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

2. Data Management WS2019/20: Exercise 02 – Query Languages and APIs

Published: November 05, 2019 (last update: Nov 05)
Deadline: November 26, 2019, 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), as well as APIs such as ODBC and JDBC. The expected result is a zip archive named `DB_Exercise02_<student_ID>.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 1.3, with the following small modifications. The frequent flyer programs are ignored in this exercise, and the ICAO code of planes has between 2 and 4 letters (not exactly 4-letters as described in Task 1.1). The schema should also include all primary keys, foreign keys, as well as `NOT NULL` and `UNIQUE` constraints. Note that the SQL script should be robust in case of partially existing tables, in which case these tables should be dropped before attempting to create the schema. If you do not want to use your own schema from Task 1.3, Appendix A provides an alternative.

Partial Results: SQL script `CreateSchema.sql`.

2.2. Data Ingestion via ODBC/JDBC and SQL (8/25 points)

Write a program `IngestData` in a programming language of your choosing (but we recommend using languages such as Python, Java, C#, or C++) that loads the data, from the provided data files [1], into the schema created in Task 2.1. The program should be called as follows:

```
IngestData ./Airlines.csv ./Airports.csv ./Planes.csv ./Routes.csv \
  <host> <port> <database> <user> <password>
```

Note that the partially denormalized inputs require deduplication (e.g., of countries, and cities). It is up to you if you handle this requirement via (1) program-local data structures (e.g., lookup tables for country-ID, city/country-ID mappings) and ODBC/JDBC, or (2) ingestion into temporary tables and transformations in SQL.

Partial Results: Source code `IngestData.*`, and a script to compile and run the program.

[1] `https://github.com/tugraz-isds/datasets/tree/master/airports_airlines`

2.3. SQL Query Processing (9/25 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:** Which airports are at negative altitude? (return airport name, altitude)
- **Q02:** Which airports are both departure and arrival of a route? (return name)
- **Q03:** Which airlines have routes starting in Graz, sort by name ascending? (return each airline name once)
- **Q04:** Which cities have more than 2 Airports, sort by city name ascending? (return city names)
- **Q05:** Compute the number of airports per EU member states, sorted first by number of airports descending, second by country ascending. (return country name, number of airports)
- **Q06:** Which countries do not have any airports? (return country name)
- **Q07:** What are the Top-7 Airlines by number of operated routes in descending order? (return airline name, count)
- **Q08:** Which plane types are used on routes operated by 'Lufthansa' but not by 'Star Aviation'? (return plane type name)

Partial Results: SQL script `Queries.sql`, with comments indicating the query numbers and the obtained results.

2.4. Query Plans and Relational Algebra (5/25 points)

Finally, pick two of the queries Q01 through Q08, and obtain an explanation of the physical execution plan (after running `ANALYZE` on the relevant tables). This should include a SQL query to return the plan, as well as a graphical representation of this plan. Furthermore, explain how the operators of the returned plan correspond to operations of extended relational algebra.

Partial Results: SQL script `ExplainQueries.sql` (with comments describing the relationship to relational algebra), and two images of the visual plan explanation.

2.5. Extra Credit (5 points)

After successful data ingestion in Task 2.2 you get another csv-file with additional but partially overlapping routes data. This two-phase ingestion aims to emulate real-world scenarios like periodic ingestion into a central data warehouse. Make the necessary extensions to your `IngestData.*` program from Task 2.2 to handle this partial overlap. The program should be called with the following parameters:

```
IngestData ./Bonus_Routes.csv <host> <port> <database> <user> <password>
```

Partial Results: Source code `IngestData.*`, and a script to compile and run the program (please, submit a single program that can handle both 2.2 and 2.5).

A. Relational Schema as Input for Task 2.1

As an alternative to your own schema from Exercise 1, the following relational schema can be used as a basis for this exercise. Also a python script is available to demonstrate how PostgreSQL is used within a programming language. The script assumes that Python 3 and pip is already installed. Note that the use of this material is optional.

`The schema will be inserted here when the 7 late days of the first exercise are over.`