Building fast data loops from insert to query response in ClickHouse®

Robert Hodges
Diego Nieto
https://altinity.com

A brief message from our sponsor…

Robert Hodges

Database geek with 30+ years on DBMS. Kubernaut since 2018. Day job: Altinity CEO

Diego Nieto

Software engineer at Altinity with interests in databases, Python, and Rust

Altinity

ClickHouse support and services including Altinity.Cloud
Authors of Altinity Kubernetes Operator for ClickHouse, Altinity clickhouse-backup and other open source projects

Welcome to ClickHouse®, a real-time analytic database

Understands SQL

Runs on bare metal to cloud

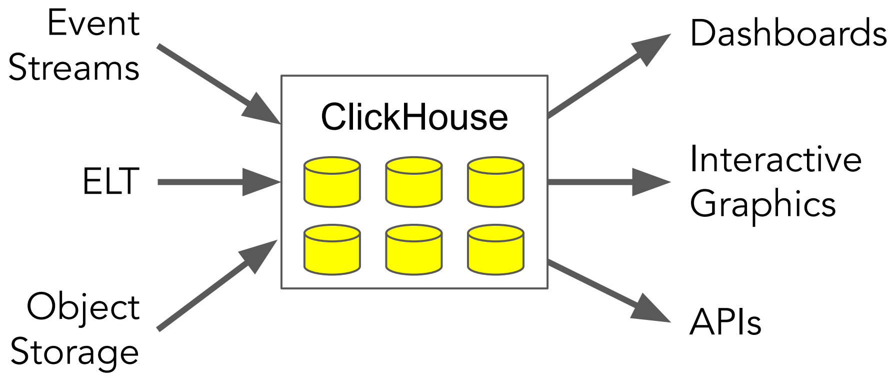
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

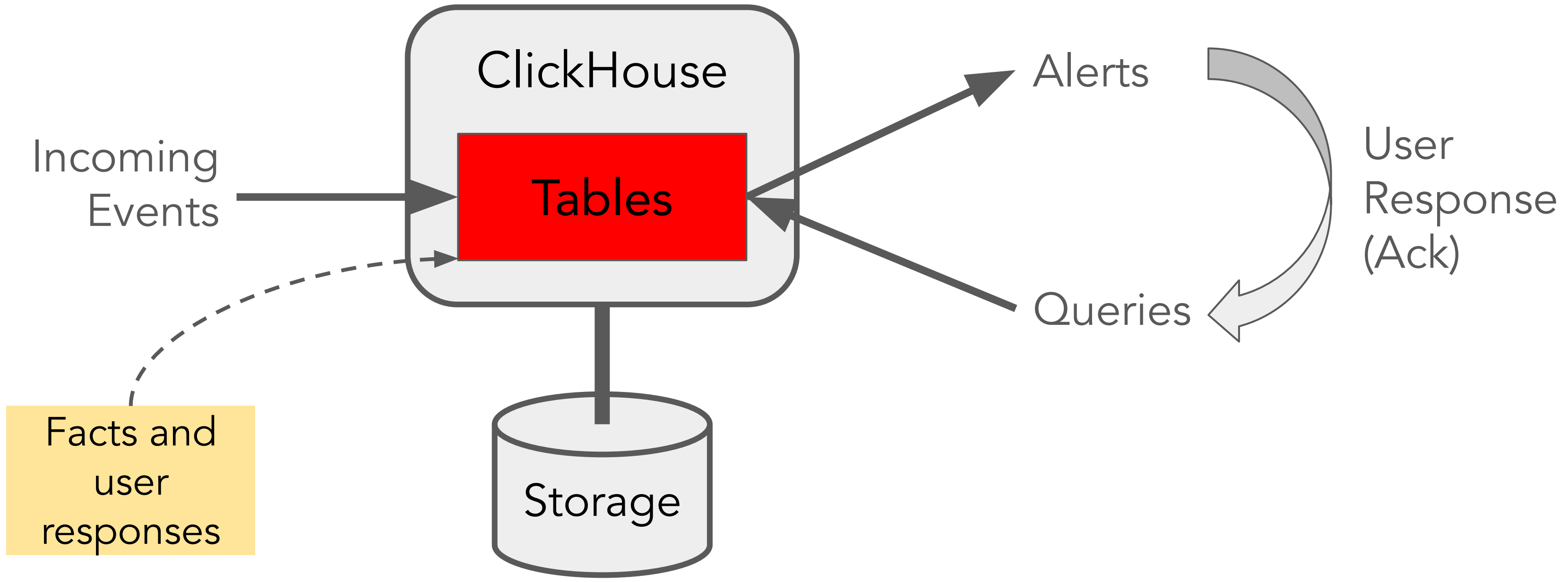
Scales to many petabytes

Is Open source (Apache 2.0)



It's the core engine for low-latency analytics

Fast data loops enable quick reaction to external events



Inserting Data Quickly

Let’s start with a simple example table

```
CREATE TABLE test (
    `sensor_id` Int32 CODEC(DoubleDelta, LZ4),
    `sensor_type` UInt8,
    `time` DateTime CODEC(DoubleDelta, LZ4),
    `date` Date ALIAS toDate(time),
    `msg_type` Enum8('reading' = 1, 'restart' = 2, 'err' = 3),
    `temperature` Decimal(5, 2) CODEC(T64, LZ4),
    `message` String DEFAULT '')
ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (msg_type, sensor_id, time)
```

The simplest way to insert data quickly

Big batches!

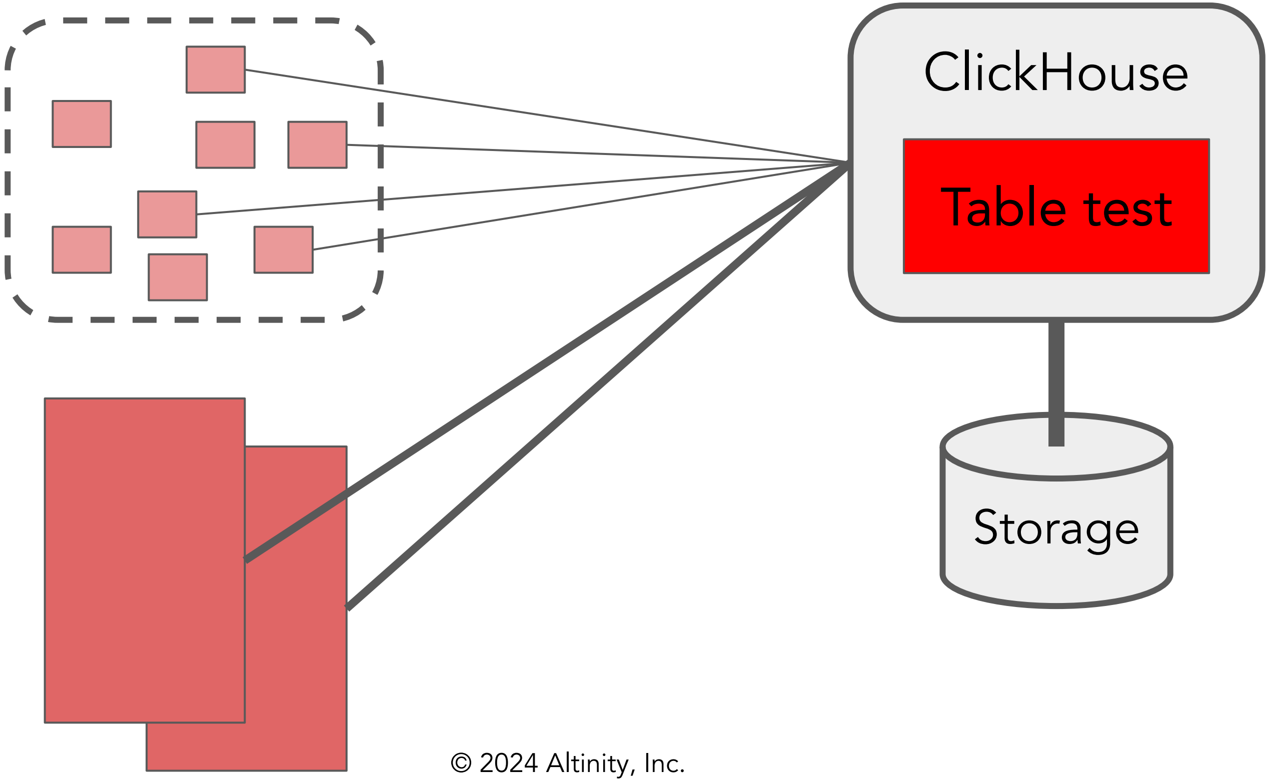
E.g.

```
#!/bin/bash
INSERT='INSERT+INTO+test+Format+CSVWithNames'
cat test.csv | curl -X POST --data-binary @- \
  "http://localhost:8123/?query=${INSERT}"
```

Where does fast insert become a problem?

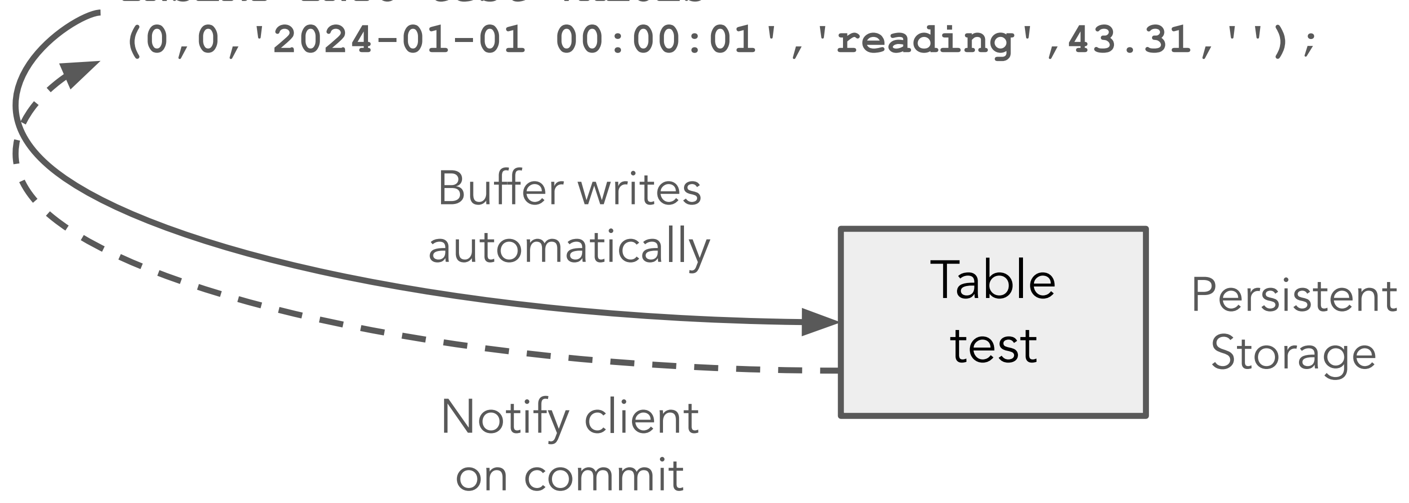
Small inserts from many clients

Extremely large inserts from clients



Async inserts are the go-to method for small updates

```
INSERT INTO test VALUES
(0,0,'2024-01-01 00:00:01','reading',43.31,'');
```



https://kb.altinity.com/altinity-kb-queries-and-syntax/async-inserts/

Enable async inserts using property settings

```
CREATE SETTINGS PROFILE IF NOT EXISTS `async_profile`
ON CLUSTER '{cluster}'
SETTINGS
  async_insert = 1,
  wait_for_async_insert=1,
  async_insert_busy_timeout_ms = 10000,
  async_insert_use_adaptive_busy_timeout = 0
;

CREATE USER IF NOT EXISTS async ON CLUSTER '{cluster}'
  IDENTIFIED WITH sha256_password BY 'topsecret' HOST ANY
  SETTINGS PROFILE `async_profile`
;
```

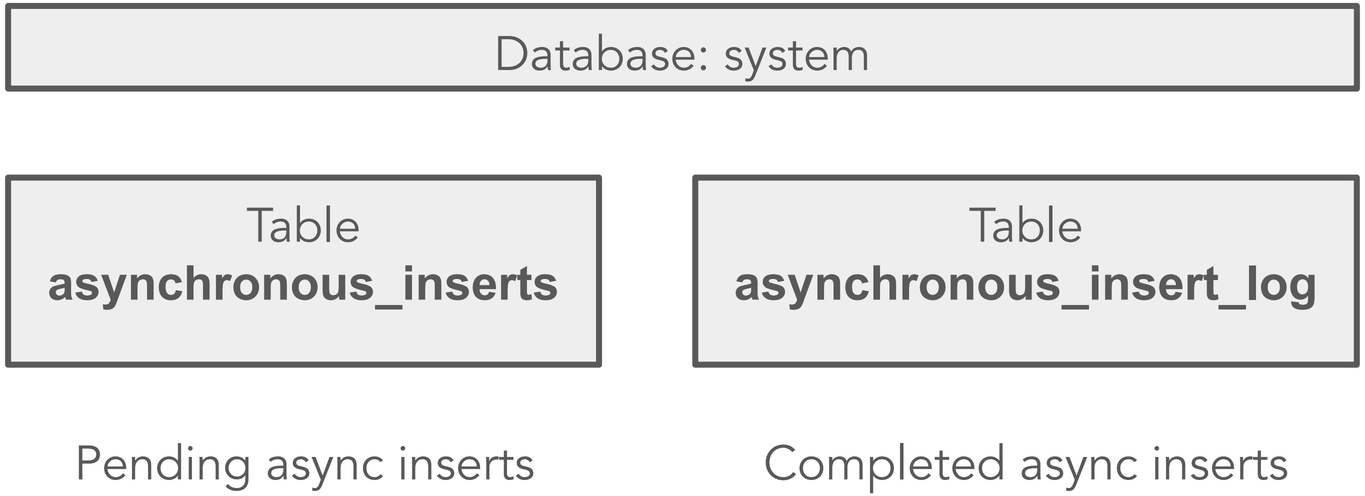
Use async insert and wait for answer

Wait this long

Don’t let ClickHouse set automatic values

User with settings

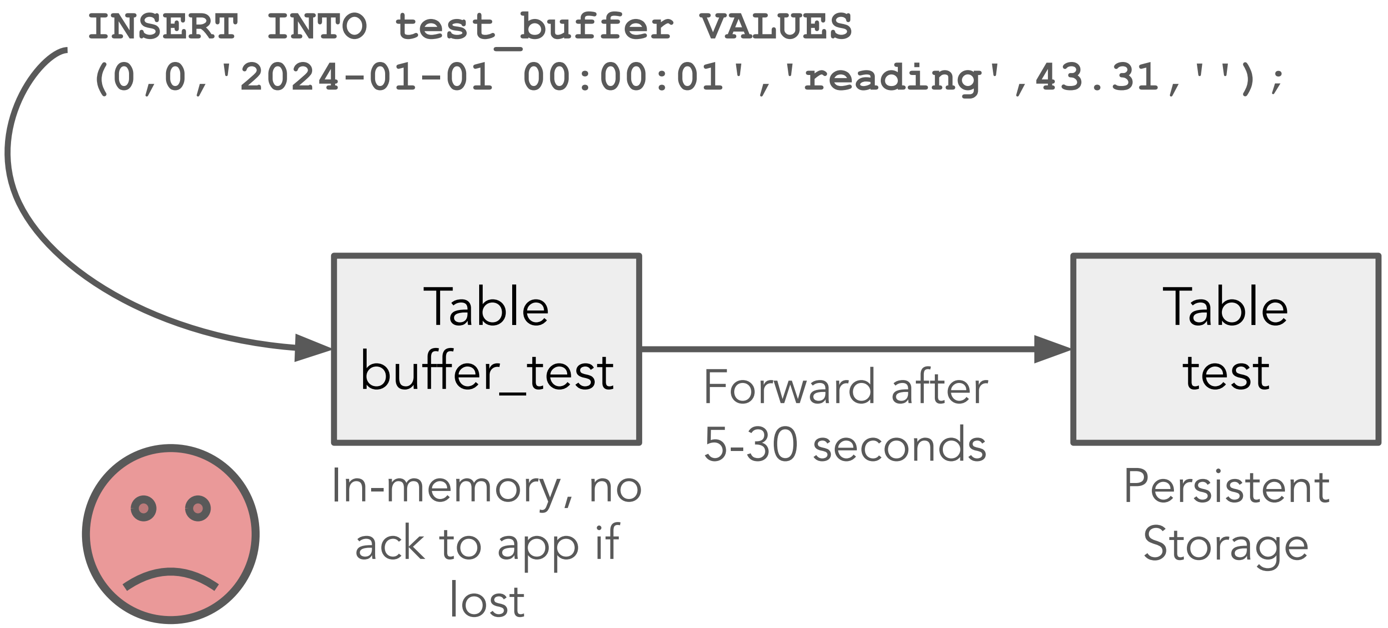
Two system tables you should know about



The old way to handle small inserts... Buffer tables

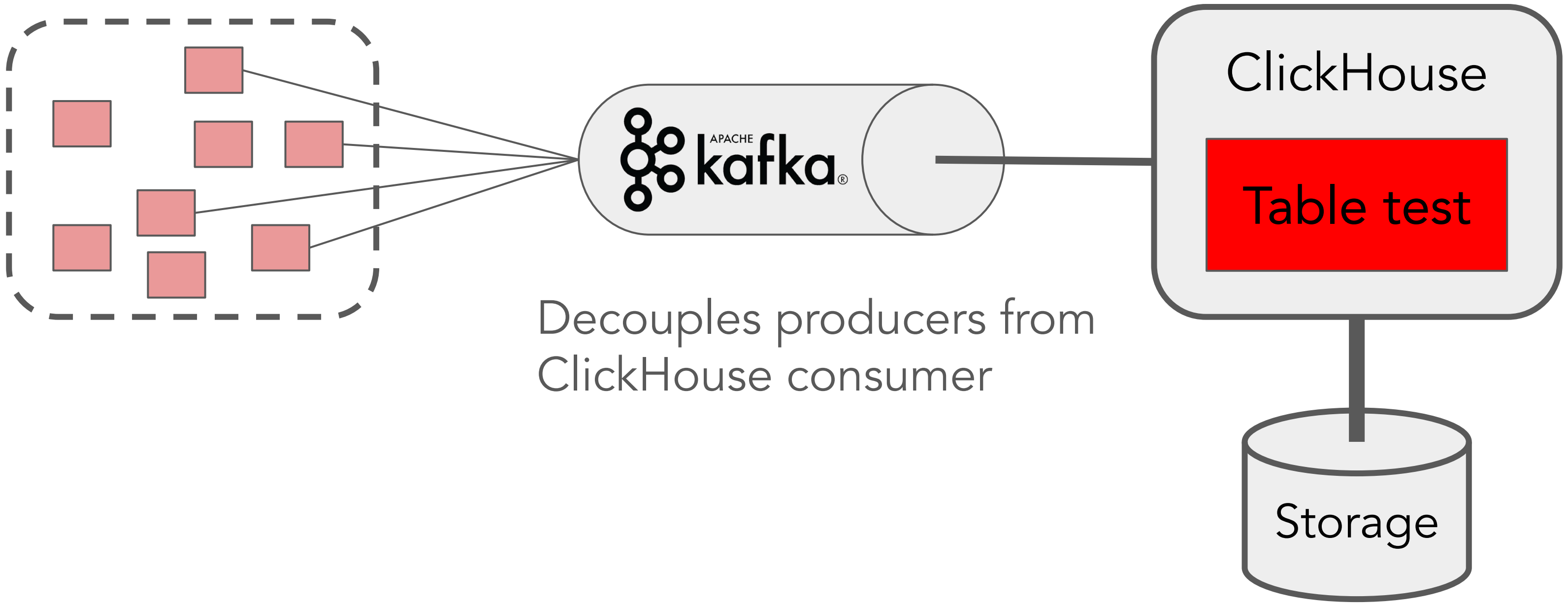
```
CREATE TABLE test_buffer AS test
ENGINE = Buffer(
  kirpi, test, -- Database and table to buffer.
  1,           -- "Layers" of independent buffers
  5,           -- Minimum time before flush
  30,          -- Maximum time ...
  1000,        -- Minimum rows ...
  1000000,     -- Maximum rows ...
  1000000,     -- Minimum bytes ...
  100000000    -- Maximum bytes ...
);

INSERT INTO test_buffer
(0,0,'2024-01-01 00:00:01','reading',43.31,'');
```

How buffer tables work



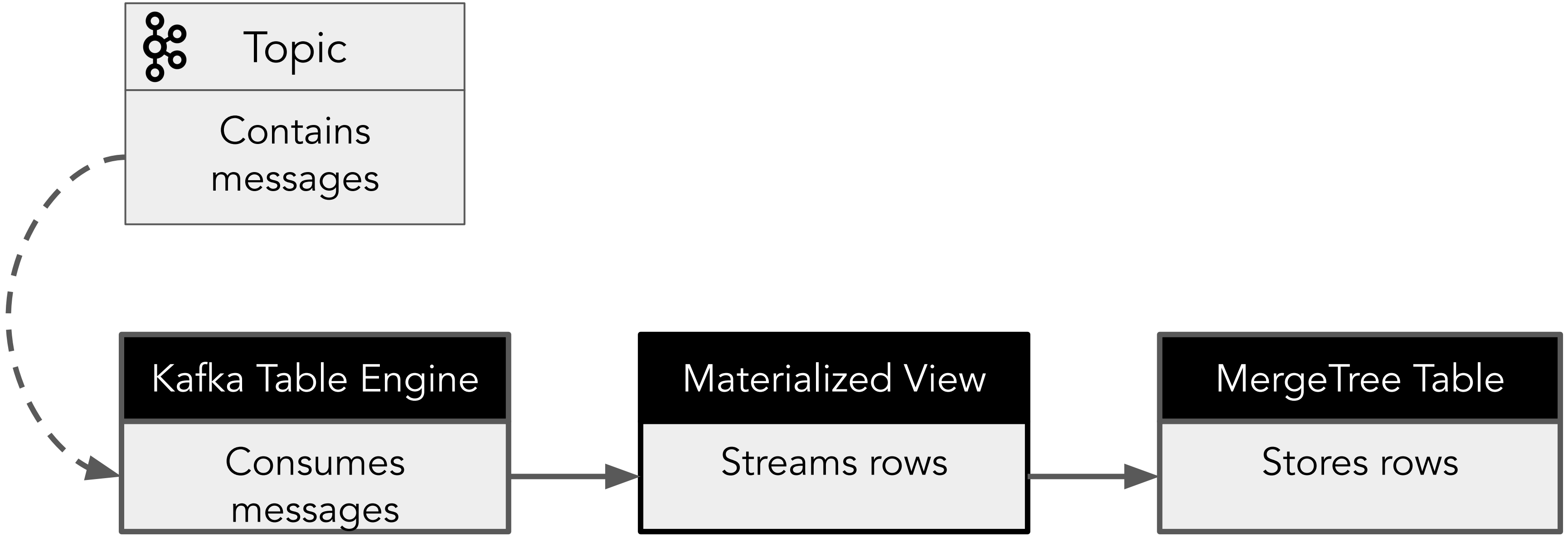
```
INSERT INTO test_buffer VALUES
(0,0,'2024-01-01 00:00:01','reading',43.31,'');
```

Another way to deal with small inserts



Decouples producers from ClickHouse consumer

Using the Kafka table engine to read from Kafka



https://kb.altinity.com/altinity-kb-integrations/altinity-kb-kafka/

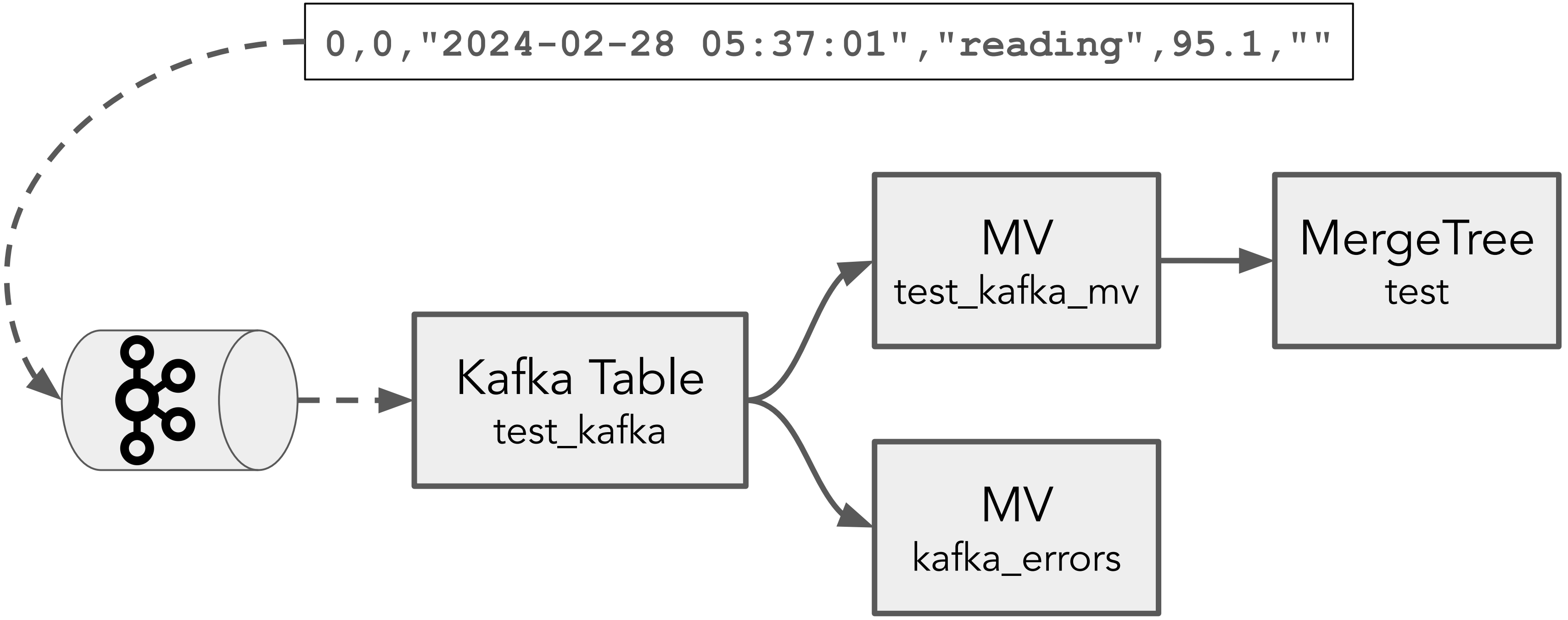
Implementing ingest with Kafka table engine

```
CREATE TABLE test_kafka (
    `sensor_id` Int32,
    `sensor_type` UInt8,
    `time` DateTime,
    `msg_type` Enum8('reading' = 1, 'restart' = 2, 'err' = 3),
    `temperature` Decimal(5, 2),
    `message` String Default ''
) ENGINE = Kafka SETTINGS
  kafka_broker_list = 'kafka-headless.kafka:9092',
  kafka_topic_list = 'test', kafka_format = 'CSVWithNames',
  ;

CREATE MATERIALIZED VIEW test_kafka_mv TO test AS SELECT
    `sensor_id`, `sensor_type`, `time`,
    `msg_type`, `temperature`,  `message`
FROM test_kafka;
```

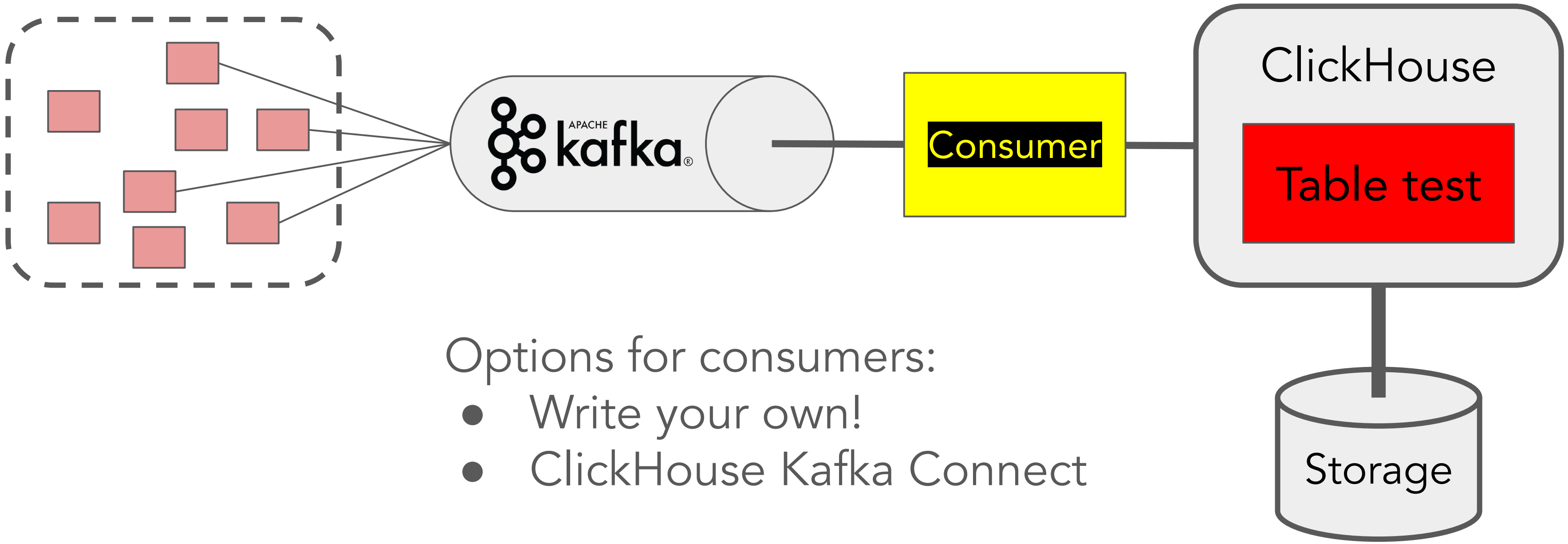
What about handling Kafka ingest errors?

```
CREATE MATERIALIZED VIEW kafka_errors
(
    `topic` String, `partition` Int64,
    `offset` Int64, `raw` String, `error` String
)
ENGINE = MergeTree
ORDER BY (topic, partition, offset)
SETTINGS index_granularity = 8192 AS
SELECT
    _topic AS topic, _partition AS partition,
    _offset AS offset, _raw_message AS raw,
    _error AS error
FROM default.kafka_engine
WHERE length(_error) > 0
```

What happens under the covers?



Use a consumer application to get custom semantics



Options for consumers:

- Write your own!
- ClickHouse Kafka Connect

Generating Alerts

The simplest way to generate alerts

Poll!

E.g.

```
SELECT * FROM test
WHERE temperature > 90.0
```

Let’s use a materialized view to make polling faster

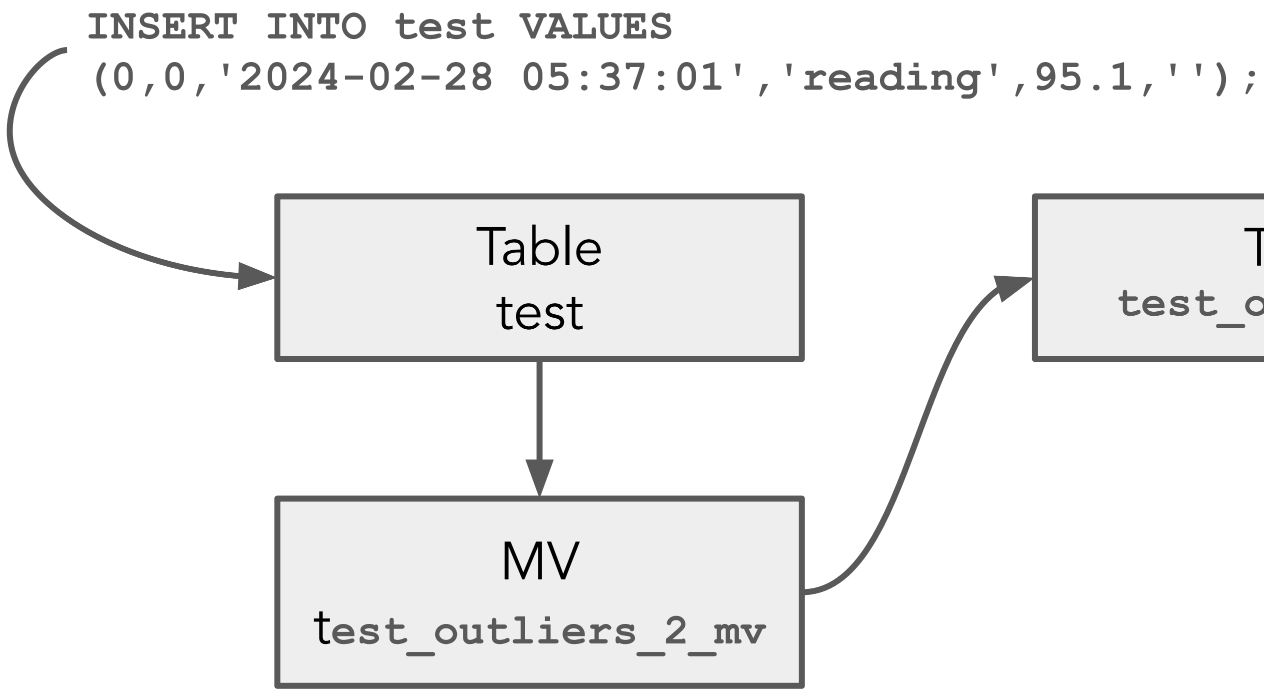
```
-- Table with max/min temperatures.
CREATE TABLE test_outliers (
    hour DateTime,
    min_temp SimpleAggregateFunction(min, Decimal(5,2)),
    max_temp SimpleAggregateFunction(max, Decimal(5,2))
) ENGINE = AggregatingMergeTree
PARTITION BY toYYYYMM(hour)
ORDER BY hour;

-- MV to compute max/min temps by hour.
CREATE MATERIALIZED VIEW test_outliers_mv TO test_outliers
AS SELECT
    toStartOfHour(time) AS hour,
    min(temperature) AS min_temp,
    max(temperature) AS max_temp
FROM test GROUP BY hour
```

We can do better! Let’s “remember” the outliers

```
-- Table with actual outlier values.
CREATE TABLE test_outliers_2 AS test
ENGINE = MergeTree
PARTITION BY toYYYYMM(time)
ORDER BY (msg_type, sensor_id, time);

-- Capture any reading whose value is over 90.
CREATE MATERIALIZED VIEW test_outliers_2_mv TO test_outliers_2
AS SELECT * FROM test WHERE temperature > 90.0;
```

Altinity

What happens under the covers?

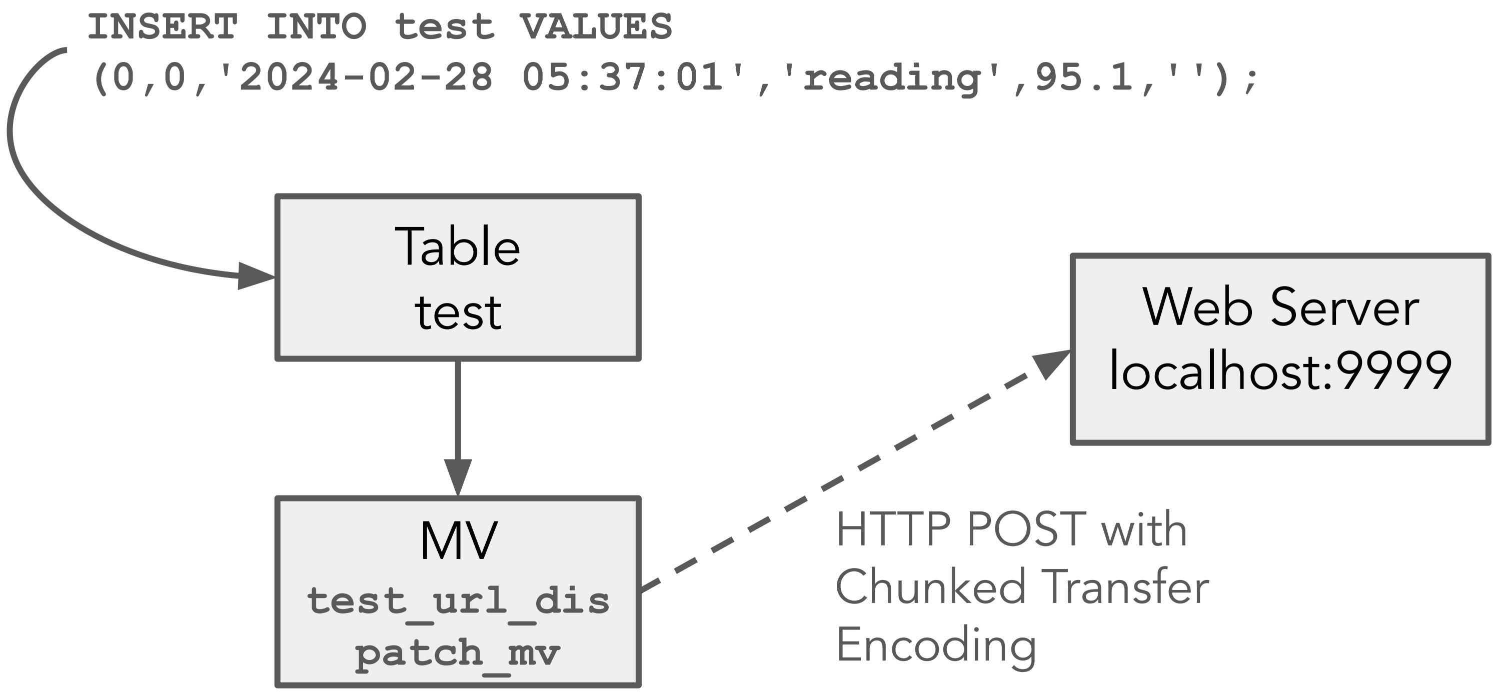


Still better, let's tell somebody about it

```
-- Use URL engine to send a notification.
CREATE TABLE test_url_dispatch (
  sensor_id Int32,
  sensor_type UInt8,
  time DateTime,
  msg_type Enum8('reading' = 1, 'restart' = 2, 'err' = 3),
  temperature Decimal(5,2),
  message String
) ENGINE = URL('http://localhost:9999/', CSV)

-- Capture any reading whose value is over 90.
CREATE MATERIALIZED VIEW test_url_dispatch_mv TO test_url_dispatch
AS SELECT * FROM test WHERE temperature > 90.0 Format CSV;
```

Altinity

What happens under the covers?



INSERT INTO test VALUES (0,0,'2024-02-28 05:37:01','reading',95.1,'');
Table test
MV test_url_dispatch_mv
Web Server localhost:9999
HTTP POST with Chunked Transfer Encoding

Example of posting to Kafka

```
CREATE TABLE kafka_consumer (
    `sensor_id` Int32,
    `sensor_type` UInt8,
    `time` DateTime,
    `msg_type` Enum8('reading' = 1, 'restart' = 2, 'err' = 3),
    `temperature` Decimal(5, 2),
    `message` String Default ''
) ENGINE = Kafka SETTINGS
  kafka_broker_list = 'kafka-headless.kafka:9092',
  kafka_topic_list = 'test-consumer', kafka_format = 'CSVWithNames';

-- MV to check if temp is higher than 90 CELSIUS & capture.
CREATE MATERIALIZED VIEW temp_alert_mv TO temp_alerts
AS SELECT
    time as timestamp, temperature as temperature
FROM test_kafka_consumer WHERE temperature > 90
```

Example of posting to Kafka (continuation)

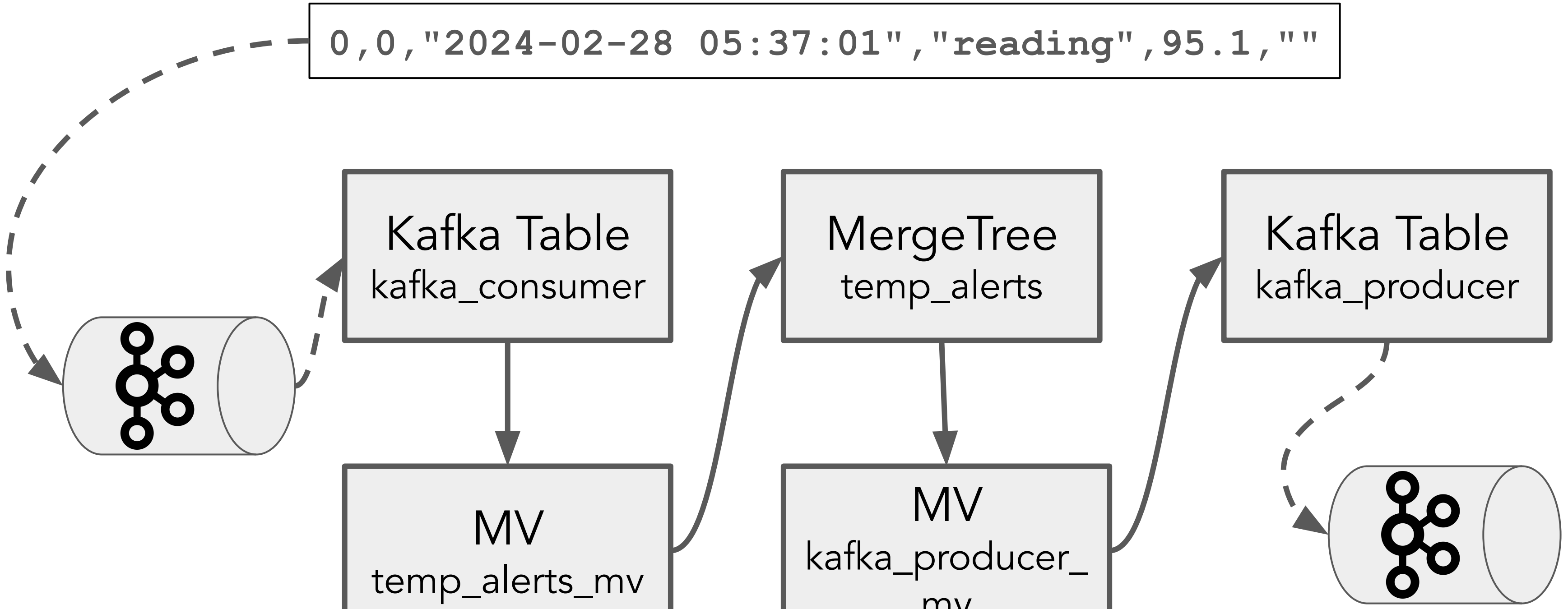
```
-- Intermediate table to store alerts
CREATE TABLE temp_alerts (
    `time` DateTime,
    `temperature` Decimal(5, 2)
    `alert_level` Enum8('moderate' = 1, 'high' = 2, 'critical' = 3),

) ENGINE = MergeTree() ORDER BY tuple()

--Push data from test_temp_alerts to kafka topic test-alerts
CREATE TABLE kafka_producer (
    `time` DateTime,
    `temperature` Decimal(5, 2),
    `severity` Enum8('moderate' = 1, 'high' = 2, 'critical' = 3),
) ENGINE = Kafka SETTINGS
  kafka_broker_list = 'kafka-headless.kafka:9092',
  kafka_topic_list = 'test-alerts', kafka_format = 'CSVWithNames';
```

Example of posting to Kafka (continuation)

```
-- MV to check push an alert to kafka topic test-alerts
-- It checks if temperature is in a range and tags the alert level
CREATE MATERIALIZED VIEW kafka_producer_mv TO kafka_producer
AS SELECT
    time as timestamp
    temperature as temperature
    multiIf(
        temperature > 90 AND < 120, ‘moderate’,
        temperature > 120 AND < 150, ‘high’,
        temperature > 150, ‘critical’, ‘Error’) AS severity
FROM temp_alerts
```

Altinity

What happens under the covers?



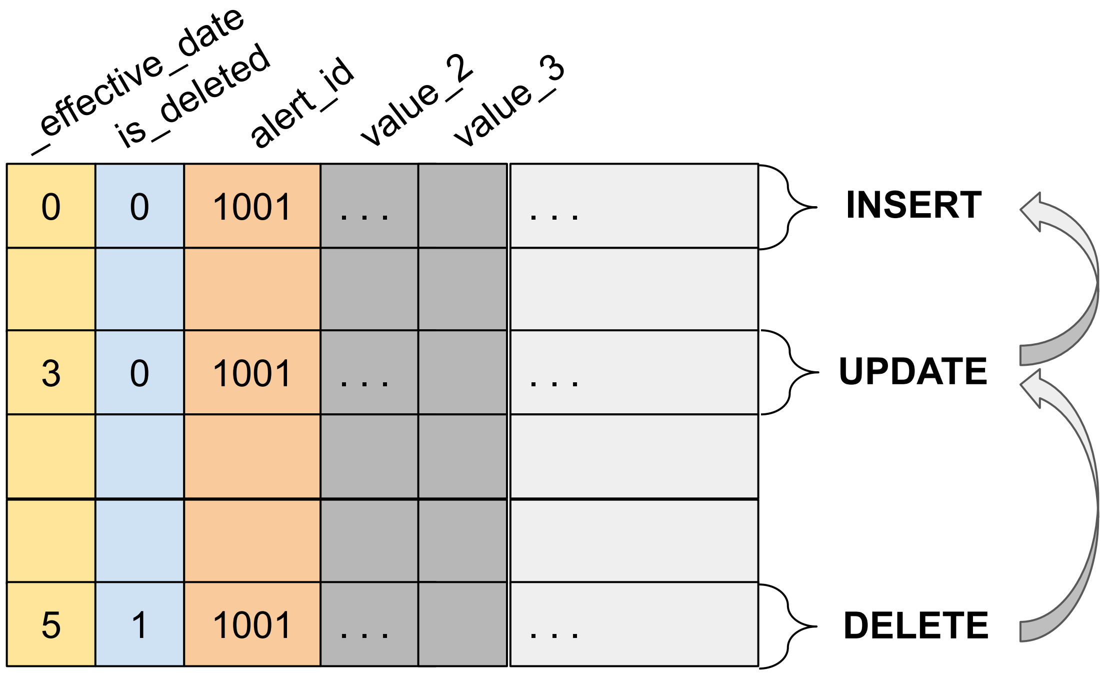
Updating State Related to Alerts

Using ReplacingMergeTree tables to track alerts

```
CREATE TABLE test_temp_alerts_rmt (
    `alert_id` UInt64,
    `time` DateTime,
    `temperature` Decimal(5, 2),
    `alert_level` Enum8('moderate' = 1, 'high' = 2, 'critical' = 3),
    `acknowledged` UInt8 DEFAULT 0,
    `effective_date` DateTime DEFAULT now(),
    is_deleted UInt8 DEFAULT 0
)
Engine=ReplacingMergeTree(effective_date, is_deleted)
PARTITION BY toDate(time)
ORDER BY alert_id;
```

How ReplacingMergeTree works

Eventually consistent replacement of rows



Using ReplicatedMergeTree tables to track alerts

```
-- Add an alert
INSERT INTO test_temp_alerts_rmt(
   alert_id, time, temperature, alert_level)
VALUES (1001, '2024-01-01 20:22:20', 95.6, 3);

SELECT * FROM test_temp_alerts_rmt SETTINGS final = 1 \G

Row 1:
──────
alert_id:       1001
time:           2024-01-01 20:22:20
temperature:    95.6
alert_level:    critical
acknowledged:   0
effective_date: 2024-11-26 05:35:43
is_deleted:     0
```

Altinity

Using ReplicatedMergeTree tables to track alerts

```
-- Acknowledge the alert.
INSERT INTO test_temp_alerts_rmt(alert_id, time, temperature,
  alert_level, acknowledged) VALUES
  (1001, '2024-01-01 20:22:20', 95.6, 3, 1);

SELECT * FROM test_temp_alerts_rmt SETTINGS final = 1 \G

Row 1:
──────
alert_id:       1001
time:           2024-01-01 20:22:20
temperature:    95.6
alert_level:    critical
acknowledged:   1
effective_date: 2024-11-26 05:38:10
is_deleted:     0
```

Altinity

What are some other options for ack’ing alerts?

- Use a materialized view to track the last value of the alert state (“last-point” query)
- Put the alert in MySQL using MySQL table engine
- Put the alert in RocksDB using EmbeddedRocksDB table engine
    - Not replicated
- Use an external application

Wrapping Up

You too could be an expert at ClickHouse!

https://altinity.com/clickhouse-training/

More reading...

- Altinity Knowledge Base (https://kb.altinity.com/altinity-kb-setup-and-maintenance/rbac)
- Altinity blog (https://altinity.com/blog)
- ClickHouse code (https://github.com/ClickHouse/ClickHouse)
- ClickHouse docs (https://clickhouse.com/docs)

Thank you! Any questions?

Website: https://altinity.com
Slack: https://www.altinity.com/slack
GitHub: https://github.com/Altinity
Ask for Diego or Robert!

Altinity Stable Builds
Enterprise Support for ClickHouse

Altinity.Cloud
Altinity Kubernetes Operator for ClickHouse