Replicating MySQL to ClickHouse

Altinity

preFOSDEM MySQL Days, January 30 2025

Who I am

Arnaud Adant

Database Team Lead

Jump Trading

Main developer and contributor

Kanthi Subramanian, Altinity

What is a transactional database ?

- Row based (tables, rows, …)
- B-tree data structure
- Speak SQL
- ACID
- Transactions
- MySQL, Postgres, … MS SQL, Oracle, …

What is ClickHouse ?

- An Open Source columnar database
- MergeTree data structure
- Created in 2009
- 38.6k stars
- Realtime Analytics
- VLDB 2024 - ClickHouse: Lightning Fast Analytics for Everyone
- Over 2000 contributors

What is replication ?

- Data synchronization between a primary and replica(s)
- Homogeneous
- Heterogeneous
- Logical replication
- Log based
- Change Data Capture

Why replication ?

- **Migration : system A to B**
- **Continuous synchronization**
- **Fault tolerance**
- **Disaster recovery**
- **Very useful building block**

Requirements

- No Data Loss
- Low Latency
- Exactly Once Delivery
- Operational Simplicity
- Fault Tolerance / HA
- Fully Open Source

Design choices

- **Keep It Simple Stupid**
  - A docker container, simple yaml config

- **Standing on the shoulders of giants :**
  - MySQL binary logs, ANTLR, PG Logical replication

- **Do not re-invent the wheel :**
  - Debezium for Change Data Capture

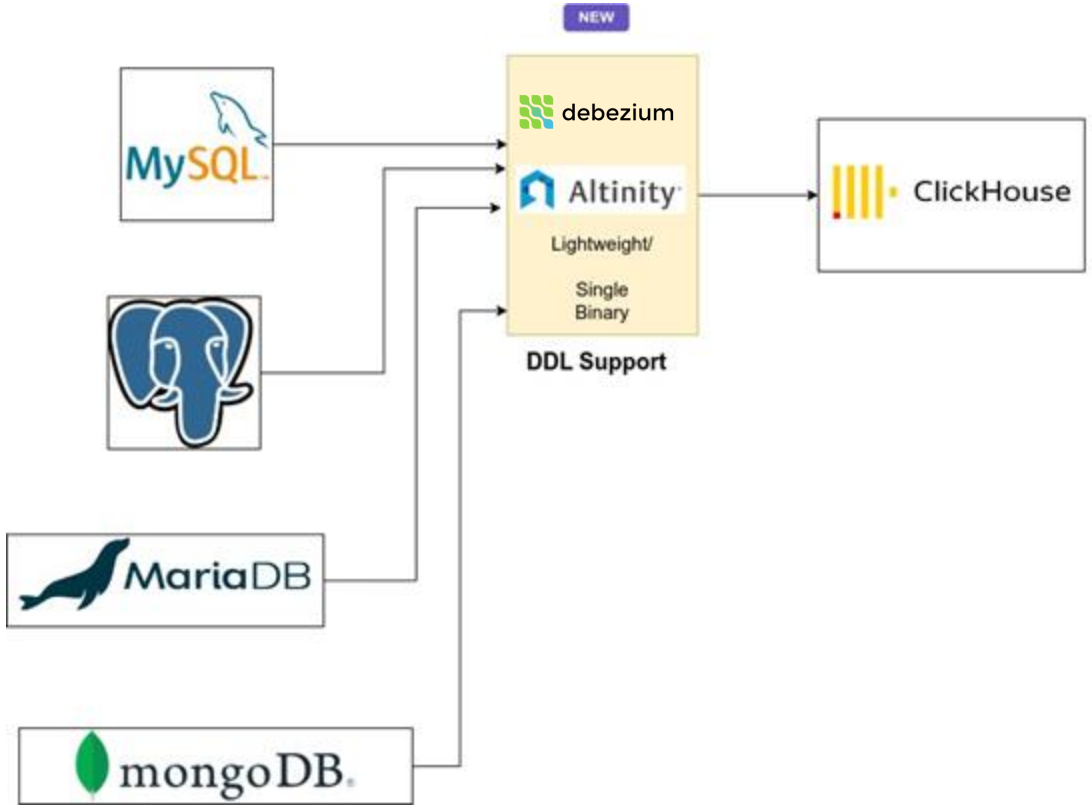
Design choices in ClickHouse (1/2)

- One to One Table mapping
- Primary key
- Data Type Mapping
- ReplacingMergeTree engine, aka RMT
- “the same queries should return the same results”
- Full DDL Support
- Timezones

Design choices in ClickHouse (1/2)

- **Replication state stored in ClickHouse**
- **MySQL replication like behavior (stop / start / status)**
- **Retry on failure, both DML and DDL**
- **Replication filters**
- **Checksums**
- **Efficient dumpers and loaders**

Architecture



Implementation : Sink Connector Lightweight

- **One container**
- **Docker compose**
- **Java and Debezium based**
- **Multi-threaded Applier**
- **Eventually consistent**
- **Low Latency**

Table Migration

MySQL

```
CREATE TABLE `orders` (
`orderNumber` int NOT NULL,
`orderDate` date NOT NULL,
`requiredDate` date NOT NULL,
`shippedDate` date DEFAULT NULL,
`status` varchar(15) NOT NULL,
`comments` text,
`customerNumber` int NOT NULL,

PRIMARY KEY (`orderNumber`),
KEY `customerNumber` (`customerNumber`) )
ENGINE=InnoDB DEFAULT CHARSET=latin1

PARTITION BY RANGE COLUMNS(orderDate)
(
PARTITION p20201231 VALUES LESS THAN ('2021-01-01'),
PARTITION p20211230 VALUES LESS THAN ('2021-12-31')
)
```

ClickHouse

```
CREATE TABLE test.orders (
`orderNumber` Int32,
`orderDate` Date32,
`requiredDate` Date32,
`shippedDate` Nullable(Date32),
`status` String,
`comments` Nullable(String),
`customerNumber` Int32,

`_version` UInt64,
`is_deleted` UInt8 )

ENGINE = ReplacingMergeTree(_version, is_deleted)
ORDER BY orderNumber
PARTITION BY orderDate
SETTINGS index_granularity = 8192
```

Altinity

Why is MySQL so good in this context ?

- **Excellent replication, outperforms the competition**
- **Excellent exporter (mysqlsh)**
- **Battle tested**
- **Excellent Debezium support**
- **Performant**
- **Fast counts**
- **pt-heartbeat**

mysqlsh

- The best tool to export data in TSV format
- Uses zstd for compression (Zstandard)
- Consistent snapshot
- Gives the replication position
- Wrapper mysql_dumper.py (dump)
- See also clickhouse_loader.py (load)

Checksums

- Compare MySQL and CH tables / result sets
- Inspired by pt-table-checksum
- Order independent
- Inspired by Sisense article
- https://www.sisense.com/blog/hashing-tables-to-ensure-consistency-in-postgres-redshift-and-mysql/
- Multi-threaded
- mysql_table_checksum.py /clickhouse_table_checksum.py

Fast counts

- MySQL count(*) is usually slow on very large tables
- Multi-threaded count available since 8.0.14
- Works on partitions (!)
- Can be further parallelized by partition
- mysql_table_count.py / clickhouse_table_count.py

Reference :

https://www.percona.com/blog/mysql-8-0-14-a-road-to-parallel-query-execution-is-wide-open/

Fast counts

```
mysql> select count(*) from ontime;
+-----------+
| count(*)  |
+-----------+
| 177920306 |
+-----------+
1 row in set (2 min 33.93 sec)

mysql> set local innodb_parallel_read_threads=32;
Query OK, 0 rows affected (0.00 sec)

mysql> select count(*) from ontime;
+-----------+
| count(*)  |
+-----------+
| 177920306 |
+-----------+
1 row in set (5.35 sec)
```

Heartbeat

A MUST HAVE

```
CREATE TABLE `heartbeat` (
`ts` varchar(26) NOT NULL,
`server_id` int unsigned NOT NULL,
 `file` varchar(255) DEFAULT NULL,
`position` bigint unsigned DEFAULT NULL,
`relay_master_log_file` varchar(255) DEFAULT NULL,
`exec_master_log_pos` bigint unsigned DEFAULT NULL,
PRIMARY KEY (`server_id`) ) ENGINE=InnoDB DEFAULT
CHARSET=latin1
```

```
root     20887  0.0  0.0 107248 25168 ?       S     2023  30:42 /usr/bin/perl
/usr/bin/pt-heartbeat --config /dev/null --interval=10 --defaults-file
/etc/mysql/percona-toolkit.conf --update --stop --sentinel /tmp/pt-heartbeat-
hourly --database server_team_replicate
```

Heartbeat

```
mysql> select * from server_team_replicate.heartbeat where ts > '202501-01'\G
*************************** 1. row ***************************
                  ts: 2025-01-29T11:25:32.000560
           server_id: 216
                file: binary.116202
            position: 452830306
relay_master_log_file: NULL
  exec_master_log_pos: NULL
1 row in set (0.00 sec)
```

Automatic Primary Key

- From 8.0.31,
- sql_generate_invisible_primary_key = 1

(automatic PK generation)

- sql_require_primary_key = 1

(force all tables to have a PK)

- Recommended in any replication setup

Demo

see also https://github.com/Altinity/clickhouse-sink-connector/discussions/810

Appendix

Features

Features

Feature	Description
Single Binary	No additional dependencies or infrastructure required
Exactly Once Processing	Offsets are committed to ClickHouse after the messages are written to ClickHouse
Supported Databases	MySQL, MariaDB, PostgreSQL, MongoDB(Experimental)
Supported ClickHouse Versions	24.8 and above
Clickhouse Tables Types	ReplacingMergeTree, MergeTree, ReplicatedReplacingMergeTree
Replication Start positioning	Using sink-connector-client to start replication from a specific offset or LSN(MySQL Binlog Position, PostgreSQL LSN)
Supported Datatypes	Refer Datatypes
Initial Data load	Scripts to perform initial data load (MySQL)
Fault Tolerance	Sink Connector Client to continue replication from the last committed offset/LSN in case of a failure
Update, Delete	Supported with ReplacingMergeTree
Monitoring	Prometheus Metrics, Grafana Dashboard
Schema Evolution	DDL support for MYSQL.
Deployment Models	Docker Compose, Java JAR file, Kubernetes
Start, Stop, Pause, Resume Replication	Supported using sink-connector-client
Filter sources databases, tables, columns	Supported using debezium configuration.
Map source databases to different ClickHouse databases	Database name overrides supported.
Column name overrides	Planned
MySQL extensive DDL support	Full list of DDL(sink-connector-lightweight/docs/mysql-ddl-support.md)
Replication Lag Monitoring	Grafana Dashboard and view to monitor lag
Batch inserts to ClickHouse	Configurable batch size/thread pool size to achieve high throughput/low latency
MySQL Generated/Alias/Materialized Columns	Supported
Auto create tables	Tables are automatically created in ClickHouse based on the source table structure.

Comparison - MySQL DB Engine

- Only SELECT and Insert queries
- Proxy table to copy data from mysql data table.

Comparison - MySQL Table Engine /MySQL DB Engine

MySQL

```
CREATE TABLE `offices` (
  `officeCode` varchar(10) NOT NULL,
  `city` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL,
  `addressLine1` varchar(50) NOT NULL,
  `addressLine2` varchar(50) DEFAULT NULL,
  `state` varchar(50) DEFAULT NULL,
  `country` varchar(50) NOT NULL,
  `postalCode` varchar(15) NOT NULL,
  `territory` varchar(10) NOT NULL,
  PRIMARY KEY (`officeCode`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

ClickHouse

```
create table mysql_table_offices(
officeCode String,
city String, phone String,
addressLine1 String,
addressLine2 String,
 state String,
country String,
postalCode String,
territory String)
ENGINE= MySQL('mysql-master:3306', 'test',
'offices','root','root')

CREATE DATABASE mysql_db ENGINE =
MySQL('localhost:3306', 'test', 'my_user',
'user_password') SETTINGS
read_write_timeout=10000,
connect_timeout=100;
```

Comparison - MySQL Engine

MySQL

```
CREATE TABLE `offices` (
  `officeCode` varchar(10) NOT NULL,
  `city` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL,
  `addressLine1` varchar(50) NOT NULL,
  `addressLine2` varchar(50) DEFAULT NULL,
  `state` varchar(50) DEFAULT NULL,
  `country` varchar(50) NOT NULL,
  `postalCode` varchar(15) NOT NULL,
  `territory` varchar(10) NOT NULL,
  PRIMARY KEY (`officeCode`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

ClickHouse

```
create table mysql_table_offices(

officeCode String,
city String, phone String,
addressLine1 String,
addressLine2 String,
 state String,
country String,
postalCode String,
territory String)

ENGINE= MySQL('mysql-master:3306', 'test',
'offices','root','root')
```

Comparison - MySQL Engine

```
d4094e4527d2 :) select * from mysql_table_offices;

SELECT *
FROM mysql_table_offices

Query id: c8cdaf36-dae4-4c40-b809-40841bffa57c
```

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans			France	75017	EMEA
5	Tokyo	+81 33 224 5000	4-1 Kioicho		Chiyoda-Ku	Japan	102-8578	Japan
6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2		Australia	NSW 2010	APAC
7	London	+44 20 7877 2041	25 Old Broad Street	Level 7		UK	EC2N 1HN	EMEA

Comparison - MySQL Engine(DDL changes)

```
mysql> alter table offices add column managerName varchar(500);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0


mysql> show create table offices;

| offices | CREATE TABLE `offices` (
  `officeCode` varchar(10) NOT NULL,
  `city` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL,
  `addressLine1` varchar(50) NOT NULL,
  `addressLine2` varchar(50) DEFAULT NULL,
  `state` varchar(50) DEFAULT NULL,
  `country` varchar(50) NOT NULL,
  `postalCode` varchar(15) NOT NULL,
  `territory` varchar(10) NOT NULL,
  `managerName` varchar(500) DEFAULT NULL,
  PRIMARY KEY (`officeCode`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+---------+-----------------------------------------------------------------------------
--------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------
---------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Comparison - MySQL DB Engine(DDL changes)

```
SHOW CREATE TABLE offices

Query id: 21ddac4f-3f8e-4bfc-aa9c-64fa4000671c

CREATE TABLE mysql_db.offices
(
   `officeCode` String,
   `city` String,
   `phone` String,
   `addressLine1` String,
   `addressLine2` Nullable(String),
   `state` Nullable(String),
   `country` String,
   `postalCode` String,
   `territory` String,
   `managerName` Nullable(String)
)
ENGINE = MySQL('mysql-master:3306', 'test', 'offices', 'root', '[HIDDEN]')
```

Comparison - MySQL DB Engine(Drawbacks)

MySQL:
create table sales_new(sales_id DECIMAL(10,3) PRIMARY KEY NOT NULL);

ClickHouse:

show create table sales_new;

SHOW CREATE TABLE sales_new

Query id: 1e734fee-fbe1-49a0-94aa-53628921260a

```
CREATE TABLE mysql_db.sales_new
(
    `sales_id` String
)
ENGINE = MySQL('mysql-master:3306', 'test', 'sales_new', 'root', '[HIDDEN]')
```

1 row in set. Elapsed: 0.003 sec.

Comparison

Feature	Altinity Sink Connector (Lightweight, Single Binary)	Airbyte	ClickHouse `mysql` Table Engine	Custom Python Script with ClickHouse Connect
Replication Type	Real-time CDC	Batch (Scheduled)	Direct Query	Batch or Scheduled
Data Freshness	Near real-time	Configurable (e.g., hourly)	Near real-time (with latency)	Configurable
Schema Change Handling	Full support(MySQL), Partial(PostgreSQL)	Manual schema refresh required	No automatic schema sync	Manual intervention needed
Complexity	Low to Medium (single binary setup)	Moderate	Low	High (requires coding and scheduling)
Ease of Setup	Easy (standalone binary, no Kafka needed)	Easy	Very easy	Complex (custom coding)
Maintenance	Low to Moderate (single binary process)	Low	Low	High
Initial Sync Support	Yes	Yes	Not applicable (direct query)	Yes
Transformation Capabilities	Limited	Basic (Airbyte transformations)	No	Full control (custom code)
Cost	Free or license-based	Free (Open-source)	Free (built-in to ClickHouse)	Free (but may require custom infrastructure)
Suitability for High Volume	High	Medium	Medium	Medium to Low
Additional Infrastructure	None	None	None	Optional (scheduling tools like Airflow)
Data Accuracy	High (real-time CDC)	Medium (depends on sync frequency)	Medium	High
Ideal Use Case	Low-latency, real-time replication without Kafka	Batch syncs, easy setup	Simple queries without replication	Custom, flexible ETL

Handling DDL

- DDL’s from MySQL cannot be translated to ClickHouse.
- Example `CREATE TRIGGER`
- Need a mechanism to ignore DDL so that
- The replication can continue.

Monitoring

Monitor Lag, DDL and CPU/Memory Usage using Grafana Dashboard.
Start/Stop replication and monitor lag using sink connector CLI.

Altinity

Development

- Actively developed with feedback from Customers with serious production workloads

- Contributors with expertise in ClickHouse, JDBC and Debezium.

Altinity

Performance

- Configurable ClickHouse writer thread pool.

- Configurable Queue size (tune between max memory usage vs max throughput)

- Configurable batch size (Batch records inserted to ClickHouse)

Roadmap

- History tables
- Override column Schema (custom mapping of data types)
- Configuration builder for non-expert users.
- Support for Transactions.
- Support for more source databases (MongoDB, Cassandra, SQL Server, Oracle)

Performance