CS 327E Lab 3: Query Interface

Learning Objectives:

1. Continue working with your database
2. Continue working with your partner
3. Revise the data loader application from Lab 2 as necessary
4. Learn to write simple and complex SQL queries
5. Gain some experience protecting your database against possible SQL injection attacks

Prerequisites:

1. Lab 1 completed
2. Lab 2 completed

Steps Outlined:

0. Perform any reworks and/or enhancements to your Lab 2 submission that were noted in your graded rubric. For example, if a table didn't get loaded correctly, fix the import script for this table before proceeding with the work for this lab. Additionally, ensure that we can populate your database by only having to set our MySQL database connection details and run your `populate_database.py` (or `main.py`) script. We should only have to set our MySQL connection details in one place (e.g. in the `db_connect.py` file, rather than in each of the separate import scripts). Make sure to include the latest version of your `create_tables.sql` file in the **lab2** folder as well. All updates to the Lab 2 submission should be done to your existing **lab2** folder.

1. Create a new folder in your local git repository called **lab3**. All the work you do for this lab will go into this folder.

2. Write some SQL queries to explore and analyze the data that is in your database. You must come up with 10 different select statements, of which at least 5 must take some user input. The collection of 10 queries must satisfy the following minimum criteria:

- 2 group bys with a having clause
- 2 aggregates (e.g. count, min, max, sum, avg)
- 1 distinct
- 3 inner joins
- 2 outer joins
- 5 where clauses
- 5 order bys

In addition to the 10 select statements, you must come up with:

- 2 create views
- 5 queries using your views

Create a script called `queries.sql` with the above SQL queries and views. Make sure that each statement is separated by a semi-colon so that the script can be run inside a SQL client tool such as MySQL Workbench. Commit the script to your team's local repo and push the commit to your remote private repo on Github. For queries that require user input, show the query with example input inside of brackets. For example, `select * from Instructor where instructor_type = ['associate'] and status = ['Active'] order by [last_name];` (Note that the queries won't run when you add these brackets, but they should all run after we remove the brackets).

3. Develop a command-line interface in Python with menu options. The user chooses an option from the menu and the appropriate select statement is run through Python with the results displayed to the user. The user can then choose another option or exit from the program. The menu options can all be at the same level or they can form a hierarchy. For example, you can have one top-level option with several sub-options and some of the sub-options may have their own options. The depth of the hierarchy is going to be dependent on the queries that you formulate.

Regardless of how many levels you have, you must ensure that there is one menu option for each of the SQL queries that is in your `queries.sql`. The select statements that require some form of user input should take the input from the command-line and plug the values into the query as appropriate. The unchanging parts of the select statements can be hard-coded into the Python program. All select statements should be executed through the PyMySQL connector.

Use the function `run_stmt()` in `db_connect.py` to execute the select statements. Do not use `run_prepared_stmt()` as we have found that it does not permit column names to be passed in as parameters. Name the Python program `query_interface.py` and add it to your repo. Commit the file to your local repo and then push the commit to your remote private repo Github.

4. Once the query interface is functional, the next step is to add some basic protection for SQL injections. Although we are not using prepared statements, we can still check for "bad" characters in the input strings. If the user input is expected to be a numeric value, ensure that a valid number was passed. If the user input is expected to be a string, check that it doesn't contain the following substrings:

- "`; drop table`",
- "`; truncate table`"
- "`; delete from`"
- "`or 1=1`".

If any of these “bad” substrings are found in the user input, print out an appropriate error message without executing the SQL statement. Commit your updates to `query_interface.py` and push the commit to your remote private repo on Github.

5. Ensure that all the scripts for the query interface are in your **lab3** folder on Github. Locate the last **commit id** that you are using for your submission and paste it into an email. Your email should also contain a link to your team's repo on Github. Address the email to the professor and both TAs and carbon copy your lab partner. The subject of the email should be: [CS327E][Lab3][<TeamName>], replacing <TeamName> with your actual team name. The email is **due Tuesday, 11/15 at 11:59pm**. If it's late, there will be a **10% grade reduction per late day**. This late policy is also documented in the syllabus. Note: only one member per team should send the submission email.

Coding Conventions:

1. Place all reusable code in functions. For example, the database connection code should be defined in its own function because it’s used throughout the program. If you are using `db_connect.py`, note that we have made some enhancements to it, so be sure to pull the latest version from Github.

2. Use basic error handling with `try-except` blocks. Catch only the errors that you can handle and exit the program when an error is fatal (e.g. missing input data file, invalid database connection, etc). Use `print` statements in the `except` block to report the error regardless of whether you choose to continue or exit the program.

Teamwork & Collaboration:

1. We will use 3 class meetings (11/07, 11/09, and 11/14) to work on this lab.
2. We expect each team to split up the work as evenly as possible between the two members.
3. We expect each team to make several commits throughout this project. We don’t want to see a single giant commit before the assignment is turned in!
4. We expect each team to use the Github Issue Tracker: opening tasks, assigning them, and tracking their status. We will be reviewing the Issue Tracker to ensure that both team members are contributing equally to the project.

Resources:

Lab 3 Grading Rubric: http://tinyurl.com/zxzsuef
Lab 3 Team Sign-up Sheet: http://tinyurl.com/j6hzgvw
Code Samples: https://github.com/cs327e-fall2016/snippets