Summaries in Quantitative Finance
No. 6

A Practitioner's Guide to ClickHouse

Yan Zeng

A Practitioner's Guide to ClickHouse

Yan Zeng, version 1.0, last updated on 2/25/2023

Downloadable at https://leanpub.com/sqf6-clickhouse-guide

Contents

Goals

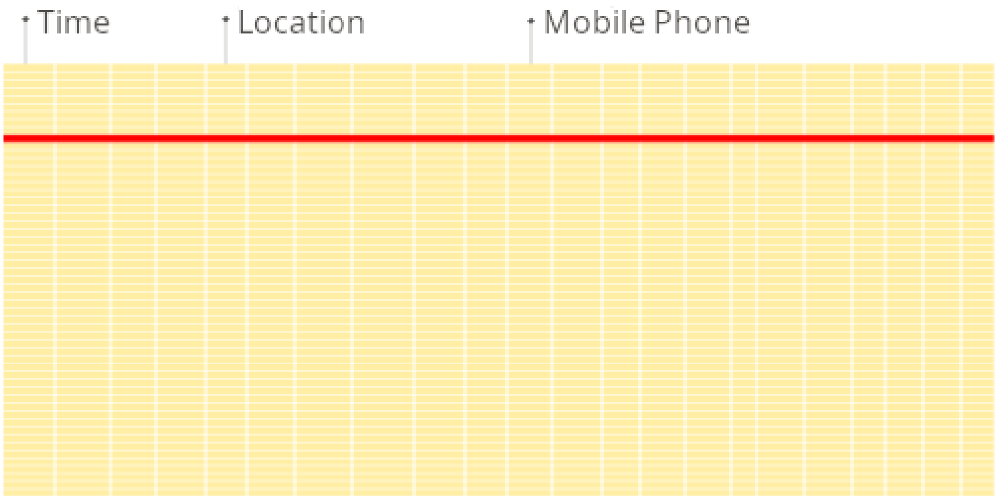
- Provide a self-contained introduction to the inner working of ClickHouse
    - How are data stored?
    - How are data queried?
- Design a suitable table schema for equity tick data
- Intended audience: engineers implementing ClickHouse; quants using ClickHouse

References

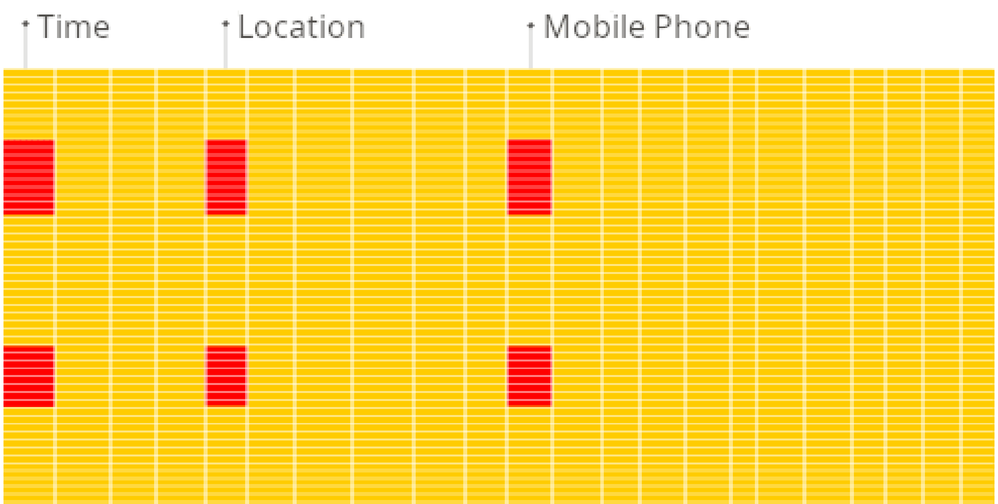
- 朱凯：《ClickHouse 原理解析与应用实践》，机械工业出版社，2020.
- Vijay Anand: *Up and Running with ClickHouse*, BPB Publications, India, 2020.
- ClickHouse Official Documentation: https://clickhouse.com/docs/en/intro

Row-Based DBMS vs. Column-Based DBMS

- Row-based DBMS



- Column-based DBMS



Sample Equity Tick Data

- Data Source: https://firstratedata.com/tick-data
- Sample data: trades of AAPL and MSFT on 2020-01-02

	id	timestamp	price	volume	exchange_code	trade_code
0	MSFT	2020-01-02 04:00:00.037305	158.70	300	8	E-M
1	AAPL	2020-01-02 04:00:00.067010	295.05	3801	8	E-M
2	MSFT	2020-01-02 04:00:00.268687	158.70	150	8	E
3	MSFT	2020-01-02 04:00:00.666680	158.50	10	8	S-E-B-M
4	AAPL	2020-01-02 04:00:02.831485	295.08	1	8	S-E-B-M

- Table schema for trade data:

table	name	type
equity_tickdata	id	String
equity_tickdata	timestamp	DateTime64
equity_tickdata	price	Float64
equity_tickdata	volume	Int64
equity_tickdata	exchange_code	Int32
equity_tickdata	trade_code	String

- Table creation for trade data

```
CREATE TABLE equity_tickdata (
    `id` String,
    `timestamp` DateTime64,
    `price` Float64,
    `volume` Int64,
    `exchange_code Int32,
    `trade_code` String,
) ENGINE = MergeTree
PARTITION BY toYYYYMMDD(timestamp)
PRIMARY KEY (id, timestamp)
ORDER BY (id, timestamp)
SETTING index_granularity = 8192
```

Basic Statistics of One Day's Tick Data

- Based on Bloomberg BPIPE equity tick data (trades & quotes combined) on 2022-05-27
  - ~19K distinct tickers
  - ~136 million distinct time stamps
  - ~595 million rows
  - ~7.4 GB of compressed data and ~55.5 GB uncompressed data
  - These numbers allow back-of-envelop estimation of query efficiency (see later)

- Equity tick data is huge, so that storage and queries need to be extremely efficient

Key Concepts in ClickHouse: Partition, Primary Key, Order BY, Skip Index

Conceptually,

- `Partition`: directory for physical storage of data
- `Order By`: sort rows by lexicographic order of sort keys
- `Primary Key`: for indexing data location
- `Data Skipping Index`: additional data indexing

Physically,

- Data are divided by "partitions" (directories)
- Within each partition, column data are stored separately in `[Column].bin`
- Rows are in lexicographic ascending order by the primary key columns (and the additional sort key columns)
- Rows from different columns are matched via `[Column].mrk`

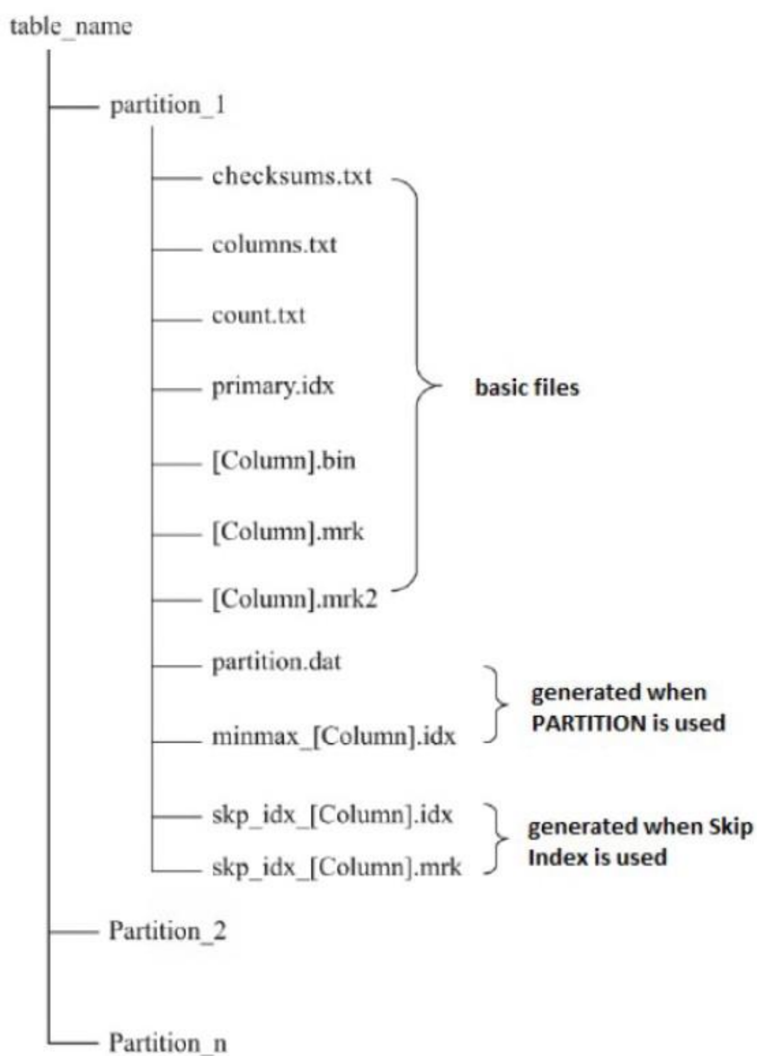
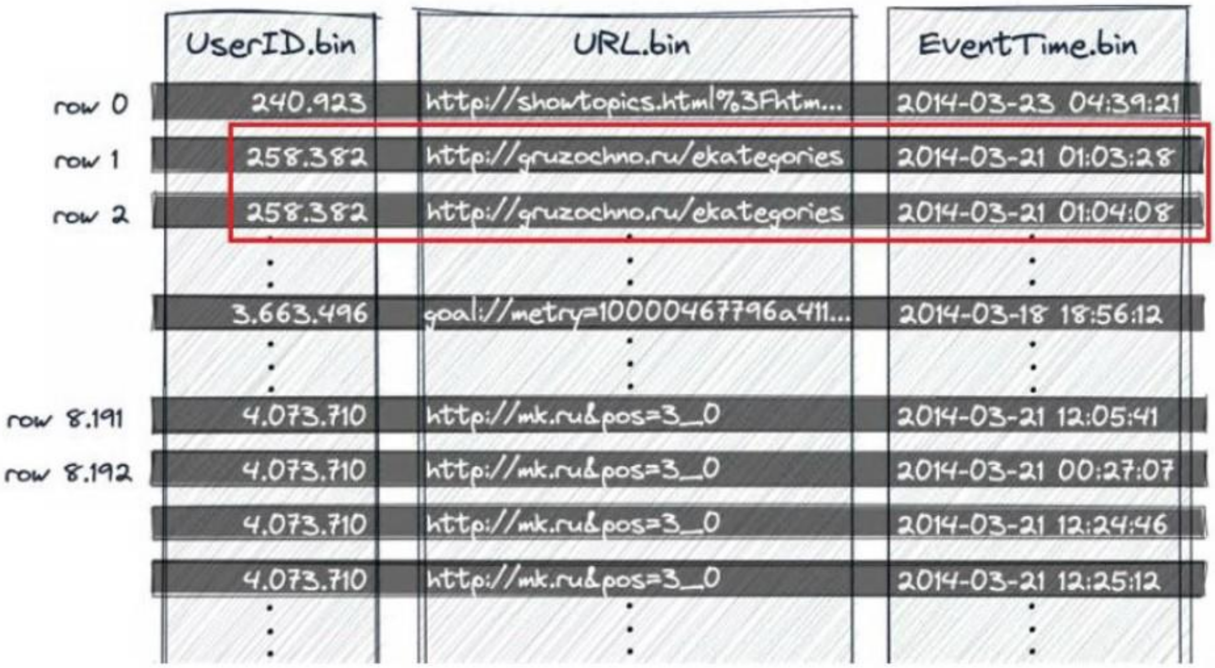


Illustration of Data Storage in ClickHouse

Using web browsing data as an illustration: `UserID`, `URL`, `EventTime`

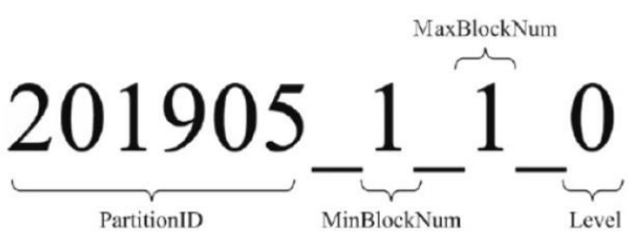
- `PRIMARY KEY (UserID, URL) ORDER BY (UserID, URL, EventTime)`
- Web browsing data are sorted first by `UserID`, then by `URL`, and lastly be `EventTime`



Partition

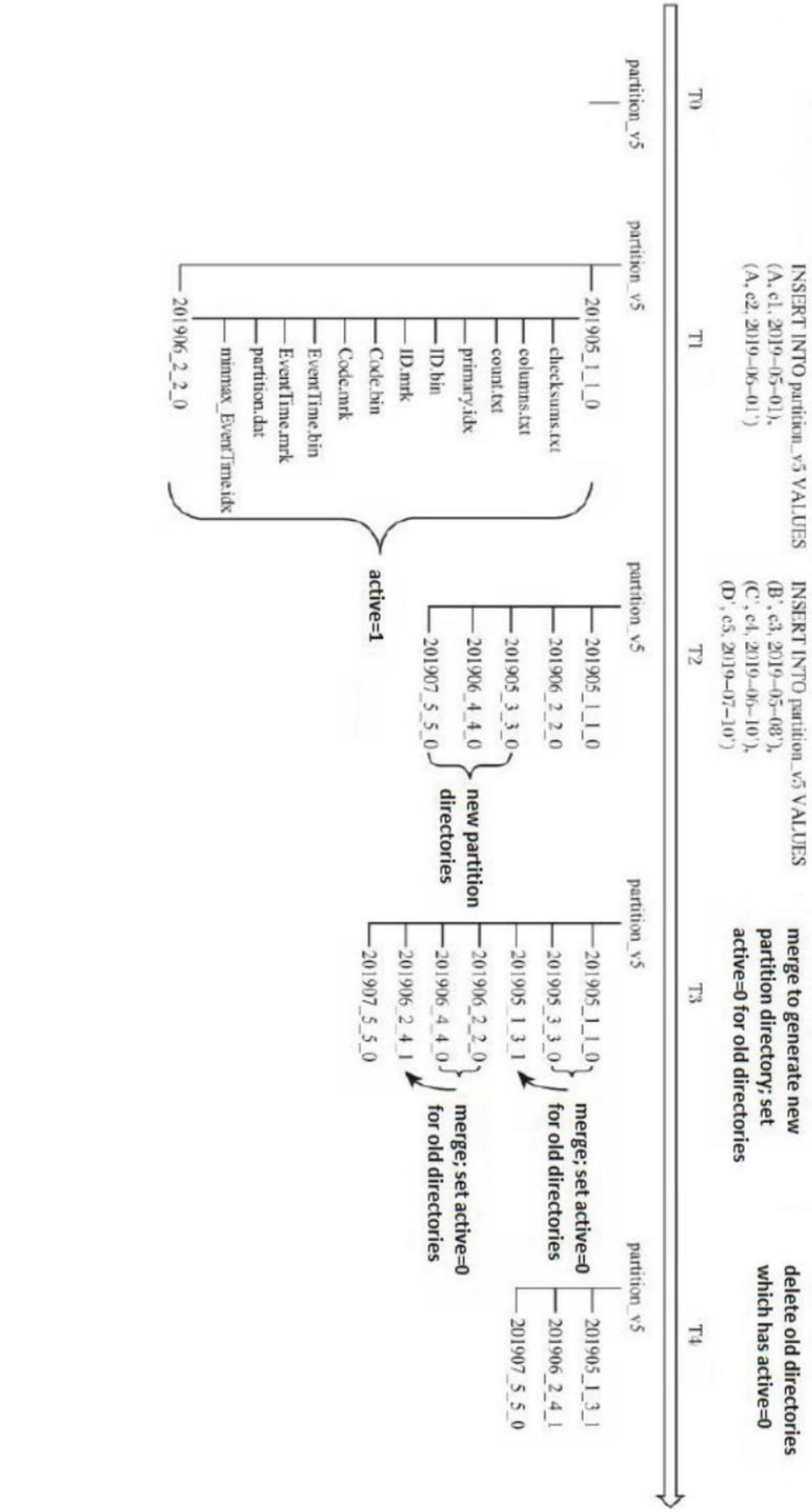
Basics

- Data are written to disk simultaneously so that table insertion is fast
- As a result, multiple directories for the same partition are created, and then merged (10-15 min. after insertion); old directories will then be deleted (~8 min. after merging)
- `MinBlockNum, MaxBlockNum`: global counter across partitions, increase by 1 if a new partition directory is generated
- `Level`: the number of merging for a particular partition; local counter of "age"
- Example: directory "201905_1_1_0" is the first directory created for the partition "201905"



Directory creation, merging, and deletion

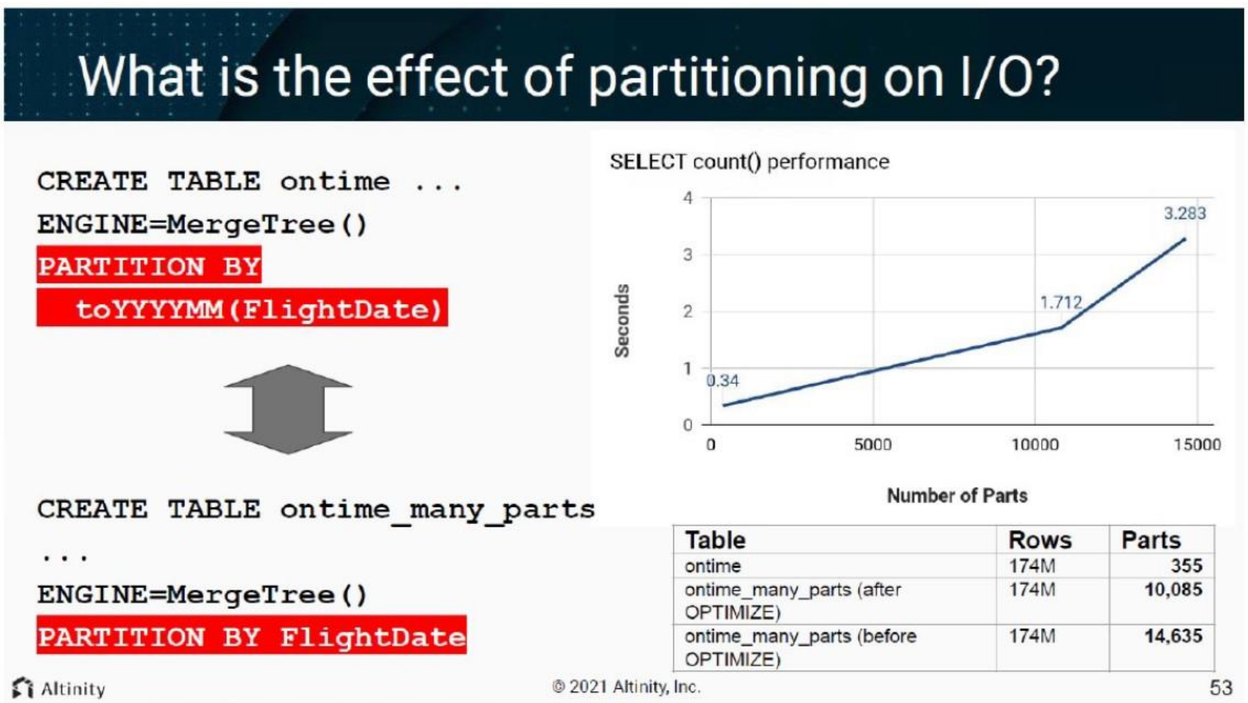
In the example below, month is used as the partition key for table "`partition_v5`", e. g. 201905, 201906, etc.


T0
partition_v5
T1
INSERT INTO partition_v5 VALUES
(A, c1, 2019-05-01'),
(A, c2, 2019-06-01')
partition_v5
201905_1_1_0
checksums.txt
columns.txt
count.txt
primary.idx
ID.bin
ID.mrk
Code.bin
Code.mrk
EventTime.bin
EventTime.mrk
partition.dat
minmax_EventTime.idx
201906_2_2_0
active=1
T2
INSERT INTO partition_v5 VALUES
(B', c3, 2019-05-08'),
(C', c4, 2019-06-10'),
(D', c5, 2019-07-10')
partition_v5
201905_1_1_0
201906_2_2_0
201905_3_3_0
201906_4_4_0
201907_5_5_0
new partition directories
T3
merge to generate new partition directory; set active=0 for old directories
partition_v5
201905_1_1_0
201905_3_3_0
201905_1_3_1
201906_2_2_0
201906_4_4_0
201906_2_4_1
201907_5_5_0
merge; set active=0 for old directories
merge; set active=0 for old directories
T4
delete old directories which has active=0
partition_v5
201905_1_3_1
201906_2_4_1
201907_5_5_0

Partition improves query performance

Back to Bloomberg equity tick data. Assume we have 3 years of daily data and we use date as the partition key

- This will lead to about 250 * 3 = 750 partitions
- Partitioning indexing (`minmax.idx`) is triggered when the partition column "`timestamp`" is used in the "`WHERE`" condition, allowing ClickHouse to skip many irrelevant partitions.
  - ➢ `SELECT * FROM equity_tickdata LIMIT 10` ⇒ full table scan, 750 partitions will be scanned
  - ➢ `SELECT * FROM equity_tickdata WHERE timestamp >= toDateTime64('2020-01-02 00:00:00.000000',6) AND timestamp <= toDateTime64('2020-01-02 23:59:59.000000',6) LIMIT 10` ⇒ scan data for 01/02/2020, 1 partition will be scanned
- Number of partitions affects efficiency, up to 10x (Source: Altinity): month vs. date as partition key



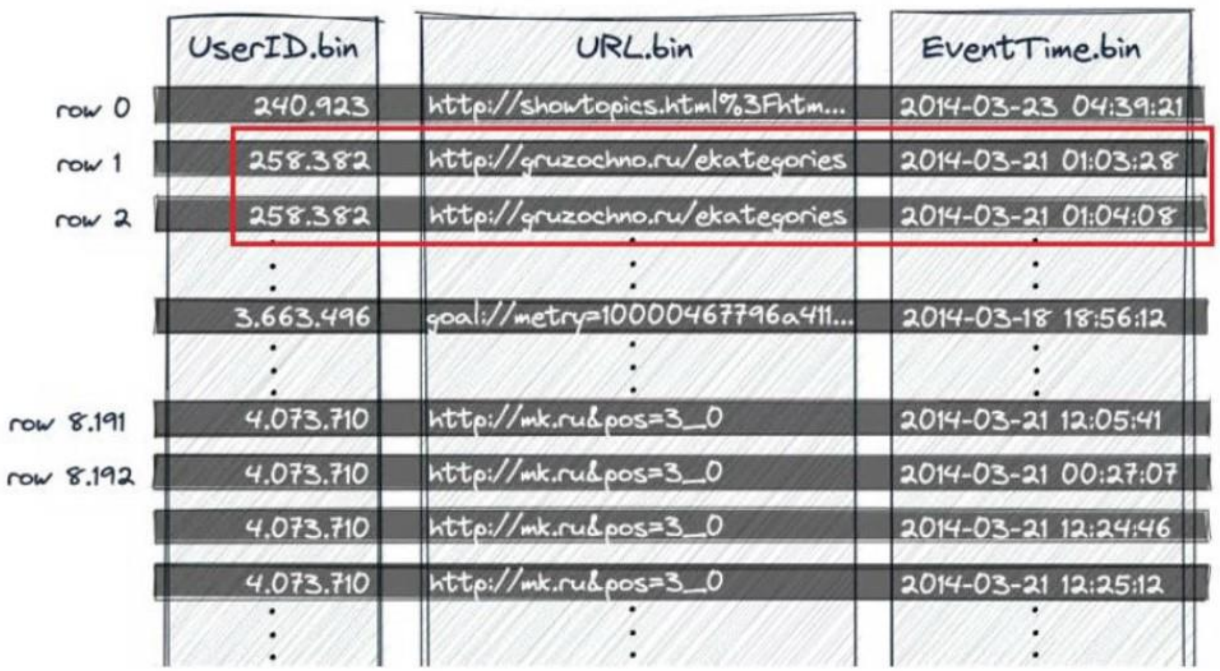
Key Takeaways for Partition

- A partition is a directory for physical storage of data
- Partition allows fast table insertion: multiple directories are created for the same partition, and then merged and deleted
- Partition allows fast data query: when the column(s) for partitioning appears in `WHERE` statement, partition indexing is triggered and only the relevant partitions are scanned for query result
- Number of partitions should not be too big: building and reading partition index files take time and memory

Primary Key, Order By

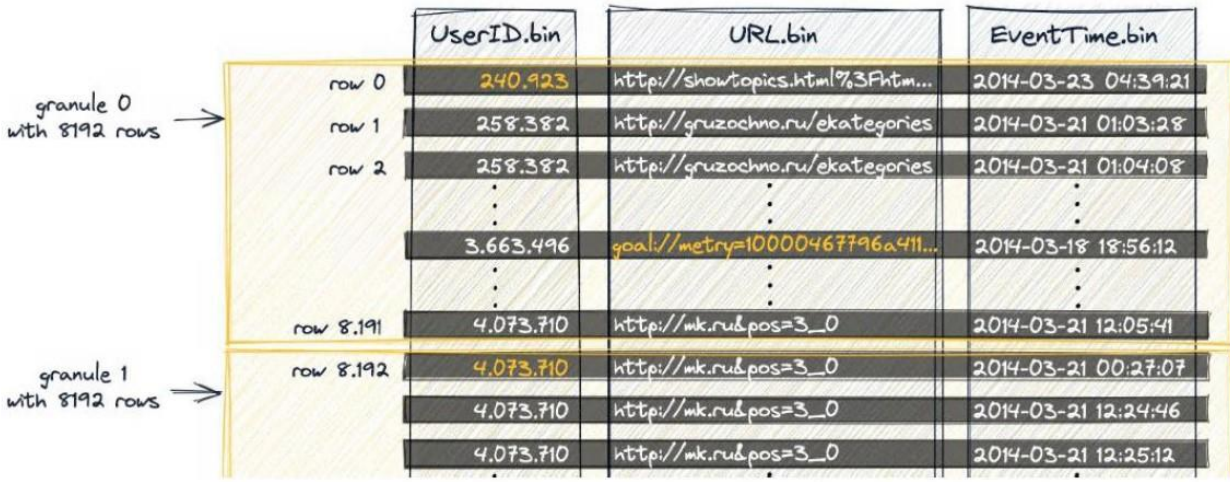
Recall that web browsing data are sorted first by `UserID`, then by `URL`, and lastly by `EventTime`:
`PRIMARY KEY (UserID, URL) ORDER BY (UserID, URL, EventTime)`

- Ordered data storage allows for efficient search algorithm, e. g. binary search algorithm
- Web browsing data are sorted first by `UserID`, then by `URL`, and lastly be `EventTime`



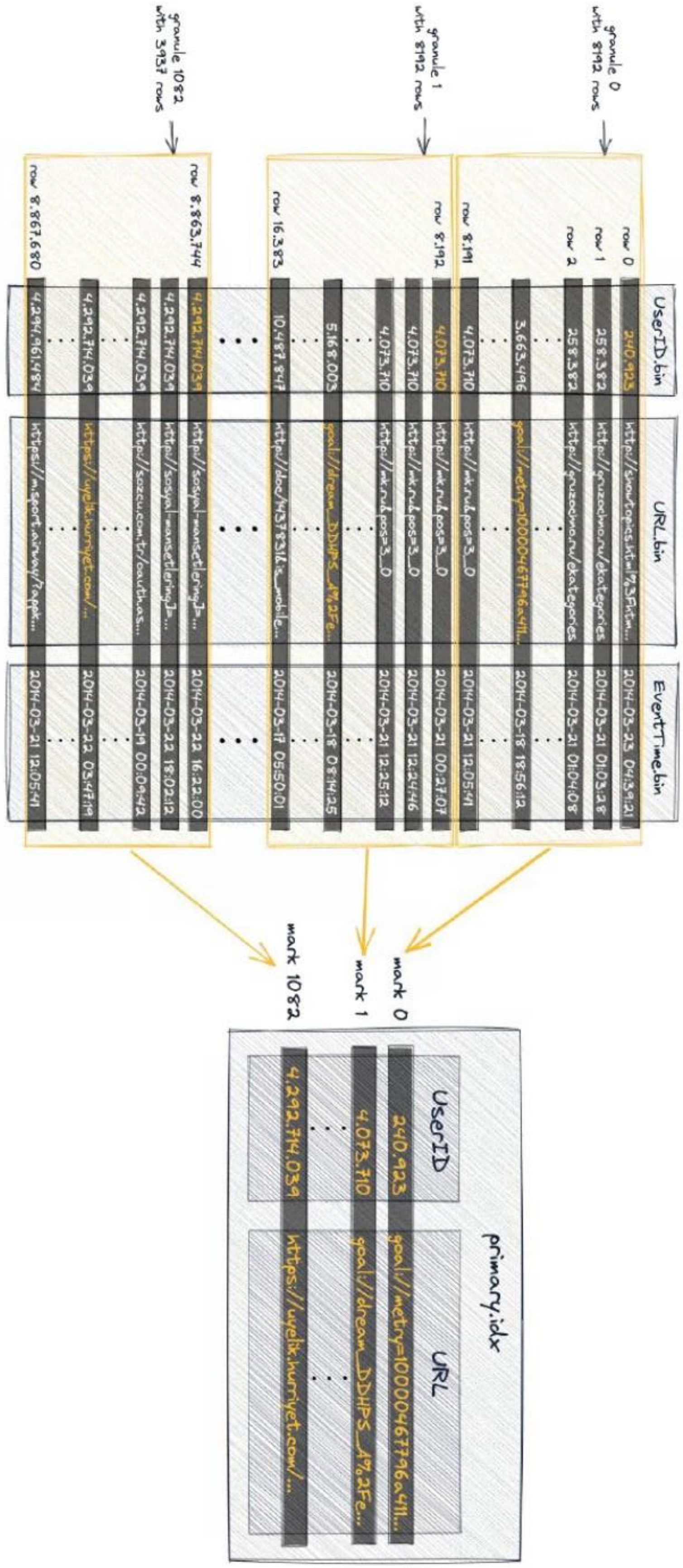
Sparse index to locate granules

- Primary key columns are used to build a sparse index, which, when combined with column level offset files ("`mark`"), can quickly locate matching data
    - First element of the primary key columns is used for binary search algorithm
    - Other elements of the primary key columns are used for *generic exclusion search algorithm* (more on this later)
- Data are *logically* grouped into "granules"
    - typically, 8192 rows, set by `index_granularity`
    - for Bloomberg equity tick data on 5/27/2022, 1 granule = 55.5 GB / 595 mil. * 8192 = 0.76 MB, 1 ticker = 55.5 GB / 19K = 3 MB = 4 granules
- After being located by the sparse index, relevant granules are loaded into memory for parallel data processing



Build and use the primary index

- The primary index has one entry per granule. The orange marked columns values are the minimum values of each primary key column in each granule; they will be the entries in the table's primary index. The primary index file is completely loaded into the main memory (~6MB for `equity_tickdata` table on 5/27, ~120MB if partitioning by month)
- The primary index is used for selecting granules: `SELECT * FROM equity_tickdata WHERE id = 'AAPL' AND timestamp >= toDateTime64('2022-05-27 00:00:00.000000,6)' AND timestamp <= '2022-05-27 23:59:59.000000,6)'`
    - "`id`" is used for binary search algorithm
    - "`timestamp`" is used for generic exclusion search algorithm to locate the relevant granules

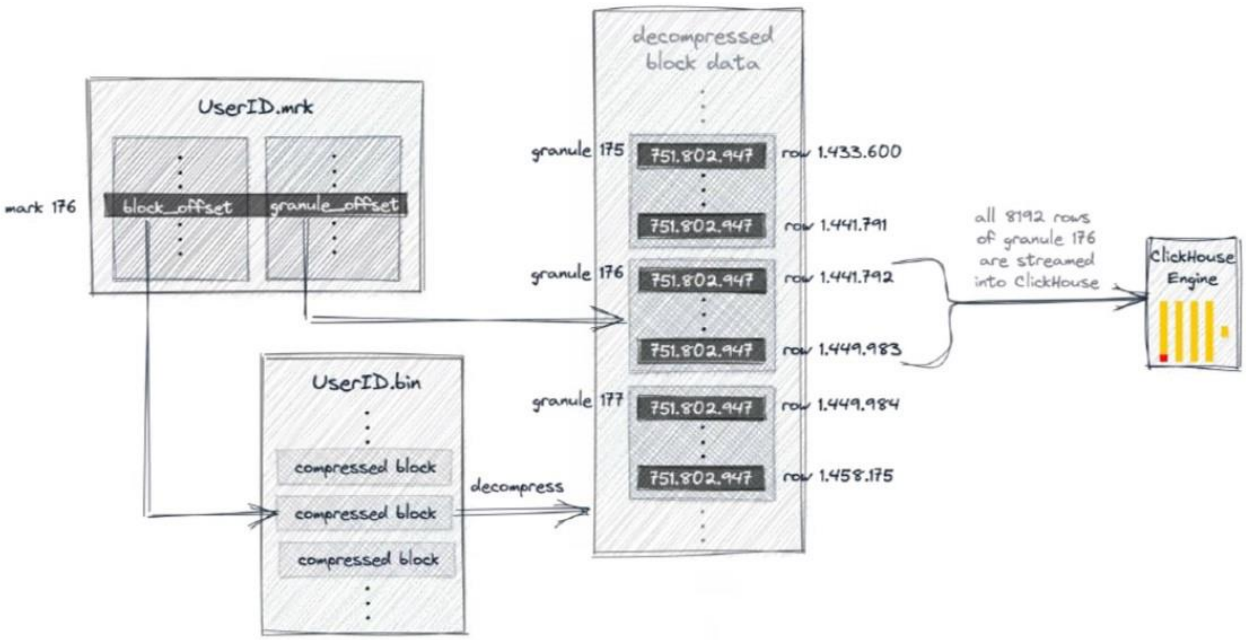


granule 0 with 8192 rows
granule 1 with 8192 rows
granule 1082 with 3937 rows
UserID.bin
URL.bin
EventTime.bin
row 0
row 1
row 2
row 8.191
row 8.192
row 16.383
row 8.863,744
row 8.867,680
mark 0
mark 1
mark 1082
primary.idx
UserID
URL
240.923
4.073.710
4.292.714.039

Use mark files

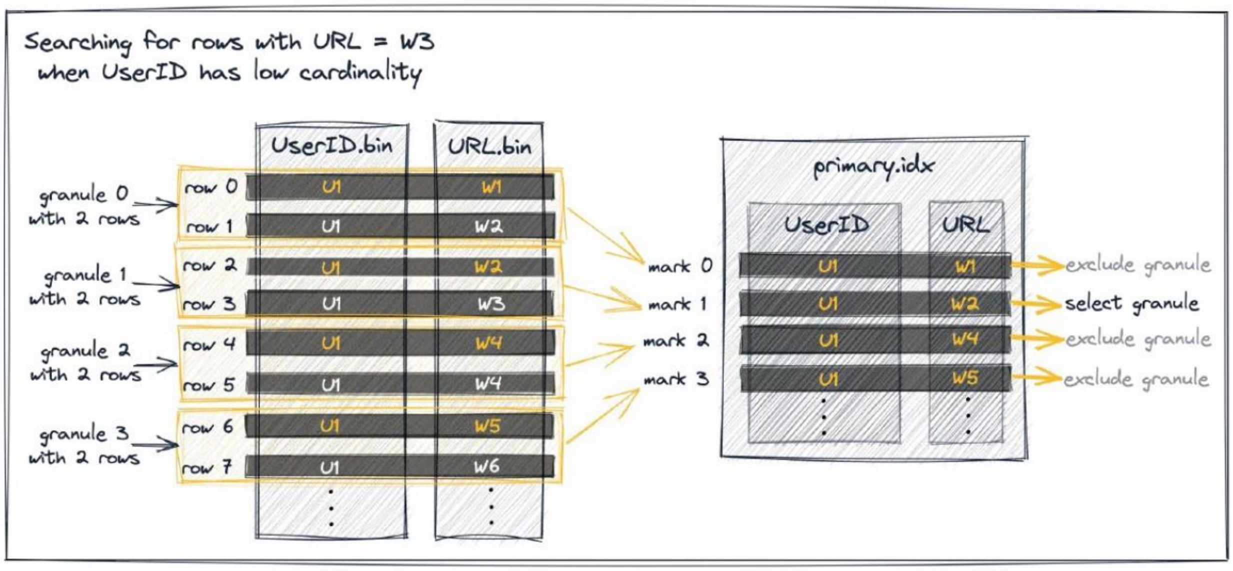
Primary index file locates the *logical* location of relevant granules, mark files locate the *physical* location of the granules

- Locating via mark files happens to each column in parallel (hence the speed)
- Why not store that information directly in primary index? The primary index file needs to fit into the main memory



Generic exclusion search algorithm

- The generic exclusion search algorithm is most effective when the predecessor key column has low(er) cardinality
- On 5/27/2022, `equity_tickdata` has ~19K distinct IDs and ~136 mil. distinct timestamps, `#id` ≪ `#timestamp`
- Details of this algorithm can be found at https://clickhouse.com/docs/en/guides/improving-query-performance/sparse-primary-indexes/sparse-primary-indexes-multiple/#generic-exclusion-search-algorithm



Data skipping index: a secondary index to group and skip granules

A secondary data skipping index on the `URL` column of the web browsing data with compound primary key `(UserID, URL)`

- A secondary data skipping index on `URL` helps with excluding granules only if the #`UserID` ≪ #`URL`
- Data skipping index should only be used after investigating other alternatives (projections, materialized views, etc.)
- Data skipping index behavior is not obvious from thought experiments alone

Test, Test, Test!

- Design of table schemas needs to be carefully considered for each business application.
- Use ClickHouse command-line client to have detailed performance information for each design.

```
SELECT * FROM skip_table WHERE my_value IN (125, 700)

┌─my_key─┬─my_value─┐
│ 512000 │      125 │
│ 512001 │      125 │
│    ... │      ... │
└────────┴──────────┘

8192 rows in set. Elapsed: 0.079 sec. Processed 100.00 million rows, 800.10 MB (1.26 billion rows/s., 10.10 GB/s.
```

Additional Resources

- ClickHouse Academy - Free self-paced ClickHouse Training (requires login to track progress): https://clickhouse.com/learn/
- Monthly ClickHouse release webinars: https://clickhouse.com/company/news-events
- Monthly newsletter: https://clickhouse.com/company/news-events
- YouTube channel - recent recordings from Monthly releases & meetups: https://www.youtube.com/c/ClickHouseDB
- Blogs - many recent articles of technical nature: https://clickhouse.com/blog
- ClickHouse Roadmap 2023: https://github.com/ClickHouse/ClickHouse/issues/44767