TCHouse-C Semi-structured Data and Real-time update

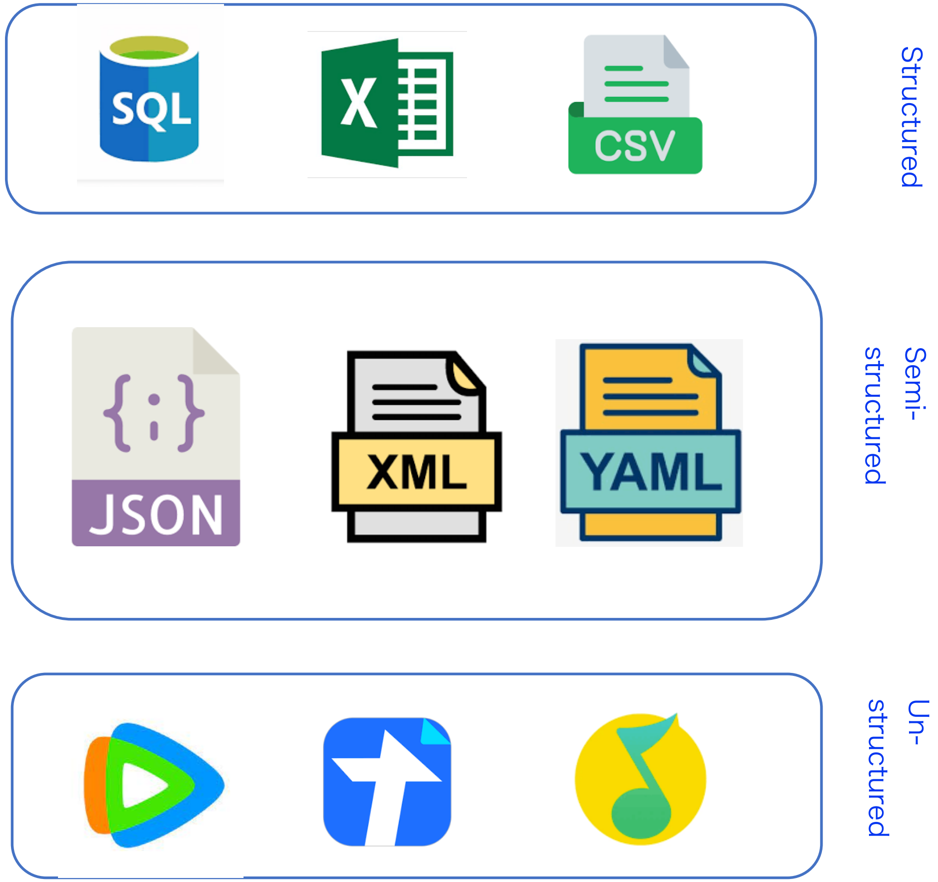
Peng Jian

2024-01-06

Semi-structured Data

Semi structured data

半结构化数据是一种数据格式，它介于结构化数据和非结构化数据之间。



来源广泛

- 互联网
- 物联网
- 社交媒体
- 移动应用
- APM等

价值潜力巨大

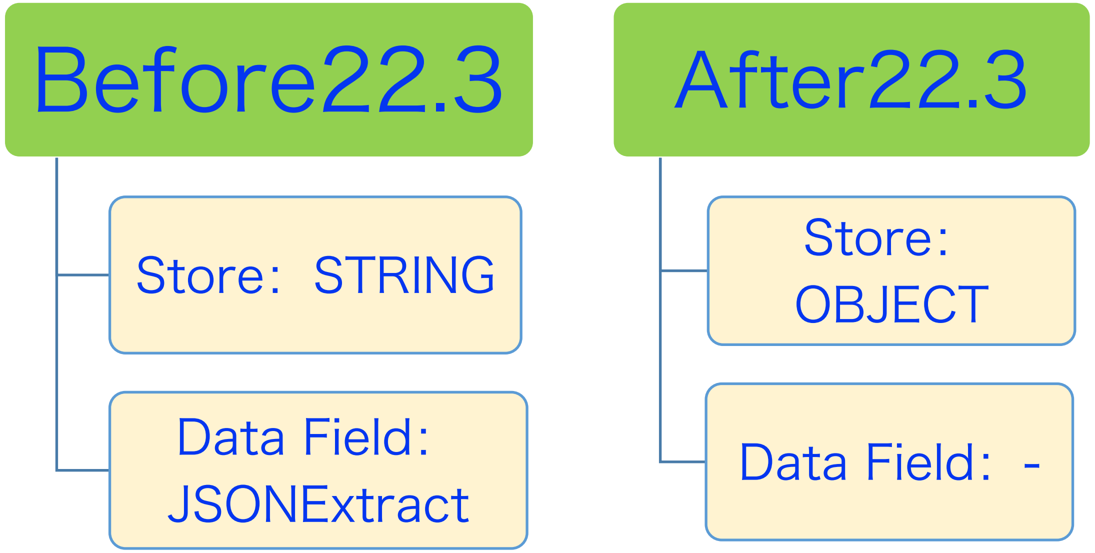
- 数据分析
- 数据挖掘
- 洞察
- 预测

应用场景广泛

- 灵活性好
- 多样性和不规则数据

ClickHouse Solution on Semi-structured Data

Solution



Before22.3

- Store: STRING
- Data Field: JSONExtract

After22.3

- Store: OBJECT
- Data Field: -

Advantage

Low Cost	High Performance	Easy Operation
Compression rate	IO for write	No dependency
performance	Query	stability

ClickHouse solution - advantage

Customer successful cases - using ClickHouse to take place ES

	Use cases	performance	Cost
小红书	Log/APM	SELECT is 20 X ES	50% of ES
B站	Log	Write10X, Select 2X P90<1s	30% of ES
携程	Log	P99<3s	42% of ES
UBER	Log	5X	-

注：据公开资料看，京东、唯品会、快手等知名的公司在使用ClickHouse处理半结构化数据。由于未披露性能和成本数据，没有收录在表中。

B站: https://cloud.tencent.com/developer/article/2143639
携程: https://www.51cto.com/article/744745.html
UBER:https://presentations.clickhouse.com/meetup40/uber.pdf

ClickHouse weak point

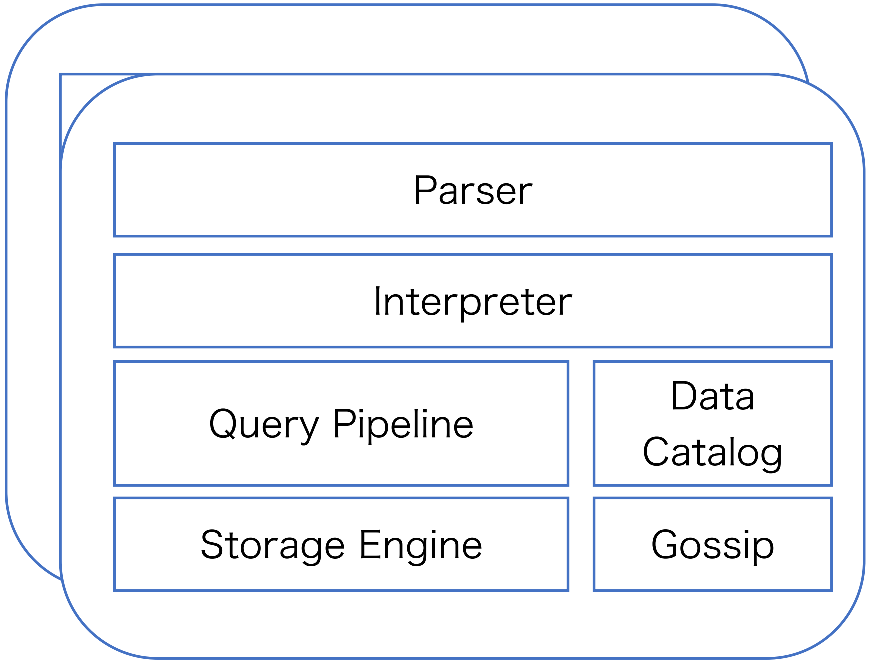
	存储	分析函数	不足
基于STRING方案	STRING	JSONExtract*	High CPU usage，low storage compression ratio，No SCHEMA
基于OBJECT方案	OBJECT	-	Not support secondary index neither Materializiced view no SCHEMA SYNC

TCHouse-C optimized solution：SCEHMA-LESS

整体方案

{JSON}

```
{
  "a":{
    "b":{
      "c":100
    },
    "e":"text"
  },
  "d": 20240106.1
}
```



a.b.c	a.e	d
100	‘text’	20240106.1

Before data injection：no need for table SCHEMA

Data injection ：accept semi-structured Data type

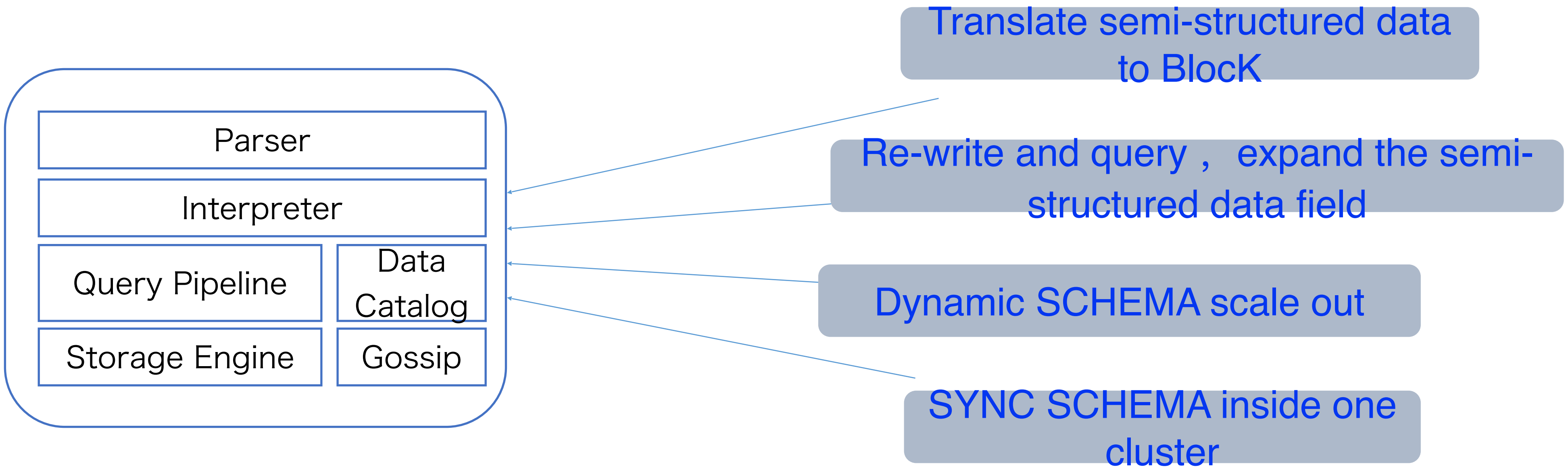
SQL：support query for prefix （a.*)

Store engine：dynamic SCHEMA

Meta data Snyc：fast SCHEMA SNYC in one cluster

TCHouse-C optimized solution： SCEHMA-LESS

Detail



Parser

Interpreter

Query Pipeline

Data Catalog

Storage Engine

Gossip

Translate semi-structured data to BlocK

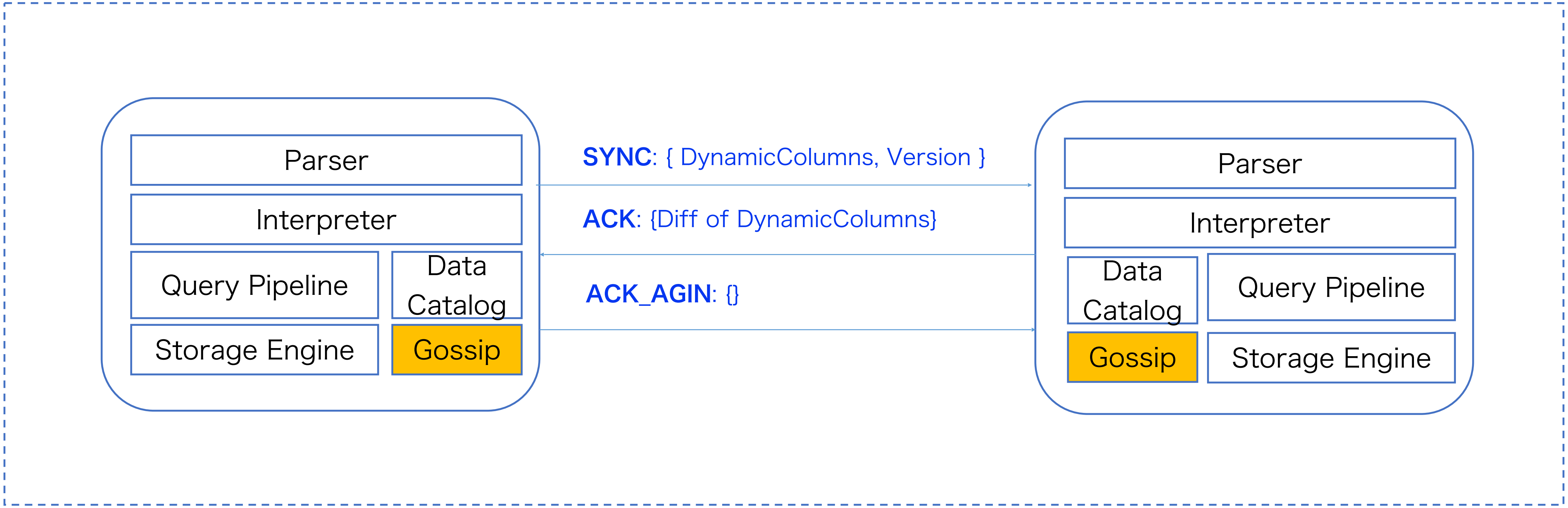
Re-write and query， expand the semi-structured data field

Dynamic SCHEMA scale out

SYNC SCHEMA inside one cluster

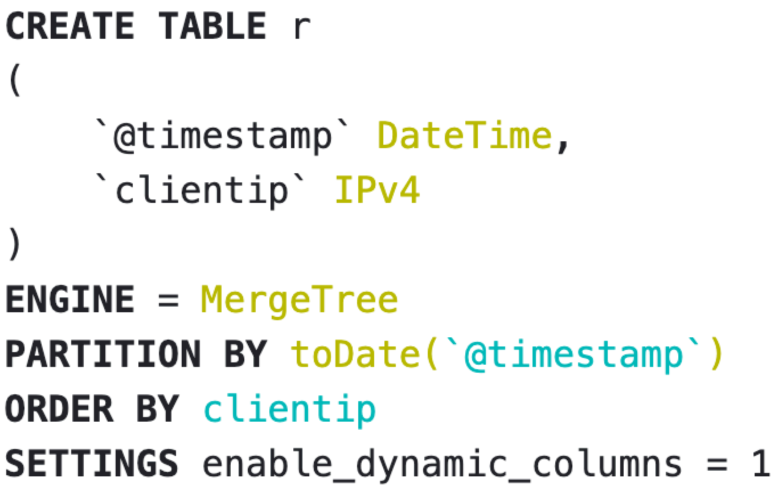
TCHouse-C optimized solution： SCEHMA-LESS

SCHEMA同步



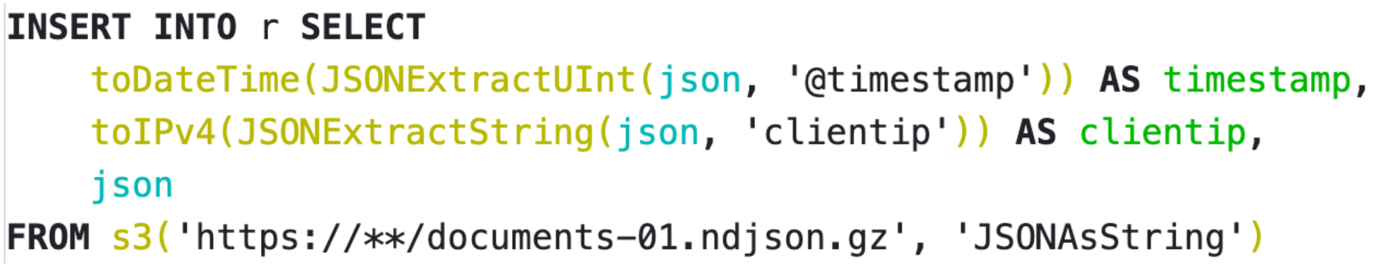
TCHouse-C处理半结构化数据新方案

1. 创建表



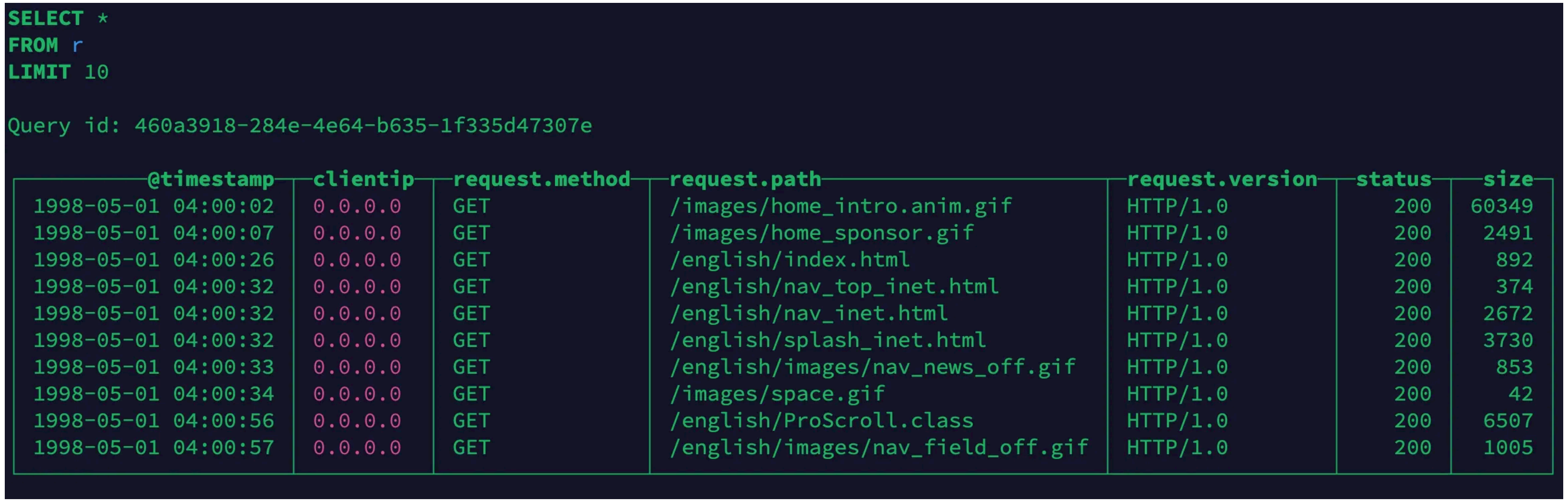
```
CREATE TABLE r
(
    `@timestamp` DateTime,
    `clientip` IPv4
)
ENGINE = MergeTree
PARTITION BY toDate(`@timestamp`)
ORDER BY clientip
SETTINGS enable_dynamic_columns = 1
```

2. 导入数据

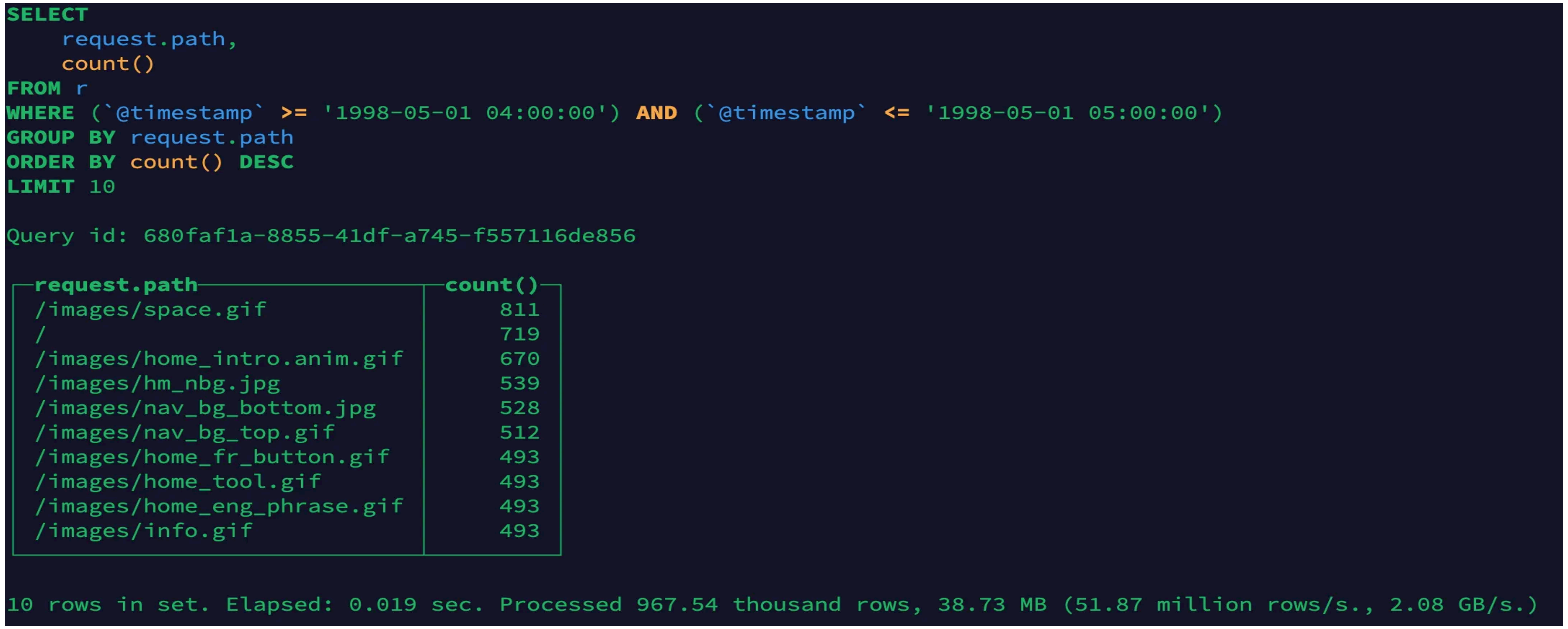


```
INSERT INTO r SELECT
    toDateTime(JSONExtractUInt(json, '@timestamp')) AS timestamp,
    toIPv4(JSONExtractString(json, 'clientip')) AS clientip,
    json
FROM s3('https://**/documents-01.ndjson.gz', 'JSONAsString')
```

TCHouse-C optimized solution



```
SELECT *
FROM r
LIMIT 10

Query id: 460a3918-284e-4e64-b635-1f335d47307e

┌──────────@timestamp─┬─clientip─┬─request.method─┬─request.path────────────────────────┬─request.version─┬─status─┬──size─┐
│ 1998-05-01 04:00:02 │ 0.0.0.0  │ GET            │ /images/home_intro.anim.gif         │ HTTP/1.0        │    200 │ 60349 │
│ 1998-05-01 04:00:07 │ 0.0.0.0  │ GET            │ /images/home_sponsor.gif            │ HTTP/1.0        │    200 │  2491 │
│ 1998-05-01 04:00:26 │ 0.0.0.0  │ GET            │ /english/index.html                 │ HTTP/1.0        │    200 │   892 │
│ 1998-05-01 04:00:32 │ 0.0.0.0  │ GET            │ /english/nav_top_inet.html          │ HTTP/1.0        │    200 │   374 │
│ 1998-05-01 04:00:32 │ 0.0.0.0  │ GET            │ /english/nav_inet.html              │ HTTP/1.0        │    200 │  2672 │
│ 1998-05-01 04:00:32 │ 0.0.0.0  │ GET            │ /english/splash_inet.html           │ HTTP/1.0        │    200 │  3730 │
│ 1998-05-01 04:00:33 │ 0.0.0.0  │ GET            │ /english/images/nav_news_off.gif    │ HTTP/1.0        │    200 │   853 │
│ 1998-05-01 04:00:34 │ 0.0.0.0  │ GET            │ /images/space.gif                   │ HTTP/1.0        │    200 │    42 │
│ 1998-05-01 04:00:56 │ 0.0.0.0  │ GET            │ /english/ProScroll.class            │ HTTP/1.0        │    200 │  6507 │
│ 1998-05-01 04:00:57 │ 0.0.0.0  │ GET            │ /english/images/nav_field_off.gif   │ HTTP/1.0        │    200 │  1005 │
└─────────────────────┴──────────┴────────────────┴─────────────────────────────────────┴─────────────────┴────────┴───────┘
```



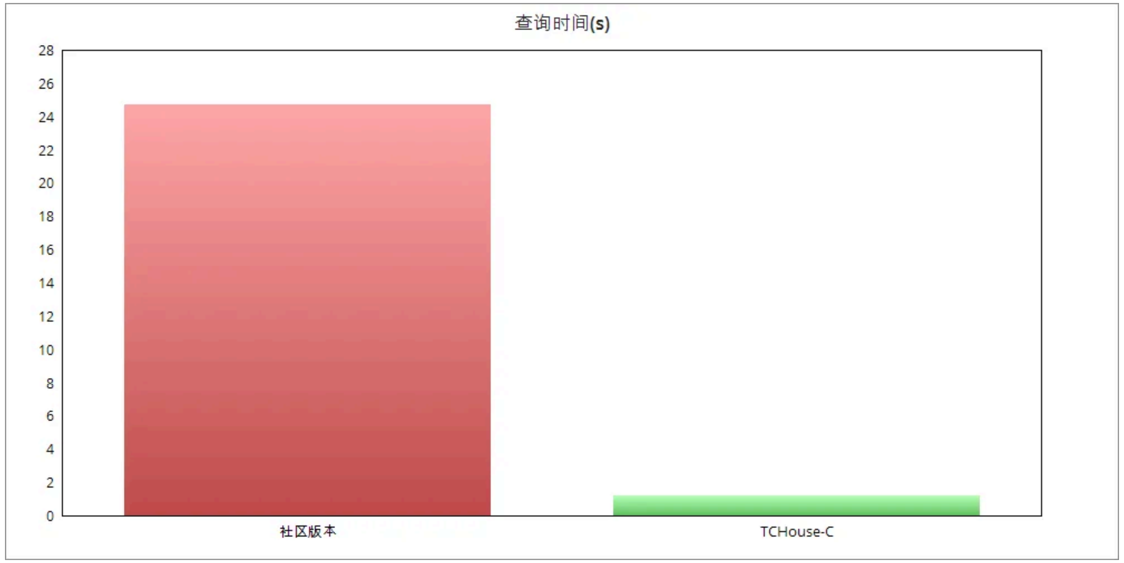
```
SELECT
    request.path,
    count()
FROM r
WHERE (`@timestamp` >= '1998-05-01 04:00:00') AND (`@timestamp` <= '1998-05-01 05:00:00')
GROUP BY request.path
ORDER BY count() DESC
LIMIT 10

Query id: 680faf1a-8855-41df-a745-f557116de856

┌─request.path──────────────┬─count()─┐
│ /images/space.gif         │     811 │
│ /                         │     719 │
│ /images/home_intro.anim.gif │   670 │
│ /images/hm_nbg.jpg        │     539 │
│ /images/nav_bg_bottom.jpg │     528 │
│ /images/nav_bg_top.gif    │     512 │
│ /images/home_fr_button.gif │    493 │
│ /images/home_tool.gif     │     493 │
│ /images/home_eng_phrase.gif │   493 │
│ /images/info.gif          │     493 │
└───────────────────────────┴─────────┘

10 rows in set. Elapsed: 0.019 sec. Processed 967.54 thousand rows, 38.73 MB (51.87 million rows/s., 2.08 GB/s.)
```

TCHouse-C

result:

1. Simplify the data injection process
2. Store by field，better compression
3. Query by field，better performance
4. Utilized secondary index、materialized view、PROJECTION
5. Data management：delete

Query improve 20X，low down cost to 50%



Real time update

TCHouse-C real time update

Performing high-frequency add, delete, and modify operations on data."

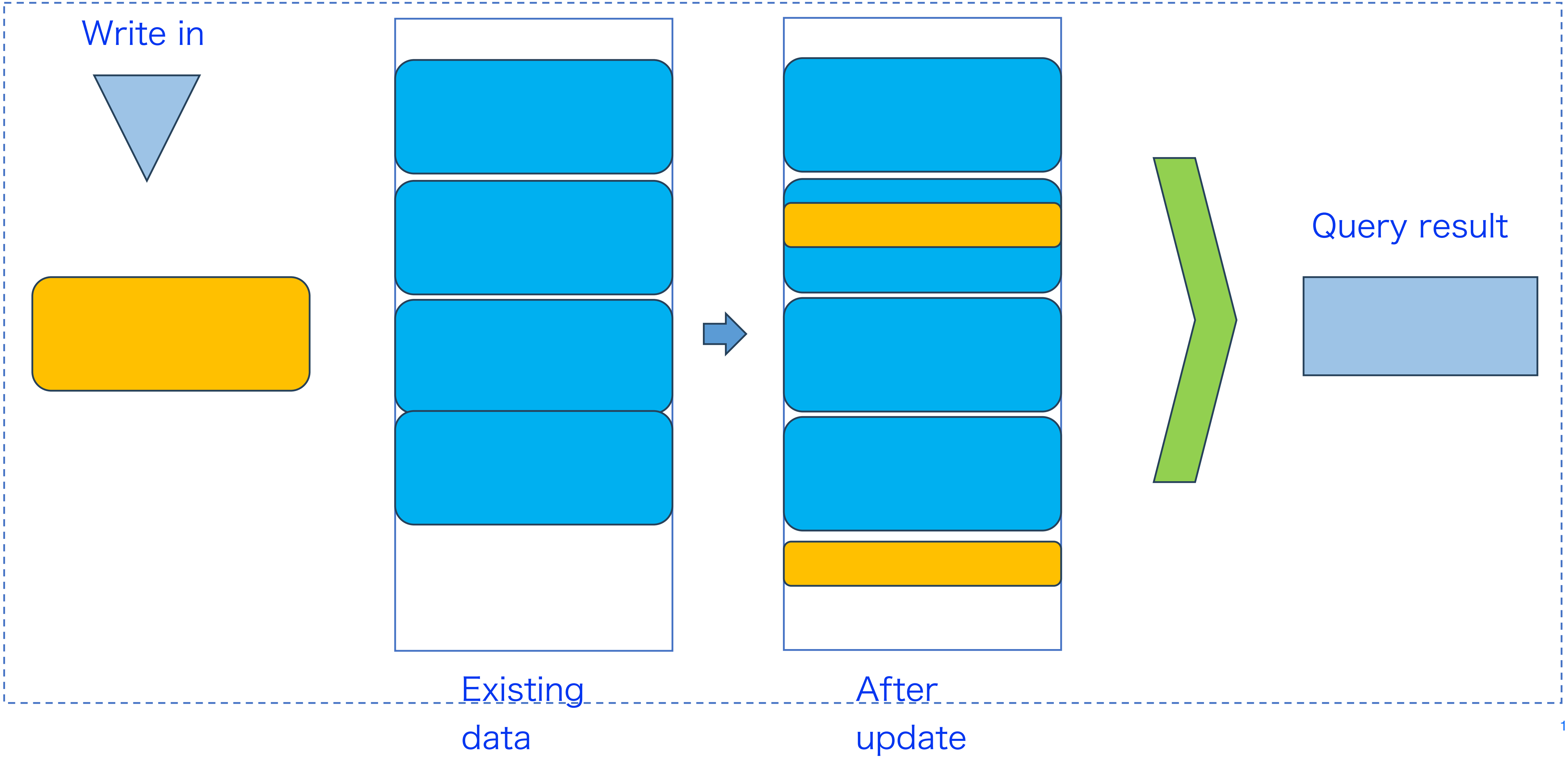
In real-time business analytics, it is necessary for the data warehouse to have the capability for real-time data updates, such as real-time dashboards, IoT device data, user behavior tracking, and e-commerce transaction scenarios. In these scenarios, there is a demand for frequent and low-latency real-time updates.

Building a large wide table using the ability to update partial columns

Typically, adopting a large wide table approach enhances multidimensional analytical capabilities. In community editions, businesses commonly utilize the Merge-On-Read solution, which is not user-friendly and has poor performance.
With the support of UPSERT functionality, leveraging the ability to update partial columns allows different upstream businesses to update columns relevant to their operations. This simplifies the data integration process

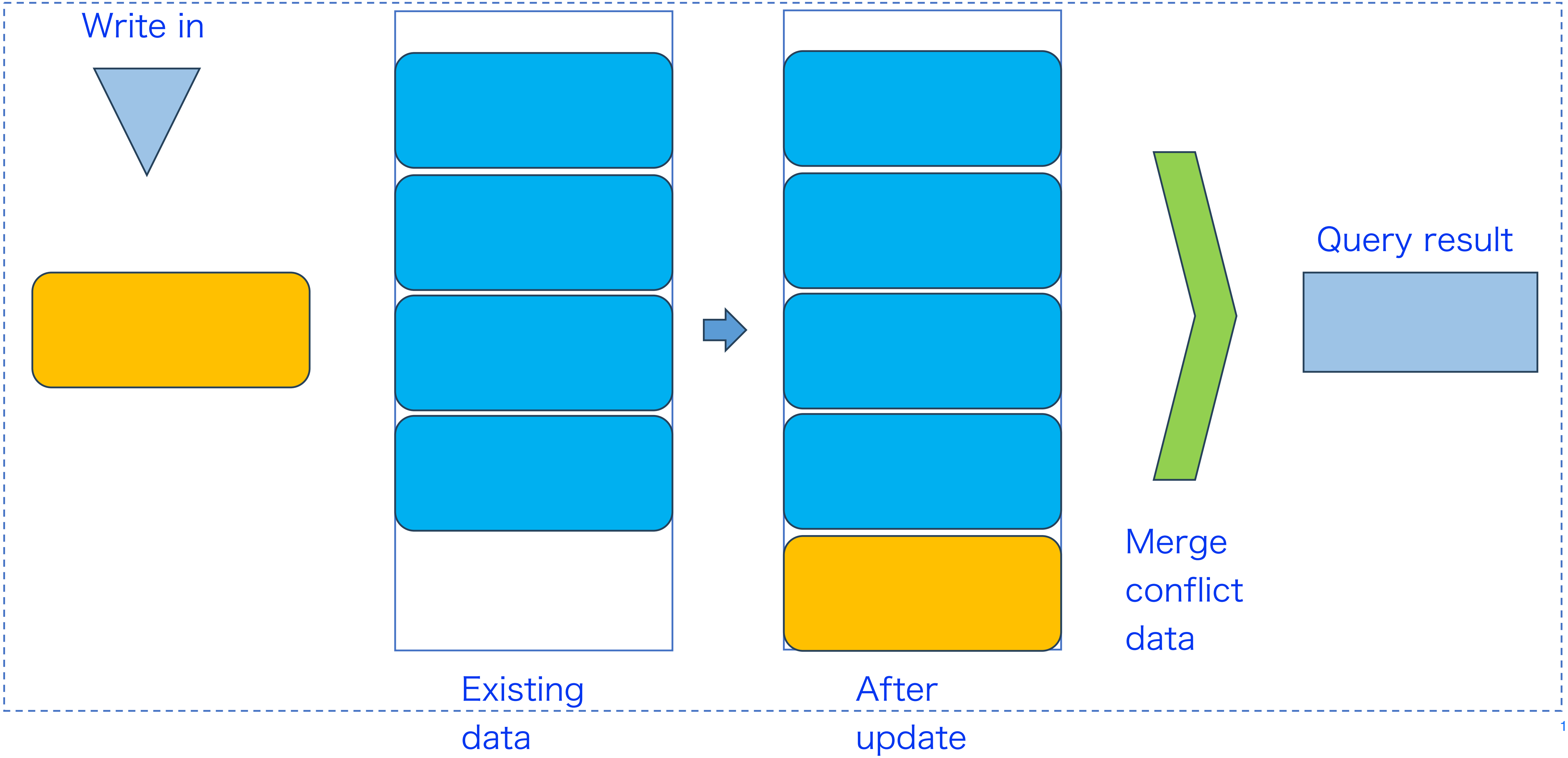
existing solution for the real time update

Copy-On-Write



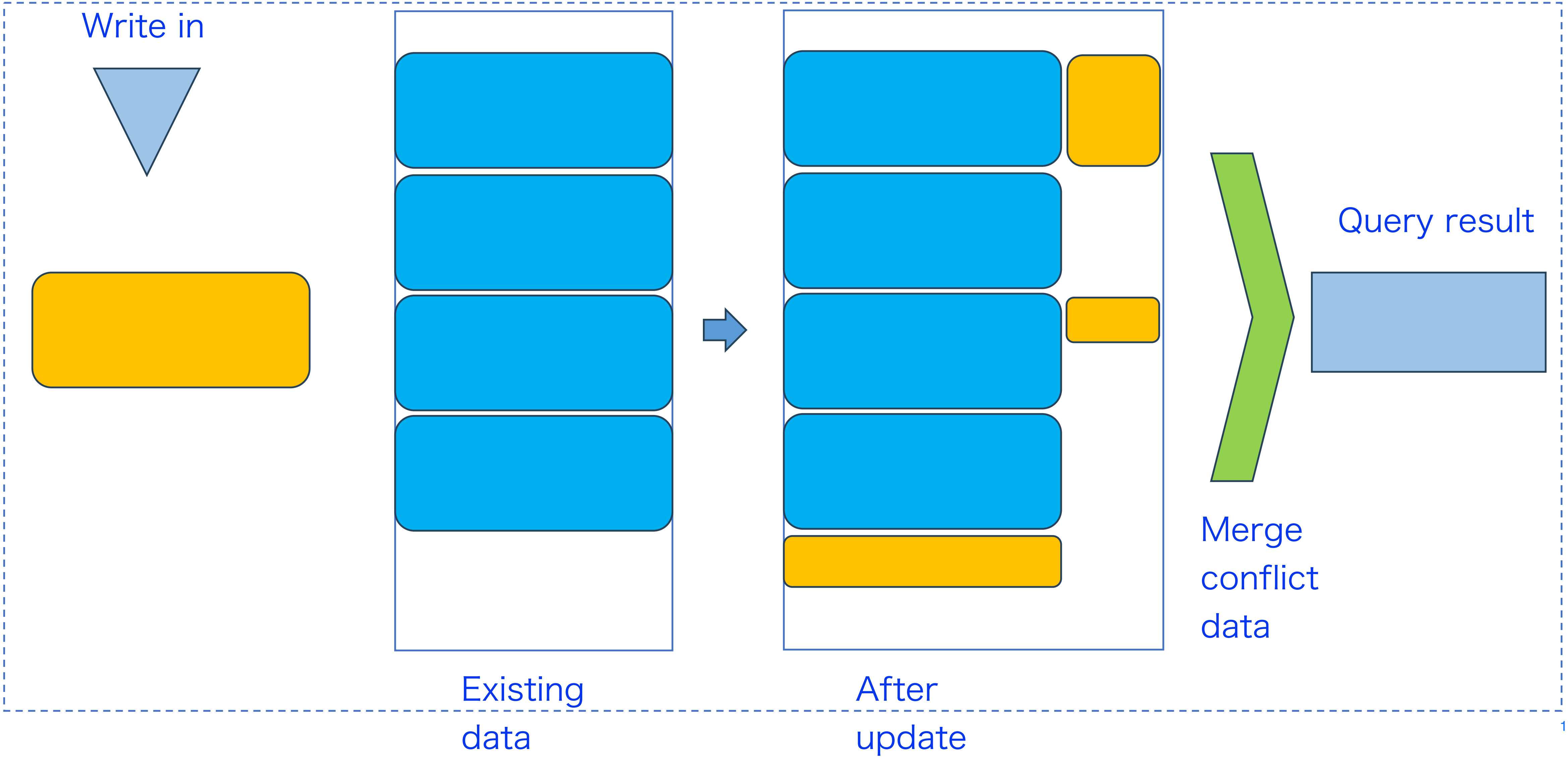
existing solution for the real time update

Merge-On-Read



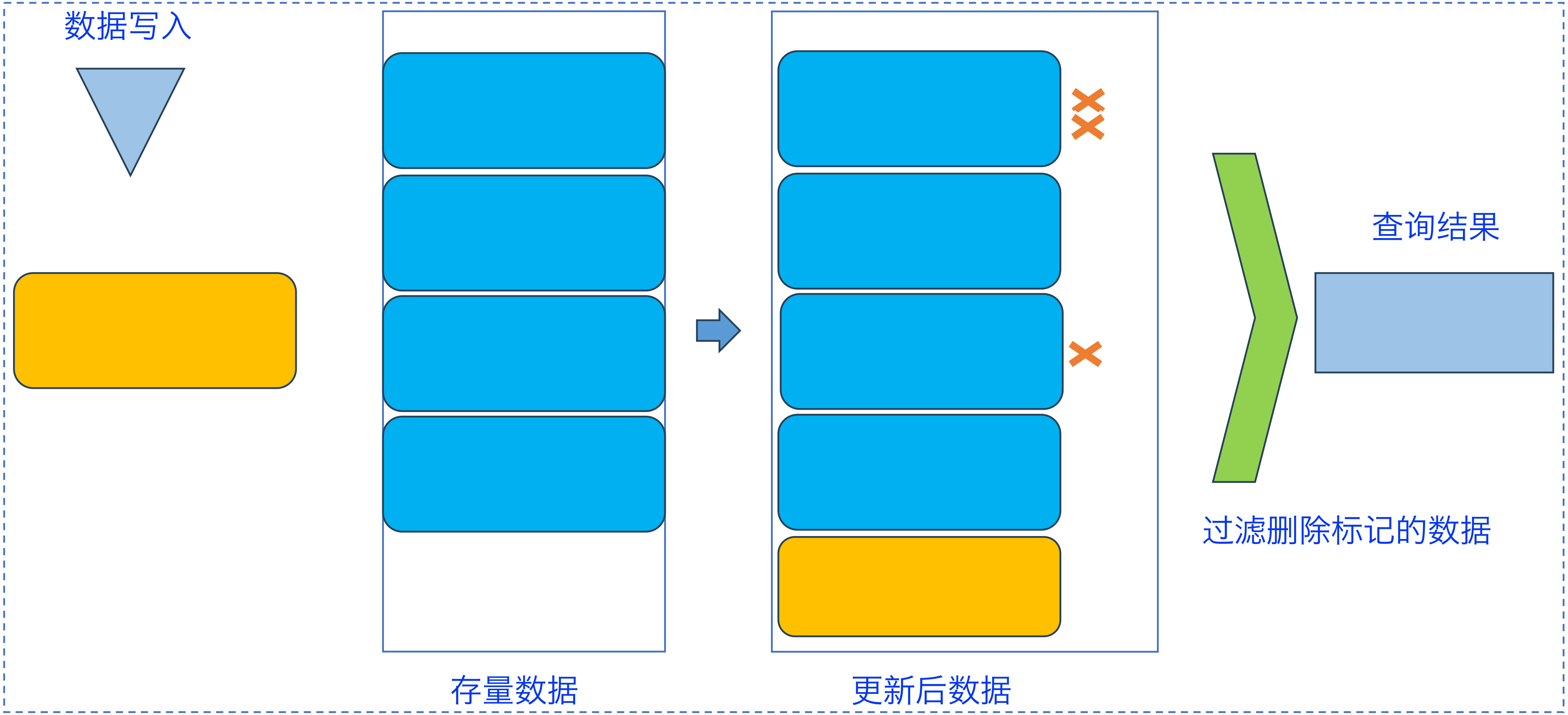
existing solution for the real time update

Delta-Store



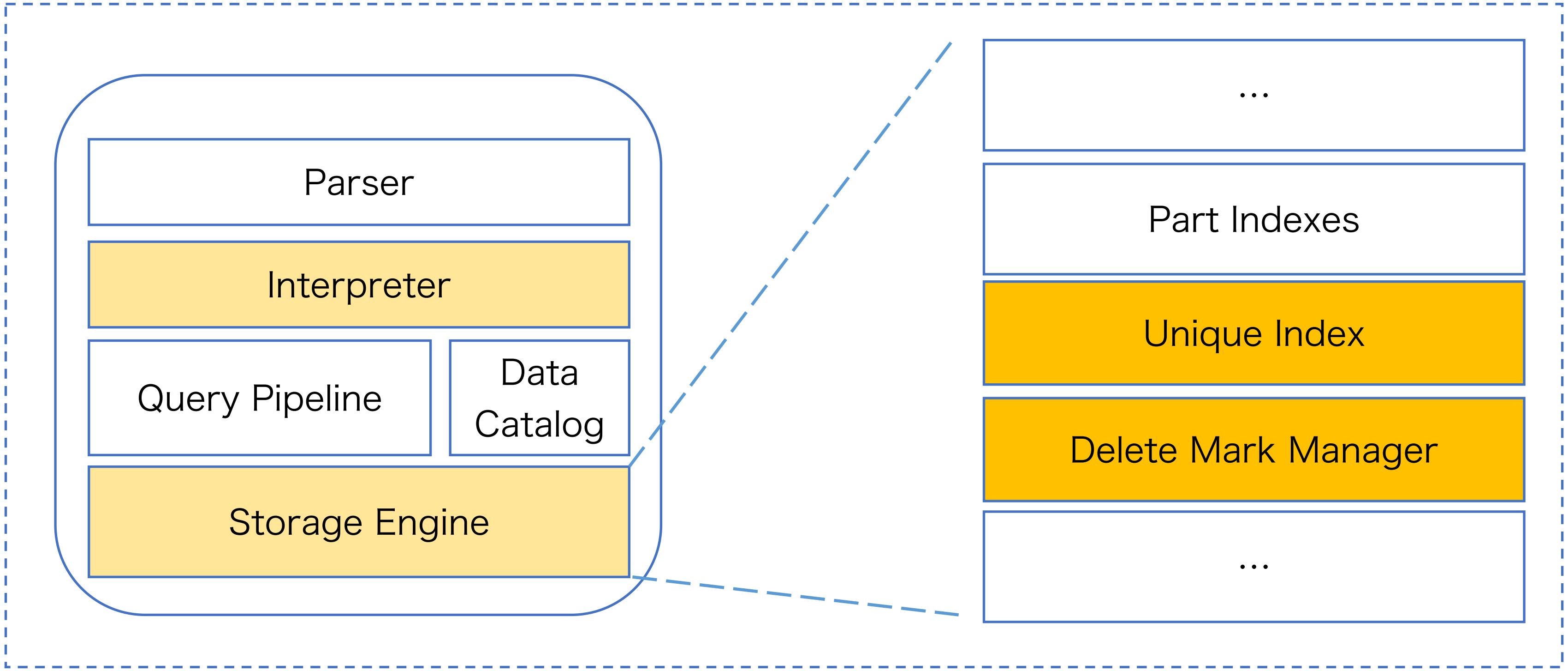
existing solution for the real time update

Delete-Insert



TCHouse-C solution for real time update: Delete-Insert

Overall solution



Interpreter: UPDATE/DELETE 对应的Pipeline构建

Storage Engine:

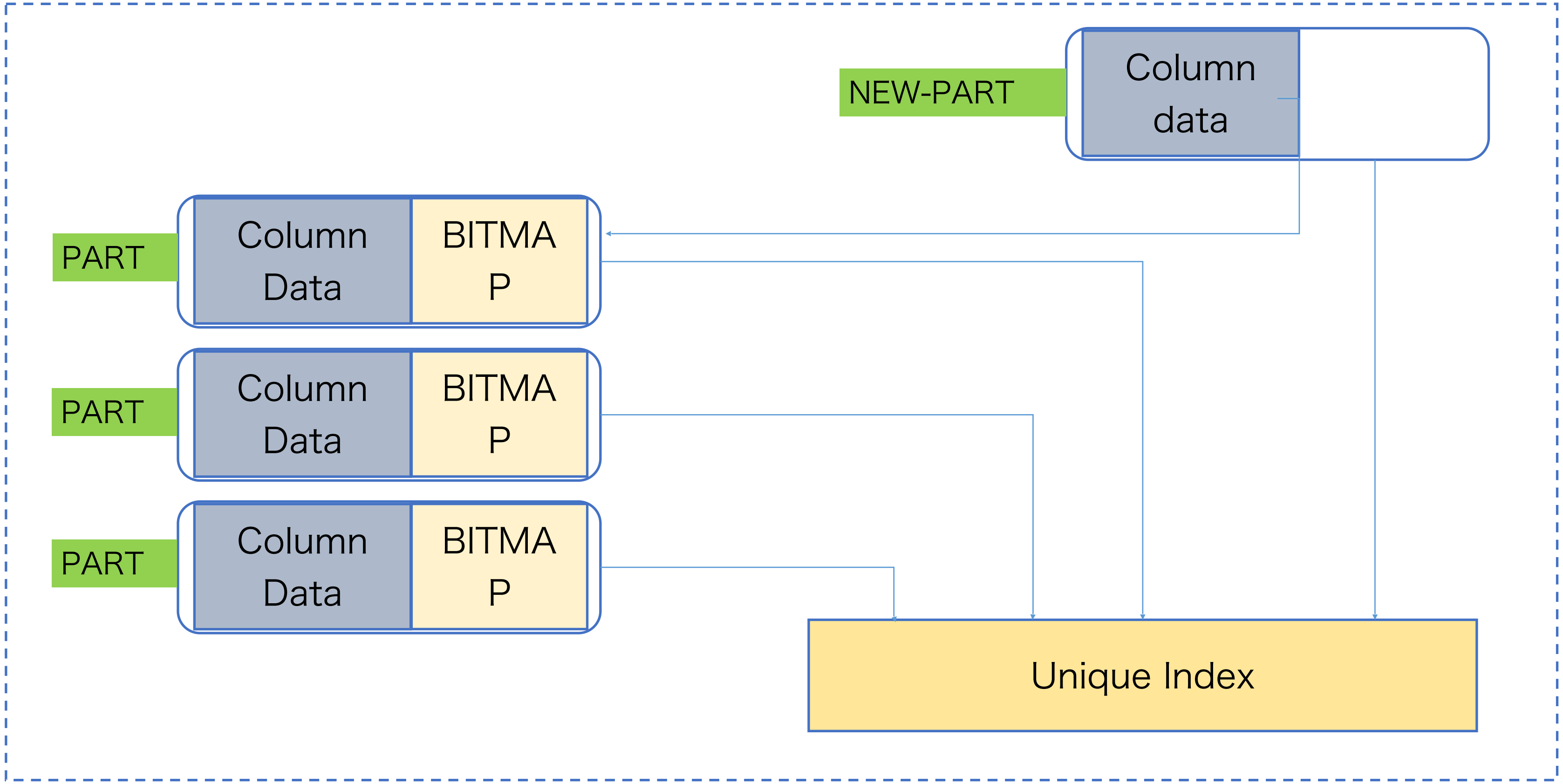
Filtering out data that has been marked as deleted during queries and ensuring synchronization of data copies is a common requirement

Unique Index: Table and line level

DeleteMarkManager: mar DELETE in parts

TCHouse-C solution for real time update: Delete-Insert

Write in and remove duplicate

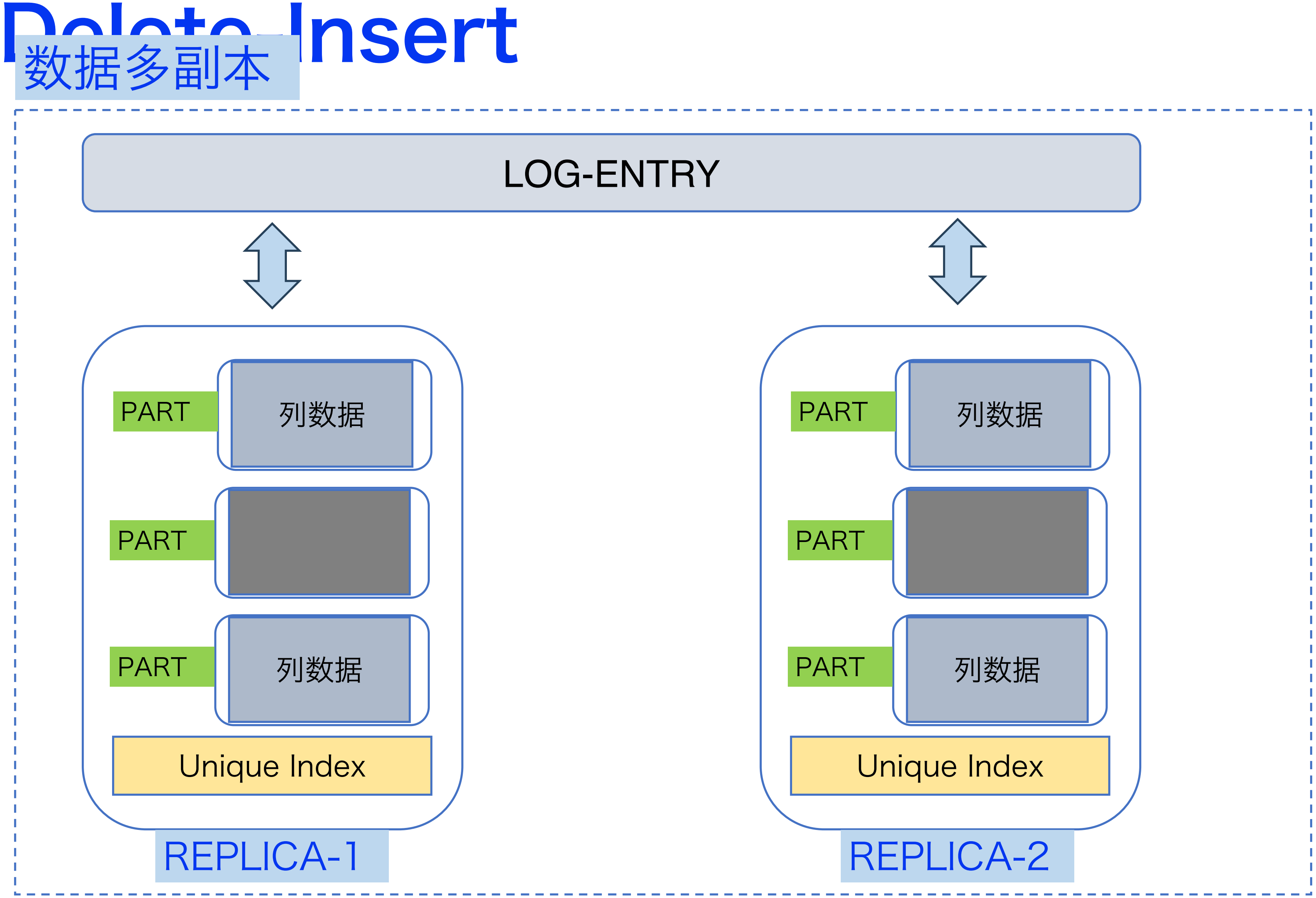


During the write process, add table-level index updates and simultaneously mark corresponding rows in the existing partitions (PART) for deletion

Writing data deduplicates through a global index

When querying, construct the values of the virtual column _exists_row using BITMAP to filter out the deleted data.

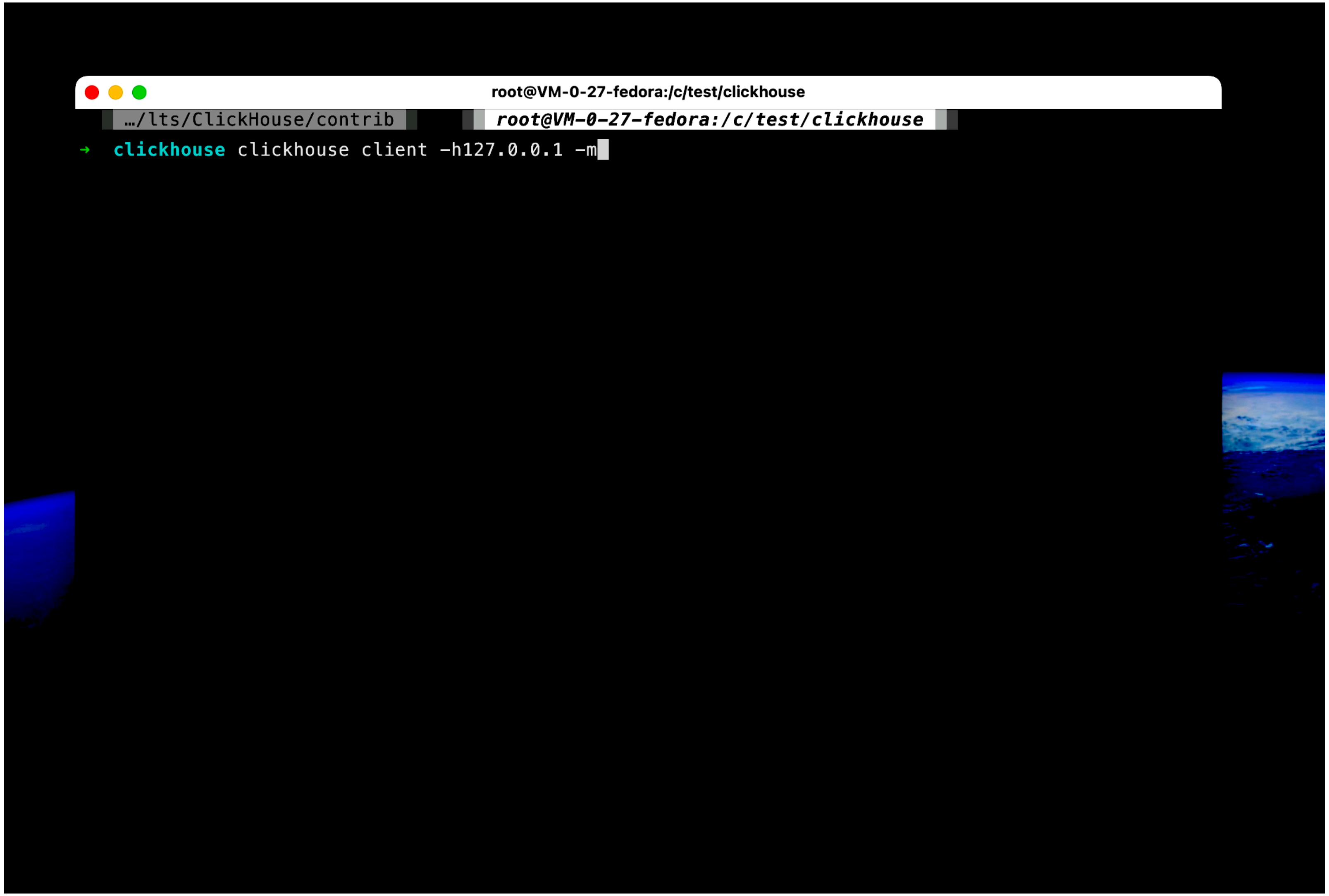
TCHouse-C solution for real time update: Delete-Insert



实现细节：

1. PART 分配唯一 递增 ID
2. DELETE 语句生成特殊 PART，通过LOG-ENTRY 队列同步。
3. 通过版本机制确保数据更新时序符合预期。

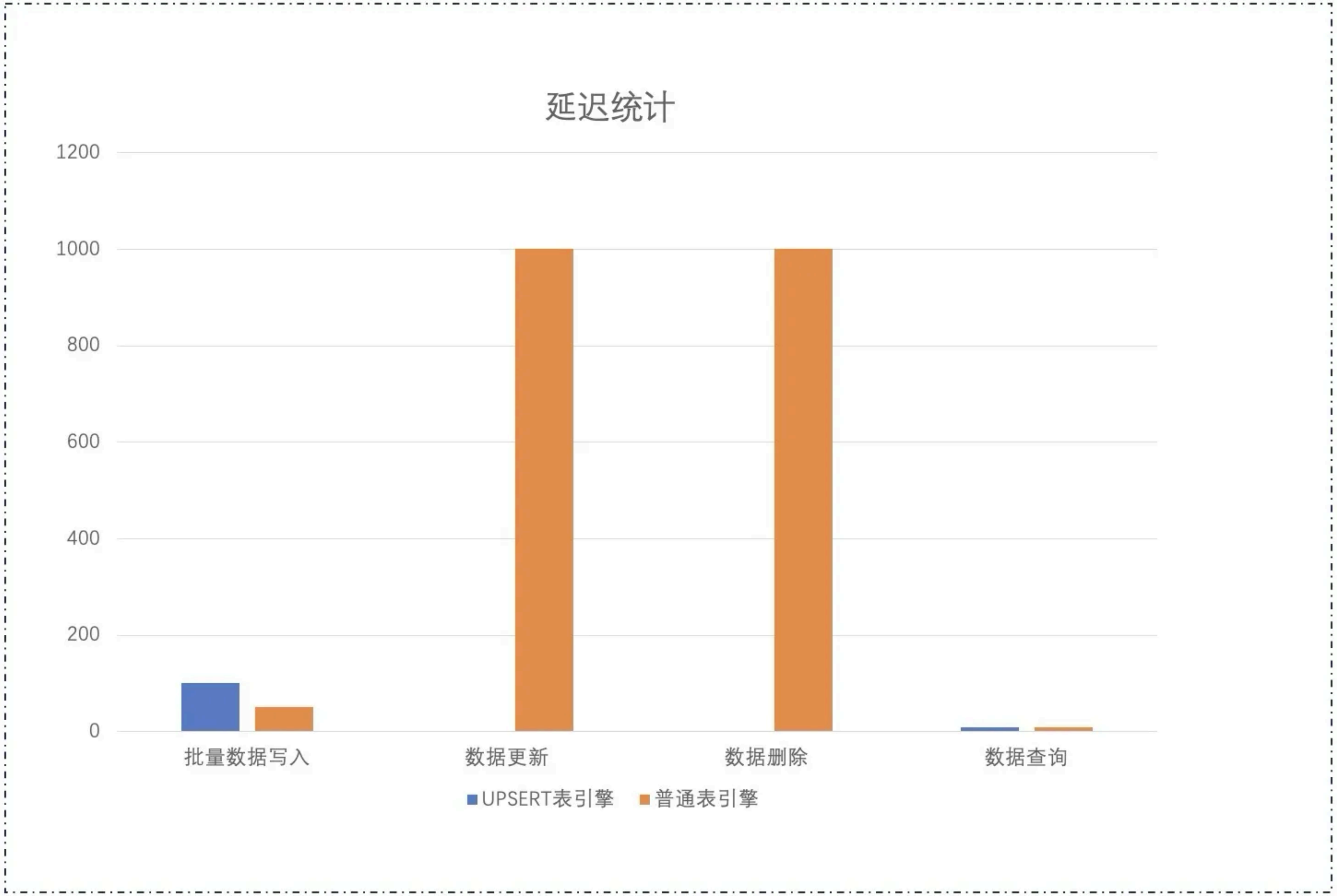
TCHouse-C demo



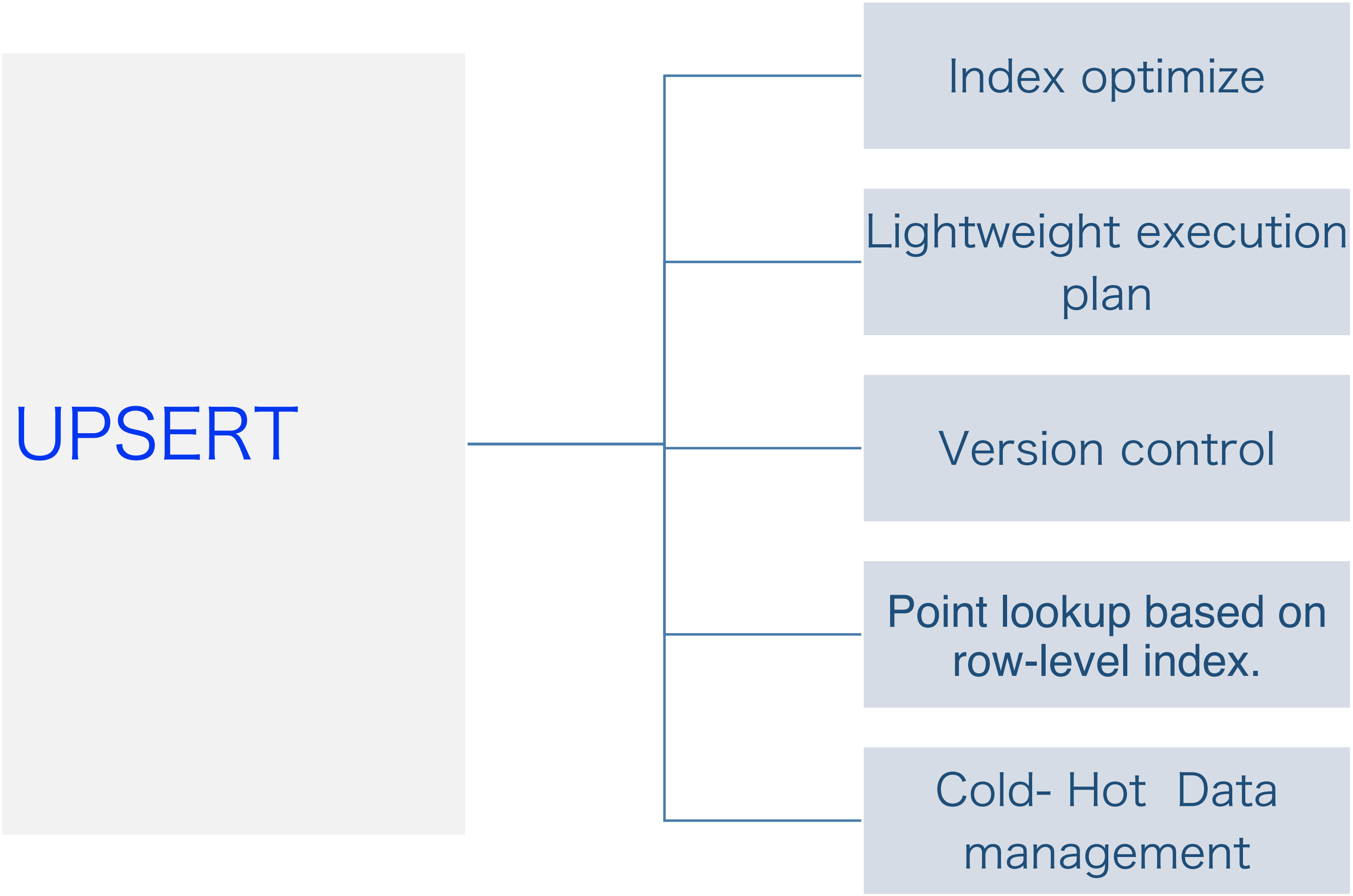
root@VM-0-27-fedora:/c/test/clickhouse

```
➜  clickhouse clickhouse client -h127.0.0.1 -m
```

TCHouse-C performance result：INSERT+DELETE



TCHouse-C real time update - roadmap



UPSERT
Index optimize
Lightweight execution plan
Version control
Point lookup based on row-level index.
Cold- Hot Data management

谢谢

腾讯云TCHouse-C技术交流群