Graph Data Management and Analysis Project
From RDBMS to Graph

Summer Semester 2022

Submission deadline: 17.07.2022

In this project you are given the data of FreeDB (`cddb`) that contains information about music records. The data is stored in a relational database, i.e., PostgreSQL. You will find the sql dump file to load the data into your local PostgresSQl at the course website. After importing the data into Postgres you need to accomplish the following tasks.

Task 1: Data Import

Write code or describe the operations in order to load the data into Neo4j. Note that it is not obligatory to write SQL Code. It is totally acceptable if you decide to export the database to CSV files and then use the CSV files to import the data into Neo4j. However, you need to provide a sufficient amount of details on how you modelled the data in Neo4j and elaborate on all the implementation and design decisions you took. Note that for this task, you are not allowed to use any external tools, e.g. the ETL tool of Neo4j.

Task 2: Cypher queries

Write the Cypher code required for the following queries:

1. Given an artist name, e.g. `'radiohead'`, list all the albums of that artist.
2. Given a song name, e.g. `'family reunion'`, list the artist and album name of CDs released in 1999 which contain that song.
3. Given a track number, e.g. `12`, show (non-duplicated) artist, song, and album names of songs that are located on that track on a cd.
4. Augment a list of all artists with self-titled albums (i.e., an album with the same name as the artist who releases it). Artists with no self-titled albums have to be included with `null` as album.
5. Find how many albums (not CDs) have been issued each year in the 20th century (years 19XY)?
6. Find all artists that have released at least 5 albums that are associated with the genre 'rock'.

Task 3: SQL to Cypher

Translate the following SQL queries to Cypher:

1.
   ```
   SELECT DISTINCT artist
   FROM artists
   NATURAL JOIN artist2album
   NATURAL JOIN cds
   WHERE ayear = 2000
   ```

2.
   ```
   SELECT MAX(ayear), AVG(ayear)
   FROM cds
   WHERE ayear BETWEEN 1900 AND 2012
   ```

3.
   ```
   SELECT genre , COUNT(cdid) AS cds
   FROM  cds NATURAL JOIN genres g
   GROUP BY g.genreid , genre
   HAVING COUNT(cdid) >= 1000
   ORDER BY cds DESC
   ```

4.
   ```
   SELECT artist, album
   FROM cddb.artists
   NATURAL JOIN cddb.artist2album
   NATURAL JOIN cddb.albums
   NATURAL JOIN cddb.cds
   NATURAL JOIN cddb.cdtracks
   NATURAL JOIN cddb.songs
   WHERE song LIKE '%moonlight shadow%'
   ```

The Cypher queries should return the same type of result (same entries). Also, use the original solutions to make sure that the number of results returned by Postgres is the same as the number of results returned by Neo4j. In case there is a difference in the number of results of Postgres and Neo4j, slight discrepancies are acceptable, i.e., if Postgres returns 110 results and Neo4j returns 109, as long as the Cypher query is correct. Such discrepancies may happen due to the data import and the migration. Ideally, in such cases you should describe why there is a difference.

Task 4: Searching and Ranking

Implement a simple search engine that enables search by artist, album and song name/title. The results must be ranked based on importance. It is up to you to come up with how the importance of each result is computed and you must justify your decision (it goes without saying that you need to come up with a meaningful definition). However, the importance should ideally take into account user preferences/likes. As such, this task is split in two parts:

1. Write a Cypher query that adds a relationship `:LIKES` between a node with label `:User` and an artist, album, or song. Every user should be identified

just by a numerical userID (no more information is necessary). If a user already exists in the system, no additional node should be added. After coming up with the necessary Cypher query, add a significant number of users and likes.

2. Implement a simple Python function that has the following arguments:
    - the userID of the user submitting the search (the user ID may not exist in the database),
    - a string that contains one or more keywords for the search, and
    - an optional argument that indicates whether the search is on all or a specific field, i.e., artist, album, song.

   The search must return exactly 10 results.

Python must only be used to call the database. You should not write any code in Python that implements functionality necessary for the task. However, submitting multiple queries in the same function call is allowed. Also, for this task of the project, you are not only allowed but also encouraged to use functions from the GDS library of Neo4j. Hence, before making any decisions, have a careful look at the available functions. Again, you have to justify the use of any function that you employ.

Submission Instructions

The submission must consist of two files:

- The first file must contain all the code for any Cypher queries you wrote. As such you can even submit the code in a text file. Optionally yet ideally, you should submit a python file (either a simple .py file or a Jupyter Notebook) with separate functions for each query/operation.

- The second file should be a project report. For each of the tasks, especially Task 1 and Task 3 you should provide adequate description about your design and implementation choices. The documentation is less important for Task 2 as it involves writing queries. However, if there are any discrepancies in the result, or you believe that a query cannot be expressed using a single Cypher query, or anything else worth mentioning, it should be included in the report.

Submit both files as a single `.zip` file named `SURNAME_NAME_PROJ_A.zip`. No group submissions are allowed. Each student must work on and submit their own project.

Note: To avoid any confusion, in this project you must work alone. While some exchange of opinions is allowed, there should be one submission per student.