CSCI 620//Brown, Introduction to Big Data, Spring 2225

Assignment 2 – SQL

Description

Create the following database schema in Postgres with appropriate types and populate it with IMDB data. Primary keys are in bold. Foreign keys are indicated with FK.

- Title (**id**, type, title, originalTitle, startYear, endYear, runtime, avgRating, numVotes)
- Genre (**id**, genre)
  Note: this table should contain individual genres
- Title_Genre (**genre**, **title**)
  - genre FK Genre(id)
  - title FK Title(id)
- Member (**id**, name, birthYear, deathYear)
- Title_Actor (**actor**, **title**)
  - actor FK Member(id)
  - title FK Title(id)
- Title_Writer (**writer**, **title**)
  - writer FK Member(id)
  - title FK Title(id)
- Title_Director (**director**, **title**)
  - director FK Member(id)
  - title FK Title(id)
- Title_Producer (**producer**, **title**)
  - producer FK Member(id)
  - title FK Title(id)
- Character (**id**, character)
  Note: this table should contain individual characters
- Actor_Title_Character (**actor**, **title**, **character**)
  - (actor,title) FK Title_Actor (actor, title)
  - character FK Character(id)

It is recommended that, to load characters, you disable the foreign keys of the database. (You should still ensure that the data is valid by removing any characters which do not satisfy the foreign key constraints).

Your tasks

1. Provide a program to load the IMDB data from the text files into the database. Your program needs to load the whole database (without adult titles) in approximately twelve hours using commodity hardware. **(10 points)**

2. Provide a program to retrieve the following data from the database. Each will consist of a single SQL query and you need to report evidence that you retrieved what was expected (e.g. example rows returned by the query). Report the time your queries took to run. **(9 points per query)**

   2.1. Number of invalid Title_Actor relationships with respect to characters. (That is, entries in Title_Actor which do not appear in Actor_Title_Character.)
   2.2. Alive actors whose name starts with "Phi" and did not participate in any title in 2014.
   2.3. Producers who have produced the most talk shows in 2017 and whose name contains "Gill". (Hint: talk show is a genre)
   2.4. Alive producers ordered by the greatest number of long-run titles produced (runtime greater than 120 minutes).
   2.5. Alive actors who have portrayed Jesus Christ (simply look for a character with this specific name).

3. Visualize and provide a brief explanation of the execution plan for each of the previous queries. (It's okay if you don't understand all the details.)

   (Hint: http://tatiyants.com/pev/ will allow you to copy and paste EXPLAIN output from Postgres to visualize query plans. You can also explore interactively to receive more information on the meaning of each step in the plan.) **(10 points)**

(continued on next page)

4. Provide relational algebra expressions for the queries below.
   (3 points per query)

   4.1. Number of invalid Title_Actor relationships with respect to characters. (That is, entries in Title_Actor which do not appear in Actor_Title_Character.)
   4.2. Alive actors whose name starts with “Phi” and did not participate in any title in 2014.
   4.3. Producers who have produced a talk show in 2017 and whose name contains “Gill”. (Hint: talk show is a genre)
   4.4. The name of long-run titles (runtime greater than 120 minutes) produced by producers who are still alive.
   4.5. Alive actors who have portrayed Jesus Christ (look for both words independently).

5. Create indexes and full-text indexes where they are required. Document your decisions and provide the scripts to generate them. Re-run all the previous queries and report performance improvement and provide a brief explanation about why such an improvement including references to execution plans. (Hint: To reason about the performance gain you may need to restart the database server in order to clear query caches.) **(20 points)**