COMP 3005: Database Management Systems **(Due: Apr. 10, 2024 (11:59 PM))**

COMP3005 Project (Winter 2024)

Instructor: Ahmed El-Roby and Abdelghny Orogat

1. Problem Statement

Design a database that stores a soccer events dataset spanning multiple competitions and seasons. The provided dataset is in JSON format and can be downloaded from `https://github.com/statsbomb/open-data/tree/0067cae166a56aa80b2ef18f61e16158d6a7359a`[1]. The documentation of the dataset is also available in the above URL. After designing the database, you need to import the data from the JSON files into your database. You will be required to use PostgreSQL[2] to store and query your database. You will be required to submit python scripts (one script per query) that does the following:

1. Connect to the database.
2. Execute the following queries (one script per query):
   (a) Q_1: In the La Liga season of 2020/2021, sort the players from highest to lowest based on their average xG scores.
   (b) Q_2: In the La Liga season of 2020/2021, find the players with the most shots. Sort them from highest to lowest.
   (c) Q_3: In the La Liga seasons of 2020/2021, 2019/2020, and 2018/2019 combined, find the players with the most first-time shots. Sort them from highest to lowest.
   (d) Q_4: In the La Liga season of 2020/2021, find the teams with the most passes made. Sort them from highest to lowest.
   (e) Q_5: In the Premier League season of 2003/2004, find the players who were the most intended recipients of passes. Sort them from highest to lowest.
   (f) Q_6: In the Premier League season of 2003/2004, find the teams with the most shots made. Sort them from highest to lowest.
   (g) Q_7: In the La Liga season of 2020/2021, find the players who made the most through balls. Sort them from highest to lowest.
   (h) Q_8: In the La Liga season of 2020/2021, find the teams that made the most through balls. Sort them from highest to lowest.
   (i) Q_9: In the La Liga seasons of 2020/2021, 2019/2020, and 2018/2019 combined, find the players that were the most successful in completed dribbles. Sort them from highest to lowest.
   (j) Q_10: In the La Liga season of 2020/2021, find the players that were least dribbled past. Sort them from lowest to highest.
3. Write the output of the previous queries into CSV files (one for each query). The files must be named Q_x.csv, where x is replaced by the query number (e.g., 1, 2, ...). The first line of the file is the header (the attributes), and the data tuples starts on the second line.

You are required to use Psycopg3[3] to connect to the database as this is the library that will be installed on the machine used to run your code. More details on this can be found in Appendix A.

Your output files will be compared to the gold standard output to verify the correctness of your answers. The grading will be based on the correctness and the efficiency[4] of your program. The grading rubric is discussed in Section 4.

[1] Please use this exact URL to download the dataset as it will be the same dataset used to verify the correctness of your answers to the given queries.

[2] `https://www.postgresql.org/`

[3] `https://www.psycopg.org/psycopg3/docs/`

[4] The efficiency will be directly impacted by the design of your database since all submissions will be using the same programming language and the code is executed on the same hardware.

2. Deliverables

You will be required to submit the following:

1. A project report as one pdf file (on Brightspace).
2. Github repository URL[5] that includes:
   (a) Your exported database from PostgreSQL named "dbexport.pgsql". **You will need to import only the data from the seasons mentioned above such that your exported database is not excessively large.** The size limit on the Github repository is 2 GB. However, you are required to import all event types encountered in the dataset.
   (b) Your source code file(s) that maps and loads the existing JSON dataset from the JSON files into your database. If there are more than one file, store them in a separate directory "Loader".
   (c) Python scripts for executing the queries. Each file should be named after the query. For example, "Q_1.py".

The Github repository will be submitted using this Google form: `https://forms.gle/VsWDwopfbjUX3L4YA`. You will be required to sign in to Google to verify your identity. If for any reason, you do not have a Gmail account, please send to me directly using ahmed.elroby@carleton.ca with the following information: Your name, id, Carleton email address, and the Github repository URL. If you implement the bonus queries (Section 5), also send the URL of the video submission in the email message. Use "COMP 3005 - Project Submission" as the email subject. The form will be closed the next morning of the deadline. You can only make **one submission**. So, please fill out the form after submitting the report and make sure that the information in the form is correct before clicking on "Submit".

You are allowed to work on this project in teams of 3 students or less. But your team should make only one submission on behalf of the team members. But you must indicate the names, IDs, and emails of the team members in the project report. Teams made up of two students will get 10% bonus of their base grade. Teams made up of one student will get 20% bonus of their base grade.

3. Project Report

You need to submit one report file that contains the following sections. You can add other sections, but the following sections **must be** in the report:

3.1. Conceptual Design

This section should explain the conceptual design of the database. That is, the ER-diagram of the database and the explanation of all the assumptions made in the diagram regarding cardinalities and participation types. Make sure that the assumptions do not contradict with the problem statement in Section 1. Note that although the queries above target a limited number of event types, your design (and database instance) should reflect (store) as many events type as possible.

3.2. Reduction to Relation Schemas

Reduce your ER-diagram into relation schemas and list these in this section.

3.3. Database Schema Diagram

This section should show the final schema diagram of the database. This diagram should be similar to the schema diagram of the university database that we study in this course (Figure 1).

4. Grading Rubric

This project will be graded based on a total of 130 points. This includes a 9% bonus towards your final grade. The breakdown of the points is as follows:

[5] Important: Make sure your repository is public at the time of submission.

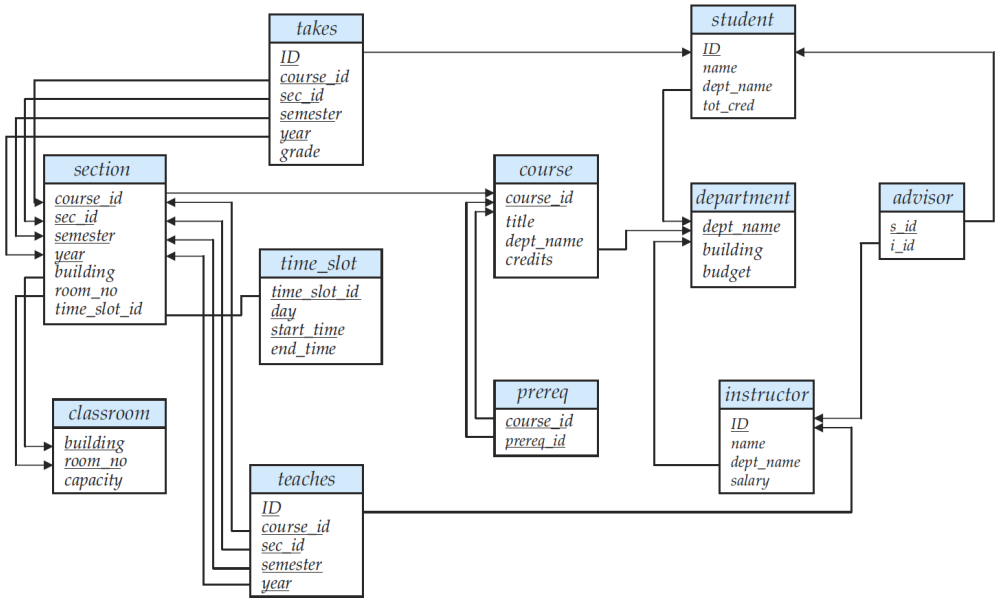

Figure 1: Database Schema Diagram

- Project Report: 30 points.
  - Conceptual Design: 10 points.
  - Reduction to Relational Schemas: 10 points.
  - Schema Diagram: 10 points.
- Query Correctness: 50 points. Each query is worth 5 points.
- Query Efficiency: 50 points.
  - The baseline for efficiency is determined by the best-performing submission successfully processing 100% of the queries. The execution time of each successfully processed query for a submission will be compared to the execution time of the baseline, and the grade for this question will be assigned based on the following formula: $grade_{Q_1} = \frac{executionTime_{baseline}}{executionTime_{submission}} \times 100$. The average grade for the successfully executed queries will be the final efficiency grade.
  - Example: One submission successfully process three queries Q_3, Q_5, and Q_{10} with the execution times t_3, t_5, and t_{10}, respectively. The corresponding execution times for the same queries from the baseline submission are t'_3, t'_5, and t'_{10}. The final grade of efficiency of this submission is $grade_{efficiency} = \frac{\left(\frac{t'_3}{t_3}+\frac{t'_5}{t_5}+\frac{t'_{10}}{t_{10}}\right)\times 100}{3}$. If the ratio of execution time $\frac{t'_i}{t_j}$ is greater than 1, it will be set to 1.
  - Note that in order to receive a grade for the efficiency component, you need to successfully process at least 3 queries. Otherwise, you will get no points for efficiency.

5. Bonus Queries

The following two queries are more complicated than the aggregate queries required above. Successfully demonstrating any of them is worth 5 extra points. For a possible total of 10 points. The demonstration of these queries will be graded based on a video submission that you can submit using the same Google form you use to submit your Github repository.

1. Divide the goal into 6 equal-size areas (top-left, top-middle, top-right, bottom-left, bottom-middle, and bottom-right). In the La Liga seasons of 2020/2021, 2019/2020, and 2018/2019 combined, find the players who shot the most in either the top-left or top-right corners. Sort them from highest to lowest.
2. In the La Liga season of 2020/2021, find the teams with the most successful passes into the box. Sort them from the highest to lowest.

6. Instructions for Submission

Please follow these instructions in your submission:

- Submit your project report as one pdf file.
- Make sure that your GitHub repository is public.
- Adhering to the naming convention for the source code files and the output files is of utmost importance as the project is fully-automatically graded and may result in the submission not being graded correctly. The code of the auto-grader will be shared with you to validate your submissions.
- The due date (April 10th) is a **hard deadline**. No submissions will be accepted after the deadline. It is your responsibility to guarantee there is a version of your report uploaded before the due date. If there is no submission after the due date, you will receive no marks for the project.

A. Psycopg

A.1. Introduction

For connecting to a PostgreSQL database from Python, Psycopg 3 is a newer and advanced choice. It offers modern features like native coroutine support and improved type handling, making it suitable for contemporary Python applications.

A.2. Installation

Install Psycopg 3 using pip:

```
pip install psycopg
```

Note: Psycopg 3 is designed to be compatible with modern Python environments and should be used for new projects or when upgrading existing ones.

A.3. Establishing a Connection

Import Psycopg and use the connection parameters as follows:

```
import psycopg

try:
    conn = psycopg.connect(
    "dbname=your_dbname user=your_username "
    "password=your_password host=your_host port=your_port"
                )
except psycopg.OperationalError as e:
    print(f"Error: {e}")
    exit(1)
```

A.4. Using a Cursor

Create a cursor from the connection to execute SQL commands:

```
with conn.cursor() as cursor:
```

A.5. Executing SQL Commands

Execute SQL commands using the cursor:

```
cursor.execute("SELECT * FROM your_table")
```

A.6. Fetching Data

Retrieve data with the cursor's fetch methods:

```
rows = cursor.fetchall()
for row in rows:
    print(row)
```

A.7. Committing Transactions

Psycopg 3 handles transactions more transparently, and the explicit commit is often unnecessary.

A.8. Closing the Connection

Connections and cursors are closed automatically when used in a 'with' block. Manually closing is optional:

```
cursor.close()
conn.close()
```

A.9. Error Handling and Security

Proper error handling remains crucial. Psycopg 3 introduces more specific exceptions for better error management.

A.10. Asynchronous Support

Psycopg 3 supports asynchronous operations:

```
import asyncio
import psycopg

async def async_query():
    async with psycopg.connect(...) as conn:
        async with conn.cursor() as cursor:
            await cursor.execute("SELECT * FROM your_table")
            rows = await cursor.fetchall()
            for row in rows:
                print(row)

asyncio.run(async_query())
```