SOEN 363 - Phase 1 Assignment

Team Members

- Anthony Chraim
- Ethan Benabou
- Justin Loh
- Mohamed Amine Kihal

Files

- `data/`: scripts for pre-processing the input `.dat` files.
- `sql/`: queries, views, etc. to answer the assignment questions.
- `results-csv/`: csv files displaying the output of all queries.
    - `results-csv/performance.csv`: Raw performance metrics from question 4.
    - `results-csv/q3-*.csv`: Raw output from the queries in each question.
- `docker-compose.yml`: docker-compose file for Postgres:latest.
- `do.sh`: commandline utility for managing the postgres docker instance.
- `schema.sql`: SQL statements for creating the table schema.
- `erd.png`: Entity-Relation diagram, exported from Postgres.
- `report.pdf`, `report.md`: This report.

Introduction

To see the queries along with detailed comments on each statement, check out the `.sql` files in the `sql/` directory

DATABASE: PostgreSQL v13 in Docker

PostgreSQL was used as the database for this phase of the project. Postgres v13 was run inside of Docker and the process was scripted as much as possible to increase repeatability, and to remove the "it works on my machine" scenario. A Python script was created to parse the provided `.dat` files and produce fat `INSERT` statements with a tuple for every row in the `.dat` file. This pre-processing step proved to be much quicker than importing the raw `.dat` files and it also provided the opportunity to replace all instance of the string '\N' with the SQL value `null`. The queries were written. You can find all queries, views, and other SQL statements that were created in the `results-csv` directory included in this submission. Finally, the performance metrics were calculated for question 4, where query performance was measured with indexing vs without indexing, and the performance of materialized views was investigated.

Procedures

The following process was followed for creating the database and developing the the queries:

1. Clean the data
    - Using Python, the `.dat` files were converted to `.sql` files containing `INSERT` statements that could easily be run to import the data. The '\N' strings were replaced with `null`.
2. Spin up a fresh instance of Postgres running in Docker.
3. Create the table schemata.
4. Execute the INSERT statements to populate the database.

5. Run the queries, save the results to `.csv` files using the `psql` commandline tool.

Performance Analysis

Question No.	Type	Execution time
3-K-2	View	25 sec 318 msec
3-L	View	2 min 29 sec
4-K-2	Materialized View	147 msec
4-L	Materialized View	226 msec

Discussion

Based on the observation obtained , it is confirmed that that the execution time of materialized views complete in proportionately less time compared to that of normal views. As materialized views exist physically as entities in disk space, the execution time is much quicker that views which are simply stored in a temporary virtual space after view creation. Hence, it is recommended to use materialized views in the storage of data with a seldom need for updates.

However unlike normal views , materialized views are not automated to be updated with each view use. Materialized views need to be manually updated with via specific commands or the use of triggers, in comparison, normal views are updated after each view use in different query statements, making normal views much more responsive in storing up-to-date information.

Assumptions

- **q3-m**:
    - There are no duplicates, other than those that appear in the movies table, but movie id is a foreign key for many other tables.
    - You might want to remove these redundant movie id records from other tables.
    - For example, actors who act in a movie that is duplicated will receive credit twice for acting in the same movie.
    - These queries are implemented with the intent that we want to remove these extra records from all tables.
    - For example, to ensure that all records in actors table correspond to unique (non-duplicate) movies (actors do not get credited or the duplicate movies they star in).