ClickHouse in Action: Powering Electrum's Electric Mobility Data Infrastructure

Andi Pangeran

- Over *10 years of experience* in the IT Industry
- Currently, *Principal Software Engineer* at ***Electrum.id***

@apangeran

@a_pangeran

@A_Pangeran

electrum

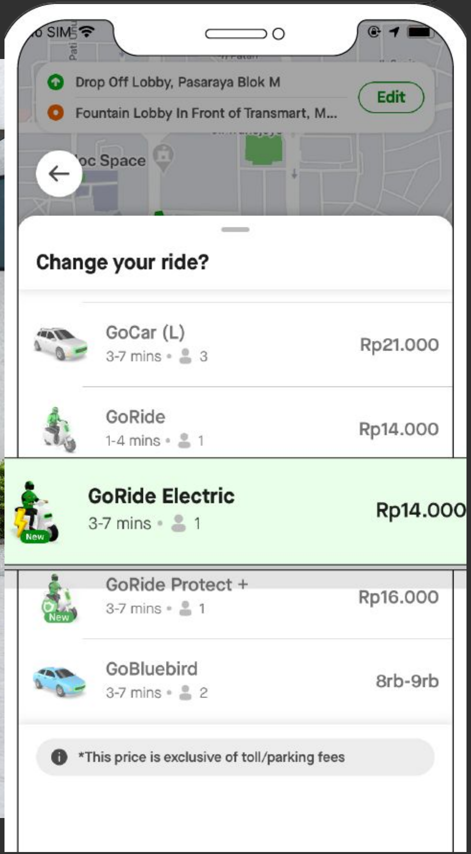
We aim to create a sustainable future where electric mobility is **accessible, efficient, and integral** to urban life. We offer electric motorcycles & mobility solutions with **convenient charging battery swap technology.**

3,000+ electric motorcycles

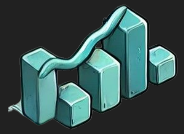
250+ swap stations across Jakarta

220k km travelled per day


electrum

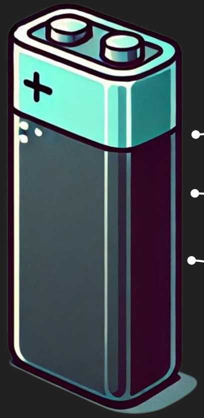

Drop Off Lobby, Pasaraya Blok M
Fountain Lobby In Front of Transmart, M...
Edit
Change your ride?
GoCar (L)
3-7 mins • 3
Rp21.000
GoRide
1-4 mins • 1
Rp14.000
GoRide Electric
3-7 mins • 1
Rp14.000
New
GoRide Protect +
3-7 mins • 1
Rp16.000
New
GoBluebird
3-7 mins • 2
8rb-9rb
*This price is exclusive of toll/parking fees

titipan
kantor
🤣🤣



Data at Electrum

Data: *Internet of Things (IoT)*



Battery Management System:
Voltage, current, temperature, state of charge (SOC), state of health (SOH), Cycle Count, Cells Information

GPS:
GPS coordinates

Vehicle:
Speed and distance traveled

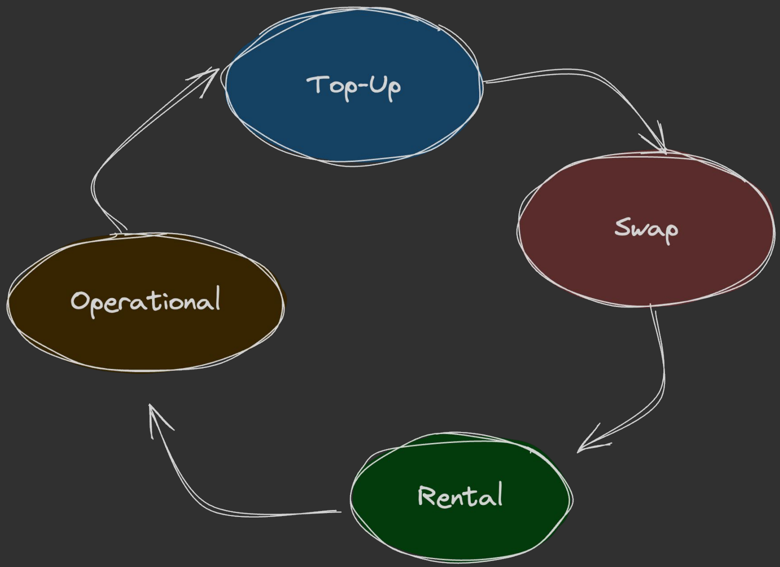
Battery

General Data:
State, Input Power, voltage, current

Doors Data:
State (lock/unlock), status (charging/idle), All batteries data

Swap Station

Data: Transactions (Order)



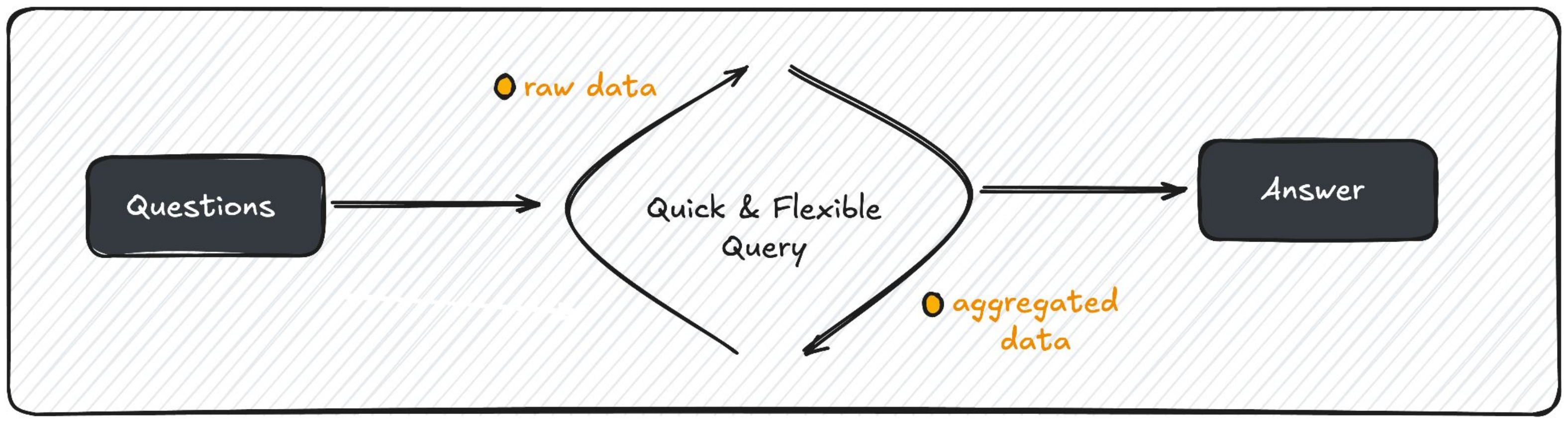
- **Top-Up Transactions:**
  Data on account recharges, including amounts, payment methods, and status.
- **Swap Transactions:**
  Details on battery swaps (station ID, door-number, battery ID, completion status).
- **Rental Transactions**:
  - Onboarding: Start data (user, vehicle, battery).
  - Offboarding: End data (user, vehicle, vehicle condition).
- **Operational Activity**:
  Maintenance, logistics, and fleet management data (e.g., technician actions, station repairs).

So what do we look for?

Flexible, Scaleable, & Low-Cost

Data Stack

Need: Quick & Flexible Query



Question: Why do one of our battery swap stations was charging slower than others?

Answer: Query showed that the station’s temperature is 4°C hotter than others, which caused slower charging to avoid overheating. Later, we learned the station was positioned under direct sunlight.

Need: Scaleable & Low-Cost

Resource Efficiency

Efficiently use resources so we're not **over-provisioning**

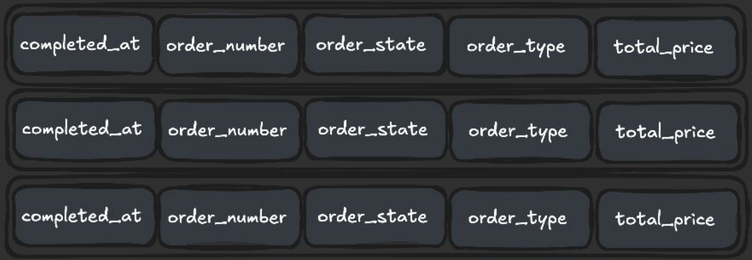
Expandable Infrastructure

Able to **scale linearly** with resources;
as datasets grow, adding more resources will keep response times fast.

Why ClickHouse?

- ✓ Columnar Storage
- ✓ Indexing & Data Skipping
- ✓ Data Organization & Merging Mechanism

Why: *It's columnar based*



Row-based

Column-Based

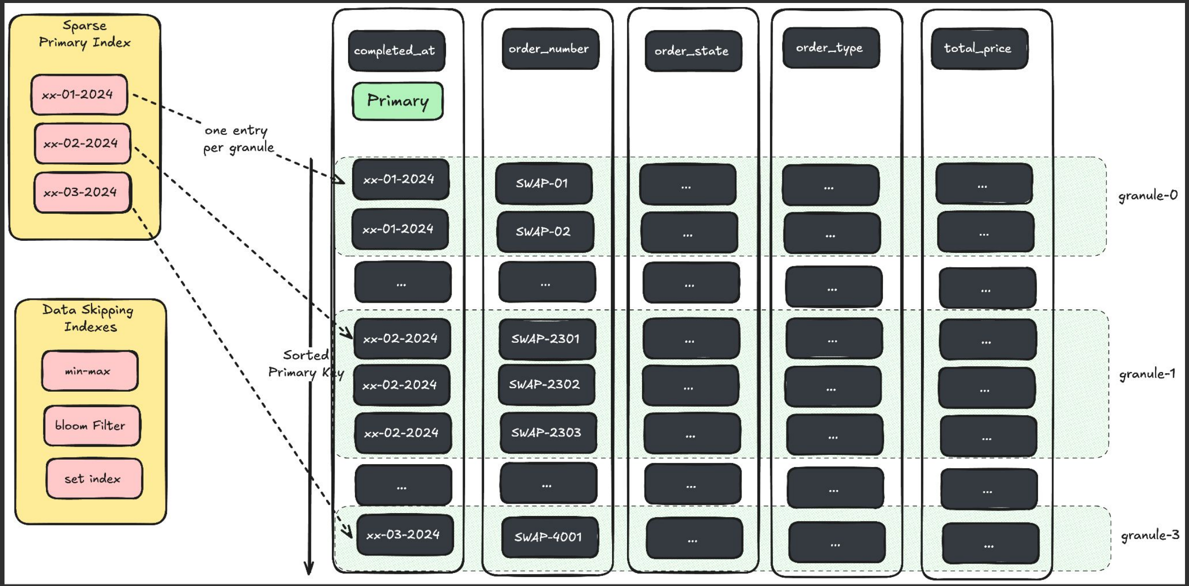
Excess disk Reads We need data in column, but it's stored in rows. So entire rows are read.

Column Pruning it only pick the subset of columns that are openly declared in the SQL query, thus eliminating 99% columns (990 of 1000 columns) from the disk read consideration.

Vectorized Execution
data processed in **large arrays**, which could be the entire length of the column loaded into RAM.

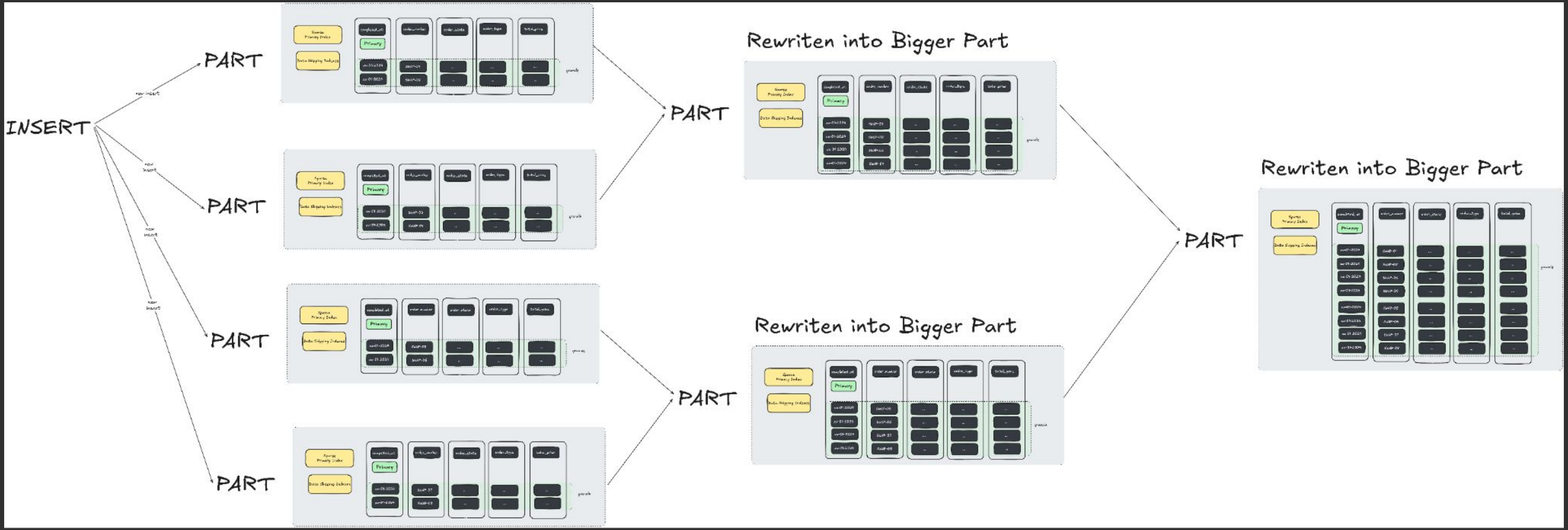
+ **reduces CPU cache miss rates**

Why: *Multiple Layers of Indexes*



- **Granules** are chunks of rows, and **ClickHouse** groups rows into granules based on the index_granularity setting.
- **Sparse Primary Index**: **ClickHouse** uses the *sparse primary index* to quickly jump to the relevant granules, skipping over large portions of the data that don't need to be read.
- **Skip Indexes**: These allow **ClickHouse** to skip entire granules if it can determine from the index that no rows in the granule satisfy the query's filter.

Why: *Efficient Merging*



Parts: Each time data is inserted, a new part is created. Multiple parts can accumulate over time, which can slow down query performance.

To improve performance and reduce the number of parts, **ClickHouse** periodically runs merging operations.

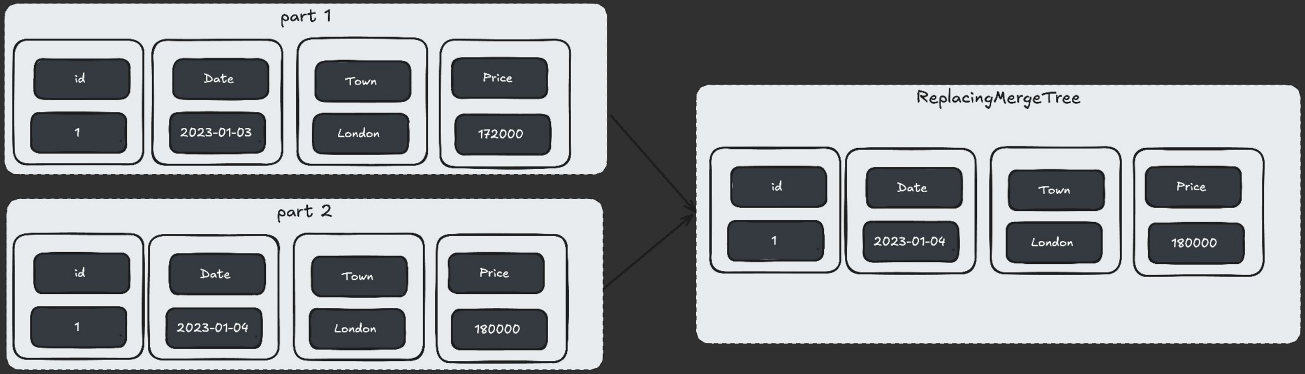
What Happens During a Merge?


```sql
CREATE TABLE uk_price_paid
(
   date  Date,
   town  String,
   price UInt32
)
   ENGINE = MergeTree
       ORDER BY (town, date);
```

MergeTree:

- Combines parts without deduplication or transformations.
- Simply reduces the number of parts but keeps all rows (including duplicates).
- Ideal for append-only data.

What Happens During a Merge?

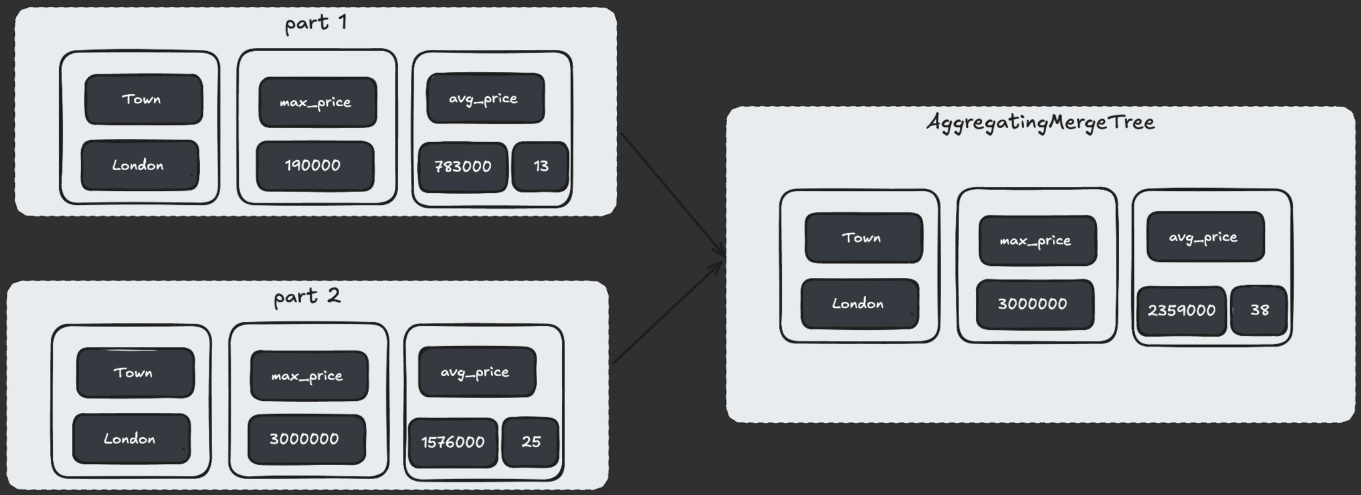



```sql
CREATE TABLE uk_listings
(
   id UInt32,
   date Date,
   town String,
   price UInt32
)
   ENGINE = ReplacingMergeTree(date)
       ORDER BY id;
```

ReplacingMergeTree:

- Combines parts and deduplicates based on a key (or version).
- Replaces older rows with newer ones during a merge.
- Suitable for upserts and scenarios where data is frequently updated.

What Happens During a Merge?




```
CREATE TABLE uk_price_paid_aggregates
(
   town String,
   max_price SimpleAggregateFunction(max, UInt32),
   avg_price AggregateFunction(avg, UInt32)
)
   ENGINE = AggregatingMergeTree
       ORDER BY town;
```

AggregatingMergeTree:

- Combines parts by aggregating rows with the same key using predefined functions like sum, count, avg, etc.
- Ideal for pre-aggregating data to reduce query time on large datasets.

Another thing we love about ClickHouse is

materialized view.

Let’s deep-dive

Materialized view 101

```sql
CREATE TABLE IF NOT EXISTS trx_orders
(
   completed_at DateTime64(3, 'UTC'),
   order_number String,
   order_type   LowCardinality(String),
   total_price  Int64
)
   ENGINE = MergeTree
       PARTITION BY toYYYYMM(completed_at)
       ORDER BY (completed_at, order_type, order_number)
       SETTINGS index_granularity = 8192;
```

Raw-Table

```sql
CREATE TABLE IF NOT EXISTS trx_agg_daily_orders
(
   order_date     String,
   order_type     LowCardinality(String),
   count_orders   AggregateFunction(count),
   total_price    AggregateFunction(sum, Int64)
)
   ENGINE = AggregatingMergeTree()
       PARTITION BY toYYYYMM(order_date)
       ORDER BY (order_date, order_type);
```

Aggregated-Table

SYNC ?

Materialized view 101

```sql
CREATE TABLE IF NOT EXISTS trx_orders
(
    completed_at DateTime64(3, 'UTC'),
    order_number String,
    order_type   LowCardinality(String),
    total_price  Int64
)
    ENGINE = MergeTree
        PARTITION BY toYYYYMM(completed_at)
        ORDER BY (completed_at, order_type, order_number)
        SETTINGS index_granularity = 8192;
```

```sql
CREATE TABLE IF NOT EXISTS trx_agg_daily_orders
(
    order_date    String,
    order_type    LowCardinality(String),
    count_orders  AggregateFunction(count),
    total_price   AggregateFunction(sum, Int64)
)
    ENGINE = AggregatingMergeTree()
        PARTITION BY toYYYYMM(order_date)
        ORDER BY (order_date, order_type);
```

Raw-Table ⟶ Materialized View ⟶ Aggregated-Table

```sql
CREATE MATERIALIZED VIEW trx_agg_daily_orders_mv

TO trx_agg_daily_orders
AS

SELECT toDate(completed_at) AS order_date,
      order_type,
      countState()             AS total_orders,
      sumState(total_price)    AS total_price
FROM trx_orders
GROUP BY order_date, order_type;
```

Materialized view 101 - insert scenario empty

New insert block

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256

Raw-Table → **Materialized View** → **Aggregated-Table**

Materialized view 101 - insert scenario empty

New insert block

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256

Raw-Table → **Materialized View** → **Aggregated-Table**

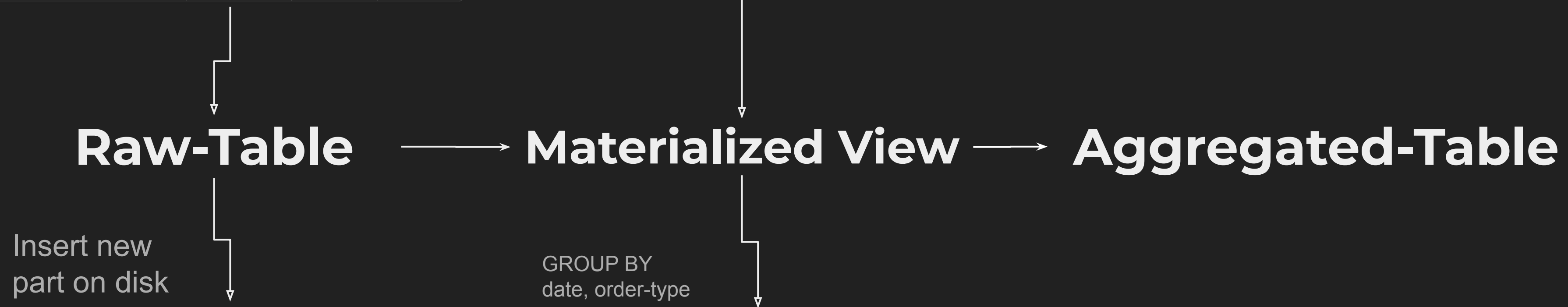
Insert new part on disk

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256

Materialized view 101 - insert scenario empty

New insert block

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256



completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256

	order_date	order_type	total_orders	total_price
1	2024-09-28	SWAP	2	2339
2	2024-09-28	TOPUP	2	4711

Materialized view 101 - insert scenario empty

New insert block

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256

Raw-Table → Materialized View → Aggregated-Table

Insert new part on disk

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256

GROUP BY
date, order-type

	order_date	order_type	total_orders	total_price
1	2024-09-28	SWAP	2	2339
2	2024-09-28	TOPUP	2	4711

STORE as new part of target table

	order_date	order_type	total_orders	total_price
1	2024-09-28	SWAP	2	2339
2	2024-09-28	TOPUP	2	4711

Materialized view 101 - insert merge scenario

Raw-Table → Materialized View → Aggregated-Table

Existing part on disk

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256

Existing part of target table

	order_date	order_type	total_orders	total_price
1	2024-09-28	SWAP	2	2339
2	2024-09-28	TOPUP	2	4711

Materialized view 101 - insert merge scenario

New insert block

completed_at	order_number	order_type	total_price
2024-09-28 14:04:32.656515	ORD144868	SWAP	1392
2024-09-28 14:04:32.656517	ORD355138	TOPUP	623
2024-09-28 14:04:32.656517	ORD286434	TOPUP	4335
2024-09-28 14:04:32.656517	ORD847260	SWAP	2624

Raw-Table → Materialized View → Aggregated-Table

Merge with existing part

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256
2024-09-28 14:04:32.656515	ORD144868	SWAP	1392
2024-09-28 14:04:32.656517	ORD355138	TOPUP	623
2024-09-28 14:04:32.656517	ORD286434	TOPUP	4335
2024-09-28 14:04:32.656517	ORD847260	SWAP	2624

	order_date	order_type	total_orders	total_price
1	2024-09-28	SWAP	2	2339
2	2024-09-28	TOPUP	2	4711

Materialized view 101 - insert merge scenario

New insert block

completed_at	order_number	order_type	total_price
2024-09-28 14:04:32.656515	ORD144868	SWAP	1392
2024-09-28 14:04:32.656517	ORD355138	TOPUP	623
2024-09-28 14:04:32.656517	ORD286434	TOPUP	4335
2024-09-28 14:04:32.656517	ORD847260	SWAP	2624

Raw-Table → **Materialized View** → **Aggregated-Table**

Merge with existing part

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256
2024-09-28 14:04:32.656515	ORD144868	SWAP	1392
2024-09-28 14:04:32.656517	ORD355138	TOPUP	623
2024-09-28 14:04:32.656517	ORD286434	TOPUP	4335
2024-09-28 14:04:32.656517	ORD847260	SWAP	2624

GROUP BY
date, order-type

completed_at	order_type	total_orders	total_price
2024-09-28	SWAP	2	4016
2024-09-28	TOPUP	2	4958

	order_date	order_type	total_orders	total_price
1	2024-09-28	SWAP	2	2339
2	2024-09-28	TOPUP	2	4711

Materialized view 101 - insert merge scenario

New insert block

completed_at	order_number	order_type	total_price
2024-09-28 14:04:32.656515	ORD144868	SWAP	1392
2024-09-28 14:04:32.656517	ORD355138	TOPUP	623
2024-09-28 14:04:32.656517	ORD286434	TOPUP	4335
2024-09-28 14:04:32.656517	ORD847260	SWAP	2624

Raw-Table → Materialized View → Aggregated-Table

Merge with existing part

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256
2024-09-28 14:04:32.656515	ORD144868	SWAP	1392
2024-09-28 14:04:32.656517	ORD355138	TOPUP	623
2024-09-28 14:04:32.656517	ORD286434	TOPUP	4335
2024-09-28 14:04:32.656517	ORD847260	SWAP	2624

GROUP BY
date, order-type

completed_at	order_type	total_orders	total_price
2024-09-28	SWAP	2	4016
2024-09-28	TOPUP	2	4958

Merge with
existing part
Of target table

	order_date	order_type	total_orders	total_price
1	2024-09-28	SWAP	2	2339
2	2024-09-28	TOPUP	2	4711

completed_at	order_type	total_orders	total_price
2024-09-28	SWAP	4	6355
2024-09-28	TOPUP	4	9669

Materialized view 101 - Query Comparison

Raw-Table → Materialized View → Aggregated-Table

completed_at	order_number	order_type	total_price
2024-09-28 13:48:26.249280	ORD953128	SWAP	59
2024-09-28 13:48:26.249282	ORD957709	TOPUP	455
2024-09-28 13:48:26.249283	ORD475309	SWAP	2280
2024-09-28 13:48:26.249283	ORD110939	TOPUP	4256
2024-09-28 14:04:32.656515	ORD144868	SWAP	1392
2024-09-28 14:04:32.656517	ORD355138	TOPUP	623
2024-09-28 14:04:32.656517	ORD286434	TOPUP	4335
2024-09-28 14:04:32.656517	ORD847260	SWAP	2624

completed_at	order_type	total_orders	total_price
2024-09-28	SWAP	4	6355
2024-09-28	TOPUP	4	9669

```
SELECT toDate(completed_at) AS order_date,
       order_type,
       count(*)             AS total_orders,
       sum(total_price)     AS total_price
FROM trx_orders
GROUP BY order_date, order_type;
```

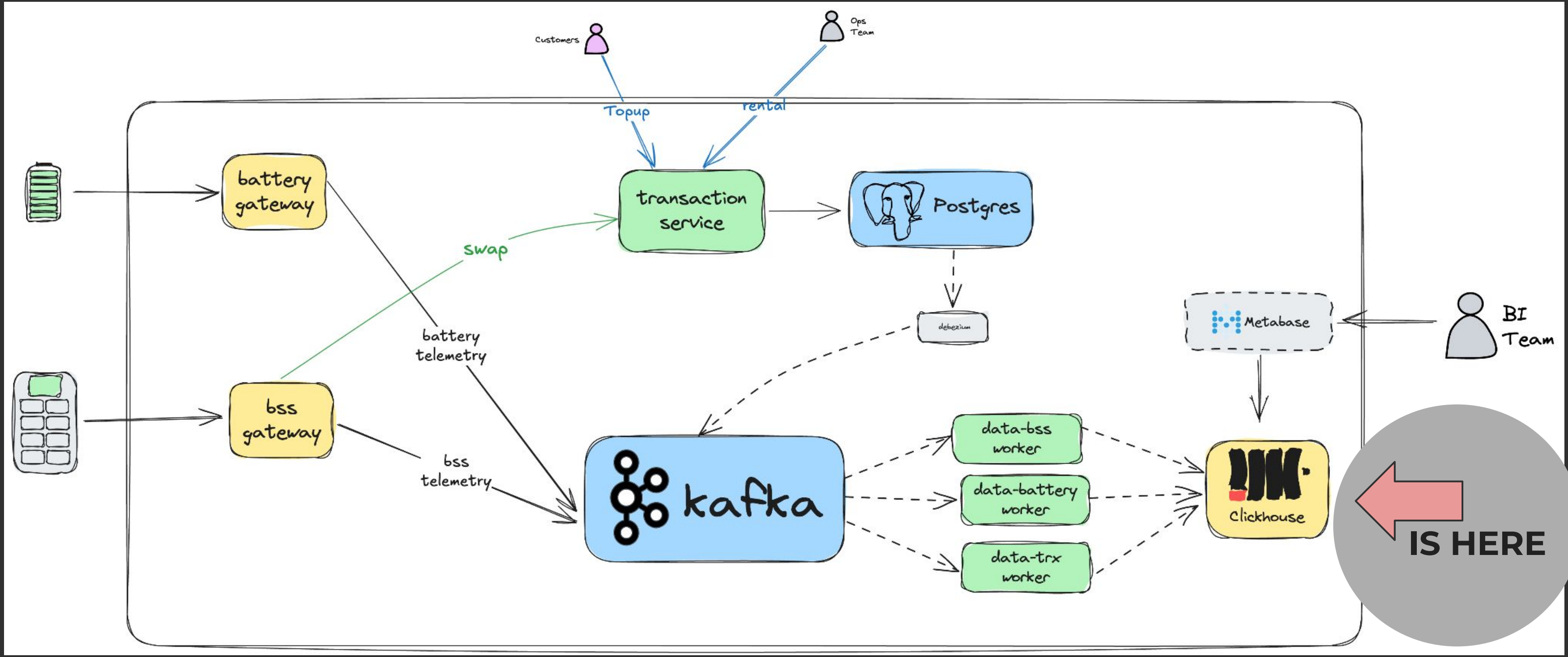
Processed 8 rows

```
SELECT
       order_date,
       order_type,
       countState() AS total_orders,
       sumState(total_price) AS total_price
FROM trx_agg_daily_orders
GROUP BY order_date, order_type;
```

Processed 2 rows

How ClickHouse Powers Our Data Platform

How ClickHouse Powers Our Data Platform



DEMO

https://github.com/meong1234/clickhouse-meetup-10-2024

Where are we?

50 million+
Records / Day

≤$500/month

*“**ClickHouse** enables us to have a data analytics capability that grows with our needs, while keeping costs low and predictable"* - Electrum.id

Key Takeaways

Columnar Format:
ClickHouse stores data in **columns**, optimizing read performance for analytical queries by reading only the relevant columns, reducing I/O and improving query speed.

Sparse Primary Index:
ClickHouse uses a **sparse index** to store pointers to data blocks (granules), allowing it to **skip irrelevant data** during queries, improving performance when filtering by the primary key.

Skip Index:
The **skip index** allows **ClickHouse** to **bypass granules** based on non-primary key column values, helping to avoid unnecessary data scanning and speeding up queries on non-primary key columns.

Merging Process in Different Engines:

- **MergeTree**: Performs basic merging of data parts without deduplication or aggregation.
- **ReplacingMergeTree**: Deduplicates data during merging, keeping only the most recent version of rows with the same primary key.
- **AggregatingMergeTree**: Aggregates data during merging based on predefined aggregation functions, storing the results in summarized form.

ClickHouse Materialized Views:

- **Triggered on Inserts**: Automatically updates the materialized view whenever new data is inserted into the source table.
- **Real-Time Pre-Aggregation**: Performs pre-aggregation or data transformation in real time, allowing for faster query performance by reducing the need for complex computations at query time.

Ref

1. **Why ClickHouse Is So Fast**
   ChistaDATA Inc. (2023). Why ClickHouse Is So Fast.
   https://chistadata.com/why-clickhouse-is-so-fast

2. **How to Use ClickHouse Indexes: A Practical Guide**
   ChistaDATA Inc. (2023). How to Use ClickHouse Indexes: A Practical Guide.
   https://chistadata.com/how-to-use-clickhouse-indexes-practical-guide

3. **ClickHouse Skipping Indexes**
   ClickHouse Documentation. (2023). Skipping Indexes.
   https://clickhouse.com/docs/en/optimize/skipping-indexes

4. **ClickHouse Overview**
   YouTube. (2023). ClickHouse Overview.
   https://www.youtube.com/watch?v=QDAJTKZT8y4

5. **ClickHouse Architecture Deep Dive**
   YouTube. (2023). ClickHouse Architecture Deep Dive.
   https://www.youtube.com/watch?v=iLXXoDaFoxs

6. **Optimizing ClickHouse Performance**
   YouTube. (2023). Optimizing ClickHouse Performance.
   https://www.youtube.com/watch?v=BHcIEszF6Fk

7. **Supercharge Your ClickHouse Data Loads - Part 1**
   ClickHouse Blog. (2023). Supercharge Your ClickHouse Data Loads - Part 1.
   https://clickhouse.com/blog/supercharge-your-clickhouse-data-loads-part1


Thanks