CPSC437 S24 - LAB 1

DUE: February 9, 2024, 11:59PM Eastern

SUBMISSION NOTES: Your submission will include the code and requested output files to Gradescope

1 Introduction

In this assignment, you will work with a simulated university database, comprising various entities like classrooms, departments, courses, instructors, sections, students, and more. Each of these entities is represented in the form of tables with specific attributes and constraints, forming a comprehensive relational database model. For details on the tables, see `sql-files/DDL.sql`.

Your task is to perform a series of data manipulations, queries, and inserts using Python with Embedded SQL, providing a hands-on experience in handling real-world data scenarios. The assignment will progress through various levels of complexity, starting from basic data retrieval to more advanced data manipulation and analysis.

By the end of the assignment, you should be able to:

1. Understand how host languages make connections to Postgresql databases.
2. Use Python to embed SQL queries for data retrieval and manipulation.
3. Analyze and interpret complex data relationships within a relational database.
4. Develop an appreciation for data integrity and the importance of database constraints.
5. Gain practical skills that bridge the gap between theoretical database concepts and real-world applications.

1.1 Python Setup

This assignment assumes you have completed Lab 0. With conda installed, we can now create virtual environments to employ separation of concerns with other local Python environments. The following step is highly recommended, but not required.

1. **(Optional but recommended) Environment set up**: Create a virtual environment with

   ```
   conda create -n 437
   ```

   To switch to this virtual environment, execute

```
conda activate 437
```

All package installations will now happen within the `'437'` environment. For more information on environment management with Conda, please visit the documentation.

2. **Package installation:** All the necessary packages for the lab are in the `requirements.txt` To install them, execute

```
conda install --file requirements.txt
```

IMPORTANT: You may only use imports from packages in the requirements.txt or the base library.

1.2 Embedded SQL

In the realm of application development, the utility of SQL is unparalleled for querying and manipulating data. However, SQL has its limitations, which necessitates the integration of SQL within a general-purpose programming language. This integration is achieved through Embedded SQL. The need for such an integration arises primarily due to:

1. **Limitations in Expressiveness of SQL:** While SQL excels in data manipulation and retrieval, there are complexities and nuances in data operations that SQL alone cannot handle efficiently. For example, SQL might fall short in scenarios requiring intricate conditional logic, iterative processing over data sets, or operations that are inherently procedural. These limitations prompt the need for a more versatile programming environment.

2. **Incapability for Non-Declarative Actions:** SQL is a declarative language, designed to specify what needs to be done, rather than how to do it. This nature makes it unsuitable for tasks that require procedural logic, such as complex calculations, data processing, or tasks like plotting and graphical representations of data. Such tasks are essential in many applications, like data analysis, reporting, or interactive data visualization.

Embedded SQL provides a means by which a program can interact with a database server. The SQL statements are translated at compile time (or runtime for interpreted languages) into function calls. At runtime, these function calls connect to the database using an API that provides dynamic SQL facilities.

We will be using the `psycopg2` PostgreSQL database adapter for Python to interact with the `university-db`. A `DatabaseConnection` class, provided in `database-connection.py`, serves as a context manager for database connections. It simplifies interactions with the database by automatically establishing a connection upon entering a context and responsibly closing it upon exit, even in the event of errors. This class abstracts the intricacies of database connectivity, allowing for concise and clean execution of SQL commands using a cursor object that is created and managed internally. You must set the `DB_NAME, USER, PASSWORD, HOST,` and `PORT` fields at the top of the file.

Task 1: Department Statistics

In `task 1`, we will append to the university database a new relation `salary_statistics` with the following specification:

```
dept_name VARCHAR(20),
median_salary NUMERIC(8, 2),
average_salary NUMERIC(8, 2),
std_dev_salary NUMERIC(8, 2),
PRIMARY KEY (dept_name),
FOREIGN KEY (dept_name) REFERENCES department (dept_name)
    ON DELETE SET NULL
```

You are to implement the `create_salary_statistics` function in `task1.py` which creates the table if it does not already exist. Following this, you are to query the `dept_name` and `salary` from the `instructor` table and:

1. Generate a plot using `matplotlib` that:

   - plots the median salary vs. department as a bar chart.
   - plots the average salary vs. department as a line chart.
   - plots the standard deviation as a vertical line from the mean. For departments with only one observation, set the standard deviation to 0.

   It will look similar to the following:

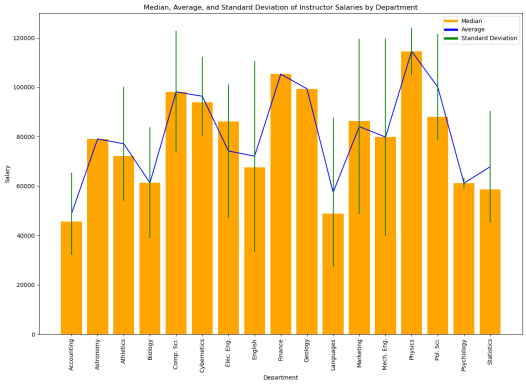

Figure 1: Plot Task 1

   Save this plot as `task1.png`.

2. Populate the `salary_statistics` table with the statistics for the department. It is important to ensure that this script could be used to update the `salary_statistics` table when future data is added, and thus the data should override old data for a `dept_name` should it exist.

3. Write the results to a csv file – per the module docstring – called `task1.csv`.

The starter code can be found in `task1.py`. You are to submit this file as part of your submission.

Task 2: Overlapping Sections

In task 2, you will find all pairs of sections that overlap. You will append to the university database a new relation `overlapping_sections` with the specification of your choice.[1] For task 2 you must:

1. Implement the function `create_overlapping_sections_table_if_not_exists`, which creates the `overlapping_sections` table if it does not exist.

2. Implement the utility function `get_overlap` that takes in two `TimeSlotInfo`:

```
TimeSlotInfo = TypedDict('TimeSlotInfo',
                         {'day': str,
                          'semester': str,
                          'year': int,
                          'start_hr': int,
                          'start_min': int,
                          'end_hr': int,
                          'end_min': int})
```

   and returns `None` if the time slots do not overlap or a tuple of the start and end time of the overlap if they do. You do not have to use this function in subsequent parts if you would prefer to handle them a different way.

3. Two sections are said to overlap if they occur on the same day, in the same semester, in the same year and the runtime of section 1 has overlap with the runtime of section 2. E.g., the following two sections overlap on Monday from 10:00 to 10:15:[2]

```
CPSC-437-001, Monday, 2017, fall, 09:00-10:15
CPSC-237-002, Monday, 2017, fall, 10:00-10:45
```

   Compute all overlapping sections, and write them to the `overlapping_sections` table. You must also write the results to a `task2.csv` file as described in the module docstring.

The boilerplate code and module docstring is provided in `task2.py`.

[1] It is recommended you use the same types in the `DDL.sql` file for the fields you choose to store.

[2] This example is used only for illustration and is not necessarily the shape of the data you will be working with.

Submission

Collaboration Policy

Assignments in this course are designed to assess your individual understanding and skills. While you are encouraged to engage in discussions with the instructor, teaching fellows, undergraduate learning assistants, or classmates to deepen your understanding of course-related material, it is imperative that the work you submit is entirely your own creation. This means that any code, solutions, or written responses must be developed and written by you without copying from others.

Collaboration on assignments, unless explicitly permitted, is not allowed. This includes not only direct copying but also closely mimicking the logic or structure of another person's work. If you find yourself benefiting substantially from a discussion or an external resource, it is your responsibility to clearly acknowledge this in your submission, detailing the nature and extent of the assistance received. However, remember that your final work must still be predominantly a product of your own efforts.

As a reminder, plagiarism in any form is a serious violation of academic integrity. This includes the unauthorized use of AI tools like ChatGPT, online forums, code repositories, or any other sources that provide direct solutions. Seeking general advice and debugging help is permissible, but the line between assistance and infringement of academic integrity should not be crossed. If in doubt, consult the instructor or teaching staff for guidance on acceptable use of external resources.

Ultimately, the goal is for you to develop and demonstrate your own skills and knowledge. Adhering to these guidelines ensures a fair and meaningful educational experience for everyone involved.

Assignment Submission

Submit the following files to Gradescope in a zip file:

- **Completed .py files:** `task1.py, task2.py`
- **Outputs** `task1.csv, task1.png, task2.csv`
- **Report:** A PDF report.

Grading

The assignment is worth **30 points**. The grade will be broken down as:

Code Quality and Organization (8 Points) The clarity and organization of your code are crucial for this assignment. A well-commented solution that succinctly explains the use of variables, functions, and any complex approaches is required. Although detailed explanations will be provided in the accompanying report, the code itself should be intuitively structured and easy to understand, ensuring that graders can follow your logic and methodology without difficulty.

PDF Report (11 points) The report, a vital component of this assignment, should detail the functionality of your code and the reasoning behind your design choices towards solving each problem. The report should convincingly argue your aggregation approach in task 1 as well as your table design choice and algorithm in task 2. Lastly, it should explain the `DatabaseConnection` class, what it does well, and would could be improved about it.

Code and Output Files (11 points): The submission should include the completed `task1.py` and `task2.py` and the required output files. These files will be assessed for correctness based on the `largeRelationsInsertFile.sql` that was loaded in lab 0 and is included in the `sql-files/` directory. Code must be executable by the grader in their conda environment that has only loaded the packages in `requirements.txt`.