阿里云
奥运会官方云服务合作伙伴

ClickHouse 索引与优化

易顺（宙游）阿里云数据库研发

本次分享尝试回答的问题

1.ClickHouse 数据在物理文件中怎么存的，不同文件之间的联系是什么？

2.针对不同业务，我们应该怎么选择 Order by key、Primary key？

3.Skipping Index 是什么？怎么选择？

4.怎么用 explain 语句查看 SQL 的执行计划？

5.ClickHouse 的一般优化策略。

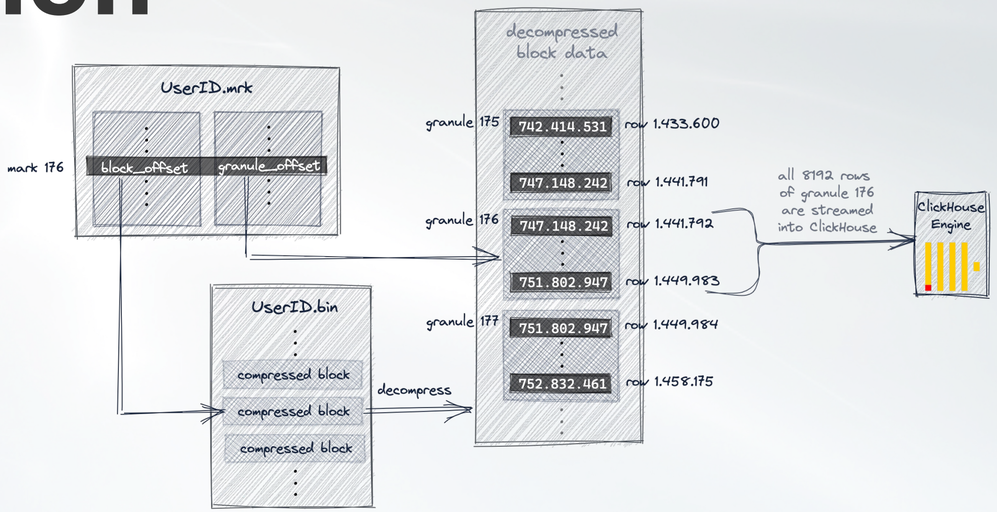
01 ClickHouse Files and Keys

02 ClickHouse Skipping Indexes

03 ClickHouse Explain

04 ClickHouse Optimization

05 Q&A



Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

01 ClickHouse Files and Keys

1. ClickHouse Files and Keys

```
CREATE TABLE hits_UserID_URL
(
    `UserID` UInt32,
    `URL` String,
    `EventTime` DateTime
)
ENGINE = MergeTree
PRIMARY KEY (UserID, URL)
ORDER BY (UserID, URL, EventTime)
SETTINGS index_granularity = 8192

Query id: f772b7c2-26fc-4b02-8ca8-45e281298bca

Ok.

0 rows in set. Elapsed: 0.007 sec.
```

```
INSERT INTO hits_UserID_URL SELECT
    intHash32(UserID) AS UserID,
    URL,
    EventTime
FROM url('https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz', 'TSV', 'WatchID UInt64,  JavaEnable UInt8,
WHERE URL != ''

Query id: a4eeba29-0f09-41f7-86d2-2a6494a4e035

Ok.

0 rows in set. Elapsed: 92.086 sec. Processed 8.87 million rows, 897.42 MB (96.37 thousand rows/s., 9.75 MB/s.)
Peak memory usage: 418.22 MiB.
```

```
part_type:                  Wide
path:                       /var/lib/clickhouse/store/805/8058f914-1a43-4228-a48d-c99aae564a16/all_1_9_2/
rows:                       8.87 million
data_uncompressed_bytes:    733.28 MiB
data_compressed_bytes:      206.92 MiB
primary_key_bytes_in_memory: 96.93 KiB
marks:                      1083
bytes_on_disk:              206.98 MiB
```

Note：

1. 使用 Merge Tree 引擎；

2. Sorting Key 选择 (UserID, URL, EventTime) 3 列；

3. Primary Key 选择 (UserID, URL)；

4. Primary Key 索引常驻内存，是 Sorting Key 的前缀；

5. 如果只指定了 sorting key，那么 primary key 跟 sorting key 相同；

6. Part 数据以 Wide 格式存储，每一列单独一个文件。

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

1. ClickHouse Files and Keys

```
[root@iZ2zecl1pe0cgqf3lyev5lZ all_1_9_2]# ll -lh
total 208M
-rw-r----- 1 clickhouse clickhouse  468 Aug 20 23:02 checksums.txt
-rw-r----- 1 clickhouse clickhouse   87 Aug 20 23:02 columns.txt
-rw-r----- 1 clickhouse clickhouse    7 Aug 20 23:02 count.txt
-rw-r----- 1 clickhouse clickhouse   10 Aug 20 23:02 default_compression_codec.txt
-rw-r----- 1 clickhouse clickhouse  33M Aug 20 23:02 EventTime.bin
-rw-r----- 1 clickhouse clickhouse 2.8K Aug 20 23:02 EventTime.cmrk
-rw-r----- 1 clickhouse clickhouse    1 Aug 20 23:02 metadata_version.txt
-rw-r----- 1 clickhouse clickhouse  48K Aug 20 23:02 primary.cidx
-rw-r----- 1 clickhouse clickhouse  238 Aug 20 23:02 serialization.json
-rw-r----- 1 clickhouse clickhouse 175M Aug 20 23:02 URL.bin
-rw-r----- 1 clickhouse clickhouse 3.8K Aug 20 23:02 URL.cmrk
-rw-r----- 1 clickhouse clickhouse 797K Aug 20 23:02 UserID.bin
-rw-r----- 1 clickhouse clickhouse 2.2K Aug 20 23:02 UserID.cmrk
```

Note：

1. 最重要的两个文件 bin 文件和 mrk 文件。

```
[root@iZ2zecl1pe0cgqf3lyev5lZ all_1_9_2]# cat columns.txt
columns format version: 1
3 columns:
`UserID` UInt32
`URL` String
`EventTime` DateTime

[root@iZ2zecl1pe0cgqf3lyev5lZ all_1_9_2]# cat count.txt
8867681

[root@iZ2zecl1pe0cgqf3lyev5lZ all_1_9_2]# cat default_compression_codec.txt
CODEC(LZ4)

[root@iZ2zecl1pe0cgqf3lyev5lZ all_1_9_2]# cat serialization.json
{"columns":[{"kind":"Default","name":"EventTime","num_defaults":0,"num_rows":8867681},
{"kind":"Default","name":"URL","num_defaults":0,"num_rows":8867681},
{"kind":"Default","name":"UserID","num_defaults":0,"num_rows":8867681}],"version":0}
```

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

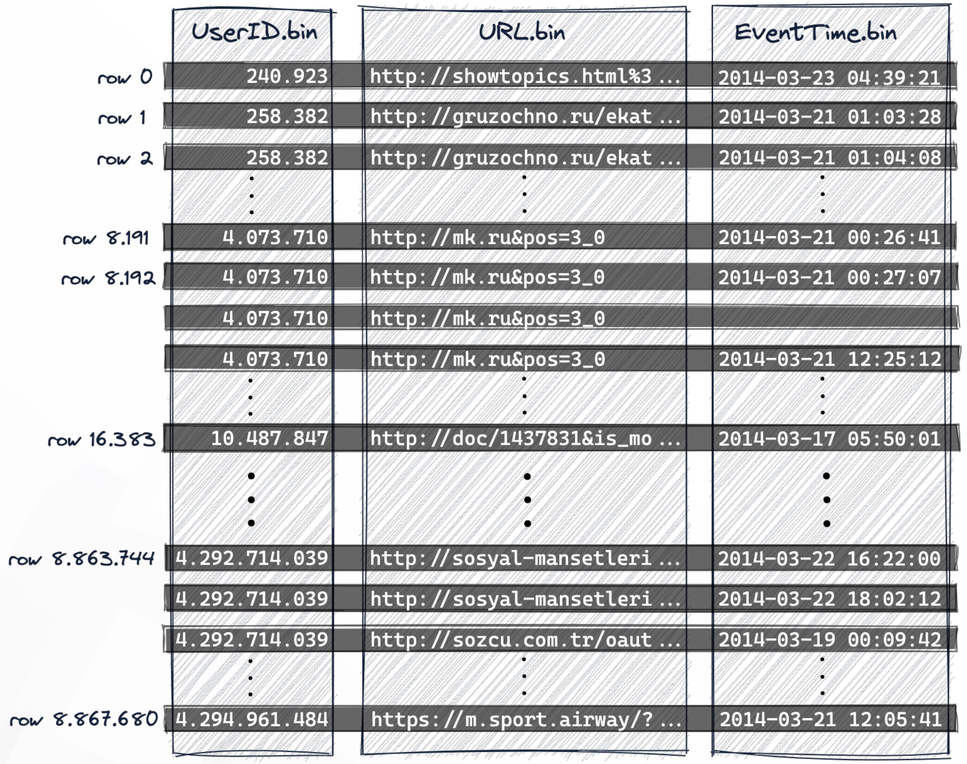
1. ClickHouse Files and Keys – bin file

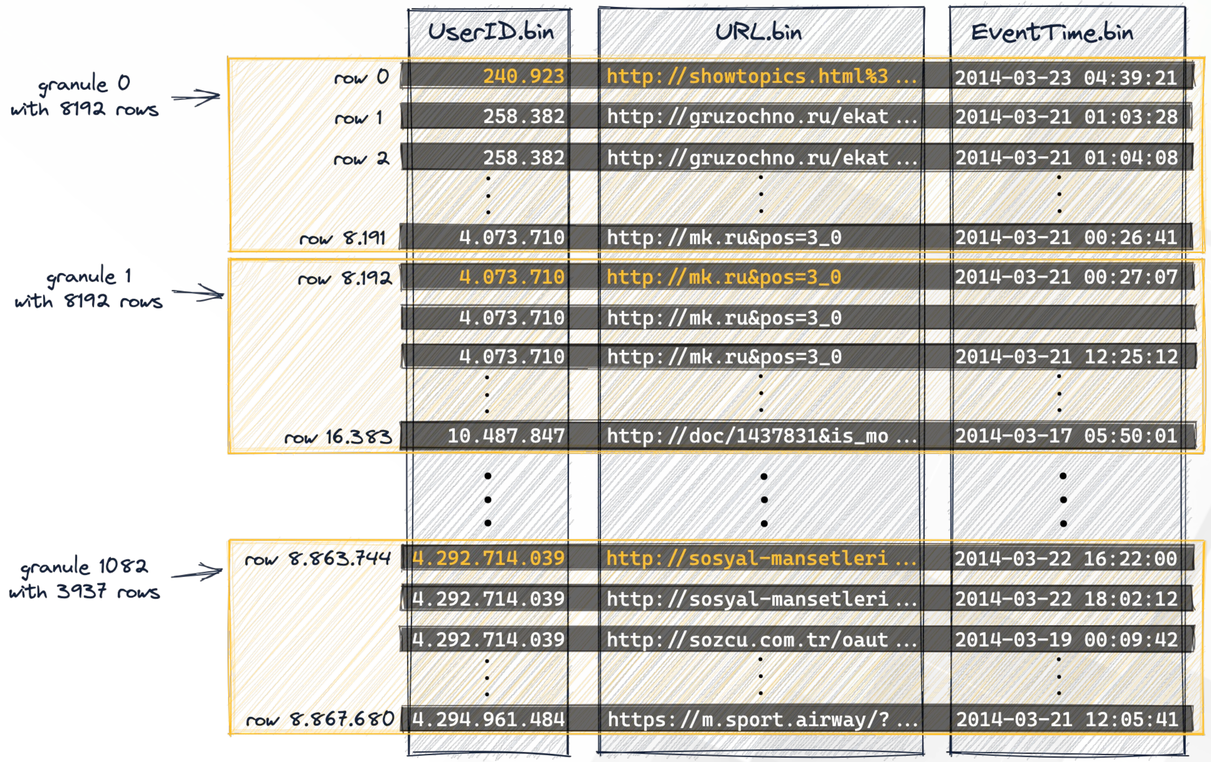
```
1   CREATE TABLE hits_UserID_URL
2   (
3       `UserID` UInt32,
4       `URL` String,
5       `EventTime` DateTime
6   )
7   ENGINE = MergeTree
8   PRIMARY KEY (UserID, URL)
9   ORDER BY (UserID, URL, EventTime)
10  SETTINGS index_granularity = 8192
```

Note：

1. 每一列在逻辑上被划分为多个 granule；

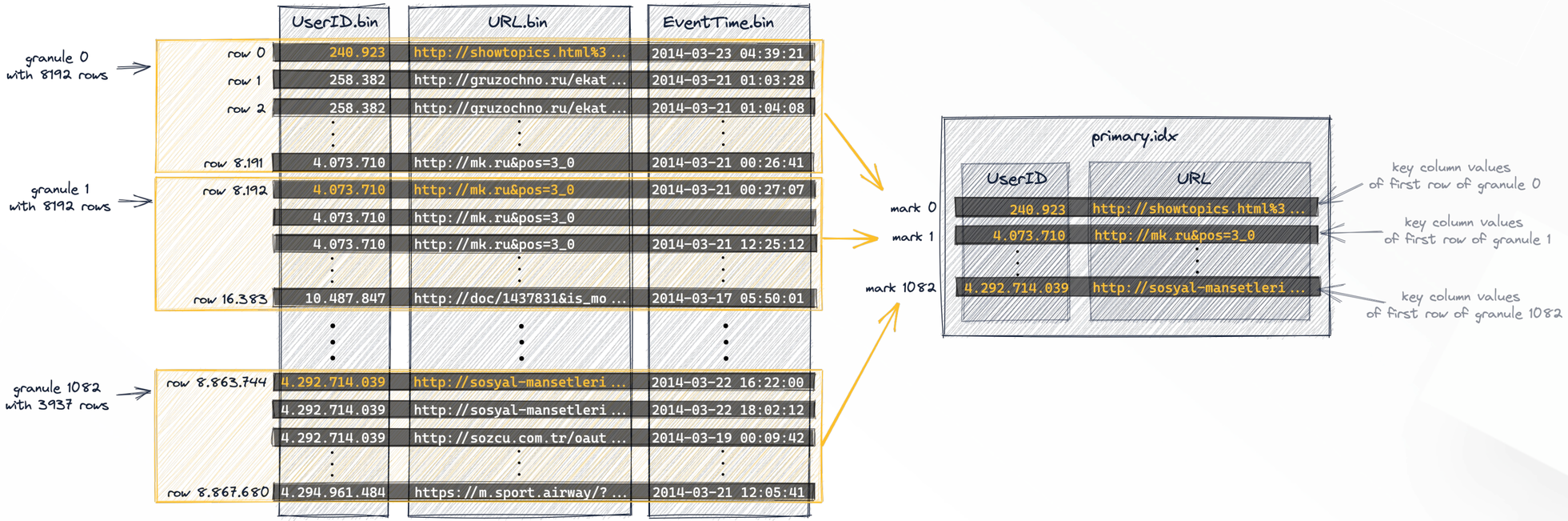
2. granule 是 ClickHouse 数据处理的最小单位。





Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

1. ClickHouse Files and Keys – idx file

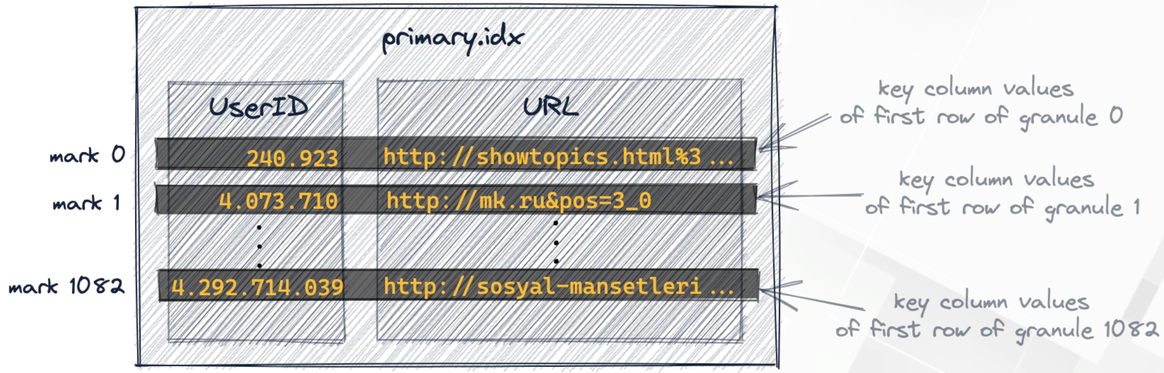


Note：

1. 只有每个 granule 的第一行会被写到 primary.idx 中；

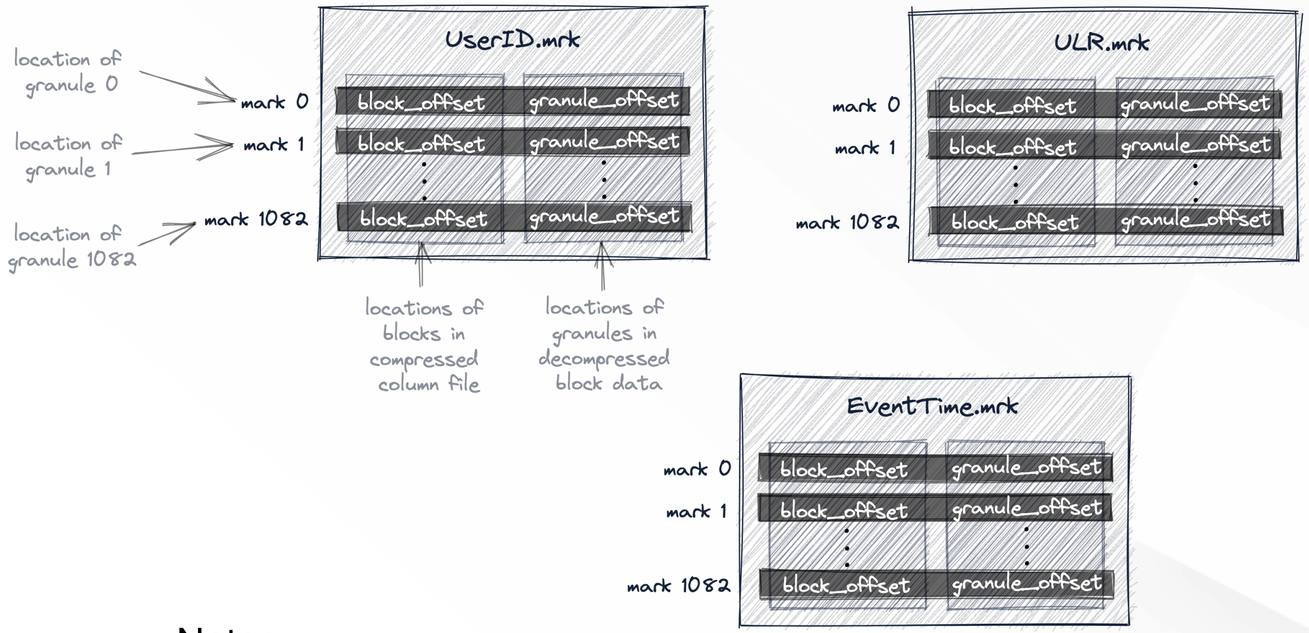
2. 总共 887万行，分为 1083个 granule，主键就有 1083 行；

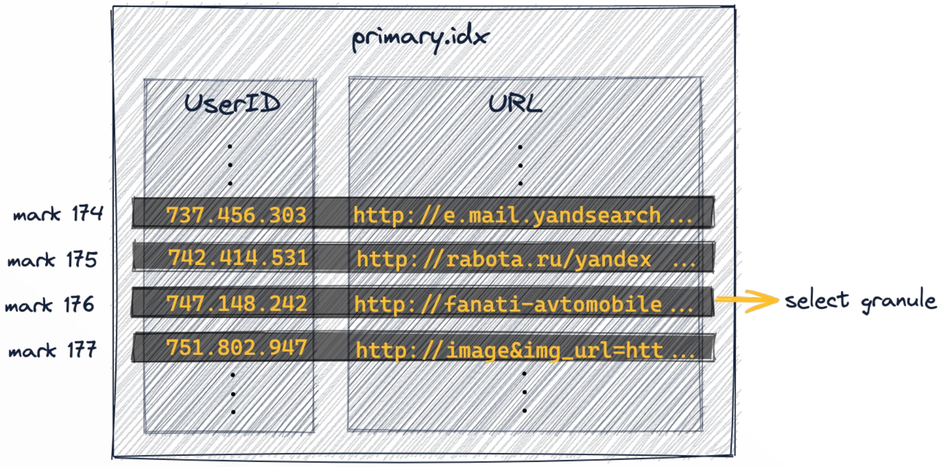
3. primary.idx 完全加载到内存中，bytes: 96.93 KiB。



Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

1. ClickHouse Files and Keys – mrk file





Note：

1. mark file 的行数跟 primary index 的行数一致，标识 granule 的物理位置；
2. mark file 有两个偏移量，block_offset 和 granule_offset；
3. bin 文件中，多个 granule 会被压缩成 1 个 block，当前数据的最大 block_number: 9;
4. block offset 是 bin 文件中 block 的偏移量；granule offset 是 block 中 granule 的偏移量；
5. mark file 不是常驻内存的，按需 load 到内存，marks_bytes: 8.61 KiB。

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

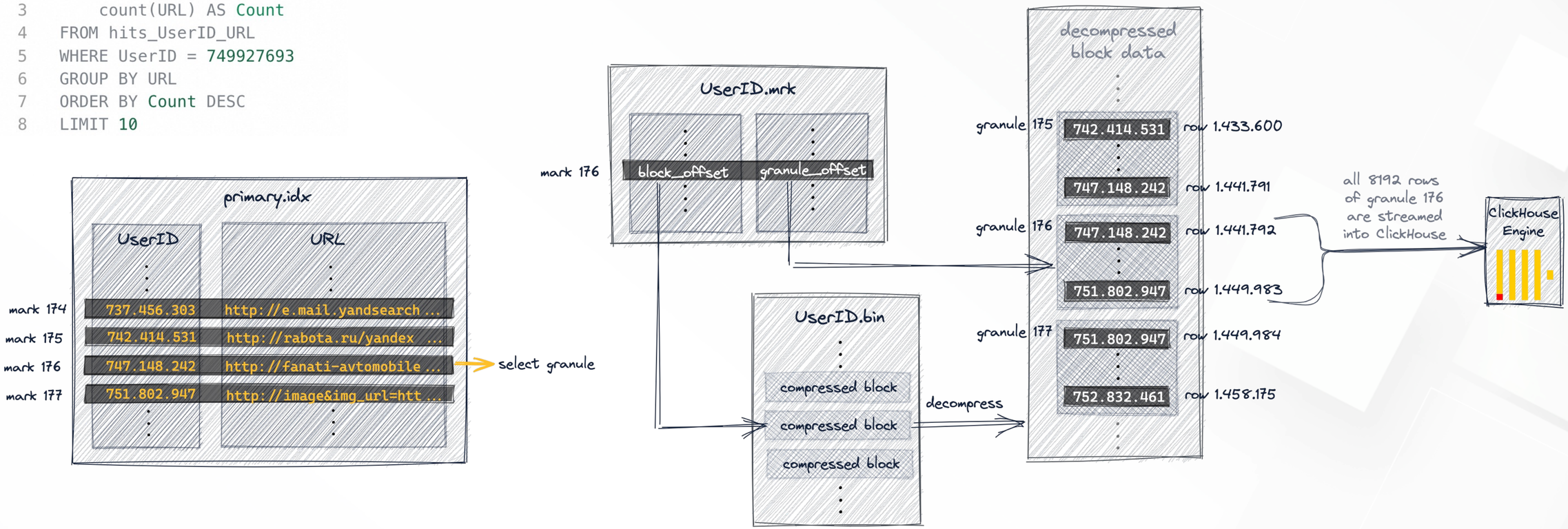
1. ClickHouse Files and Keys – 查询示例

```
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 85706ec8-14cb-40b8-b851-85ef0de3f5eb

    ┌─URL────────────────────────────────┬─Count─┐
 1. │ http://auto.ru/chatay-barana...    │   170 │
 2. │ http://auto.ru/chatay-id=371...    │    52 │
 3. │ http://public_search               │    45 │
 4. │ http://kovrik-medvedevushku-...    │    36 │
 5. │ http://forumal                     │    33 │
 6. │ http://korablitz.ru/L_1OFFER...    │    14 │
 7. │ http://auto.ru/chatay-id=371...    │    14 │
 8. │ http://auto.ru/chatay-john-D...    │    13 │
 9. │ http://auto.ru/chatay-john-D...    │    10 │
10. │ http://wot/html?page/23600_m...    │     9 │
    └────────────────────────────────────┴───────┘

10 rows in set. Elapsed: 0.007 sec. Processed 8.19 thousand rows, 740.18 KB (1.10 million rows/s., 99.05 MB/s.)
Peak memory usage: 243.55 KiB.
```

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

1. ClickHouse Files and Keys – 查询示例

```sql
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10
```

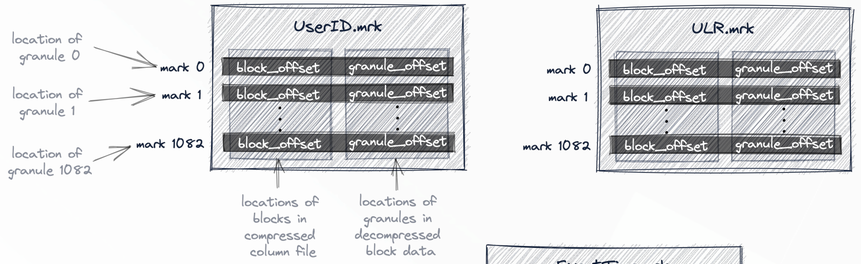


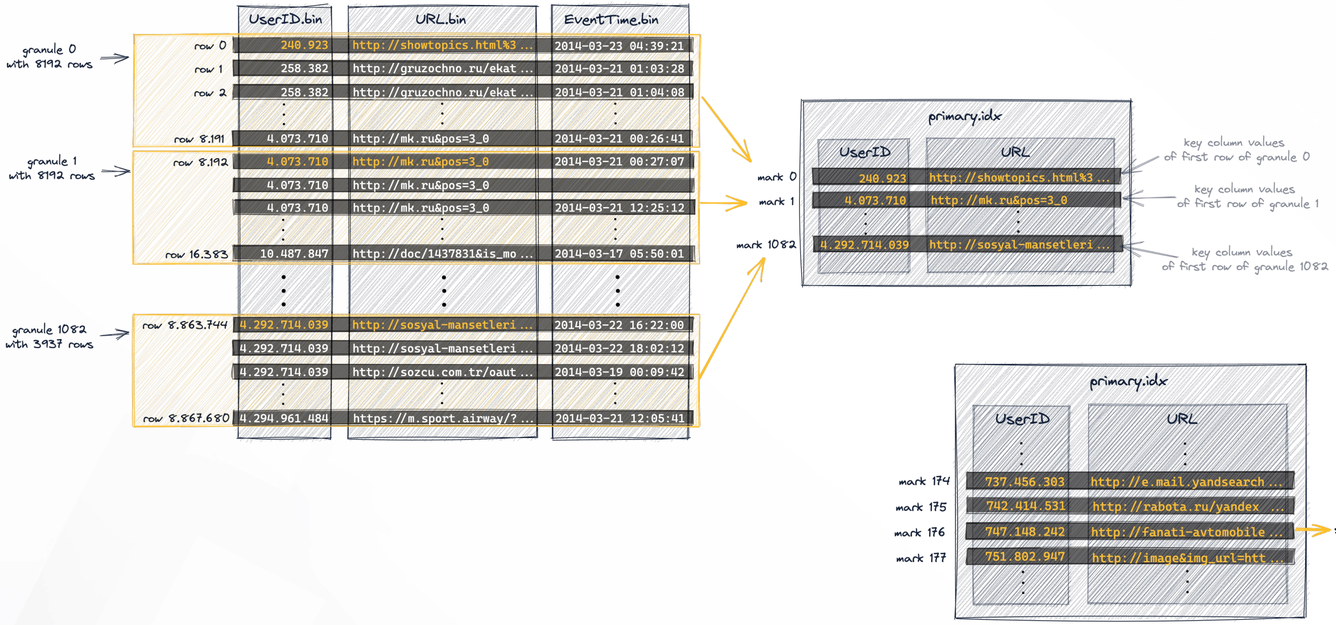

```
8-a48d-c99aae564a16) (SelectExecutor): Key condition: (column 0 in [749927693, 749927693])
8-a48d-c99aae564a16) (SelectExecutor): Filtering marks by primary and secondary keys
8-a48d-c99aae564a16) (SelectExecutor): Running binary search on index range for part all_1_9_2 (1083 marks)
8-a48d-c99aae564a16) (SelectExecutor): Found (LEFT) boundary mark: 176
8-a48d-c99aae564a16) (SelectExecutor): Found (RIGHT) boundary mark: 177
8-a48d-c99aae564a16) (SelectExecutor): Found continuous range in 19 steps
8-a48d-c99aae564a16) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1/1083 marks by primary key, 1 marks to read from 1 ranges
8-a48d-c99aae564a16) (SelectExecutor): Spreading mark ranges among streams (default reading)
8-a48d-c99aae564a16) (SelectExecutor): Reading 1 ranges in order from part all_1_9_2, approx. 8192 rows starting from 1441792
```

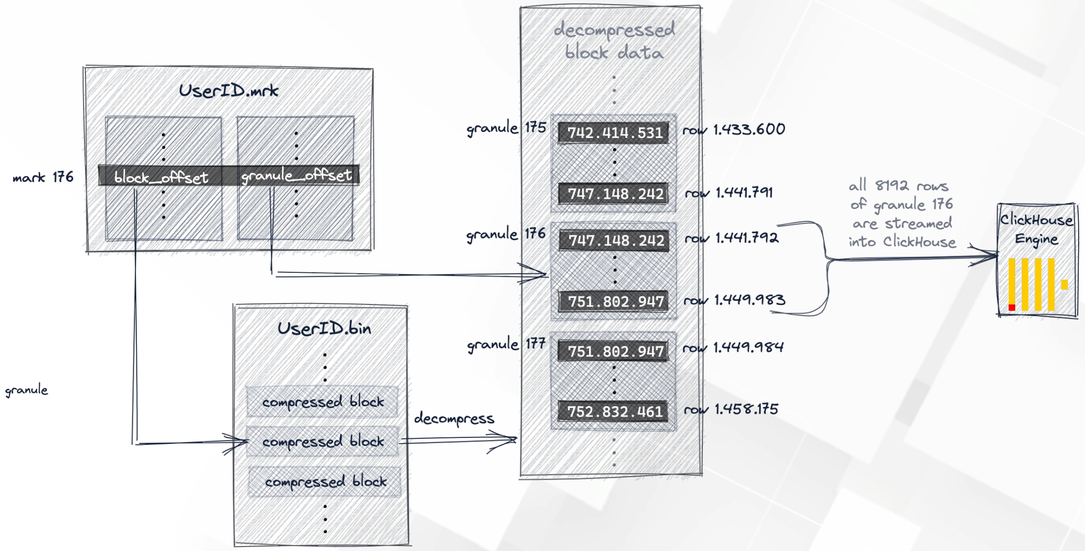
Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

1. ClickHouse Files and Keys 小结

```
[root@iZ2zecl1pe0cgqf3lyev5lZ all_1_9_2]# ll -lh
total 208M
-rw-r----- 1 clickhouse clickhouse  468 Aug 20 23:02 checksums.txt
-rw-r----- 1 clickhouse clickhouse   87 Aug 20 23:02 columns.txt
-rw-r----- 1 clickhouse clickhouse    7 Aug 20 23:02 count.txt
-rw-r----- 1 clickhouse clickhouse   10 Aug 20 23:02 default_compression_codec.txt
-rw-r----- 1 clickhouse clickhouse  33M Aug 20 23:02 EventTime.bin
-rw-r----- 1 clickhouse clickhouse 2.8K Aug 20 23:02 EventTime.cmrk
-rw-r----- 1 clickhouse clickhouse    1 Aug 20 23:02 metadata_version.txt
-rw-r----- 1 clickhouse clickhouse  48K Aug 20 23:02 primary.cidx
-rw-r----- 1 clickhouse clickhouse  238 Aug 20 23:02 serialization.json
-rw-r----- 1 clickhouse clickhouse 175M Aug 20 23:02 URL.bin
-rw-r----- 1 clickhouse clickhouse 3.8K Aug 20 23:02 URL.cmrk
-rw-r----- 1 clickhouse clickhouse 797K Aug 20 23:02 UserID.bin
-rw-r----- 1 clickhouse clickhouse 2.2K Aug 20 23:02 UserID.cmrk
```







Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

1. ClickHouse Files and Keys 小结

排序建、主键、分区键的一般选择方式：

1. Sorting Key 是 ClickHouse 最主要依赖的扫描加速技术，Part 内的数据按 Sorting Key 严格有序，查询使用二分查找；

2.按照业务场景，将查询最频繁 SQL 命中的那些列，按照基数升序放在 Primary Key 中，取得相对较好的压缩率和性能；

3. Primary Key 是 Sorting key 的前缀，为了提高内存使用效率，Primary Key 只包含查询要过滤的列；

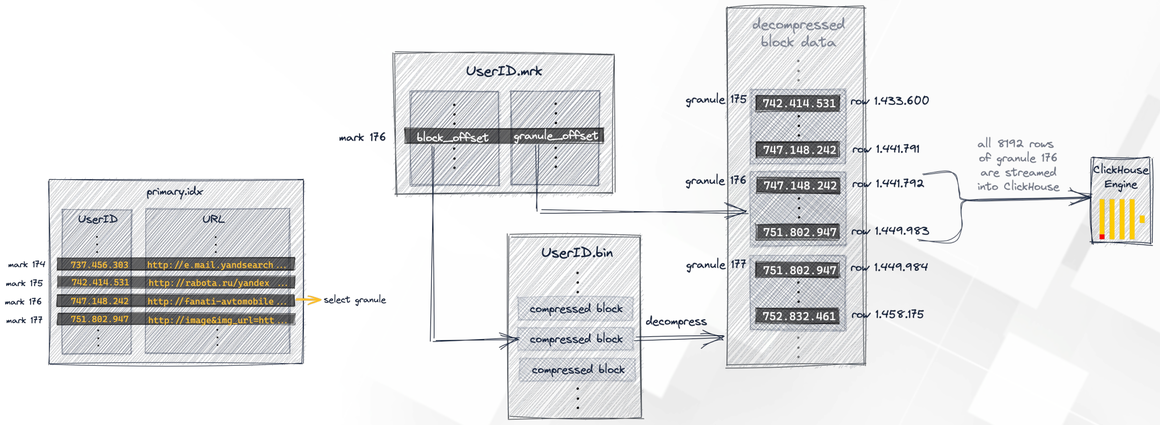
4. 因为相同的数据放在一起能够得到更好的压缩率，所以 Sorting Key 中，除了 Primary Key 还能放基数低的列，降低磁盘使用率；

5. Partition Key 定义了不同分区，不同分区的数据是在物理上隔离开的，每个 Part 只属于一个分区；

6. 由于 Partition Key 会对 Part 起到物理隔离，所以数据分区过细的情况下会导致 Part 数量膨胀，文件IO一多，就可能影响查询性能。

Question：

Primary Key 查找是很快，但是如果有些 SQL 无法命中 Primary Key 怎么办？



Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

02 ClickHouse Skipping Indexes

2. Skipping Indexes

```
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10

Query id: 4aed4fe5-4d36-47c0-8ee0-e757d2db87af

    ┌─────UserID─┬─Count─┐
 1. │ 2459550954 │  3741 │
 2. │ 1084649151 │  2484 │
 3. │  723361875 │   729 │
 4. │ 3087145896 │   695 │
 5. │ 2754931092 │   672 │
 6. │ 1509037307 │   582 │
 7. │ 3085460200 │   573 │
 8. │ 2454360090 │   556 │
 9. │ 3884990840 │   539 │
10. │  765730816 │   536 │
    └────────────┴───────┘

10 rows in set. Elapsed: 0.157 sec. Processed 8.81 million rows, 799.34 MB (56.16 million rows/s., 5.10 GB/s.)
Peak memory usage: 25.27 MiB.
```

```
(SelectExecutor): Key condition: (column 1 in ['http://public_search', 'http://public_search'])
(SelectExecutor): Filtering marks by primary and secondary keys
(SelectExecutor): Used generic exclusion search over index for part all_1_9_2 with 1537 steps
(SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1076/1083 marks by primary key, 1076 marks to read from 5 ranges
(SelectExecutor): Spreading mark ranges among streams (default reading)
(SelectExecutor): Reading approx. 8810337 rows with 8 streams
```

2. Skipping Indexes

```
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10

Query id: 4aed4fe5-4d36-47c0-8ee0-e757d2db87af

    ┌─────UserID─┬─Count─┐
 1. │ 2459550954 │  3741 │
 2. │ 1084649151 │  2484 │
 3. │  723361875 │   729 │
 4. │ 3087145896 │   695 │
 5. │ 2754931092 │   672 │
 6. │ 1509037307 │   582 │
 7. │ 3085460200 │   573 │
 8. │ 2454360090 │   556 │
 9. │ 3884990840 │   539 │
10. │  765730816 │   536 │
    └────────────┴───────┘

10 rows in set. Elapsed: 0.157 sec. Processed 8.81 million rows, 799.34 MB (56.16 million rows/s., 5.10 GB/s.)
Peak memory usage: 25.27 MiB.
```

Question：

1.从日志来看，为什么 SQL 基本上执行全表扫?

2.被过滤掉的那几个 marks 是什么情况?

3.怎么加速这种查询?

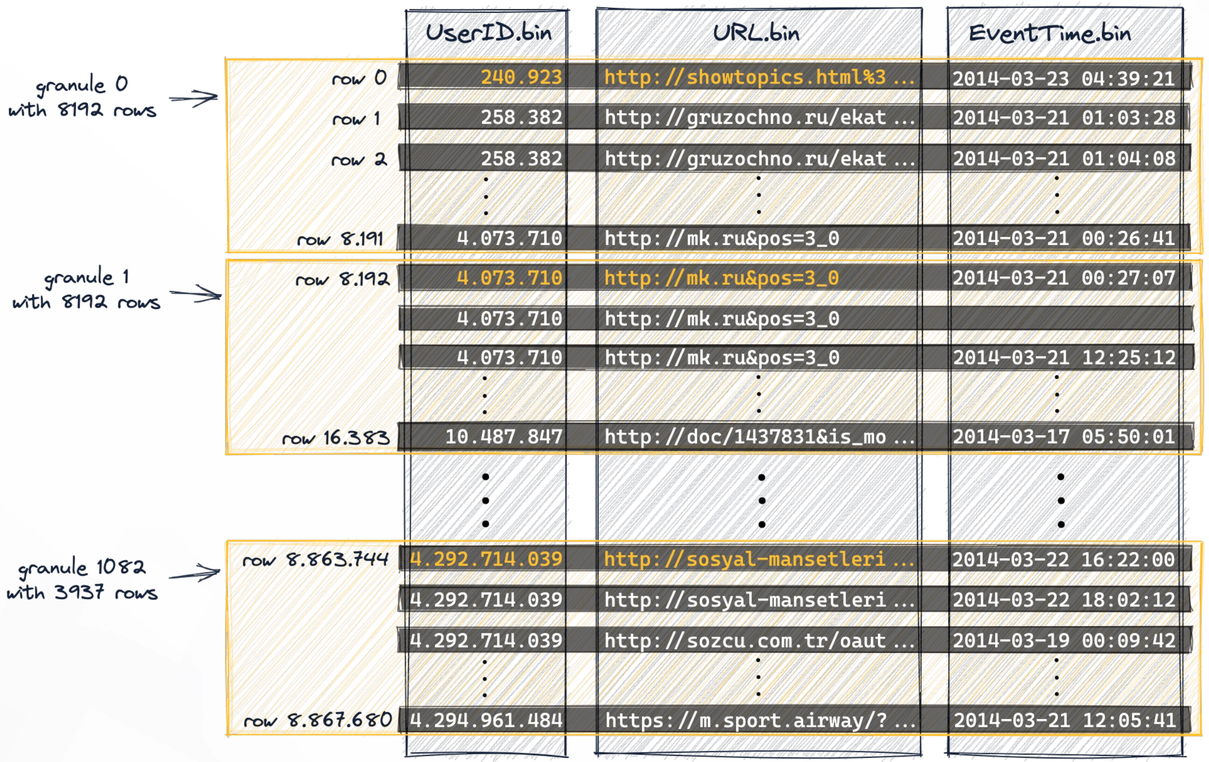
```
(SelectExecutor): Key condition: (column 1 in ['http://public_search', 'http://public_search'])
(SelectExecutor): Filtering marks by primary and secondary keys
(SelectExecutor): Used generic exclusion search over index for part all_1_9_2 with 1537 steps
(SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1076/1083 marks by primary key, 1076 marks to read from 5 ranges
(SelectExecutor): Spreading mark ranges among streams (default reading)
(SelectExecutor): Reading approx. 8810337 rows with 8 streams
```

2. Skipping Indexes

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Question：

Q：从日志来看，为什么 SQL 基本上执行全表扫?
A：因为 URL 这一列没有排序。



Question：

1.从日志来看，为什么 SQL 基本上执行全表扫?

2.被过滤掉的那几个 marks 是什么情况?

3.怎么加速这种查询?

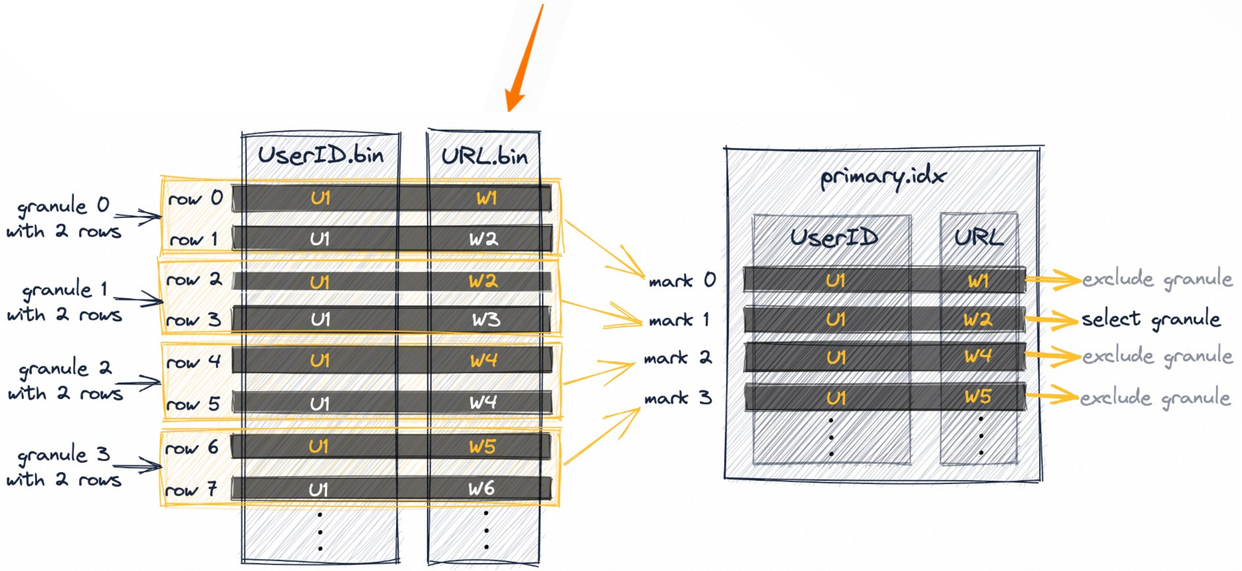
Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

2. Skipping Indexes

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Question：

Q：被过滤掉的那几个 marks 是什么情况?
A：因为在 UserID 相同的时候，URL 局部有序。



Question：

1.从日志来看，为什么 SQL 基本上执行全表扫?

2.被过滤掉的那几个 marks 是什么情况?

3.怎么加速这种查询?

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

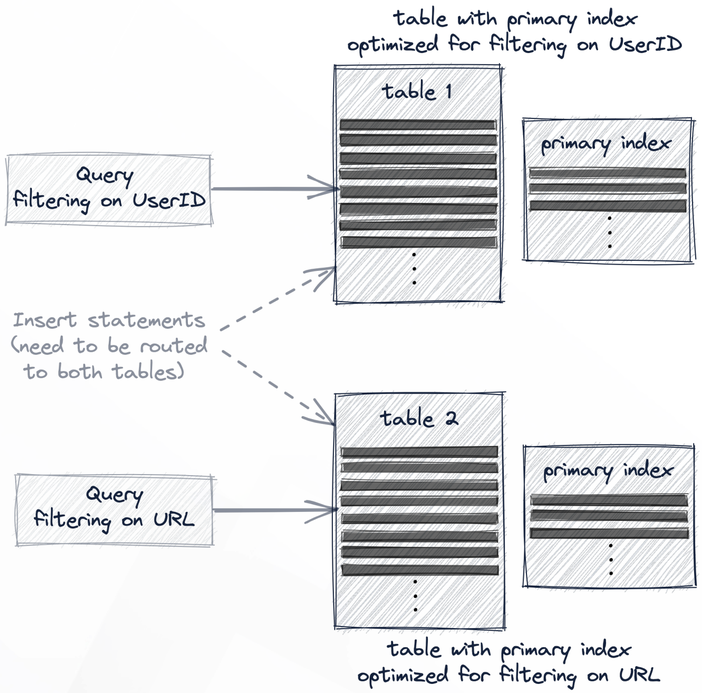
2. Skipping Indexes

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

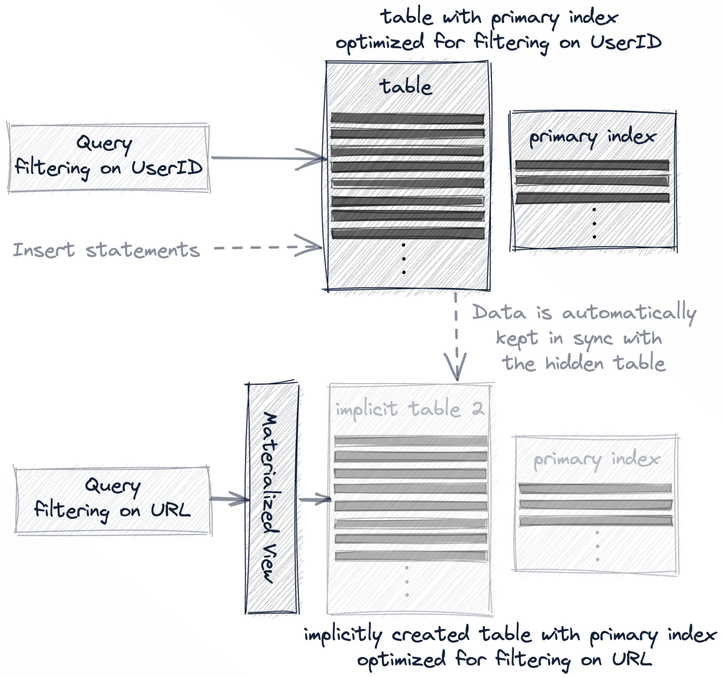
Question：

Q：怎么加速这种查询？
A：Secondary Table、MV、Projection、Skipping Index。

给另外的一份数据重新设置 Sorting Key，让 SQL 命中排好序的列。



Secondary Table



Materialized View

Question：

1.从日志来看，为什么 SQL 基本上执行全表扫？

2.被过滤掉的那几个 marks 是什么情况？

3.怎么加速这种查询？

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

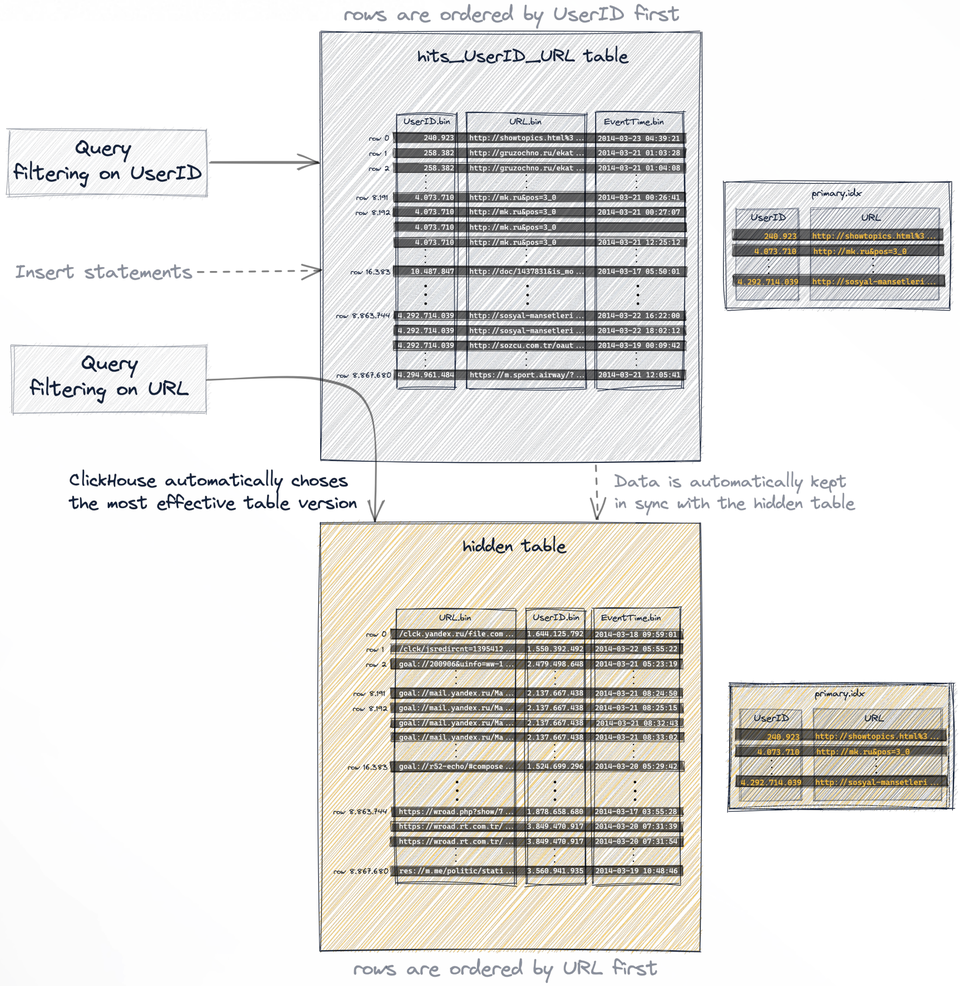
2. Skipping Indexes

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Question：

Q：怎么加速这种查询？
A：Secondary Table、MV、Projection、Skipping Index。

给另外的一份数据重新设置 Sorting Key，让 SQL 命中排好序的列。



Projection

Question：

1.从日志来看，为什么 SQL 基本上执行全表扫？

2.被过滤掉的那几个 marks 是什么情况？

3.怎么加速这种查询？

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

2. Skipping Indexes

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Question：

Q：怎么加速这种查询？
A：Secondary Table、MV、Projection、Skipping Index。

数据不冗余的情况下，可能可行的解法。

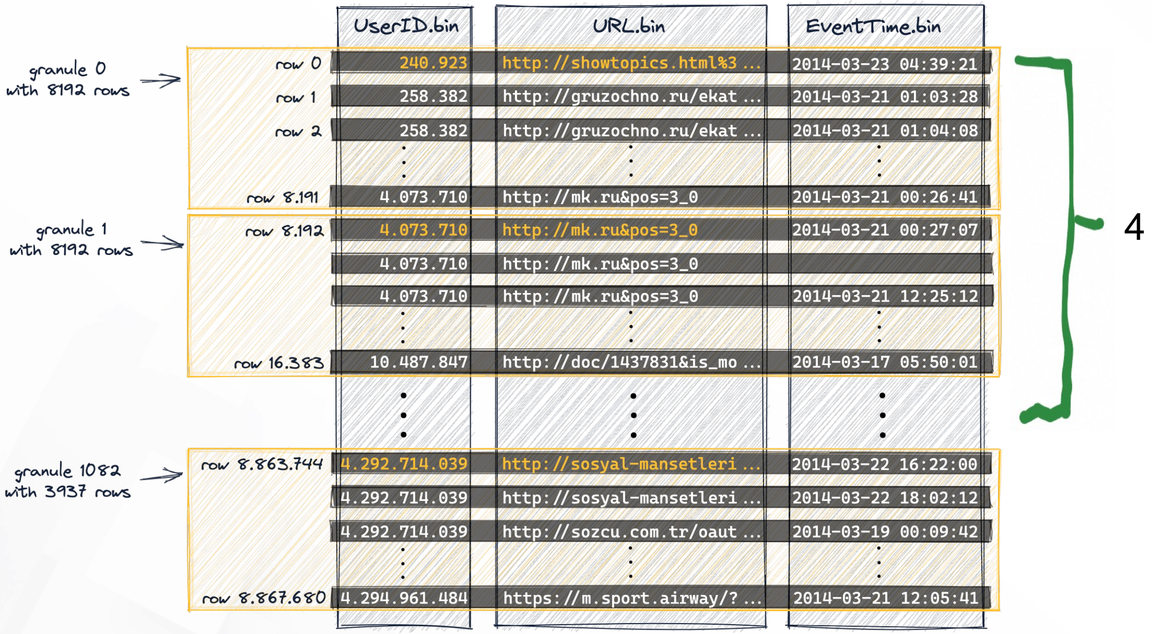
```sql
ALTER TABLE hits_UserID_URL ADD INDEX url_skipping_index URL TYPE minmax GRANULARITY 4;
ALTER TABLE hits_UserID_URL MATERIALIZE INDEX url_skipping_index;
```

Question：

1.从日志来看，为什么 SQL 基本上执行全表扫？

2.被过滤掉的那几个 marks 是什么情况？

3.怎么加速这种查询？



Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

2. Skipping Indexes

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Question：

Q：怎么加速这种查询？
A：Secondary Table、MV、Projection、Skipping Index。

数据不冗余的情况下，可能可行的解法。

```sql
ALTER TABLE hits_UserID_URL ADD INDEX url_skipping_index URL TYPE minmax GRANULARITY 4;
ALTER TABLE hits_UserID_URL MATERIALIZE INDEX url_skipping_index;
```

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10

Query id: 49fc94ff-8876-4b4d-b67b-bcbbc3a8f89b

    ┌─────UserID─┬─Count─┐
 1. │ 2459550954 │  3741 │
 2. │ 1084649151 │  2484 │
 3. │  723361875 │   729 │
 4. │ 3087145896 │   695 │
 5. │ 2754931092 │   672 │
 6. │ 1509037307 │   582 │
 7. │ 3085460200 │   573 │
 8. │ 2454360090 │   556 │
 9. │ 3884990840 │   539 │
10. │  765730816 │   536 │
    └────────────┴───────┘

10 rows in set. Elapsed: 0.179 sec. Processed 8.81 million rows, 799.34 MB (49.21 million rows/s., 4.46 GB/s.)
Peak memory usage: 35.30 MiB.
```

Question：

1.从日志来看，为什么 SQL 基本上执行全表扫？

2.被过滤掉的那几个 marks 是什么情况？

3.怎么加速这种查询？

Note：

1. 选择 URL 值可能介于 skipping index 为每组 granules 存储的最小值和最大值之间，导致这个 skipping index 没有起到过滤效果，甚至在数据量大的情况下，扫描 Skipping index 带来的开销，使得 SQL 变得“更慢”了。

2. 一般来说，当主键和该列存在强相关，且查找的值在很多 granule 中都不会出现的时候，用 Skipping Index 就能得到较好的过滤效果。

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

2. Skipping Indexes - types

```
INDEX index_name expr TYPE type(...) [GRANULARITY granularity_value]
```

```
MinMax —— 存储 granules 内的最大最小值

Syntax: minmax
```

```
Set —— 存储指定表达式的唯一值(不超过 max_rows 行, max_rows=0 表示 “无限制”)

Syntax: set(max_rows)
```

```
Bloom Filter ——  false_positive 默认 0.025.
Supported data types: Int*, UInt*, Float*, Enum, Date, DateTime, String, FixedString, Array,
LowCardinality, Nullable, UUID and Map.

Syntax: bloom_filter([false_positive])
```

```
N-gram Bloom Filter
Only works with datatypes: String, FixedString and Map.

Syntax: ngrambf_v1(n, size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)
```

```
Token Bloom Filter
The same as ngrambf_v1, but stores tokens instead of ngrams. Tokens are sequences separated by non-
alphanumeric characters.

Syntax: tokenbf_v1(size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)
```

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

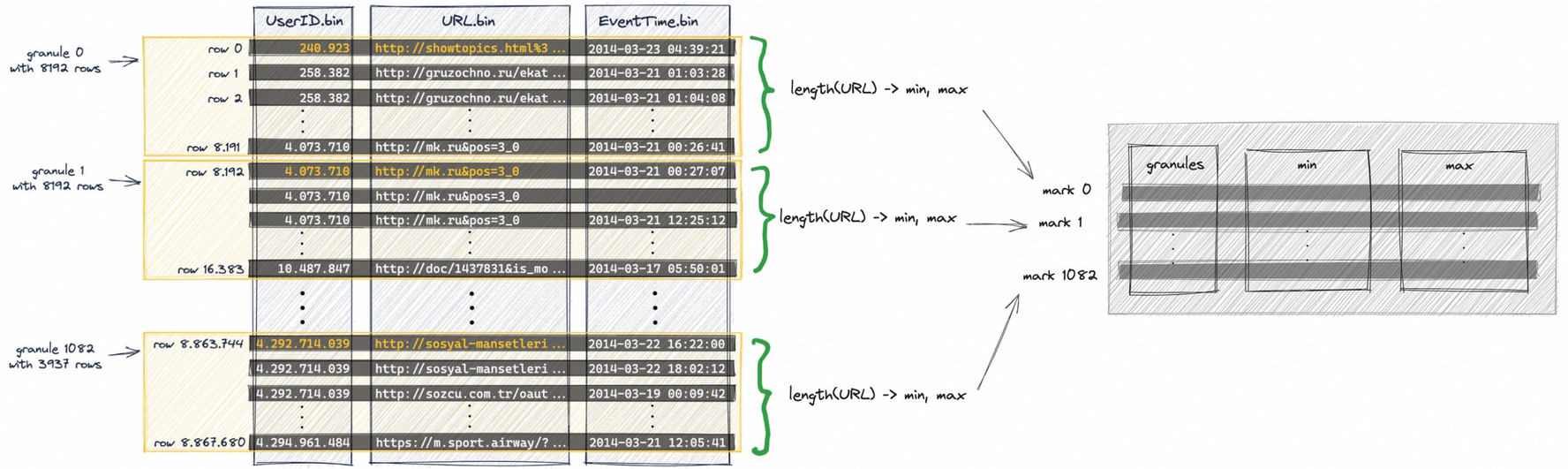
2. Skipping Indexes - minmax

```sql
ALTER TABLE hits_UserID_URL ADD INDEX url_len_minmax length(URL) TYPE minmax GRANULARITY 1

ALTER TABLE hits_UserID_URL MATERIALIZE INDEX url_len_minmax;
```

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE length(URL) > 1000
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Note：

1. minmax 适用于局部区间内的极值，比如日期、数量、长度等




```
aae564a16) (SelectExecutor): Filtering marks by primary and secondary keys
 aae564a16) (SelectExecutor): Index `url_skipping_index_len_minmax` has dropped 166/1083 granules.
 aae564a16) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1083/1083 marks by primary key, 917 marks to read from 139 ranges
 aae564a16) (SelectExecutor): Spreading mark ranges among streams (default reading)
 aae564a16) (SelectExecutor): Reading approx. 7507809 rows with 8 streams
```

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

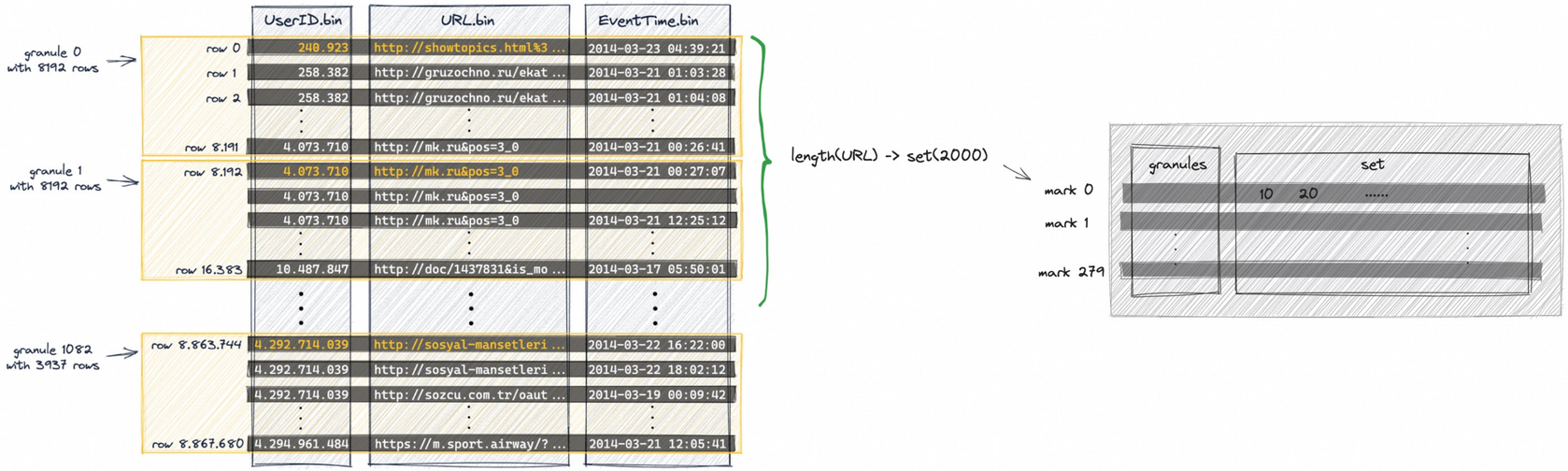
2. Skipping Indexes - set

```
ALTER TABLE hits_UserID_URL ADD INDEX url_set_len length(URL) TYPE set(2000) GRANULARITY 4
ALTER TABLE hits_UserID_URL MATERIALIZE INDEX url_set_len;
```

```
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE length(URL) = 500
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Note：

1. set 适用于局部区间内重复率较多的固定数据，比如城市、报错码等




```
9aae564a16) (SelectExecutor): Key condition: unknown
9aae564a16) (SelectExecutor): Filtering marks by primary and secondary keys
9aae564a16) (SelectExecutor): Index `url_set_len` has dropped 459/1083 granules.
9aae564a16) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1083/1083 marks by primary key, 624 marks to read from 64 ranges
9aae564a16) (SelectExecutor): Spreading mark ranges among streams (default reading)
9aae564a16) (SelectExecutor): Reading approx. 5111808 rows with 8 streams
```

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

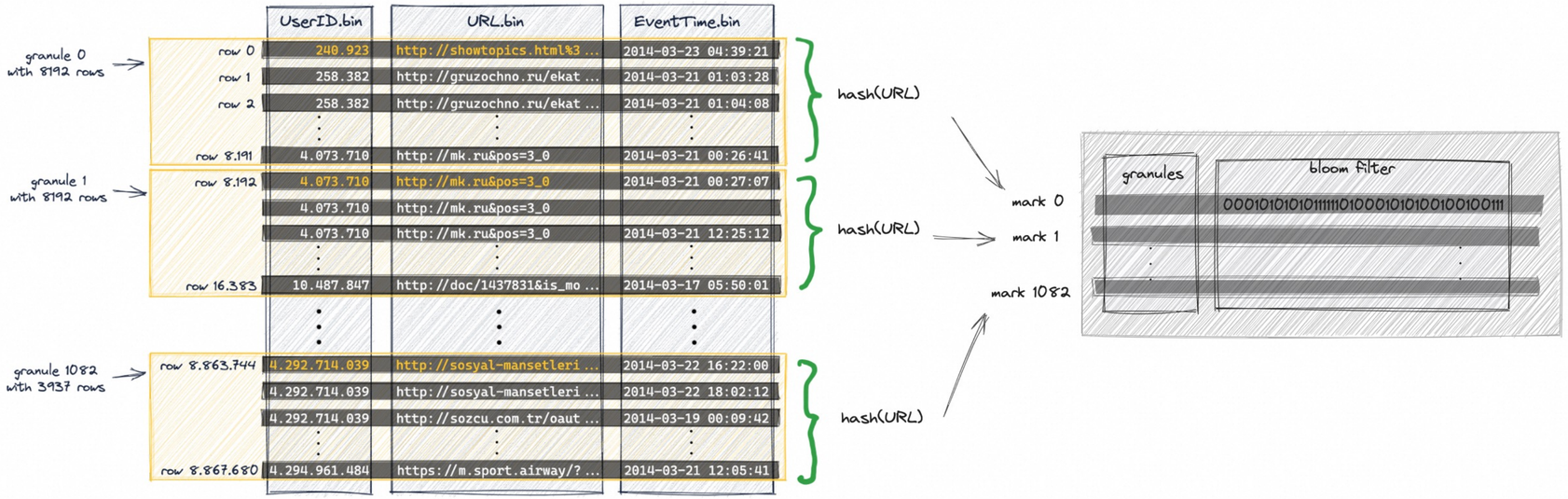
2. Skipping Indexes – bloom filter

```
ALTER TABLE hits_UserID_URL ADD INDEX url_bf URL TYPE bloom_filter GRANULARITY 1
ALTER TABLE hits_UserID_URL MATERIALIZE INDEX url_bf;
```

```
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public/?hash=8xqk3hMN9aDmK6l+K2HMuVhTDUnF0T5ugr7ITsXOPDI%3DfdSMTQ0ZDEwMDkuaHRtbD9
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Note：

1. Bloom filter 适用于局部区间内重复率较多的数据，比如手机号、query、日志等




```
99aae564a16) (SelectExecutor): Filtering marks by primary and secondary keys
 99aae564a16) (SelectExecutor): Used generic exclusion search over index for part all_1_9_2_48 with 1537 steps
 99aae564a16) (SelectExecutor): Index `url_bf` has dropped 1050/1076 granules.
 99aae564a16) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1076/1083 marks by primary key, 26 marks to read from 26 ranges
 99aae564a16) (SelectExecutor): Spreading mark ranges among streams (default reading)
 99aae564a16) (SelectExecutor): Reading approx. 212992 rows with 4 streams
```

Reference: https://clickhouse.com/docs/en/optimize/sparse-primary-indexes

2. Skipping Indexes – tokenbf_v1, ngrambf_v1

```
Token Bloom Filter
The same as ngrambf_v1, but stores tokens instead of ngrams. Tokens are sequences separated by non-alphanumeric characters. - 非字母数字的字符

Syntax: tokenbf_v1(size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)
```

```
N-gram Bloom Filter
Only works with datatypes: String, FixedString and Map.

Syntax: ngrambf_v1(n, size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)
```

```
SELECT tokens('https://www.zanbil.ir/m/filter/b113')

┌─tokens──────────────────────────────────────────────┐
│ ['https','www','zanbil','ir','m','filter','b113'] │
└─────────────────────────────────────────────────────┘

SELECT ngrams('https://www.zanbil.ir/m/filter/b113', 3)

┌─ngrams('https://www.zanbil.ir/m/filter/b113', 3)──────────────────────────────────────────────────────────────────────────────────────────────────────┐
│ ['htt','ttp','tps','ps:','s:/','://','//w','/ww','www','ww.','w.z','.za','zan','anb','nbi','bil','i                                              │
└─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

Note：

1. 当需要执行**文本搜索**时， tokenbf_v1, ngrambf_v1 类型的 skipping index 比较有用，操作符包括 LIKE, IN, hasToken

2. 基于 token 的索引使用非字母数字字符作为分隔符生成 token，这在查询时只能匹配 token (或者说整个单词)

3. 对于更多的粒度匹配，可以使用 N-gram Bloom Filter，把字符串分割成指定大小的子串，从而完成子串匹配

4. 适用于文本搜索

Reference:https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/mergetree

2. Skipping Indexes – tokenbf_v1, ngrambf_v1

```
ALTER TABLE hits_UserID_URL ADD INDEX url_token URL TYPE tokenbf_v1(10480, 20, 20) GRANULARITY 1
OPTIMIZE TABLE hits_UserID_URL FINAL
```

```
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE hasToken(URL, 'c291cmNlPWxjJnV0bV9jYW1wYW5lbWEudWElMkY')
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

```
) AS Count FROM hits_UserID_URL WHERE hasToken(URL, 'c291cmNlPWxjJnV0bV9jYW1wYW5lbWEudWElMkY') GROUP BY UserID ORDER BY Count DESC LIMIT 10 (stage: Complete)


 for moving to the tail of PREWHERE is -1
PWxjJnV0bV9jYW1wYW5lbWEudWElMkY'_String) moved to PREWHERE
) (SelectExecutor): Key condition: unknown
) (SelectExecutor): Filtering marks by primary and secondary keys
) (SelectExecutor): Index `url_token` has dropped 807/1083 granules.
) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1083/1083 marks by primary key, 276 marks to read from 211 ranges
) (SelectExecutor): Spreading mark ranges among streams (default reading)
) (SelectExecutor): Reading approx. 2260992 rows with 8 streams
```

2. Skipping Indexes – 小结

```sql
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10
```

Question：

Q：怎么加速这种查询？
A：Secondary Table、MV、Projection、Skipping Index。

```
INDEX index_name expr TYPE type(...) [GRANULARITY granularity_value]
  - MinMax
  - set(max_rows)
  - bloom_filter([false_positive])
  - tokenbf_v1(size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)
  - ngrambf_v1(n, size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)
```

03 ClickHouse Explain

3. EXPLAIN Statement

```
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 85706ec8-14cb-40b8-b851-85ef0de3f5eb

    ┌─URL────────────────────────────────┬─Count─┐
 1. │ http://auto.ru/chatay-barana...    │   170 │
 2. │ http://auto.ru/chatay-id=371...    │    52 │
 3. │ http://public_search               │    45 │
 4. │ http://kovrik-medvedevushku-...    │    36 │
 5. │ http://forumal                     │    33 │
 6. │ http://korablitz.ru/L_1OFFER...    │    14 │
 7. │ http://auto.ru/chatay-id=371...    │    14 │
 8. │ http://auto.ru/chatay-john-D...    │    13 │
 9. │ http://auto.ru/chatay-john-D...    │    10 │
10. │ http://wot/html?page/23600_m...    │     9 │
    └────────────────────────────────────┴───────┘

10 rows in set. Elapsed: 0.007 sec. Processed 8.19 thousand rows, 740.18 KB (1.10 million rows/s., 99.05 MB/s.)
Peak memory usage: 243.55 KiB.
```

Reference: https://clickhouse.com/docs/en/sql-reference/statements/explain

3. EXPLAIN Statement

Question：

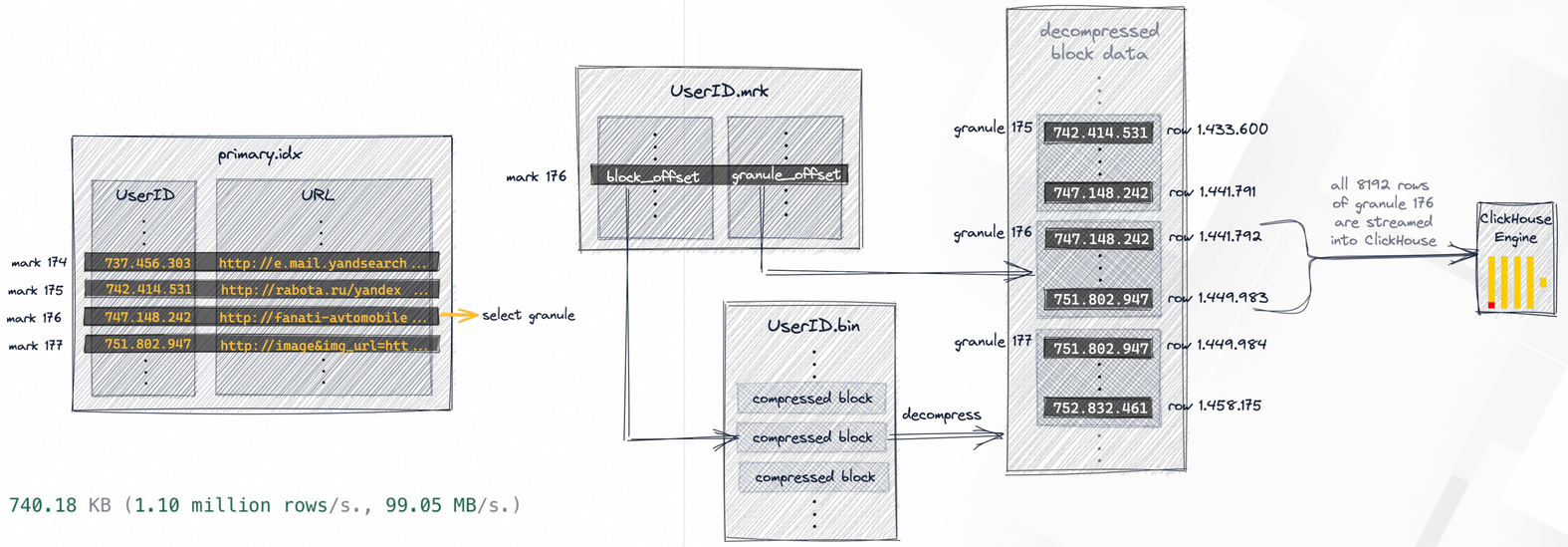
1.怎么了解 SQL 命中了哪些索引呢？索引过滤了多少数据呢？

2.怎么看 SQL 的并行查询情况呢？索引慢是不是因为计算资源不够？

3.怎么优化这些 SQL 呢？是增加跳数索引、MV、或者重新给主键重新排序？

→ **EXPLAIN Statement**

```
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 85706ec8-14cb-40b8-b851-85ef0de3f5eb

    ┌─URL────────────────────────────────┬─Count─┐
 1. │ http://auto.ru/chatay-barana...    │   170 │
 2. │ http://auto.ru/chatay-id=371...    │    52 │
 3. │ http://public_search               │    45 │
 4. │ http://kovrik-medvedevushku-...    │    36 │
 5. │ http://forumal                     │    33 │
 6. │ http://korablitz.ru/L_1OFFER...    │    14 │
 7. │ http://auto.ru/chatay-id=371...    │    14 │
 8. │ http://auto.ru/chatay-john-D...    │    13 │
 9. │ http://auto.ru/chatay-john-D...    │    10 │
10. │ http://wot/html?page/23600_m...    │     9 │
    └────────────────────────────────────┴───────┘

10 rows in set. Elapsed: 0.007 sec. Processed 8.19 thousand rows, 740.18 KB (1.10 million rows/s., 99.05 MB/s.)
Peak memory usage: 243.55 KiB.
```



Reference: https://clickhouse.com/docs/en/sql-reference/statements/explain

3. EXPLAIN Statement

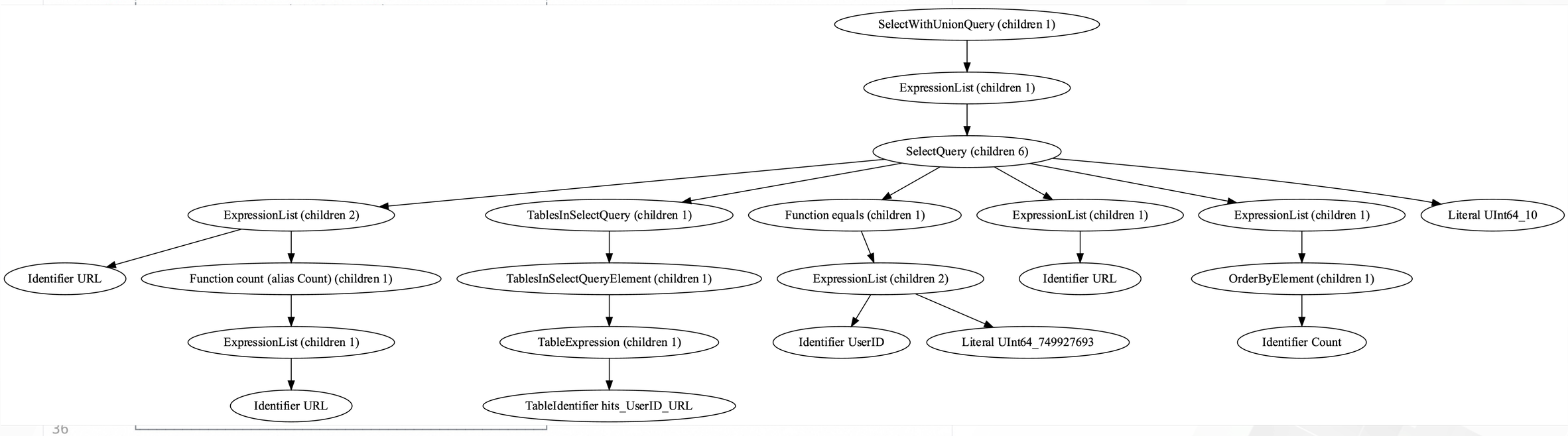
```
EXPLAIN [AST | SYNTAX | QUERY TREE | PLAN | PIPELINE | ESTIMATE | TABLE OVERRIDE] [setting = value, ...]
    [
      SELECT ... |
      tableFunction(...) [COLUMNS (...)] [ORDER BY ...] [PARTITION BY ...] [PRIMARY KEY] [SAMPLE BY ...] [TTL ...]
    ]
    [FORMAT ...]
```

```
EXPLAIN Types
  AST — Abstract syntax tree.
  SYNTAX — Query text after AST-level optimizations.
  PLAN — Query execution plan.
  PIPELINE — Query execution pipeline.
```

Reference: https://clickhouse.com/docs/en/sql-reference/statements/explain

3.1 EXPLAIN AST

```
EXPLAIN AST
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: a238bed9-b0fa-4db2-8d07-c81a91e0f84a

┌─explain─────────────────────────────────────┐
1. │ SelectWithUnionQuery (children 1)           │
```



https://dreampuf.github.io/GraphvizOnline

3.2 EXPLAIN SYNTAX

```
EXPLAIN SYNTAX
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: f732feb6-0bc1-4d91-b90d-10079d1308af

    ┌─explain─────────────────────┐
 1. │ SELECT                      │
 2. │     URL,                    │
 3. │     count(URL) AS Count     │
 4. │ FROM hits_UserID_URL        │
 5. │ WHERE UserID = 749927693    │
 6. │ GROUP BY URL                │
 7. │ ORDER BY Count DESC         │
 8. │ LIMIT 10                    │
    └─────────────────────────────┘

8 rows in set. Elapsed: 0.001 sec.
```

```
EXPLAIN SYNTAX
SELECT *
FROM hits_UserID_URL AS a, hits_UserID_URL AS b
LIMIT 10

Query id: b829740a-f89f-4d22-b9c9-4a8c0877b6fd

    ┌─explain─────────────────────┐
 1. │ SELECT                      │
 2. │     UserID,                 │
 3. │     URL,                    │
 4. │     EventTime,              │
 5. │     b.UserID,               │
 6. │     b.URL,                  │
 7. │     b.EventTime             │
 8. │ FROM hits_UserID_URL AS a   │
 9. │ CROSS JOIN hits_UserID_URL AS b │
10. │ LIMIT 10                    │
    └─────────────────────────────┘

10 rows in set. Elapsed: 0.002 sec.
```

3.3 EXPLAIN PLAN

Plan 就是执行计划，默认的 explain 语句就是执行的 explain plan，查看 SQL 执行计划。

```
EXPLAIN
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 34877b39-7118-41f4-a246-16eda100404b

     ┌─explain──────────────────────────────────────────────┐
  1. │ Expression (Project names)                           │
  2. │   Limit (preliminary LIMIT (without OFFSET))         │
  3. │     Sorting (Sorting for ORDER BY)                   │
  4. │       Expression ((Before ORDER BY + Projection))    │
  5. │         Aggregating                                  │
  6. │           Expression (Before GROUP BY)               │
  7. │             Expression                               │
  8. │               ReadFromMergeTree (default.hits_UserID_URL) │
     └──────────────────────────────────────────────────────┘


8 rows in set. Elapsed: 0.002 sec.
```

```
EXPLAIN actions = 1
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10
Query id: f029496d-5ad8-4c59-8130-01f02e0c2839
     ┌─explain──────────────────────────────────────────────
  1. │ Expression (Project names)
  2. │ Actions: INPUT : 0 -> count(__table1.URL) UInt64 : 0
  3. │          INPUT : 1 -> __table1.URL String : 1
  4. │          ALIAS count(__table1.URL) :: 0 -> Count UInt64 : 2
  5. │          ALIAS __table1.URL :: 1 -> URL String : 0
  6. │ Positions: 0 2
  7. │   Limit (preliminary LIMIT (without OFFSET))
  8. │   Limit 10
  9. │   Offset 0
 10. │     Sorting (Sorting for ORDER BY)
 11. │     Sort description: count(__table1.URL) DESC
 12. │     Limit 10
 13. │       Expression ((Before ORDER BY + Projection))
 14. │       Actions: INPUT :: 0 -> count(__table1.URL) UInt64 : 0
 15. │                INPUT :: 1 -> __table1.URL String : 1
 16. │       Positions: 0 1
 17. │         Aggregating
 18. │         Keys: __table1.URL
 19. │         Aggregates:
 20. │             count(__table1.URL)
 21. │               Function: count(String) → UInt64
 22. │               Arguments: __table1.URL
 23. │         Skip merging: 0
 24. │           Expression (Before GROUP BY)
 25. │           Actions: INPUT :: 0 -> __table1.URL String : 0
 26. │           Positions: 0
 27. │             Expression
 28. │             Actions: INPUT : 0 -> URL String : 0
 29. │                      INPUT :: 1 -> UserID UInt32 : 1
 30. │                      ALIAS URL :: 0 -> __table1.URL String : 2
 31. │             Positions: 2
 32. │               ReadFromMergeTree (default.hits_UserID_URL)
 33. │               ReadType: Default
 34. │               Parts: 1
 35. │               Granules: 1
 36. │               Prewhere info
 37. │               Need filter: 1
 38. │                 Prewhere filter
 39. │                 Prewhere filter column: equals(__table1.UserID, 749927693_UInt32) (removed)
 40. │                 Actions: INPUT : 0 -> UserID UInt32 : 0
 41. │                          COLUMN Const(UInt32) -> 749927693_UInt32 UInt32 : 1
 42. │                          FUNCTION equals(UserID : 0, 749927693_UInt32 :: 1) -> equals(__table1.UserID, 749927693_UInt32) U
 43. │                 Positions: 0 2
     └──────────────────────────────────────────────────────
```

3.3 EXPLAIN PLAN

Plan 就是执行计划，默认的 explain 语句就是执行的 explain plan，查看 SQL 执行计划。

```
EXPLAIN
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 34877b39-7118-41f4-a246-16eda100404b

    ┌─explain──────────────────────────────────────────────────────┐
 1. │ Expression (Project names)                                   │
 2. │   Limit (preliminary LIMIT (without OFFSET))                 │
 3. │     Sorting (Sorting for ORDER BY)                           │
 4. │       Expression ((Before ORDER BY + Projection))            │
 5. │         Aggregating                                          │
 6. │           Expression (Before GROUP BY)                       │
 7. │             Expression                                       │
 8. │               ReadFromMergeTree (default.hits_UserID_URL)    │
    └──────────────────────────────────────────────────────────────┘

8 rows in set. Elapsed: 0.002 sec.
```

```
EXPLAIN indexes = 1
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 43bbf6d0-48a5-4f64-9f4e-39c3aa3809d6

    ┌─explain──────────────────────────────────────────────────────┐
 1. │ Expression (Project names)                                   │
 2. │   Limit (preliminary LIMIT (without OFFSET))                 │
 3. │     Sorting (Sorting for ORDER BY)                           │
 4. │       Expression ((Before ORDER BY + Projection))            │
 5. │         Aggregating                                          │
 6. │           Expression (Before GROUP BY)                       │
 7. │             Expression                                       │
 8. │               ReadFromMergeTree (default.hits_UserID_URL)    │
 9. │               Indexes:                                       │
10. │                 PrimaryKey                                   │
11. │                   Keys:                                      │
12. │                     UserID                                   │
13. │                   Condition: (UserID in [749927693, 749927693])
14. │                   Parts: 1/1                                 │
15. │                   Granules: 1/1083                           │
    └──────────────────────────────────────────────────────────────┘
```

```
8-a48d-c99aae564a16) (SelectExecutor): Key condition: (column 0 in [749927693, 749927693])
8-a48d-c99aae564a16) (SelectExecutor): Filtering marks by primary and secondary keys
8-a48d-c99aae564a16) (SelectExecutor): Running binary search on index range for part all_1_9_2 (1083 marks)
8-a48d-c99aae564a16) (SelectExecutor): Found (LEFT) boundary mark: 176
8-a48d-c99aae564a16) (SelectExecutor): Found (RIGHT) boundary mark: 177
8-a48d-c99aae564a16) (SelectExecutor): Found continuous range in 19 steps
8-a48d-c99aae564a16) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 1/1083 marks by primary key, 1 marks to read from 1 ranges
8-a48d-c99aae564a16) (SelectExecutor): Spreading mark ranges among streams (default reading)
8-a48d-c99aae564a16) (SelectExecutor): Reading 1 ranges in order from part all_1_9_2, approx. 8192 rows starting from 1441792
```

3.4 EXPLAIN PIPELINE

```
EXPLAIN PIPELINE
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 1aa3dc5a-aae0-434e-a9f0-075f3a4705a2

    ┌─explain─────────────────────────────────────────────────────────────────────────────────────────────┐
 1. │ (Expression)                                                                                        │
 2. │ ExpressionTransform                                                                                 │
 3. │   (Limit)                                                                                           │
 4. │   Limit                                                                                             │
 5. │     (Sorting)                                                                                       │
 6. │     MergingSortedTransform 8 → 1                                                                    │
 7. │       MergeSortingTransform × 8                                                                     │
 8. │         LimitsCheckingTransform × 8                                                                 │
 9. │           PartialSortingTransform × 8                                                               │
10. │             (Expression)                                                                            │
11. │             ExpressionTransform × 8                                                                 │
12. │               (Aggregating)                                                                         │
13. │               Resize 1 → 8                                                                          │
14. │                 AggregatingTransform                                                                │
15. │                   (Expression)                                                                      │
16. │                   ExpressionTransform                                                               │
17. │                     (Expression)                                                                    │
18. │                     ExpressionTransform                                                             │
19. │                       (ReadFromMergeTree)                                                           │
20. │                       MergeTreeSelect(pool: ReadPoolInOrder, algorithm: InOrder) 0 → 1              │
    └─────────────────────────────────────────────────────────────────────────────────────────────────────┘

20 rows in set. Elapsed: 0.003 sec.
```

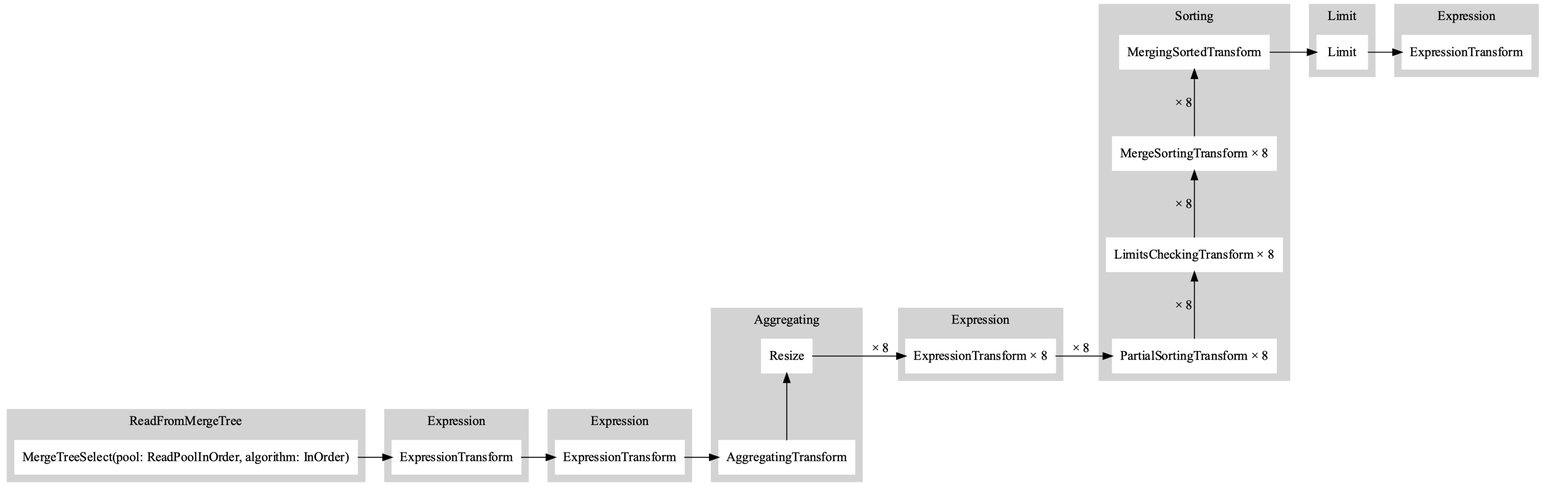
3.4 EXPLAIN PIPELINE

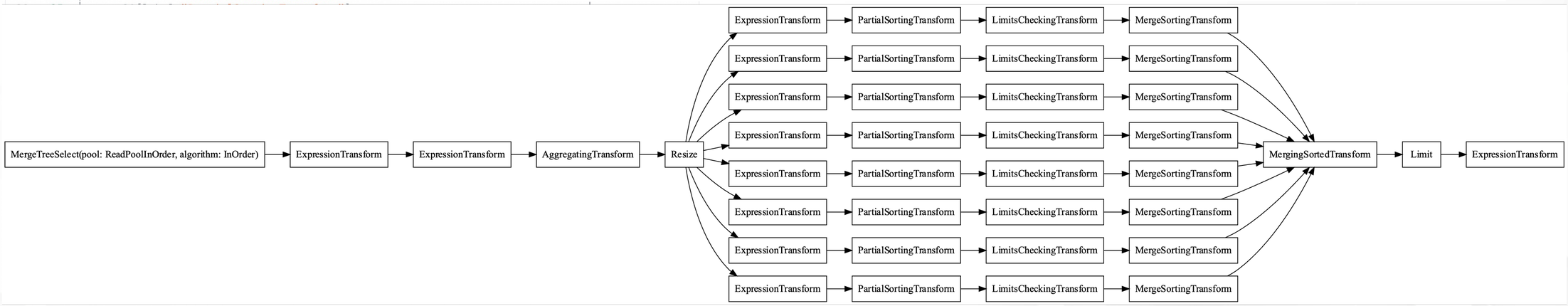
```
EXPLAIN PIPELINE graph = 1, compact = 0
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 43ed1457-9cb9-450a-845e-25b70163ecfe
```

1.Explain pipeline graph=1

2.Explain pipeline graph=1, compact=0

```
┌─explain──────────────────────────────────────────────────┐
1. │ digraph                                                  │
2. │ {                                                        │
3. │   rankdir="LR";                                          │
4. │   { node [shape = rect]                                  │
5. │     n0[label="MergeTreeSelect(pool: ReadPoolInO          │
6. │     n1[label="ExpressionTransform"];                     │
7. │     n2[label="ExpressionTransform"];                     │
8. │     n3[label="AggregatingTransform"];                    │
9. │     n4[label="Resize"];                                  │
10. │     n5[label="ExpressionTransform"];                    │
11. │     n6[label="ExpressionTransform"];                    │
12. │     n7[label="ExpressionTransform"];                    │
13. │     n8[label="ExpressionTransform"];                    │
14. │     n9[label="ExpressionTransform"];                    │
15. │     n10[label="ExpressionTransform"];                   │
16. │     n11[label="ExpressionTransform"];                   │
17. │     n12[label="ExpressionTransform"];                   │
18. │     n13[label="PartialSortingTransform"];               │
19. │     n14[label="PartialSortingTransform"];               │
20. │     n15[label="PartialSortingTransform"];               │
21. │     n16[label="PartialSortingTransform"];               │
22. │     n17[label="PartialSortingTransform"];               │
23. │     n18[label="PartialSortingTransform"];               │
24. │     n19[label="PartialSortingTransform"];               │
```





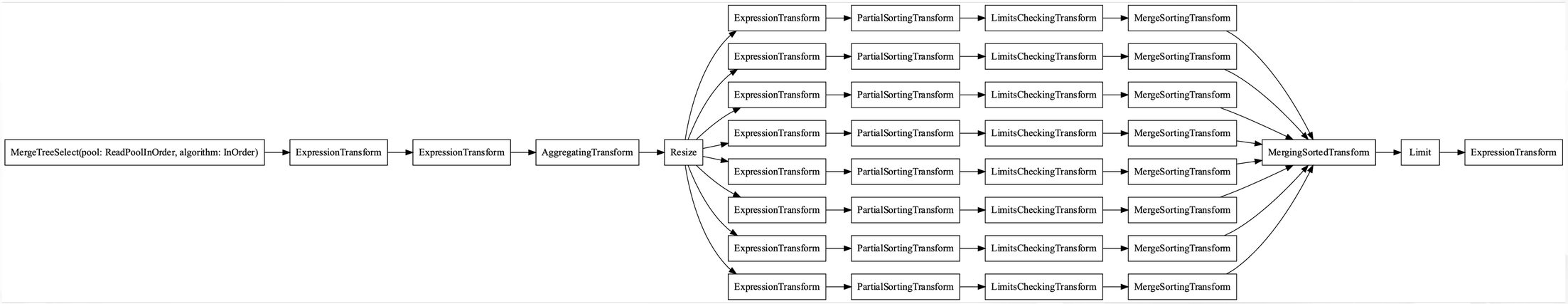
3.4 EXPLAIN PIPELINE

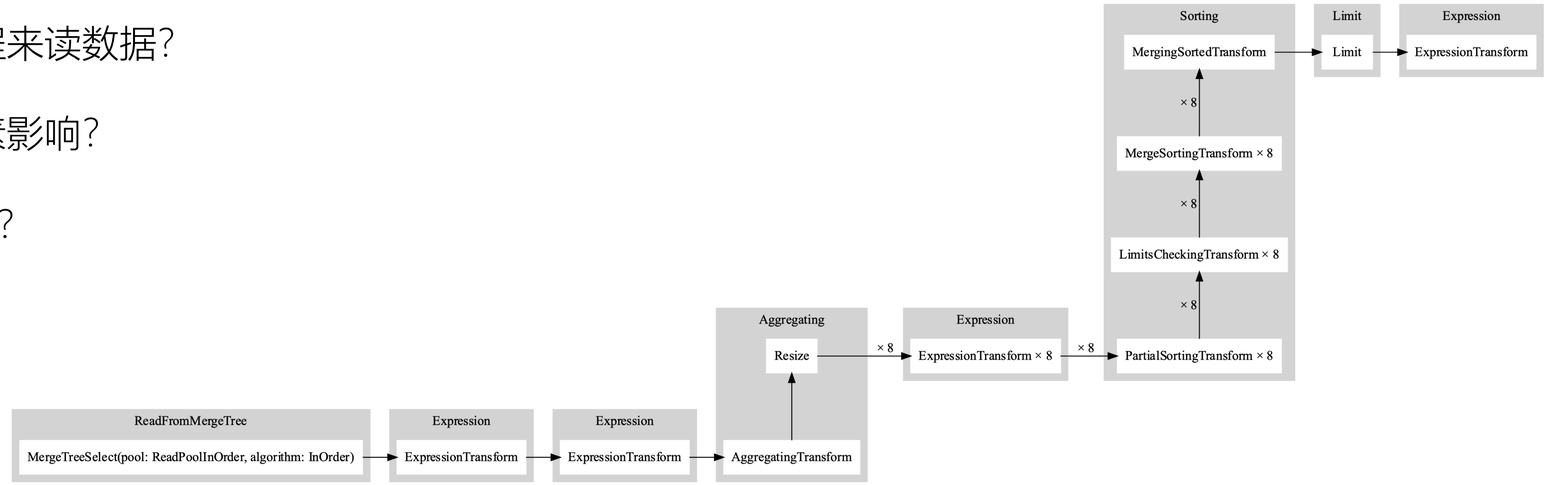
Question：

1. 为什么只有单线程来读数据？

2. 线程数受什么因素影响？

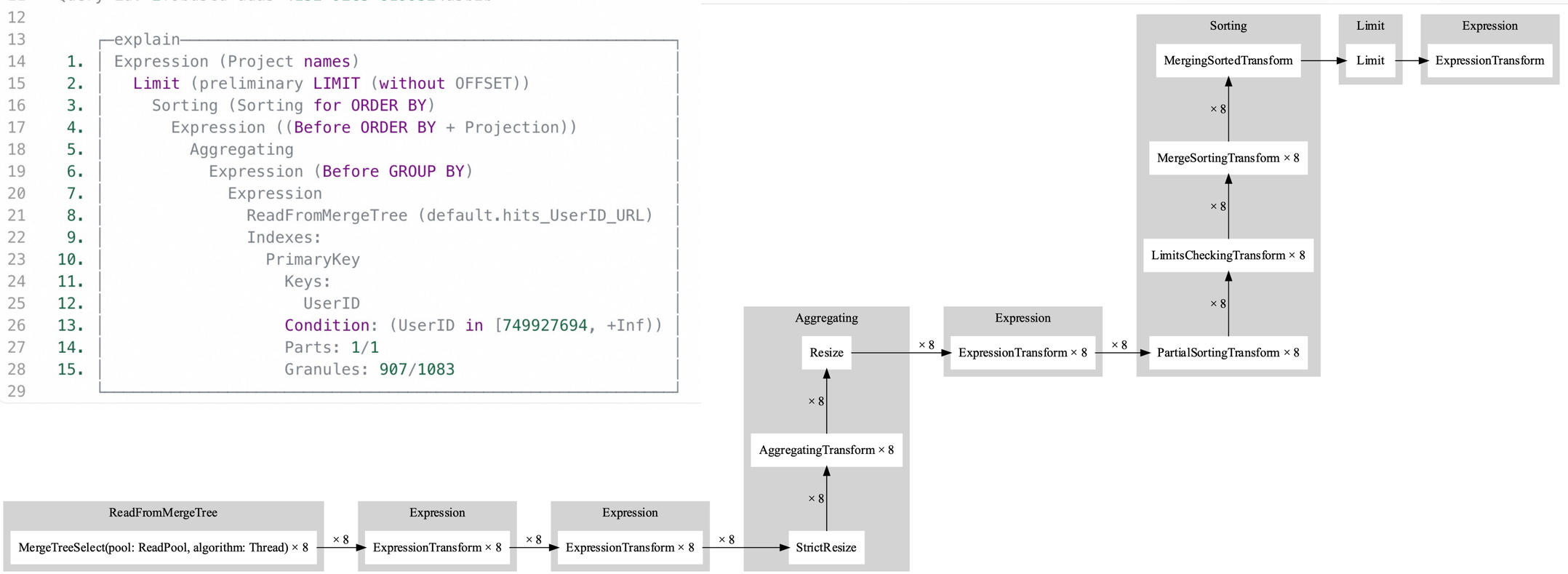
3.对性能有多大影响？





3.4 EXPLAIN PIPELINE

```
EXPLAIN indexes = 1
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID > 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 2f9bd5cd-ada3-4132-92e3-8199324a5b2b

┌─explain─────────────────────────────────────────────┐
 1. │ Expression (Project names)                      │
 2. │   Limit (preliminary LIMIT (without OFFSET))    │
 3. │     Sorting (Sorting for ORDER BY)              │
 4. │       Expression ((Before ORDER BY + Projection)) │
 5. │         Aggregating                             │
 6. │           Expression (Before GROUP BY)          │
 7. │             Expression                          │
 8. │               ReadFromMergeTree (default.hits_UserID_URL) │
 9. │               Indexes:                          │
10. │                 PrimaryKey                      │
11. │                   Keys:                         │
12. │                     UserID                      │
13. │                   Condition: (UserID in [749927694, +Inf)) │
14. │                   Parts: 1/1                    │
15. │                   Granules: 907/1083            │
    └─────────────────────────────────────────────────┘
```



3.4 EXPLAIN PIPELINE

```
explain
1. Expression (Project names)
2.   Limit (preliminary LIMIT (without OFFSET))
3.     Sorting (Sorting for ORDER BY)
4.       Expression ((Before ORDER BY + Projection))
5.         Aggregating
6.           Expression (Before GROUP BY)
7.             Expression
8.               ReadFromMergeTree (default.hits_UserID_URL)
9.               Indexes:
10.                PrimaryKey
11.                  Keys:
12.                    UserID
13.                  Condition: (UserID in [749927694, +Inf))
14.                  Parts: 1/1
15.                  Granules: 907/1083
```

Question：

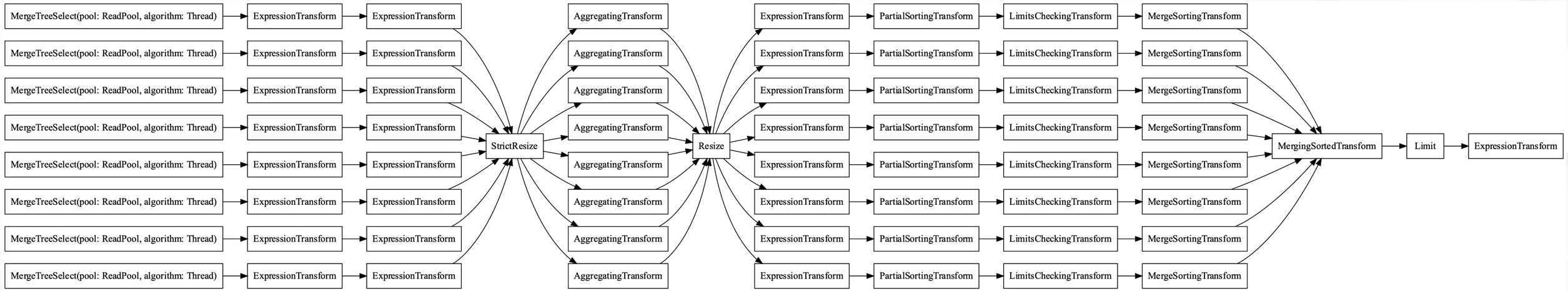
1. 为什么只有单线程来读数据?

2. 线程数受什么因素影响?

3.对性能有多大影响?

1. merge_tree_min_rows_for_concurrent_read [default 163840 - 20 granules]

2. merge_tree_min_bytes_for_concurrent_read [default 251658240 - 240MB]



3.4 EXPLAIN PIPELINE

```
explain
1. Expression (Project names)
2.   Limit (preliminary LIMIT (without OFFSET))
3.     Sorting (Sorting for ORDER BY)
4.       Expression ((Before ORDER BY + Projection))
5.         Aggregating
6.           Expression (Before GROUP BY)
7.             Expression
8.               ReadFromMergeTree (default.hits_UserID_URL)
9.               Indexes:
10.                PrimaryKey
11.                  Keys:
12.                    UserID
13.                  Condition: (UserID in [749927694, +Inf))
14.                  Parts: 1/1
15.                  Granules: 907/1083
```

Question：

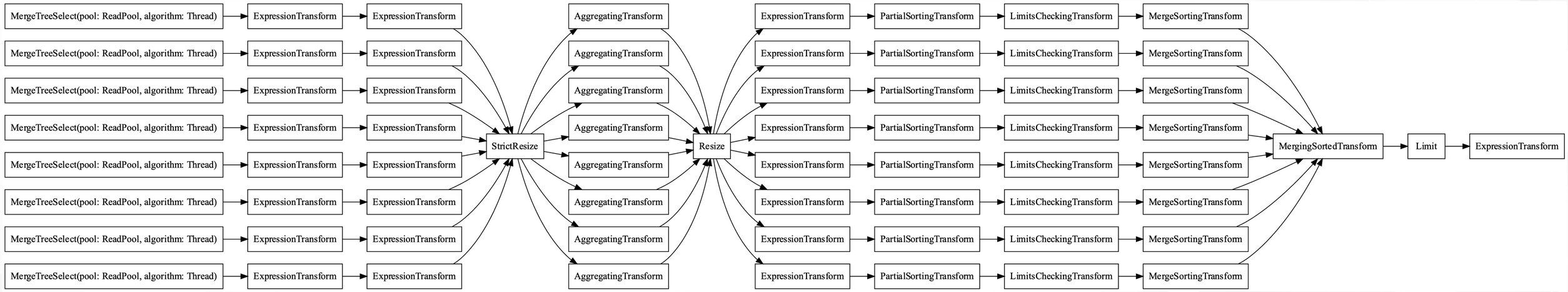
1. 为什么只有单线程来读数据?

2. 线程数受什么因素影响?

3.对性能有多大影响?

1. max_threads [default CPU cores]

```
SHOW SETTINGS LIKE 'max_threads'

Query id: b195e9ac-5a0f-49e3-a7da-2454f268305f

┌─name────────┬─type───────┬─value─┐
│ max_threads │ MaxThreads │ auto(8) │
└─────────────┴────────────┴─────────┘
```



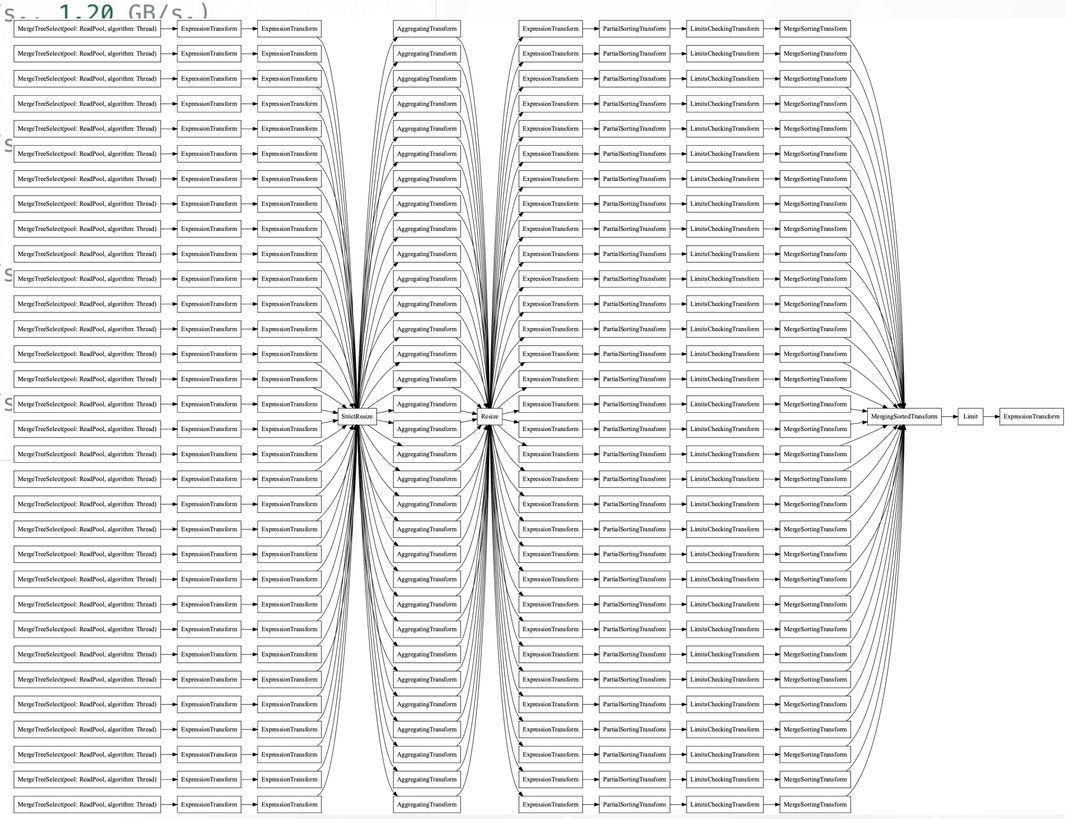
3.4 EXPLAIN PIPELINE

```sql
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID > 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10 settings max_threads = N;
```

```
SETTINGS max_threads = 8
10 rows in set. Elapsed: 0.559 sec. Processed 7.43 million rows, 672.92 MB (13.28 million rows/s., 1.20 GB/s.)
Peak memory usage: 623.05 MiB.

SETTINGS max_threads = 16
10 rows in set. Elapsed: 0.535 sec. Processed 7.43 million rows, 672.92 MB (13.89 million rows/s
Peak memory usage: 666.82 MiB.

SETTINGS max_threads = 32
10 rows in set. Elapsed: 0.586 sec. Processed 7.43 million rows, 672.92 MB (12.67 million rows/s
Peak memory usage: 774.41 MiB.

SETTINGS max_threads = 64
10 rows in set. Elapsed: 0.622 sec. Processed 7.43 million rows, 672.92 MB (11.94 million rows/s
Peak memory usage: 854.14 MiB.
```

Question：

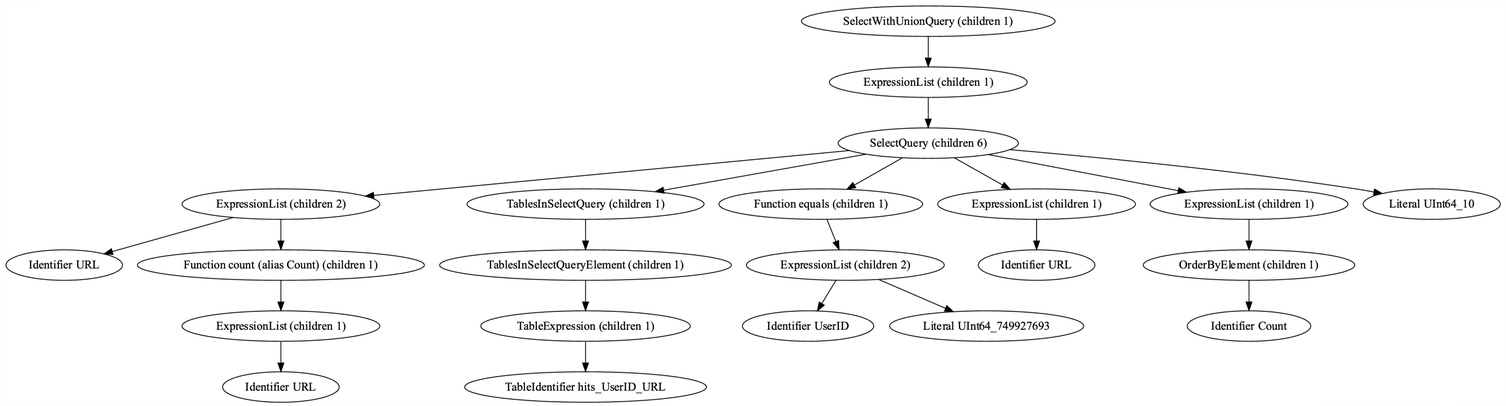
1. 为什么只有单线程来读数据?
2. 线程数受什么因素影响?
3. 对性能有多大影响?



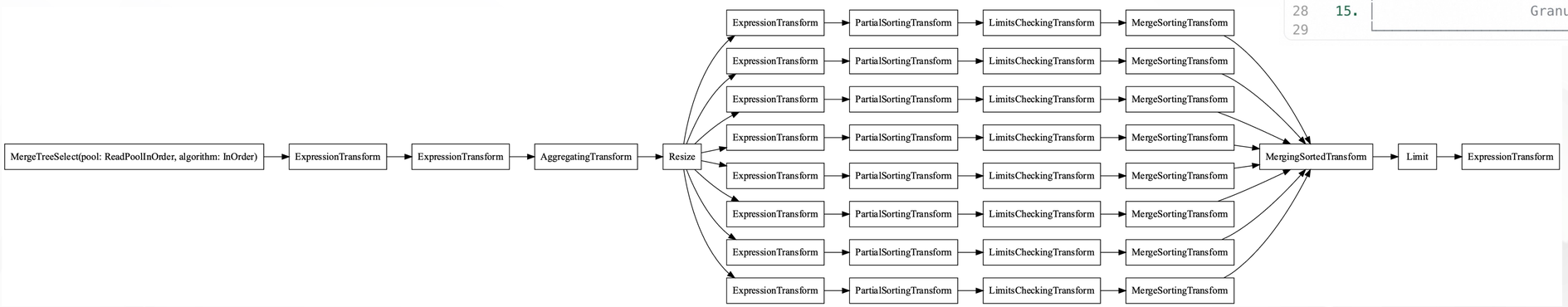
1. max_threads 越大，并行执行线程越多；
2. 并行线程越多，同一时刻读到内存的数据越多，使用内存量越大；
3. Max_threads 超过 CPU cores 越多，线程上下文切换带来的影响越大，执行效率越低。

3. EXPLAIN Statement 小结

```
EXPLAIN Types
    AST – Abstract syntax tree.
    SYNTAX – Query text after AST-level optimizations.
    PLAN – Query execution plan.
    PIPELINE – Query execution pipeline.
```




```
EXPLAIN indexes = 1
SELECT
    URL,
    count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10

Query id: 43bbf6d0-48a5-4f64-9f4e-39c3aa3809d6

    ┌─explain──────────────────────────────────────────────────────────────────┐
 1. │ Expression (Project names)                                               │
 2. │   Limit (preliminary LIMIT (without OFFSET))                             │
 3. │     Sorting (Sorting for ORDER BY)                                       │
 4. │       Expression ((Before ORDER BY + Projection))                        │
 5. │         Aggregating                                                      │
 6. │           Expression (Before GROUP BY)                                   │
 7. │             Expression                                                   │
 8. │               ReadFromMergeTree (default.hits_UserID_URL)                │
 9. │               Indexes:                                                   │
10. │                 PrimaryKey                                               │
11. │                   Keys:                                                  │
12. │                     UserID                                               │
13. │                   Condition: (UserID in [749927693, 749927693])          │
14. │                   Parts: 1/1                                             │
15. │                   Granules: 1/1083                                       │
    └──────────────────────────────────────────────────────────────────────────┘
```



04 ClickHouse Optimization

4. Optimization

步骤：

1. 根据最频繁的业务场景，调整 order by key 的顺序；
2. 根据数据分布，增加 skipping indexes；
3. 根据特殊业务场景，增加第二张表、物化视图、Projection；
4. 根据 query plan，查看实际执行步骤、索引命中情况；
5. 根据 query pipeline，查看 plan 中每一步的并行执行情况。

```
EXPLAIN indexes = 1
SELECT
    UserID,
    count(UserID) AS Count
FROM hits_UserID_URL
WHERE URL = 'http://public_search'
GROUP BY UserID
ORDER BY Count DESC
LIMIT 10

Query id: 600f5e3d-a8e6-4a84-b36f-7015276248ec

┌─explain─────────────────────────────────────────────────────────────────────────────────────────────────────────┐
 1. │ Expression (Project names)                                                                                  │
 2. │   Limit (preliminary LIMIT (without OFFSET))                                                                │
 3. │     Sorting (Sorting for ORDER BY)                                                                          │
 4. │       Expression ((Before ORDER BY + Projection))                                                           │
 5. │         Aggregating                                                                                         │
 6. │           Expression (Before GROUP BY)                                                                      │
 7. │             Expression                                                                                      │
 8. │               ReadFromMergeTree (default.hits_UserID_URL)                                                   │
 9. │               Indexes:                                                                                      │
10. │                 PrimaryKey                                                                                  │
11. │                   Keys:                                                                                     │
12. │                     URL                                                                                     │
13. │                   Condition: (URL in ['http://public_search', 'http://public_search'])                       │
14. │                   Parts: 1/1                                                                                │
15. │                   Granules: 1076/1083                                                                       │
16. │                 Skip                                                                                        │
17. │                   Name: url_skipping_index                                                                  │
18. │                   Description: minmax GRANULARITY 4                                                         │
19. │                   Parts: 1/1                                                                                │
20. │                   Granules: 1076/1076                                                                       │
└─────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

20 rows in set. Elapsed: 0.004 sec.
```

05 Q&A

奥运会全球指定云服务商