Univ.-Prof. Dr.-Ing. Matthias Boehm
Graz University of Technology
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management

2. Data Management WS2020/21: Exercise 02 – Queries and APIs

Published: November 09, 2020 (last update Nov 08, changed Q06-Q08)
Deadline: December 01, 2020, 11.59pm

This exercise on query languages and APIs aims to provide practical experience with the open-source database management system (DBMS) PostgreSQL, the Structured Query Language (SQL), and call-level APIs such as ODBC and JDBC (or their Python equivalents). The expected result is a zip archive named **DBExercise02_ <studentID>.zip**, submitted in TeachCenter.

2.1. Database and Schema Creation via SQL (3/25 points)

As a preparation step, setup the DBMS PostgreSQL (free, pre-built packages are available for Windows, Linux, Solaris, BSD, macOS). The task is to create a new database named `db<student_ID>` and setup the normalized relational schema from Task 1.2. In contrast to Exercise 1, there are some small modifications though: please ignore the derived budget classification of movies, and the location of actors (countries they live in). The schema should also include all primary keys, foreign keys, as well as `NOT NULL` and `UNIQUE` constraints. Your SQL script should be robust in case of partially existing tables and drop them before attempting to create the schema. If you do not want to use your own schema from Task 1.2, we will provide a recommended `CreateSchema.sql` by November 11 (seven days past Exercise 1).

Partial Results: SQL script `CreateSchema.sql`.

2.2. Data Ingestion via ODBC/JDBC and SQL (10/25 points)

Write a program `IngestData` in a programming language of your choosing (but we recommend Python, Java, C#, or C++) that loads the data from the provided data files [1], and ingests them into the schema created in Task 2.1. Please, further provide a script `runIngestData.sh` that sets up prerequisites, compiles and runs your program, and can be invoked as follows[2]:

```
./runIngestData.sh ./Movies.csv ./Persons.csv ./Ratings.csv \
  <host> <port> <database> <user> <password>
```

Note that the partially denormalized input files require deduplication of countries, languages, and genres. It is up to you if you handle this requirement via (1) program-local data structures (e.g., lookup tables like `Genre-GenreID`) and call-level interfaces like ODBC/JDBC, or (2) ingestion into temporary tables and transformations in SQL.

Partial Results: Source code `IngestData.*` and script `runIngestData.sh`.

[1] `https://github.com/tugraz-isds/datasets/tree/master/movies`
[2] The concrete paths are irrelevant. In this example, the `./` just refers to a relative path from the current working directory and the backslash is a Linux line continuation.

2.3. SQL Query Processing (10/25 points)

Having populated the created database in Task 2.2, it is now ready for query processing. Create SQL queries to answer the following questions and tasks (Q01-O06: 1 point, Q07/Q08: 2 points). The expected results per query will be provided on the course website.

- **Q01:** Obtain the detailed information of the movie(s) with English title `Interstellar`. (return English title, release date, runtime, budget, and revenue)
- **Q02:** Which movies were produced in `Austria`? (return original title, release date; sorted ascending by original title)
- **Q03:** How many movies were filmed per year? (return year, count; sorted ascending by year)
- **Q04:** Which movie had the most unique actors? (return English title, actor count)
- **Q05:** Compute the top-20 highest-rated movies by average rating. (return English title, release date, average rating; sorted descending by the average rating, and in case of equal ratings, further sorted ascending by English title)
- **Q06:** How many movies of genre `Science Fiction` have been released each year between 2012 and 2016 (both inclusive)? (return year, count; sorted ascending by year)
- **Q07:** Who are the top-10 most successful actors by number of movies they played in, and profits these movies generated? (return actor name, count, profit as union distinct of top-10 actors by maximum count and top-10 actors by maximum profit).
- **Q08:** Which pairs of actors co-appeared most-frequently in movies each year between 2010 and 2013 (both inclusive)? (return name of actor1, name of actor2, year, count, sorted ascending by year)

Partial Results: SQL script for each query `Q01.sql`, `Q02.sql`, ..., `Q08.sql`.

2.4. Query Plans and Relational Algebra (2/25 points)

Obtain a detailed explanation of the physical execution plan of **Q06** using `EXPLAIN`. Then annotate how the operators of this plan correspond to operations of extended relational algebra.

Partial Results: SQL script `ExplainQ06.sql` with output and annotations in comments.

A. Recommended Schema and Examples

As an alternative to your own relational schema from Exercise 1, we will provide a recommended schema (by Nov 11) as a fresh start for this exercise. Even when using the provided schema, please include it in the submission. Furthermore, we also provide an additional example Python script that demonstrates how to access PostgreSQL through a call-level interface from an application program. This script assumes that Python 3 and pip is already installed. Note that both the schema and Python scripts are made available on the course website.