DSAI Lab - Group 3

Project : GenAI powered SQL Refactoring Assistant

Problem Statement

Business organizations are increasingly migrating data from traditional, on-premise, or lightweight databases (like SQLite/MySQL) to cloud-native, distributed big data warehouses (like Apache Hive/Spark). However, this migration requires refactoring thousands of legacy SQL queries.

These platforms differ significantly not just in syntax, but in underlying semantics (e.g., date handling, string manipulation, and window functions). Manual refactoring is error-prone, time-consuming, and requires dual expertise in both source and target dialects.

We propose a GenAI-powered SQL Refactoring Assistant that automates the translation of SQL queries from a source dialect (SQLite) to a target big-data dialect (Hive), utilising Retrieval-Augmented Generation (RAG) to ensure syntax compliance.

Project Scope & Justification

Selected Migration Pair: SQLite → Apache Hive

Justification (Why this pair?):

1. Local-to-Distributed Paradigm: This pair represents a classic "Scale-up" scenario where code written for local prototyping (SQLite) must be refactored for a production distributed cluster (Hive).
2. High Complexity (Technical Depth):
    a. Typing: SQLite is dynamically typed (loose), while Hive is statically typed (strict).
    b. Functionality: Date/Time functions differ drastically (e.g., SQLite uses strftime, Hive uses from_unixtime or date_format).
    c. Syntax: Quote handling (double vs single ticks) and join optimization hints differ.

This complexity ensures the project meets the "Technical Expertise" criteria (5 marks) defined in the guidelines.

Scalability:

While we focus on SQLite → Hive for the 8-week timeline, the architecture will be designed to be dialect-agnostic. Since our dataset supports 20+ dialects (One-to-Many), the model can theoretically be fine-tuned for other pairs (e.g., MySQL → SparkSQL) in the future.

Input / Output Definition

Input: A valid SQL query written in SQLite dialect.

Output: A functionally equivalent SQL query written in HiveQL that yields the same result set when executed.

Concrete Example:

Input (SQLite):

```
SELECT strftime('%Y-%m', order_date) as month, sum(total)
FROM orders
GROUP BY strftime('%Y-%m', order_date);
```

Refactoring Challenges: Hive does not support strftime. It requires specific date parsing functions.

Output (Hive):

```
SELECT from_unixtime(unix_timestamp(order_date, 'yyyy-MM-dd'),
'yyyy-MM') as month, sum(total)
FROM orders
GROUP BY from_unixtime(unix_timestamp(order_date, 'yyyy-MM-dd'),
'yyyy-MM');
```

Dataset & Reference Paper Usage

Primary Dataset: PARROT (Hugging Face: weizhoudb/PARROT)
Reference Paper: Is It Time to Say Goodbye to Cross-domain Text-to-SQL? (ArXiv: 2509.23338)

How we will use the Reference Paper & Data:

1. Data Sourcing (Milestone 2): The paper provides a standardized parallel corpus of SQL queries across 24 dialects. We will filter this dataset to extract the (SQLite, Hive) pairs.
2. Baselines (Milestone 1 & 4): The paper provides benchmark translation accuracy scores. We will use these as the "Baseline" to compare our model against.
3. Training Strategy (Milestone 4):
    a. We will fine-tune an open-weights LLM (e.g., CodeLlama-7B or Llama-3-8B) on the PARROT subset.
    b. We will implement RAG (Retrieval Augmented Generation) using official Hive Documentation as the Knowledge Base. This allows the model to look up specific Hive syntax rules that might be hallucinated by a standard model.

Evaluation Plan (Milestone 5)

To ensure the "Minimal Friction" and "Correctness" mentioned in our objective, we will use two layers of evaluation:

1. **Execution Accuracy (The Gold Standard):**
   We will set up a local testing environment (using Docker containers for SQLite and Hive). We will run the generated query.
   - Score 1: Does it compile/run without error?
   - Score 2: Does the output data match the ground truth?
2. **Syntax Matching (CodeBLEU):**
   Since strict text matching is unreliable for code, we will use CodeBLEU (or similar semantic similarity metrics) to compare the generated AST (Abstract Syntax Tree) against the reference solution.