## CS 327E Lab 3: Query Interface

**Learning Objectives:**

1. Team up with your partner on this project
2. Gain practice writing SQL queries against your IMDB database
3. Learn how to access Postgres from Python
4. Learn how to implement a simple command-line interface in Python

**Prerequisites:**

1. Lab 1 complete
2. Lab 2 complete
3. Lab 3 setup complete [1]

**Steps Outlined:**

1. Perform any **revisions** to your Lab 2 submission that were noted in your team's graded rubric. This includes updating the DDL, re-creating tables, reloading data, altering and updating tables as well as correcting the JSON document in the Stache entry. Once all the necessary revisions have been made, commit and push all code changes to your **lab2** folder on Github. You may now start on Lab 3.

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

3. Design 10 **search options** to search your IMDB database. A search option is implemented by a single `SELECT` statement when it is a simple search or a sequence of `SELECT` statements when it is a complex search. A search option can optionally accept filter conditions from the user. The filter conditions must appear in the `WHERE` and `HAVING` clauses of the `SELECT` statement. In the expressions `WHERE col1 =<> abc` and `HAVING col2 =<> xyz`, `abc` and `xyz` can both be used as input parameters, but `col1` and `col2` cannot. For example, in the query `"select * from Actors where fname = 'Kevin' and lname = 'Spacey'"`, the inputs would be `'Kevin'` and `'Spacey'`, but not `fname` or `lname`.

The collection of 10 search options must satisfy the following minimum requirements:

a) contain 3 inner joins
b) contain 1 outer join
c) contain 3 aggregate functions
d) contain 2 group bys
e) contain 2 having conditions
f) contain 5 where conditions
g) contain 3 order bys

Create a file `queries.txt` with the 10 search options written in English. Create a file `queries.sql` with the search options written in SQL. You should hard-code any input values that are needed for the

time being. Run `queries.sql` against your IMBD database using the psql command `\i queries.sql;` and make sure that it runs error-free. Add and commit both files to your local and remote repos.

4. Pull the latest Lab 3 snippets from Github [3] and read through the **sample code**. We have provided you a simple command-line interface in Python with a few search options. Notice how each option calls a function that runs the appropriate SQL query (or queries) and prints out the results.

5. Using the Lab 3 snippets as a starting point, implement your own command-line **interface** based on the SQL that you developed in Step 3. When the interface starts up, it should list the 10 search options you designed. The user selects one of the options and is prompted to enter any required input. The interface validates the user input and runs the appropriate SQL query. The interface returns the query results and lets the user select another search option. If the user selects an invalid option or enters bad input, the interface returns a descriptive error message and reprompts the user for input. The interface does not need to authenticate the user or remember what actions the user has taken.

Create a file `interface.py` that contains the menu of search options. Create a file `queries.py` that contains the logic to process the SQL queries. The database credentials should be stored in a separate `config.py` file and imported as a module. Add and commit the `interface.py` and queries.py files to your repos as well as any other Python modules you have implemented except for `config.py`. Please **do not** add `config.py` to the repo as we do not want your database credentials leaking out into the Web.

6. For each SQL query that contains a filtered condition, create an **index** on the filtered column(s) to speed up the search. For example, in the query `"select * from Actors where fname = 'Kevin' and lname = 'Spacey'"`, add a concatenated index on `fname` and `lname` as follows:

```
CREATE INDEX actor_name_idx ON Actors (fname, lname);
```

The `create index` command is documented in the Postgres manual [5] and several examples are provided there. Note that for concatenated indexes, it is important that the order of the columns in the `create index` statement match the order of the columns in the `where` clause.

Once you have decided which indexes to build, create a `indexes.sql` file will all the `create index` statements and run them against your IMDB database. Add and commit `indexes.sql` to your local and remote repos.

7. Review the **Stache** entry that you submitted for Lab 2 and ensure that it is up-to-date. The secret field should contain the following JSON document with the appropriate values filled in:

```
{
  "aws-username": "shouldbeAdmin",
  "aws-password": "mypassword",
  "aws-console-link": "myconsolelink",
  "rds-endpoint-link": "myrdsendpoint_without_port_number",
  "rds-username": "shouldbeMaster",
  "rds-password": "myrdspassword"
}
```

Please use a JSON validator like JSONLint [4] to ensure that your JSON document is properly formatted.

In addition, the Stache entry should a read-only API endpoint and read key (through the option: allow this item to be accessed by read-only API calls). Make sure to save the Stache entry if you made any changes to it.

8. Locate the **commit id** that you will be using for your team's submission. This is a long 40-character that shows up on your main Github repo page next to the heading "Latest commit" (e.g. commit 6ca6f695bca36f7fc2c33485d1080ae30f8b9928). Locate the link to your team's repo. This is the URL to your private repo on Github (e.g. https://github.com/cs327e-spring2017/xyz.git where xyz is your repo name). Go back to the Stache entry and locate the read-only API endpoint and read key. Replace the commit id, repo link, API endpoint, and read key in the JSON string below with your own:

```
{
  "repository-link": "https://github.com/cs327e-spring2017/xyz.git",
  "commit-id": "6ca6f695bca36f7fc2c33485d1080ae30f8b9928",
  "stache-endpoint": "/api/v1/item/read/62021",
  "stache-read-key": "ec1f815a603234eb8c5e2c02b474839f0b6d3b9e76b103f1ab0463b655e6661b"
}
```

Create a file called **submission.json** that contains your modified JSON string.

Click on the Lab 3 Assignment in Canvas and upload submission.json. This submission is due by **Friday, 03/03 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: there should be one submission per team.**

**Teamwork:**

1. We will use 2 class meetings (02/27 and 03/01) to work on this lab.
2. We expect each team-member to contribute equally to the assignment and we will be checking the commit history to ensure that this is happening as expected.
3. We want you to use the Github Issue Tracker to assign and track the status of tasks.

**Resources:**

[1] Lab 3 Setup Guide: https://github.com/wolfier/CS327E/wiki/Setting-up-Lab-3
[2] Lab 3 Grading Rubric: http://www.cs.utexas.edu/~scohen/assignments/rubric3.pdf
[3] Lab 3 Snippets: https://github.com/cs327e-spring2017/snippets/tree/master/lab3_template
[4] JSONLint: http://jsonlint.com/
[5] Create index commands: https://www.postgresql.org/docs/9.6/static/sql-createindex.html