Heinrich Heine
University Düsseldorf

Institut für Informatik
Datenbanken und Informationssysteme
Prof. Dr. Stefan Conrad
Thomas Germer

WiSe 24/25
Assignment 5
Due November 19, 8:00 AM

Relational Databases and Data Analysis

- As usual, use `zipme.py` to create the archive to upload in ILIAS.

Exercise 1 *Joins* (2 + 2 + 1 + 1 + 1 Points)

In `exercise_1_example.py`, you can find an inefficient Python-Implementation of a naive natural join implemented with nested loops.

a	b
1	3
2	4

$\bowtie$

b	c
3	5
3	5
4	7

$=$

a	b	c
1	3	5
1	3	5
2	4	7

The relations are represented as dictionaries where the keys are the attribute names and the values are the corresponding columns. We will allow duplicate rows here. The result of the natural join is:

```
result = {
    "a": [1, 1, 2],
    "b": [3, 3, 4],
    "c": [5, 5, 7],
}
```

Fortunately, there are much faster ways to implement joins, such as the hash join and the merge join.

(a) Implement a natural hash join in `exercise_1_a.py` in Python without using additional libraries. Consider that a natural join joins over all attributes with the same name, not just one. Also keep in mind that a natural join of two relations without common attributes degrades to a cross product.

(b) Implement a natural merge join in `exercise_1_b.py` without using additional libraries.

(c) Express the time complexity of the hash join algorithm in terms of the number of rows n and m of the two input relations and the number of rows k and columns c of the result. For example, the naive join has complexity of $O((n \cdot m + k) \cdot c)$, which can be simplified to $O(n \cdot m \cdot c)$ since $k \leq n \cdot m$. Explain your answer in `exercise_1_c.txt`.

(d) In `exercise_1_d.txt`, express the time complexity of the merge join algorithm in terms n, m, k and c and explain your answer.

(e) The time complexity of the merge join algorithm is worse than the hash join algorithm. Give two reasons when one would prefer to use a merge join anyway in `exercise_1_e.txt`.

Exercise 2 *Simple Web Server* (3 Points)

In `exercise_2_a.py`, you will find a simple Flask-based web server. After running the file, you can visit `http://127.0.0.1:5000/python` and press the login button. You will then be greeted with the first name and last name corresponding to the login user name.

(a) The user data is currently stored as a Python dictionary. Modify `exercise_2_a.py` to implement the same functionality using a PostgreSQL database under the endpoint `/sql`.

- You need to set the environment variables `PGHOST`, `PGUSER` and `PGPASSWORD` and then start a PostgreSQL server. Ensure that the host, user and password which your PostgreSQL server uses are read from the environment variables and not hardcoded as fixed values.
- The test script will start the server automatically and will randomly modify `lastname` and `firstname` of the `users` table to ensure that you are actually using data from the database. Name your table and its attributes accordingly to make it work.
- The server will be started automatically on port 5000. It will not work if another program is already using that port. You have to stop those programs first.
- The server will print the following warning. We will ignore it for now (which is generally frowned upon) and deal with it in a later exercise.

  ```
  WARNING: This is a development server. Do not use it in a
  production deployment. Use a production WSGI server instead.
  ```

  Flask's development server should not be used in a production setting because it has not been designed to be particularly secure and it is quite slow.