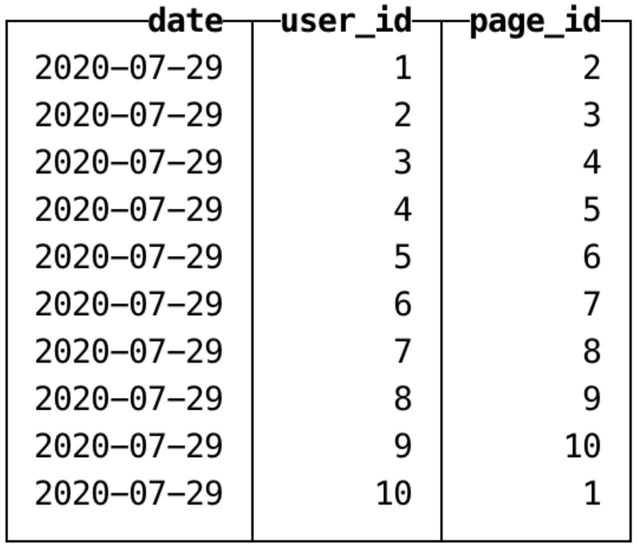
7 查询增量数据



```
SELECT
    user_id,
    maxMerge(max_when),
    minMerge(min_when),
    avgMerge(avg_duration)
FROM users_online_agg
GROUP BY user_id
LIMIT 10

┌─user_id─┬──maxMerge(max_when)─┬──minMerge(min_when)─┬─avgMerge(avg_duration)─┐
│      88 │ 2020-08-01 00:07:06 │ 2020-07-28 12:47:36 │              4967.3095 │
│      99 │ 2020-08-04 11:52:15 │ 2020-07-28 13:13:15 │                49.7171 │
│      77 │ 2020-08-11 10:31:41 │ 2020-07-28 13:13:41 │             50528.1997 │
└─────────┴─────────────────────┴─────────────────────┴────────────────────────┘
```

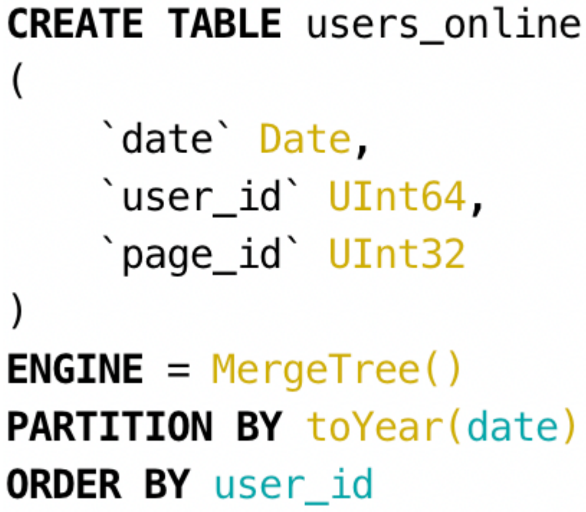
场景4： Aggregate Function 应用举例（bitmap系列）案例1：用户留存计算

Bitmap 在处理用户/会员统计，广告投放/用户画像等领域有非常便捷的应用。



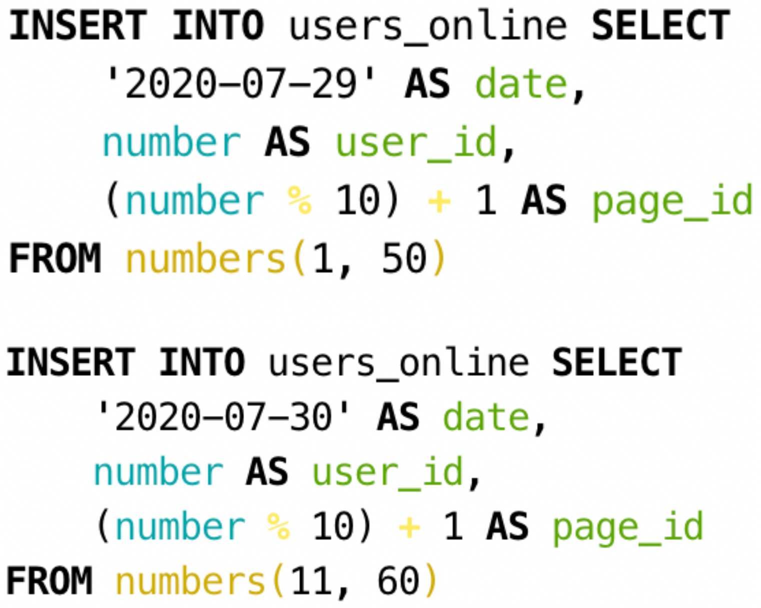
```
┌───────date─┬─user_id─┬─page_id─┐
│ 2020-07-29 │       1 │       2 │
│ 2020-07-29 │       2 │       3 │
│ 2020-07-29 │       3 │       4 │
│ 2020-07-29 │       4 │       5 │
│ 2020-07-29 │       5 │       6 │
│ 2020-07-29 │       6 │       7 │
│ 2020-07-29 │       7 │       8 │
│ 2020-07-29 │       8 │       9 │
│ 2020-07-29 │       9 │      10 │
│ 2020-07-29 │      10 │       1 │
└────────────┴─────────┴─────────┘
```

1 创建明细表



```
CREATE TABLE users_online
(
    `date` Date,
    `user_id` UInt64,
    `page_id` UInt32
)
ENGINE = MergeTree()
PARTITION BY toYear(date)
ORDER BY user_id
```

2 插入模拟数据



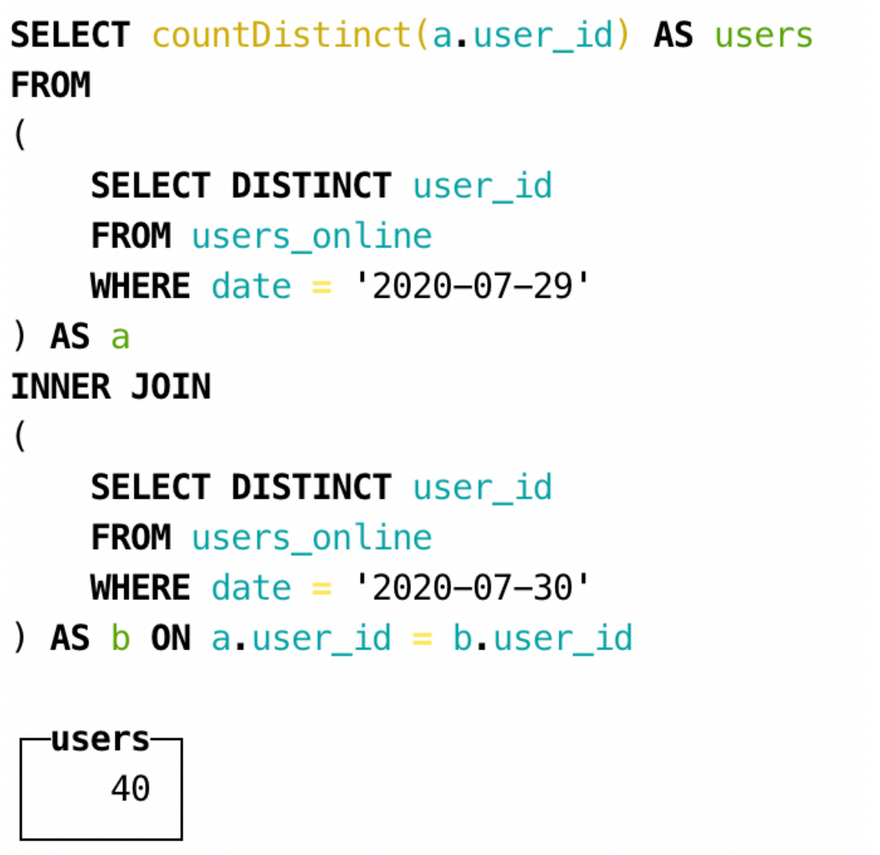
```
INSERT INTO users_online SELECT
    '2020-07-29' AS date,
    number AS user_id,
    (number % 10) + 1 AS page_id
FROM numbers(1, 50)

INSERT INTO users_online SELECT
    '2020-07-30' AS date,
    number AS user_id,
    (number % 10) + 1 AS page_id
FROM numbers(11, 60)
```

三、 常见ClickHouse实时分析场景举例

场景4： Aggregate Function 应用举例（bitmap系列）案例1：用户留存计算

2 查询用户日存留(常规方法)

举个例子~



```
SELECT countDistinct(a.user_id) AS users
FROM
(
    SELECT DISTINCT user_id
    FROM users_online
    WHERE date = '2020-07-29'
) AS a
INNER JOIN
(
    SELECT DISTINCT user_id
    FROM users_online
    WHERE date = '2020-07-30'
) AS b ON a.user_id = b.user_id
```

```
┌users┐
│   40 │
└──────┘
```

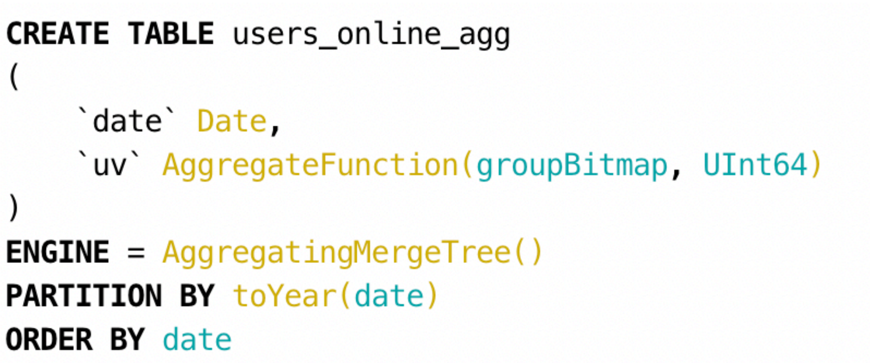
大表JOIN，COUNT DISTINCT 都很慢，并且容易OOM

三、 常见ClickHouse实时分析场景举例

腾讯云 | 云+社区 intel

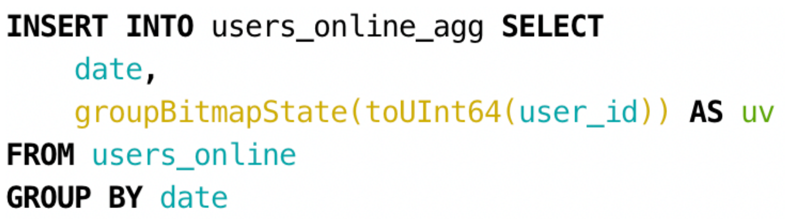
场景4： Aggregate Function 应用举例（bitmap系列）案例1：用户留存计算

3 使用聚合函数bitmap方案， 创建聚合表



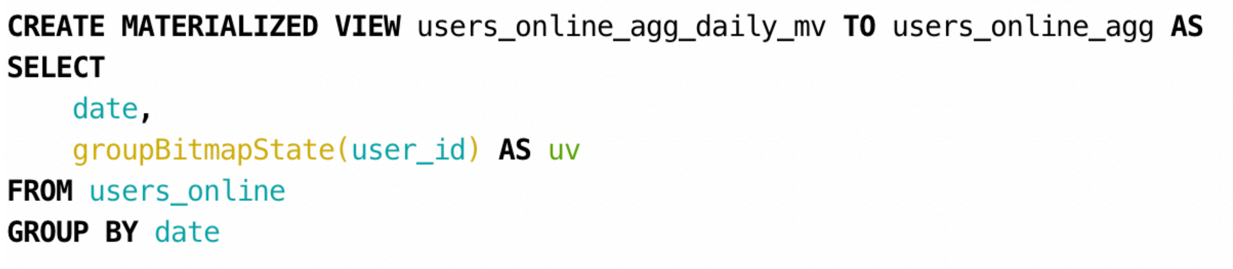
```
CREATE TABLE users_online_agg
(
    `date` Date,
    `uv` AggregateFunction(groupBitmap, UInt64)
)
ENGINE = AggregatingMergeTree()
PARTITION BY toYear(date)
ORDER BY date
```

4 导入历史数据



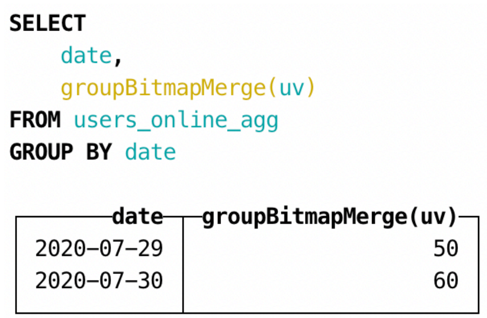
```
INSERT INTO users_online_agg SELECT
    date,
    groupBitmapState(toUInt64(user_id)) AS uv
FROM users_online
GROUP BY date
```

4 创建物化视图管理明细表与聚合表



```
CREATE MATERIALIZED VIEW users_online_agg_daily_mv TO users_online_agg AS
SELECT
    date,
    groupBitmapState(user_id) AS uv
FROM users_online
GROUP BY date
```

5 查询聚合表数据



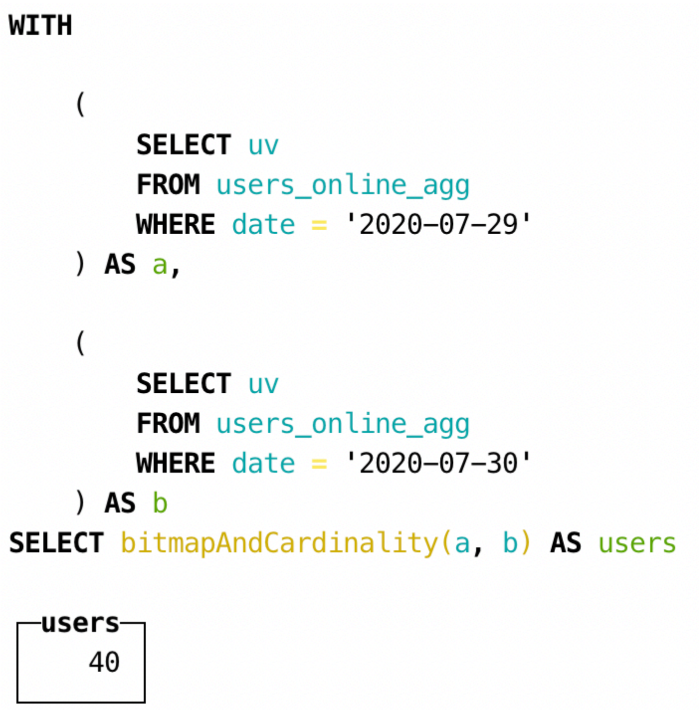
```
SELECT
    date,
    groupBitmapMerge(uv)
FROM users_online_agg
GROUP BY date

┌───────date─┬─groupBitmapMerge(uv)─┐
│ 2020-07-29 │                   50 │
│ 2020-07-30 │                   60 │
└────────────┴──────────────────────┘
```

通过BITMAP的位运算，对大规模用户(亿级）而言，位运算秒出结果

三、 常见ClickHouse实时分析场景举例

场景4： Aggregate Function 应用举例（bitmap系列）案例1：用户留存计算

6 使用聚合函数



```
WITH

    (
        SELECT uv
        FROM users_online_agg
        WHERE date = '2020-07-29'
    ) AS a,

    (
        SELECT uv
        FROM users_online_agg
        WHERE date = '2020-07-30'
    ) AS b
SELECT bitmapAndCardinality(a, b) AS users
```

```
┌users┐
│   40│
└─────┘
```

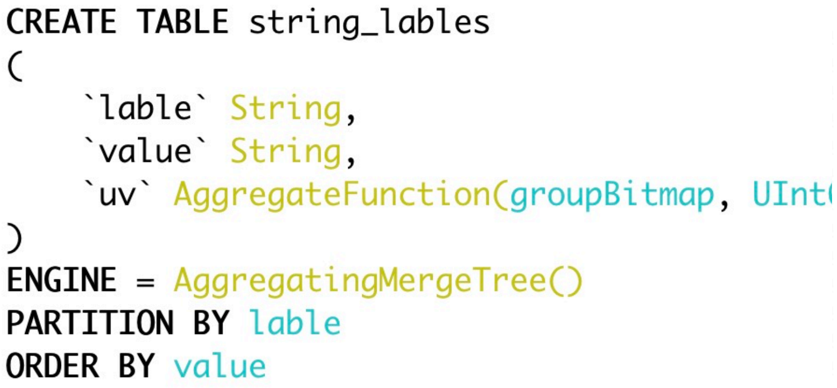
优势：

1. 计算量降低，速度快，资源消耗小

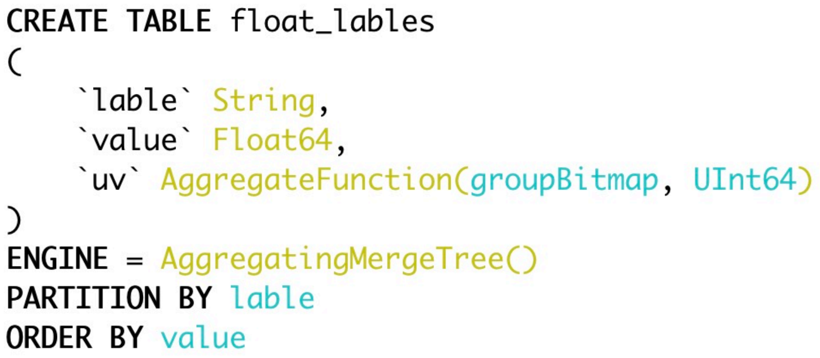
通过BITMAP的位运算，对大规模用户(亿级）而言，位运算秒出结果

场景4： Aggregate Function 应用举例（bitmap系列）案例2：广告投放/用户画像应用

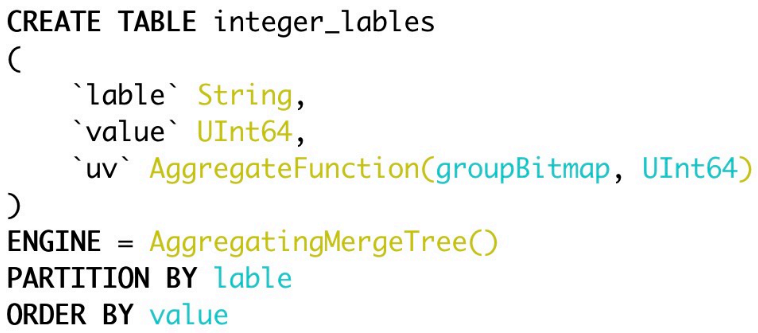
1 创建标签表



```
CREATE TABLE string_lables
(
    `lable` String,
    `value` String,
    `uv` AggregateFunction(groupBitmap, UInt
)
ENGINE = AggregatingMergeTree()
PARTITION BY lable
ORDER BY value
```



```
CREATE TABLE float_lables
(
    `lable` String,
    `value` Float64,
    `uv` AggregateFunction(groupBitmap, UInt64)
)
ENGINE = AggregatingMergeTree()
PARTITION BY lable
ORDER BY value
```

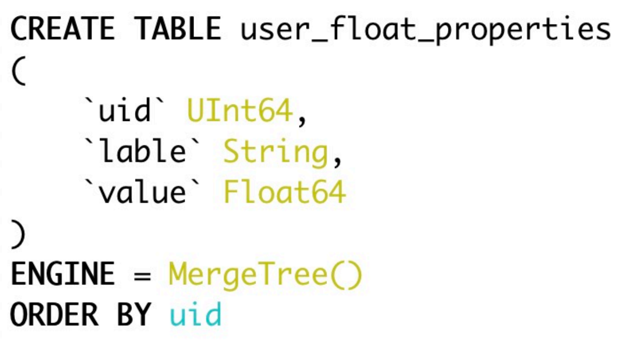


```
CREATE TABLE integer_lables
(
    `lable` String,
    `value` UInt64,
    `uv` AggregateFunction(groupBitmap, UInt64)
)
ENGINE = AggregatingMergeTree()
PARTITION BY lable
ORDER BY value
```

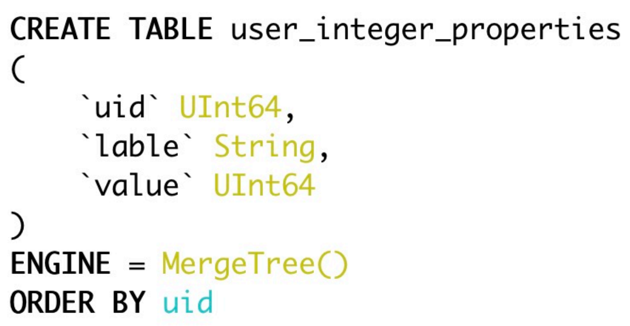
三、 常见ClickHouse实时分析场景举例

场景4： Aggregate Function 应用举例（bitmap系列）案例2：广告投放/用户画像应用

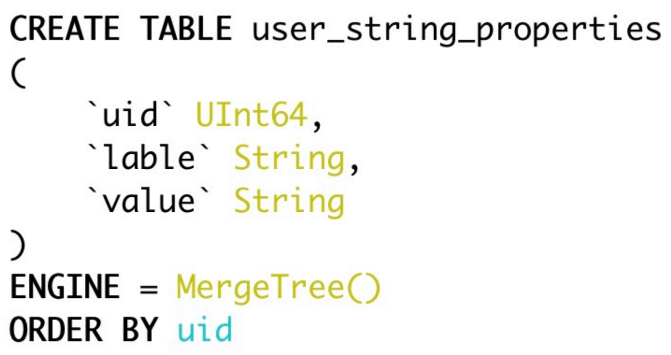
2 创建用户属性表



```
CREATE TABLE user_float_properties
(
    `uid` UInt64,
    `lable` String,
    `value` Float64
)
ENGINE = MergeTree()
ORDER BY uid
```



```
CREATE TABLE user_integer_properties
(
    `uid` UInt64,
    `lable` String,
    `value` UInt64
)
ENGINE = MergeTree()
ORDER BY uid
```



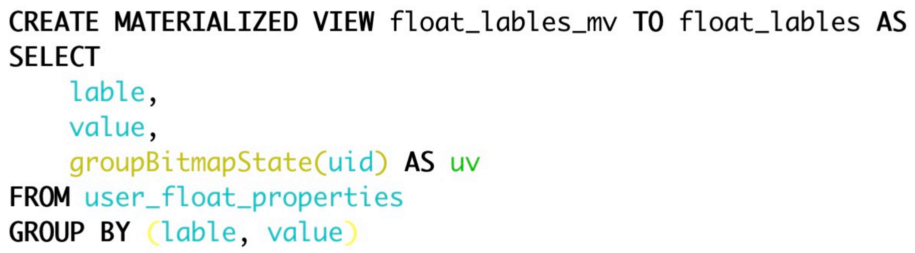
```
CREATE TABLE user_string_properties
(
    `uid` UInt64,
    `lable` String,
    `value` String
)
ENGINE = MergeTree()
ORDER BY uid
```

三、 常见ClickHouse实时分析场景举例

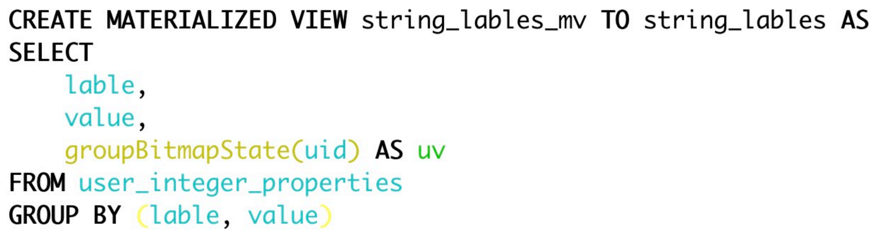
腾讯云 | 云+社区 intel

场景4： Aggregate Function 应用举例（bitmap系列）案例2：广告投放/用户画像应用

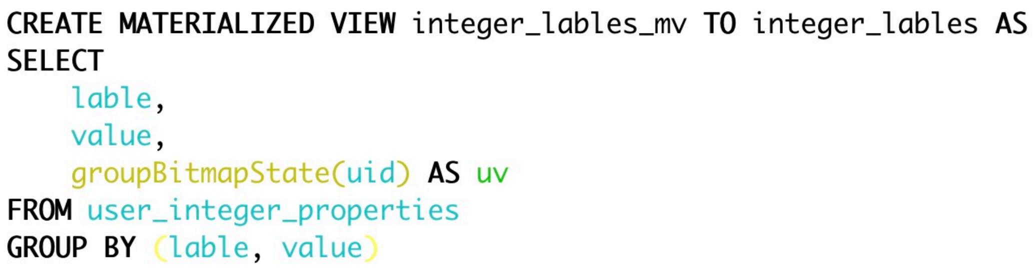
3 创建物化视图，自动聚合数据



```
CREATE MATERIALIZED VIEW float_lables_mv TO float_lables AS
SELECT
    lable,
    value,
    groupBitmapState(uid) AS uv
FROM user_float_properties
GROUP BY (lable, value)
```



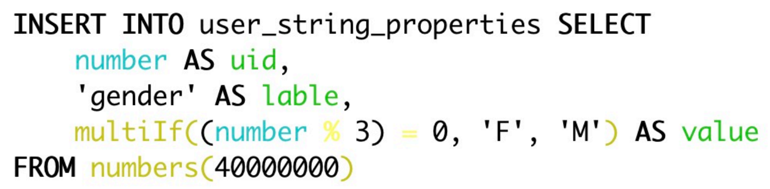
```
CREATE MATERIALIZED VIEW string_lables_mv TO string_lables AS
SELECT
    lable,
    value,
    groupBitmapState(uid) AS uv
FROM user_integer_properties
GROUP BY (lable, value)
```



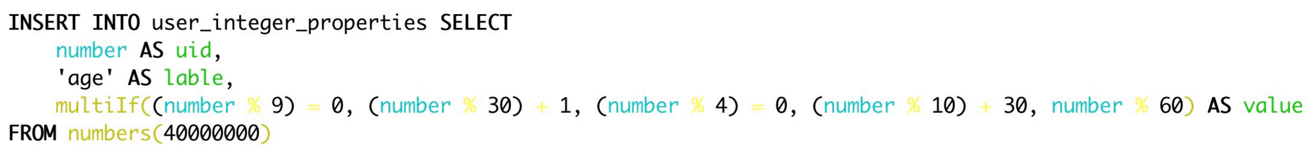
```
CREATE MATERIALIZED VIEW integer_lables_mv TO integer_lables AS
SELECT
    lable,
    value,
    groupBitmapState(uid) AS uv
FROM user_integer_properties
GROUP BY (lable, value)
```

场景4： Aggregate Function 应用举例（bitmap系列）案例2：广告投放/用户画像应用

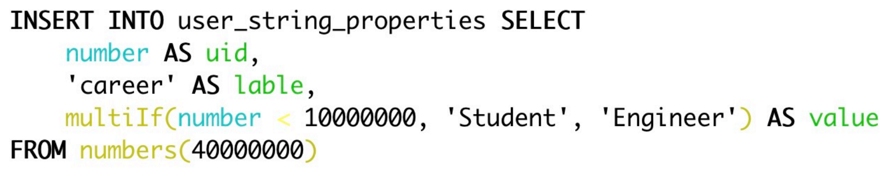
4 插入模拟数据， 4千万条



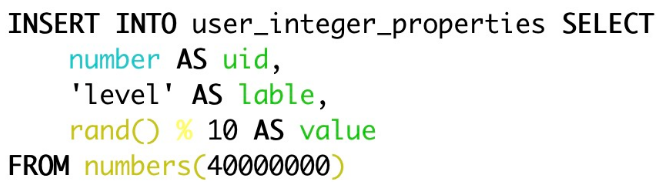
```sql
INSERT INTO user_string_properties SELECT
    number AS uid,
    'gender' AS lable,
    multiIf((number % 3) = 0, 'F', 'M') AS value
FROM numbers(40000000)
```



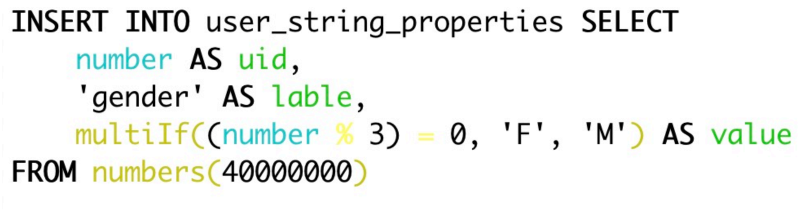
```sql
INSERT INTO user_integer_properties SELECT
    number AS uid,
    'age' AS lable,
    multiIf((number % 9) = 0, (number % 30) + 1, (number % 4) = 0, (number % 10) + 30, number % 60) AS value
FROM numbers(40000000)
```



```sql
INSERT INTO user_string_properties SELECT
    number AS uid,
    'career' AS lable,
    multiIf(number < 10000000, 'Student', 'Engineer') AS value
FROM numbers(40000000)
```



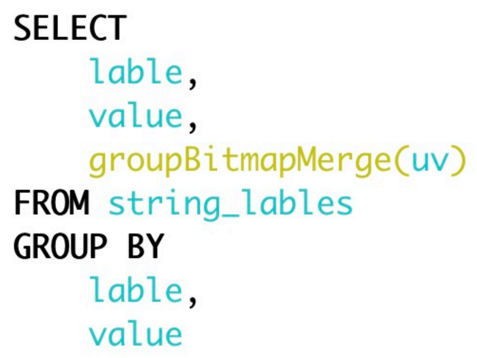
```sql
INSERT INTO user_integer_properties SELECT
    number AS uid,
    'level' AS lable,
    rand() % 10 AS value
FROM numbers(40000000)
```



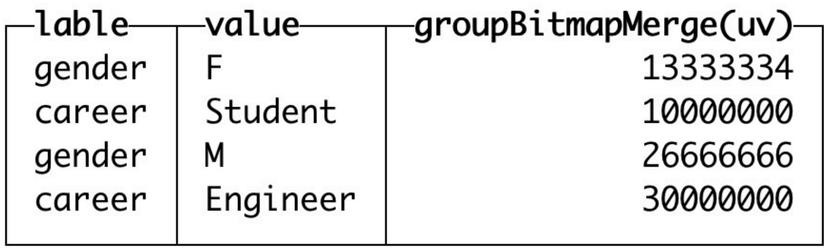
```sql
INSERT INTO user_string_properties SELECT
    number AS uid,
    'gender' AS lable,
    multiIf((number % 3) = 0, 'F', 'M') AS value
FROM numbers(40000000)
```

场景4： Aggregate Function 应用举例（bitmap系列）案例2：广告投放/用户画像应用

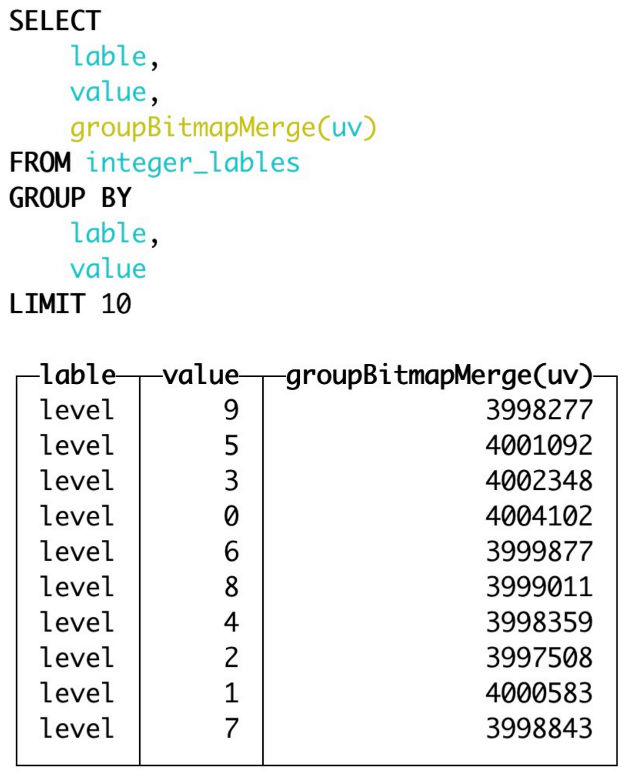
5 观察标签数据



```
SELECT
    lable,
    value,
    groupBitmapMerge(uv)
FROM string_lables
GROUP BY
    lable,
    value
```



```
┌─lable──┬─value────┬─groupBitmapMerge(uv)─┐
│ gender │ F        │             13333334 │
│ career │ Student  │             10000000 │
│ gender │ M        │             26666666 │
│ career │ Engineer │             30000000 │
└────────┴──────────┴──────────────────────┘
```



```
SELECT
    lable,
    value,
    groupBitmapMerge(uv)
FROM integer_lables
GROUP BY
    lable,
    value
LIMIT 10

┌─lable─┬─value─┬─groupBitmapMerge(uv)─┐
│ level │     9 │              3998277 │
│ level │     5 │              4001092 │
│ level │     3 │              4002348 │
│ level │     0 │              4004102 │
│ level │     6 │              3999877 │
│ level │     8 │              3999011 │
│ level │     4 │              3998359 │
│ level │     2 │              3997508 │
│ level │     1 │              4000583 │
│ level │     7 │              3998843 │
└───────┴───────┴──────────────────────┘
```

三、 常见ClickHouse实时分析场景举例

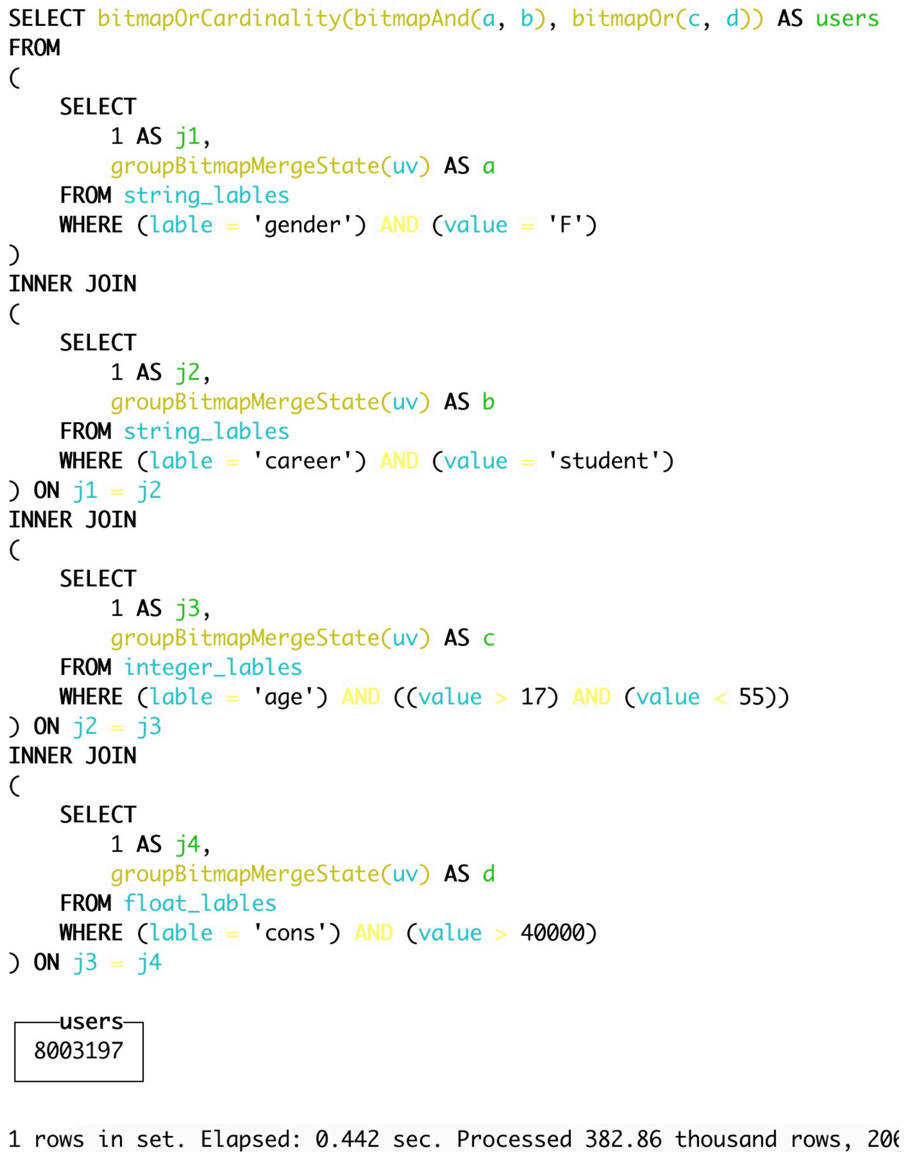
6 查询，按条件圈用户

条件：

（性别 = 女 AND 职业 = 工程师）

OR

(17 < 年龄 < 55 OR 消费 > 40000)



```
SELECT bitmapOrCardinality(bitmapAnd(a, b), bitmapOr(c, d)) AS users
FROM
(
    SELECT
        1 AS j1,
        groupBitmapMergeState(uv) AS a
    FROM string_lables
    WHERE (lable = 'gender') AND (value = 'F')
)
INNER JOIN
(
    SELECT
        1 AS j2,
        groupBitmapMergeState(uv) AS b
    FROM string_lables
    WHERE (lable = 'career') AND (value = 'student')
) ON j1 = j2
INNER JOIN
(
    SELECT
        1 AS j3,
        groupBitmapMergeState(uv) AS c
    FROM integer_lables
    WHERE (lable = 'age') AND ((value > 17) AND (value < 55))
) ON j2 = j3
INNER JOIN
(
    SELECT
        1 AS j4,
        groupBitmapMergeState(uv) AS d
    FROM float_lables
    WHERE (lable = 'cons') AND (value > 40000)
) ON j3 = j4

┌─users─┐
│ 8003197 │
└─────────┘

1 rows in set. Elapsed: 0.442 sec. Processed 382.86 thousand rows, 20(
```

目录

一、 腾讯云ClickHouse在QQ音乐实践

二、 常见ClickHouse实时分析场景

三、 腾讯云ClickHouse性能调优

四、 腾讯云ClickHouse现状与规划

三、 腾讯云ClickHouse性能调优

MergeTree engines

1. INSERT操作是原子的

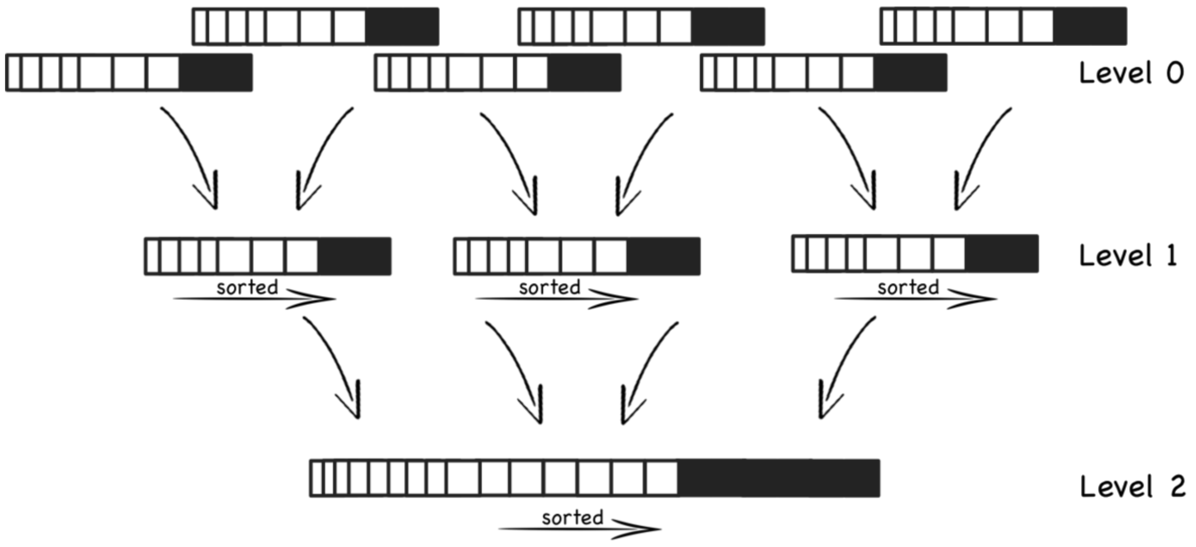
2. SELECT操作异常快

3. 支持主索引/二级索引

4. INSERT/SELECT相互不影响

5. 后台合并数据

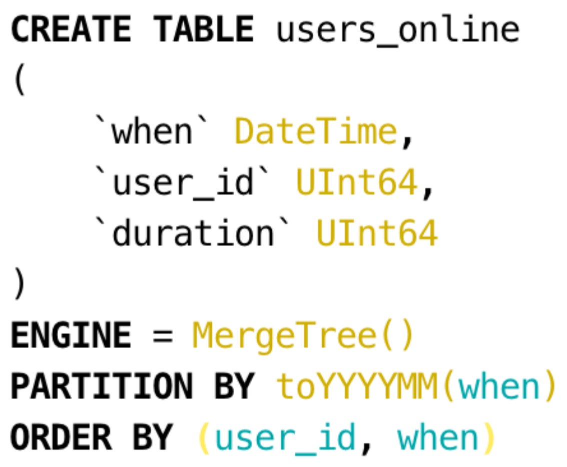
6. 主键不唯一



Compaction continues creating fewer, larger and larger files

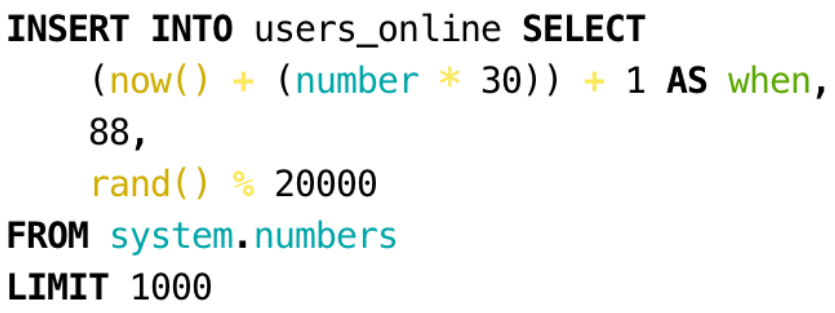
三、 腾讯云ClickHouse性能调优

1 CREATE



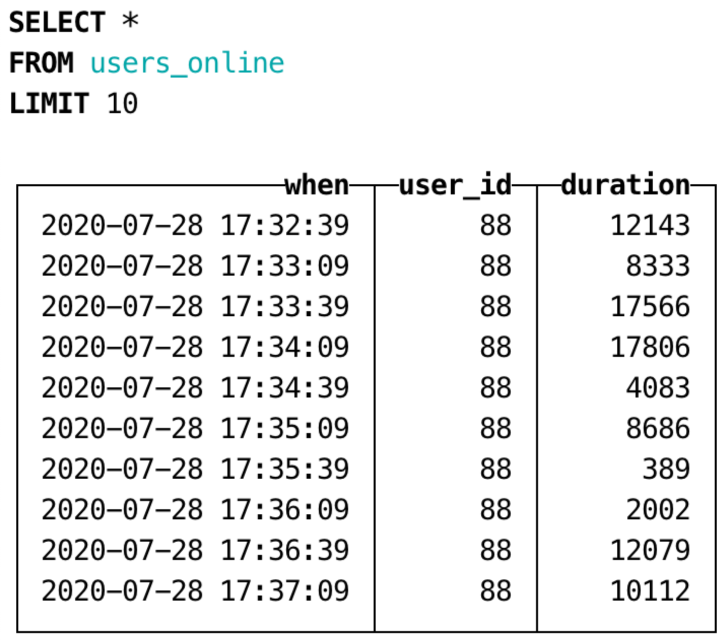
```
CREATE TABLE users_online
(
    `when` DateTime,
    `user_id` UInt64,
    `duration` UInt64
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(when)
ORDER BY (user_id, when)
```

2 INSERT



```
INSERT INTO users_online SELECT
    (now() + (number * 30)) + 1 AS when,
    88,
    rand() % 20000
FROM system.numbers
LIMIT 1000
```

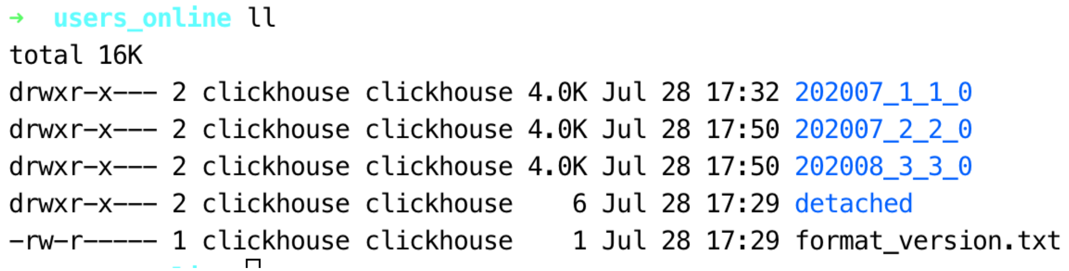
3 SELECT



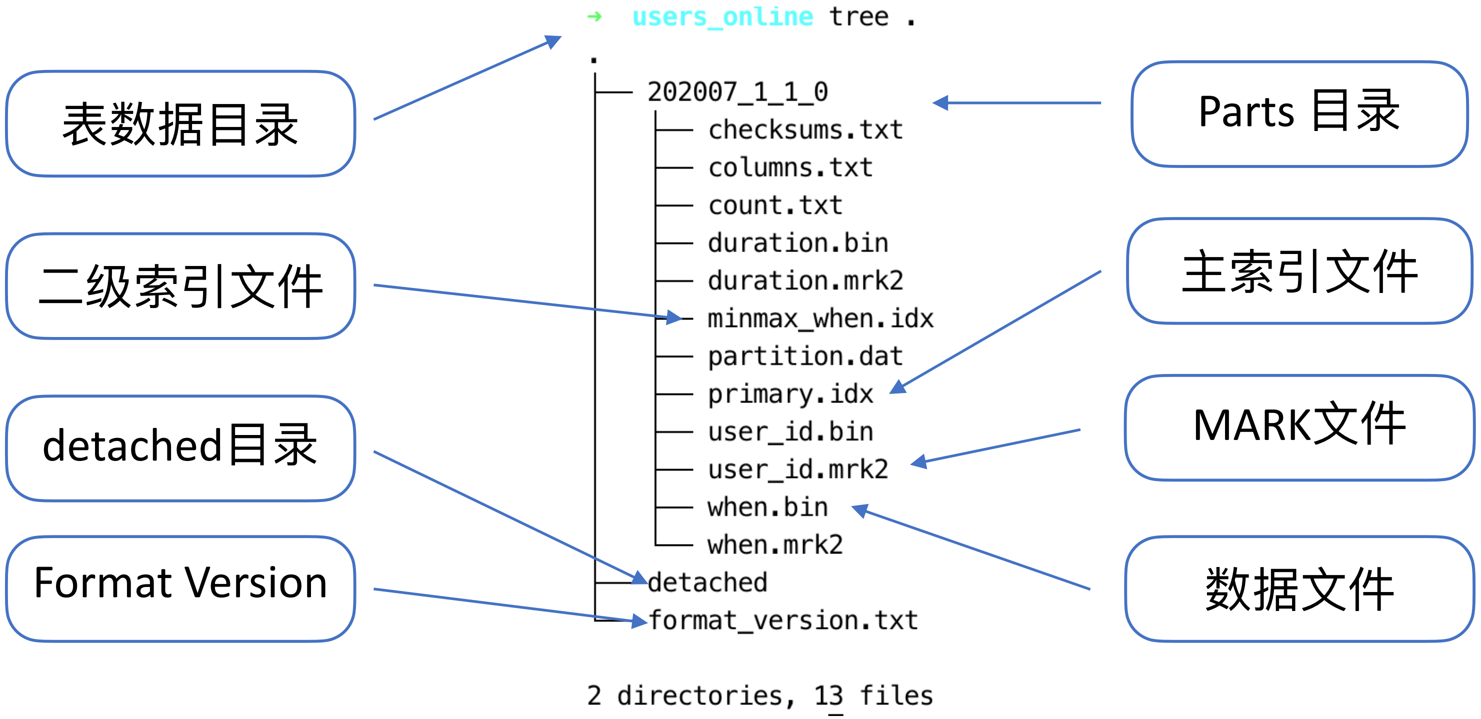
```
SELECT *
FROM users_online
LIMIT 10
```

```
┌────────────────when─┬─user_id─┬─duration─┐
│ 2020-07-28 17:32:39 │      88 │    12143 │
│ 2020-07-28 17:33:09 │      88 │     8333 │
│ 2020-07-28 17:33:39 │      88 │    17566 │
│ 2020-07-28 17:34:09 │      88 │    17806 │
│ 2020-07-28 17:34:39 │      88 │     4083 │
│ 2020-07-28 17:35:09 │      88 │     8686 │
│ 2020-07-28 17:35:39 │      88 │      389 │
│ 2020-07-28 17:36:09 │      88 │     2002 │
│ 2020-07-28 17:36:39 │      88 │    12079 │
│ 2020-07-28 17:37:09 │      88 │    10112 │
└─────────────────────┴─────────┴──────────┘
```

数据目录



```
➜  users_online ll
total 16K
drwxr-x--- 2 clickhouse clickhouse 4.0K Jul 28 17:32 202007_1_1_0
drwxr-x--- 2 clickhouse clickhouse 4.0K Jul 28 17:50 202007_2_2_0
drwxr-x--- 2 clickhouse clickhouse 4.0K Jul 28 17:50 202008_3_3_0
drwxr-x--- 2 clickhouse clickhouse    6 Jul 28 17:29 detached
-rw-r----- 1 clickhouse clickhouse    1 Jul 28 17:29 format_version.txt
```



```
➜  users_online tree .
.
├── 202007_1_1_0
│   ├── checksums.txt
│   ├── columns.txt
│   ├── count.txt
│   ├── duration.bin
│   ├── duration.mrk2
│   ├── minmax_when.idx
│   ├── partition.dat
│   ├── primary.idx
│   ├── user_id.bin
│   ├── user_id.mrk2
│   ├── when.bin
│   └── when.mrk2
├── detached
└── format_version.txt

2 directories, 13 files
```

表数据目录

二级索引文件

detached目录

Format Version

Parts 目录

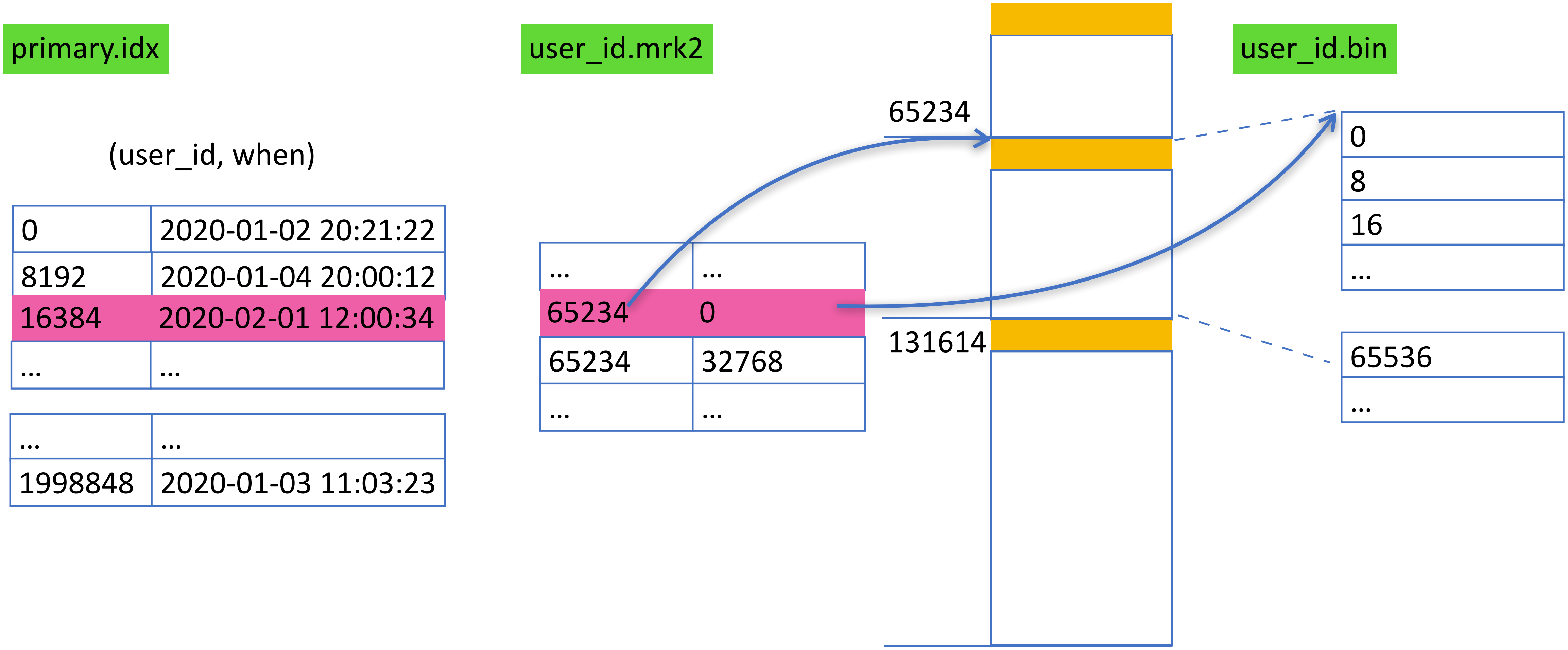
主索引文件

MARK文件

数据文件

三、 腾讯云ClickHouse性能调优

数据索引



primary.idx

(user_id, when)

0	2020-01-02 20:21:22
8192	2020-01-04 20:00:12
16384	2020-02-01 12:00:34
…	…

…	…
1998848	2020-01-03 11:03:23

user_id.mrk2

…	…
65234	0
65234	32768
…	…

user_id.bin

0
8
16
…

65536
…

配置选型

数据节点

CPU： 主频越高越好，通常也建议配置32cores以上的机型

内存：当然内存越大越好，PageCache加速作用越明显。通常建议配置128GB以上内存

磁盘：HDD盘即可，RAID-10，RAID-5，RAID-6或RAID-50。如果查询数据量大，邀请延迟低，上SSD/NVME

ZK节点

ZK节点的负载与集群的数据量成正比。当数据规模在TB级别时，建议为ZK节点选择SSD盘。

基础参数

max_threads: 查询使用的线程数量，默认为核数一半

max_memory_usage: 单次查询允许使用的内存量

max_memory_usage_for_all_users： clickhouse进程允许使用的内存量, 通常需要考虑为OS预留内存

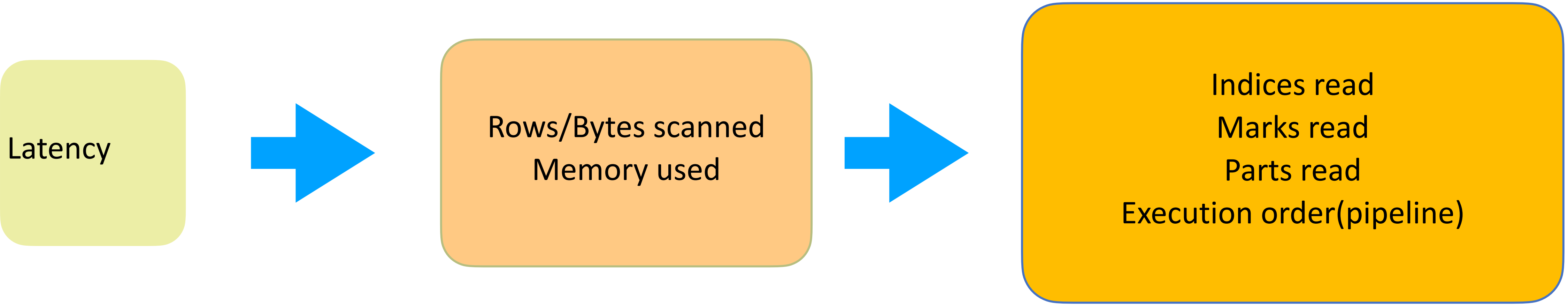
max_bytes_before_external_group_by: GROUP BY 操作使用内存超过该阈值后, 数据会写入磁盘，建议设置为max_memory_usage/2

max_concurrent_queries: 最大并发数限制

max_bytes_before_external_sort: order by 排序溢写磁盘阈值

background_pool_size: 后台线程组

查询优化



Latency

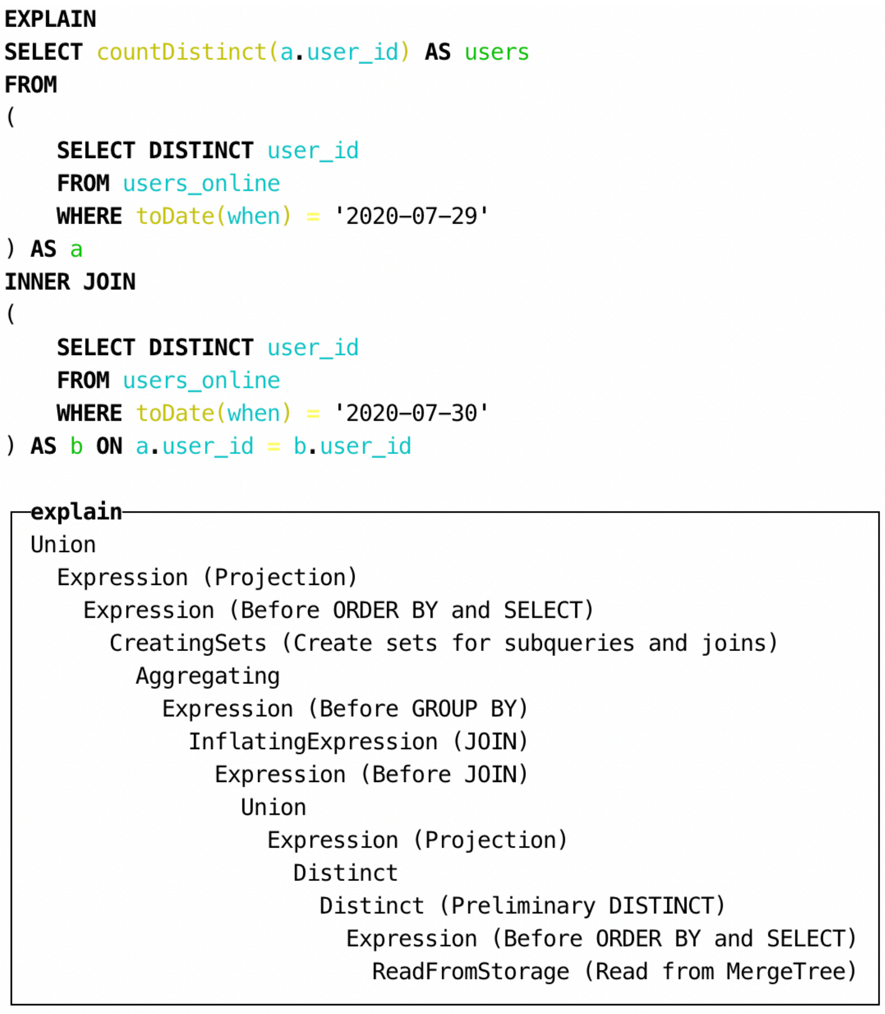
Rows/Bytes scanned
Memory used

Indices read
Marks read
Parts read
Execution order(pipeline)

查询优化

EXPLAIN [AST | SYNTAX | PLAN | PIPELINE] [settings = value, ...,] SELECT ... [FORMAT ...]

Let’s EXPLAIN.



```
EXPLAIN
SELECT countDistinct(a.user_id) AS users
FROM
(
    SELECT DISTINCT user_id
    FROM users_online
    WHERE toDate(when) = '2020-07-29'
) AS a
INNER JOIN
(
    SELECT DISTINCT user_id
    FROM users_online
    WHERE toDate(when) = '2020-07-30'
) AS b ON a.user_id = b.user_id
```

```
┌─explain──────────────────────────────────────────────────────────────┐
│ Union                                                                │
│   Expression (Projection)                                            │
│     Expression (Before ORDER BY and SELECT)                          │
│       CreatingSets (Create sets for subqueries and joins)            │
│         Aggregating                                                  │
│           Expression (Before GROUP BY)                               │
│             InflatingExpression (JOIN)                               │
│               Expression (Before JOIN)                               │
│                 Union                                                │
│                   Expression (Projection)                            │
│                     Distinct                                         │
│                       Distinct (Preliminary DISTINCT)                │
│                         Expression (Before ORDER BY and SELECT)      │
│                           ReadFromStorage (Read from MergeTree)      │
└──────────────────────────────────────────────────────────────────────┘
```

目录

一、 腾讯云ClickHouse在QQ音乐实践

二、 常见ClickHouse实时分析场景

三、 腾讯云ClickHouse性能调优

四、 腾讯云ClickHouse现状与规划

五、 腾讯云ClickHouse现状与规划

1

分钟级构建

- 10分钟构建上百节点大数据集群
- 支持控制台/程序API灵活构建

2

极致弹性

- 集群横向扩展
- 数据均衡 服务

3

极致性能

- 组件深度优化，与物理机构建性能接近
- PB至EB级COS数据高速分析

4

云端运维基础设施保障

- 百余监控指标覆盖(服务器级、服务级)
- 异常事件秒级触达
- Ddos/VPC安全加固

5

云端数据服务无缝连接

- 多源数据支持(云数据库、Ckafka)
- 云端可视化BI工具无缝对接

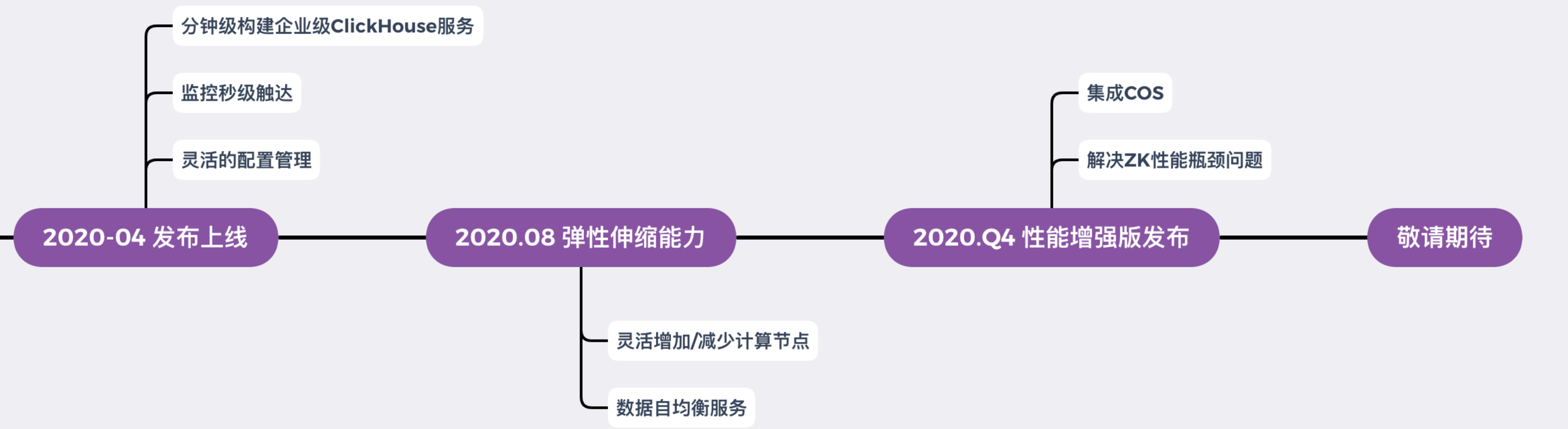
6

持续性技术支撑

- 云厂商雄厚技术支撑
- 线上技术交流(论坛、视频、指南)
- 线下技术沙龙

腾讯云EMR-ClickHouse 提供了集群快速部署，监控运维等企业级功能。方便用户快速构建云上海量数据分析平台。

五、 腾讯云ClickHouse现状与规划



2020-04 发布上线

- 分钟级构建企业级ClickHouse服务
- 监控秒级触达
- 灵活的配置管理

2020.08 弹性伸缩能力

- 灵活增加/减少计算节点
- 数据自均衡服务

2020.Q4 性能增强版发布

- 集成COS
- 解决ZK性能瓶颈问题

敬请期待

欢迎讨论ClickHouse相关问题 ~

pengjian.uestc@gmail.com