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 WS21/22: Exercise 02 – Queries and APIs

Published: November 06, 2021
Deadline: November 30, 2021, 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) or use the provided Docker container. The task is to create a new database named `db<student_ID>` and setup the provided schema[1]. You may partially customize this schema but it should be in third-normal form; include all primary keys, foreign keys, as well as `NOT NULL` and `UNIQUE` constraints; and be robust in case of partially existing tables and drop them before attempting to create the schema.

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 [2], 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[3]:

```
./runIngestData.sh ./Locations.csv ./Parties.csv ./Votes.csv ./Elections.csv \
  <host> <port> <database> <user> <password>
```

It is up to you if you perform necessary transformations of the denormalized input files via (1) program-local data structures (e.g., lookup tables like `PartyShort-PKey`), or (2) ingestion into temporary tables and transformations in SQL. However, all inserts should be performed via call-level interfaces like ODBC, JDBC, or Python's DB-API.

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

[1] `https://mboehm7.github.io/teaching/ws2122_dbs/CreateSchema.sql` (available by November 09)
[2] `https://github.com/tugraz-isds/datasets/tree/master/elections_at`
[3] 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. For any queries requiring you to return a real number, you should round the number to two decimal places.

- **Q01:** What is the ID of location `Graz(Stadt)`? (return LocationID)
- **Q02:** Select all parties of the election `NR2017`. (return ShortName, LongName, Ballot Position, sorted ascending by Ballot Position with NULLs last)
- **Q03:** Compute the voter turnout rate (total-votes/eligible) for all districts of `Graz(Stadt)` in election `NR2019`. (return location name, turnout rate, sorted descending by turnout)
- **Q04:** Compute the top 10 locations of election `NR2019` by voter turnout rate. (return name, turnout, sorted descending by turnout)
- **Q05:** Which parties from the election `NR2019` did not participate in `NR2017`? (return ShortName, LongName, sorted ascending by ShortName)
- **Q06:** Compute the support (fraction of received votes) in `NR2019` of all parties that received more than 4% of votes. (return ShortName, support; sorted descending by support)
- **Q07:** Find the parties that won (with highest support rate) at least one location in `NR2019`. (return ShortName, count of won locations, total Austrian support rate; sorted descending by won locations)
- **Q08:** Compare for each state of `Österreich` (e.g., `Steiermark`) the total number of votes with the sum of votes in all last-level child locations. (return the state name, total votes, sum of votes in child locations, difference in votes, sorted ascending by state name)

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

Please include—even if unmodified—the schema (see Task 2.2) into your 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 are already installed. Note that the schema, Docker container, Python scripts, and expected results are made available on the course website.