SQL and Neo4j Data Management for Social Media Analytics

Naveen Prabakar

August 3, 2025

1 Overview

This report describes the design and management of a social media database system implemented using both SQL (MySQL) and Neo4j (a graph database). The management system is tailored to efficiently process, analyze, and retrieve complex relationships and broadcast interactions found in a Twitter-like environment—including tweets, users, hashtags, mentions, and external URLs.

The SQL portion employs structured tables—ideal for transactional data integrity and bulk loading—while Neo4j is well suited for advanced relationship queries such as influence ranking, network traversal, and hashtag co-occurrence analysis.

2 SQL Database Design and Management

2.1 Schema Overview

The MySQL schema is organized around key social media concepts:

- **Users** (`Users`): Stores user metadata and their network metrics (screen name, name, category, state, followers, following, etc.).
- **Tweets** (`Tweet`): Contains tweet identifiers, content, retweet info, timestamps, and the posting user.
- **Hashtags, URLs, Mentions**:
  - `HashTag`: List of all hashtags.
  - `URL`: List of URLs appearing in tweets.
  - `Mentioned`: Many-to-many relation linking tweets and users mentioned within them.
  - `hasURL`: Mapping of tweets to URLs.
  - `hasTag`: Mapping of tweets to hashtags.

Referential integrity is enforced via foreign key constraints, ensuring data consistency (`Mentioned`, `hasURL`, `hasTag` all use foreign keys to link to the main tables).

2.2 Data Ingestion

Bulk data is loaded efficiently using MySQL's `LOAD DATA LOCAL INFILE` command on CSV files (e.g., `user.csv`, `tweets.csv`, `mentioned.csv`, `tagged.csv`, `urlused.csv`). Duplicate rows in hashtag and URL entity tables are safely ignored to preserve uniqueness. Post-load, integrity is maintained through database constraints.

2.3 Query Management

A series of SQL queries were crafted to support analytic and reporting needs, including:

- Retrieving top retweeted tweets in a given year.
- Counting hashtag spread across geographic states.
- Identifying highly followed users by political category/subcategory.
- Analyzing hashtag co-occurrence and user mention statistics.

These queries involve multiple table joins, aggregations, grouping, and filtering to extract meaningful insights.

3 Neo4j Graph Database Integration

3.1 Motivation

SQL databases excel at structured, transactional data storage but can be cumbersome for relationship-heavy queries involving many degrees of connections (e.g., social networks, influence graphs). Neo4j, a property-graph database, provides efficient graph traversal operations for queries like:

- Identifying influencers connected across different topics.
- Investigating hashtag co-occurrence networks.
- Discovering shortest connection paths between users.

3.2 Data Modeling

Key graph elements include:

- **Nodes:** Users, Tweets, Hashtags, URLs.
- **Relationships:**
  - `POSTED` (User →Tweet)
  - `TAGGED` (Tweet →Hashtag)
  - `MENTIONED` (Tweet →User)
  - `CONTAINS_URL` (Tweet →URL)
  - `FOLLOWS` (User →User)
- **Properties:** Attributes like follower counts, timestamps, category, and state stored on nodes or relationships to enable rich querying.

3.3 Typical Cypher Data Load Example

```
LOAD CSV WITH HEADERS FROM 'file:///users.csv' AS row
CREATE (u:User {screen_name: row.Screen_name, name: row.name, category: row.category,
    ...});
```

Other relationships (posts, tags, mentions) are similarly created to model interactions.

3.4 Sample Cypher Query Patterns

- **User Influence:** Counting tweets per user

```
MATCH (u:User)-[:POSTED]->(t:Tweet)
RETURN u.screen_name, count(t) AS tweet_count
ORDER BY tweet_count DESC LIMIT 10;
```

- **Mention Network:** Finding users mentioning each other

```
MATCH (a:User)-[:POSTED]->(:Tweet)-[:MENTIONED]->(b:User)
RETURN a.screen_name, b.screen_name;
```

- **Hashtag Co-occurrence:** Pairs of hashtags used together

```
MATCH (t:Tweet)-[:TAGGED]->(h1:Hashtag), (t)-[:TAGGED]->(h2:Hashtag)
WHERE h1 <> h2
RETURN h1.hname, h2.hname, count(*) AS usage_count
ORDER BY usage_count DESC;
```

4 Management Practices

- **Data Consistency:** Regular export/import pipelines keep SQL and Neo4j data synchronized.
- **Constraints and Indexes:** Unique constraints and indexes in both databases prevent duplicates and optimize query performance.
- **Hybrid Workflow:** SQL is used for transactional storage and bulk analysis, while Neo4j enhances graph queries and network analytics.

5 Use Cases and Workflows

- Use SQL queries to generate traditional reports and summaries based on tweets, users, and hashtag usage.
- Use Neo4j for advanced graph investigations like influencer detection, social network paths, and hashtag relationship analysis.
- Develop applications or dashboards leveraging both databases according to query complexity and performance.

6 Conclusion

By leveraging MySQL for structured, reliable, and bulk analytical workflows and Neo4j for rich, relationship-focused graph queries, this social media data management solution efficiently supports complex social network analysis and reporting. This dual-database strategy benefits applications requiring both strong data integrity and sophisticated graph traversals.