Introduction to ClickHouse for Gophers

Nikolai Lebedev

Helsinki Gophers Meetup | Mar. 18, 2025

About me

- Senior Software Developer
  API development & Data engineering
  - Go
  - PHP
  - Python
  - ElasticSearch
  - **ClickHouse**
  - BigQuery

Problem

- High ingestion rate
- Need to aggregate data on-the-fly
- Preferably on-premises

Problem

- High ingestion rate
- Need to aggregate data on-the-fly
- Preferably on-premises

Solution:

What is ClickHouse?

- Open-source
- OLAP
- Column-oriented

Column-oriented

- Row-oriented:
  - Good for transactions
  - Slow for analytics
- Columnar:
  - Faster aggregations
  - Better compression

dt	user_id	event
1742232733	1	login
1742232733	2	login
1742232733	2	logout

dt	user_id	event
1742232733	1	login
1742232733	2	login
1742232733	2	logout

Why ClickHouse?

- Fast inserts
- Fast queries on large datasets
- Cost-effective storage with compression
- Open-source & scalable
- SQL (-like) query syntax

Drawbacks

- Not fully ACID-compliant
- Transactions are experimental
- No usual Delete/Update operations
- Might be not-so-obvious in configuration

Where can ClickHouse be used?

- Real-time analytics
- Data warehousing
- Observability
- Machine-learning, GenAI

Engines

MergeTree family	Integrations	Log family	Special
● MergeTree ● Summing ● Aggregating ● Collapsing ● Replacing	● S3 ● PostgreSQL ● MySQL ● Kafka ● etc.	● Log ● StripeLog ● TinyLog	● Null ● Distributed ● URL ● etc.

Engines

MergeTree family	Integrations	Log family	Special
● MergeTree ● Summing ● Aggregating ● Collapsing ● Replacing	● S3 ● PostgreSQL ● MySQL ● Kafka ● etc.	● Log ● StripeLog ● TinyLog	● Null ● Distributed ● URL ● etc.

Engines

MergeTree family	Integrations	Log family	Special
● MergeTree ● Summing ● Aggregating ● Collapsing ● Replacing	● S3 ● PostgreSQL ● MySQL ● Kafka ● etc.	● Log ● StripeLog ● TinyLog	● Null ● Distributed ● URL ● etc.

Engines

MergeTree family	Integrations	Log family	Special
● MergeTree ● Summing ● Aggregating ● Collapsing ● Replacing	● S3 ● PostgreSQL ● MySQL ● Kafka ● etc.	● Log ● StripeLog ● TinyLog	● Null ● Distributed ● URL ● etc.

Engines

MergeTree family	Integrations	Log family	Special
● MergeTree ● Summing ● Aggregating ● Collapsing ● Replacing	● S3 ● PostgreSQL ● MySQL ● Kafka ● etc.	● Log ● StripeLog ● TinyLog	● Null ● Distributed ● URL ● etc.

MergeTree

```
CREATE TABLE events (
   dt Date,
   user_id Int32,
   ev_type Enum('type1', 'type2', 'type3'),
   name String,
   INDEX ev_name (name) TYPE set(100) GRANULARITY 2,
   PROJECTION by_event_type (SELECT * FROM events ORDER BY ev_type)
) ENGINE = MergeTree()
ORDER BY user_id, dt
PRIMARY KEY user_id, dt
PARTITION BY toYYYYMM(dt)
TTL dt + INTERVAL 1 MONTH
```

MergeTree

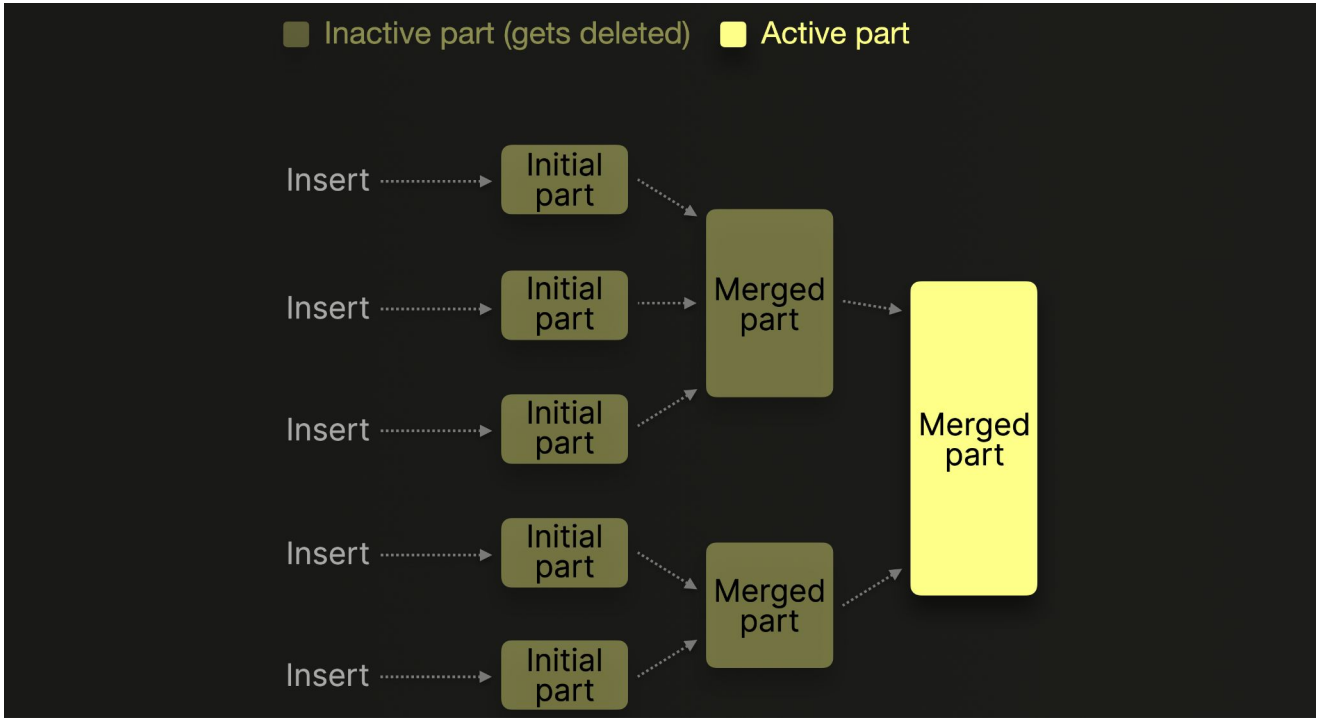

Image: https://clickhouse.com/

Indexes: Primary key

```
CREATE TABLE events (
    dt Date,
    user_id Int32,
    ev_type Enum('type1', 'type2', 'type3'),
    name String,
    INDEX ev_name (name) TYPE set(100) GRANULARITY 2,
    PROJECTION by_event_type (SELECT * FROM events ORDER BY ev_type)
) ENGINE = MergeTree()
ORDER BY user_id, dt
PRIMARY KEY user_id, dt
PARTITION BY toYYYYMM(dt)
TTL dt + INTERVAL 1 MONTH
```

Indexes: Sorting key

```
CREATE TABLE events (
    dt Date,
    user_id Int32,
    ev_type Enum('type1', 'type2', 'type3'),
    name String,
    INDEX ev_name (name) TYPE set(100) GRANULARITY 2,
    PROJECTION by_event_type (SELECT * FROM events ORDER BY ev_type)
) ENGINE = MergeTree()
ORDER BY user_id, dt
PRIMARY KEY user_id
PARTITION BY toYYYYMM(dt)
TTL dt + INTERVAL 1 MONTH
```

Indexes: Projections

```
CREATE TABLE events (
    dt Date,
    user_id Int32,
    ev_type Enum('type1', 'type2', 'type3'),
    name String,
    INDEX ev_name (name) TYPE set(100) GRANULARITY 2,
    PROJECTION by_event_type (SELECT * FROM events ORDER BY ev_type)
) ENGINE = MergeTree()
ORDER BY user_id, dt
PRIMARY KEY user_id, dt
PARTITION BY toYYYYMM(dt)
TTL dt + INTERVAL 1 MONTH
```

Indexes: Data skipping

```
CREATE TABLE events (
    dt Date,
    user_id Int32,
    ev_type Enum('type1', 'type2', 'type3'),
    name String,
    INDEX ev_name (name) TYPE set(100) GRANULARITY 2,
    PROJECTION by_event_type (SELECT * FROM events ORDER BY ev_type)
) ENGINE = MergeTree()
ORDER BY user_id, dt
PRIMARY KEY user_id, dt
PARTITION BY toYYYYMM(dt)
TTL dt + INTERVAL 1 MONTH
```

Partitions

```
CREATE TABLE events (
    dt Date,
    user_id Int32,
    ev_type Enum('type1', 'type2', 'type3'),
    name String,
    INDEX ev_name (name) TYPE set(100) GRANULARITY 2,
    PROJECTION by_event_type (SELECT * FROM events ORDER BY ev_type)
) ENGINE = MergeTree()
ORDER BY user_id, dt
PRIMARY KEY user_id, dt
PARTITION BY toYYYYMM(dt)
TTL dt + INTERVAL 1 MONTH
```

Materialized Views: Refreshable

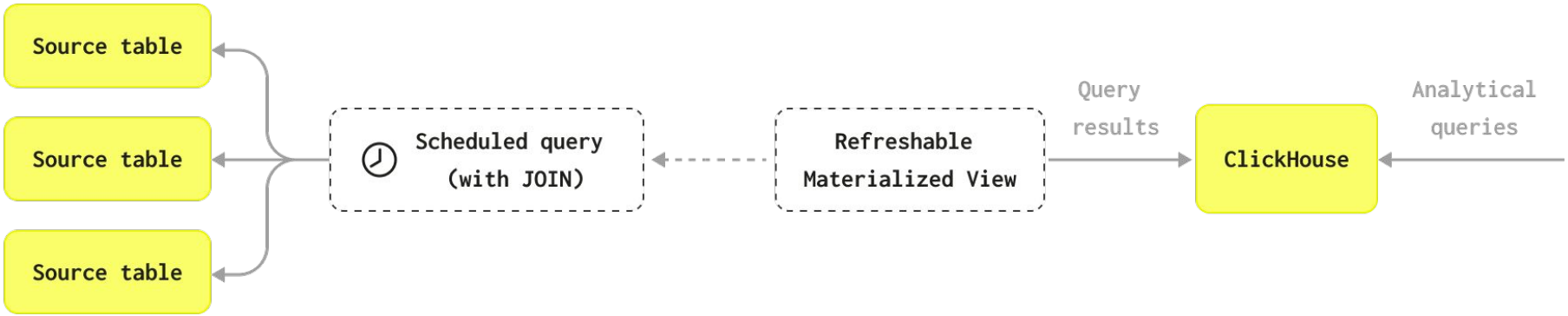

Image: https://clickhouse.com/

Materialized Views: Incremental

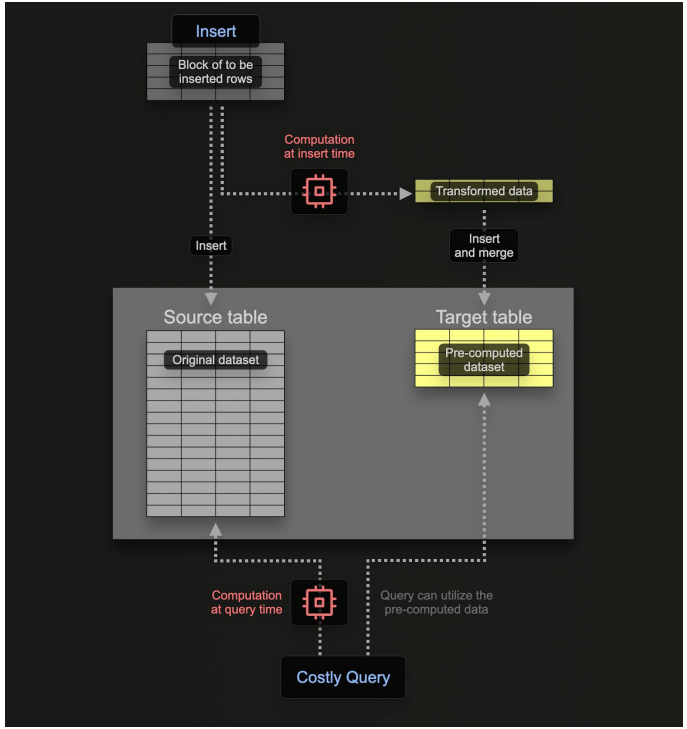

Image: https://clickhouse.com/

AggregatingMergeTree

-State

sumState

countState

uniqState

...

```
CREATE TABLE uniq_users (
   dt DateTime64 NOT NULL,
   users_uniq_cnt AggregateFunction(uniq, Nullable(Int32))
)
ENGINE = AggregatingMergeTree() ORDER BY (StartDate,
CounterID);



CREATE MATERIALIZED VIEW uniq_users_mv TO uniq_users
AS SELECT
   dt,
   uniqState(user_id) AS users_uniq_cnt
FROM events
GROUP BY dt;
```

AggregatingMergeTree

-State ➤ -Merge

sumMerge

countMerge

uniqMerge

...

```
CREATE MATERIALIZED VIEW uniq_users_mv TO uniq_users
AS SELECT
   dt,
   uniqState(user_id) AS users_uniq_cnt
FROM events
GROUP BY dt;

SELECT
   dt,
   uniqMerge(users_uniq_cnt) AS users_uniq_cnt
FROM uniq_users_mv
GROUP BY dt
ORDER BY dt;
```

Go

Official clients:

- github.com/ClickHouse/ch-go
- github.com/ClickHouse/clickhouse-go

Alternative clients:

- github.com/vahid-sohrabloo/chconn
- github.com/uptrace/go-clickhouse

ch-go

- Column-oriented
- Suitable for heavy inserts
- No reflection or interface{}
- More efficient
- No pooling (available via chpool)
- Not goroutine-safe
- Not database/sql compatible

clickhouse-go

- More high-level (struct tags, helper functions, pooling, etc.)
- Suitable for queries
- *database/sql* -compatible

github.com/ClickHouse/ch-bench

Name	Time	RAM	Ratio
ClickHouse/ch-go (Go)	401ms	9M	~1x
clickhouse-client (C++)	387ms	91M	~1x
vahid-sohrabloo/chconn (Go)	472ms	9M	~1x
clickhouse-cpp (C++)	516ms	6.9M	1.47x
clickhouse_driver (Rust)	614ms	9M	1.72x
curl (C, HTTP)	3.7s	10M	9x
clickhouse-client (Java, HTTP)	6.4s	121M	16x
clickhouse-jdbc (Java, HTTP)	7.2s	120M	18x
loyd/clickhouse.rs (Rust, HTTP)	10s	7.2M	28x
uptrace (Go)[1]	22s	13M	55x
clickhouse-driver (Python)	37s	60M	106x
ClickHouse/clickhouse-go (Go)[1]	46.8s	23M	117x
mailru/go-clickhouse (Go, HTTP)	4m13s	13M	729x

Struct tags

```
type Event struct {
   Dt          time.Time `ch:"dt"`
   UserID      string    `ch:"user_id"`
   EType       string    `ch:"ev_type"`
   Name        string    `ch:"name"`
}

var result []Event

err := conn.Select(ctx, &result, "SELECT dt, user_id, ev_type, name FROM events")
...
```

Batch insert: Clickhouse API

```
batch, err := conn.PrepareBatch(ctx, "INSERT INTO events")
...
for i := 0; i < 1000; i++ {
   err := batch.Append(v1,v2,v3,v4)
   ...
}
batch.Send()
```

Batch insert: database/sql

```
tx, err := conn.Begin()
...
tx, err := scope.Prepare("INSERT INTO events")
...
for i := 0; i < 1000; i++ {
   _, err := batch.Exec(v1,v2,v3,v4)
   ...
}
tx.Commit()
```

Async insert: Clickhouse API

```
err := conn.AsyncInsert(ctx, `INSERT INTO events VALUES (?, ?, ?, ?)`, ...);
```

Async insert: database/sql

```
// false - wait_for_async_inserts=false
ctx := clickhouse.Context(context.Background(), clickhouse.WithStdAsync(false))
for i := 0; i < 100; i++ {
   _, err := conn.ExecContext(ctx, `INSERT INTO events VALUES (?, ?, ?, ?)`, ...)
   ...
}
```

Columnar insert

```
batch, err := conn.PrepareBatch(context.Background(), "INSERT INTO events")
...
var (
   tsCol []time.Time
   ...
)
for i := 0; i < 1_000; i++ {
   col1 = append(col1, time.Now())
   ...
}
err := batch.Column(0).Append(col1)
...
```

Context

```
ctx, cancel := context.WithCancel(context.Background())
…
cancel()
…
```

```
ctx, cancel = context.WithDeadline(
      context.Background(),
      time.Now().Add(time.Second * 10)
)
defer cancel()
```

Context

```
ctx = clickhouse.Context(
      context.Background(),
      clickhouse.WithQueryID(queryId.String())
)
```

Context

```
ctx := clickhouse.Context(
   context.Background(),
   clickhouse.WithSettings(clickhouse.Settings{
      "allow_experimental_object_type": "1",
   }))
```

Thank you for your attention!

LinkedIn: https://linkedin.com/in/nikolai-lebedev/