Lab Report: Cassandra vs. MySQL for Event Tracking

---

1. Dataset Description

Generation Method A synthetic dataset was generated using `generate_data.py`. The script used the `pandas` library for data handling and `faker` for realistic user, product, and session data.

Statistics - Total events: 4,000 - Unique users: 100 (user_1 to user_100) - Unique products: 250 (prod_1 to prod_250) - Columns (10): event_id, user_id, session_id, event_time, event_type, product_id, category, price, city, device_type

Example Rows [INSERT SCREENSHOT: Output of `generate_data.py` showing "Successfully generated" and `df.head()`]

---

2. MySQL (Relational Model) Design

Table Design A single table, `events`, was created in `quickkart_db`. `event_id` is the PRIMARY KEY.

```
CREATE TABLE events (
    event_id INT PRIMARY KEY,
    user_id VARCHAR(50),
    session_id VARCHAR(50),
    event_time DATETIME,
    event_type VARCHAR(50),
    product_id VARCHAR(50) NULL,
    category VARCHAR(50) NULL,
    price DECIMAL(10,2) NULL,
    city VARCHAR(50),
    device_type VARCHAR(50),
    is_flagged BOOLEAN DEFAULT FALSE,
    payment_method VARCHAR(20) NULL
);
```

CRUD Scenarios

- **CREATE:** Loaded 4,000 events using `load_mysql.py` via `df.to_sql()`. [INSERT SCREENSHOT: Terminal output showing "--- SUCCESS! ---"]
- **READ:**
- User History: `SELECT * FROM events WHERE user_id = '...' ORDER BY event_time DESC;` [INSERT SCREENSHOT: Adminer query result]

- Product Popularity: `SELECT COUNT(*) FROM events WHERE event_type = 'purchase' GROUP BY product_id;` [INSERT SCREENSHOT]
- City-wise Purchases: `SELECT COUNT(*) FROM events WHERE event_type = 'purchase' GROUP BY city;` [INSERT SCREENSHOT]
- **UPDATE:** Flagging a session with `UPDATE events SET is_flagged = TRUE WHERE session_id = '...';` [INSERT SCREENSHOT]
- **DELETE:** Deleting all events for a user with `DELETE FROM events WHERE user_id = '...';` [INSERT SCREENSHOT]

---

3. Cassandra (Wide-Column Model) Design

Table Design (Query-First) Keyspace: `quickkart_keyspace` Table: `events_by_user` - Partition Key: `user_id` - Clustering Keys: `(event_time DESC, event_id ASC)`

```
CREATE TABLE IF NOT EXISTS events_by_user (
    user_id text,
    event_time timestamp,
    event_id int,
    session_id text,
    event_type text,
    product_id text,
    category text,
    price decimal,
    city text,
    device_type text,
    is_flagged boolean,
    payment_method text,
    PRIMARY KEY (user_id, event_time, event_id)
) WITH CLUSTERING ORDER BY (event_time DESC, event_id ASC);
```

CRUD Scenarios

- **CREATE:** Loaded row by row with `load_cassandra.py`. Optimized for high-throughput continuous writes. [INSERT SCREENSHOT]
- **READ:**
- User History: `SELECT * FROM events_by_user WHERE user_id = 'user_5' LIMIT 10;` (very fast) [INSERT SCREENSHOT]

- Analytics (e.g., Product Popularity, City-wise Purchases): Full table scan required with `ALLOW FILTERING` (slow) [INSERT SCREENSHOT]

- **UPDATE:** Flagging session required scanning all rows (`ALLOW FILTERING`) and individual updates (slow) [INSERT SCREENSHOT]

- **DELETE:**

- Delete User: `DELETE FROM events_by_user WHERE user_id = 'user_10';` (fast, single partition)
- Old Data Cleanup: Full scan + individual deletes (slow; TTL recommended) [INSERT SCREENSHOT]

---

4. Final Comparison & Analysis

Operation	MySQL	Cassandra	Winner
Insert many events	Easy bulk insert via `to_sql`	Row-by-row insert; designed for high write throughput	MySQL (easy one-time), Cassandra (high-throughput)
Read recent events of one user	Index lookup; slows with size	Partition key; always fast	Cassandra
Analytics (purchases per city)	Fast with `GROUP BY`	Requires full scan; slow	MySQL
Update/Delete by non-PK	Flexible and fast	Slow; read-modify-write pattern	MySQL
Delete by partition	Slower; row-by-row	Instant (partition delete)	Cassandra

Key Takeaways

- **MySQL:** Best for ad-hoc queries, relational data, data consistency (ACID), and flexible updates/deletes.
- **Cassandra:** Best for massive scale, continuous high write throughput, known query patterns, high availability, and partition-based deletes.
