Project Part 3

(worth 40% of overall Project grade)

Your team

You will do Part 3 with the same team as for Part 1. If your team partner dropped the class and you did not submit a contingency plan for this with your Part 1 submission, then unfortunately you will still have to complete the whole project by yourself. If you team partner dropped the class and you did submit a contingency plan for this with your Part 1 submission, then you are welcome to switch to this reduced version of your project.

Overview of Part 3

As you recall from Part 1, you had two options for Part 3 of the project: you could either follow the Web Front-End Option or the Expanded-Design Option. You will follow for Part 3 the option that you stated in Part 1.

Important notes

- You should make sure that you loaded sufficient data into your database to show off all functionality of your application.
- You can make (hopefully small) changes to the SQL schema that you created in Part 2. If for some strange reason you feel the need to make any radical changes, please check with your project mentor ahead of time to avoid any last-minute surprises.

Web Front-End Option

If you are following the Web Front-End option, you will finish building the application that you proposed in Part 1, on top of the database that you created in Part 2. For the final evaluation of Project Part 3, you will need to submit your code for your application and a README file on CourseWorks by **Monday March 22** (see below for further instructions). Also, both team members will meet with your project mentor between March 23rd and March 26th. Your project mentor will contact you shortly to **schedule a 15-minute meeting** for one of those days. Your implementation will be on Python 3 using Flask, and should satisfy these requirements:

- Your application must execute SQL query strings on your database on our class's PostgreSQL server. You cannot use an Object-Relational Mapper, or ORM. An important goal of this project is that you practice writing and debugging SQL queries as part of your application, so tools that attempt to make this "too easy" are not permitted.
- Your application must provide a way to view or interact with all the entities and relationships in your final E/R diagram.
- Your application's web interface does not need to be beautiful or sophisticated. Plain text pages are acceptable. You will not get additional credit for fancy interfaces, as this is not the focus of our course.
- In general, you can use any third-party libraries you want except for ORMs or other libraries that simplify database access, which are not allowed. If you are unsure if a library is permitted, ask your project mentor.

The following resources may be helpful for learning both Python and Flask:

- Java-to-Python Primer
- Python tutorial
- Flask documentation
- Flask tutorial
- Jinja template documentation
- Jinja tutorial

Getting started

Your job is to implement your proposed web application. To help you out, we have provided a bare-bones Flask web application, `server.py`, available here. It provides code that connects to a database URL, and a default index page. Take a look at the comments in `server.py` to see how to use or modify the server. In particular, what you will need to modify the value of `DATABASEURI` inside `server.py`, to refer to your PostgreSQL username and password. You will need to connect to your database from Part 2. Please read all these directions. Once you get it running, you should start working on your custom logic.

Important: Please run `python --version` to figure out what version of Python is the default in your VM/virtual environment. If the version is 2.7.*, you will need to run Python as `python3` to make sure you are using Python 3.

A short introduction to SQLAlchemy

We use a Python package called `SQLAlchemy` to simplify our work for connecting to the database. For example, `server.py` contains the following code to load useful functions from the package:

```
from sqlalchemy import *
```

`SQLAlchemy` is able to connect to many different types of DBMSs, including SQLite, PostgreSQL, MySQL, and Oracle. Each such DBMS is called an "engine." The `create_engine()` function sets up the configuration to the specific DBMS we want to connect to:

```
engine = create_engine(DATABASEURI)
```

Given an engine, we can then connect to it (this is similar to how `psql` connects to our class's PostgreSQL database server):

```
conn = engine.connect()
```

At this point, the `conn` connection object can be used to execute queries to the database. This is basically what `psql` is doing under the covers:

```
cursor = conn.execute("select 1")
```

The `execute` function takes a SQL query string as input and returns a `cursor` object. You can think of this as an iterator over the result relation. This means you can run `SELECT *` on, say, a million-row table and not run out of memory. Instead of receiving the entire result at once, this object lets you treat the result as an iterator and call `next()` on it, or loop through it. See the SQLAlchemy documentation for a detailed description.

```
# this fetches the first row if called right after
# the execute function above. It also moves the
# iterator to the next result row.
record = cursor.fetchone()
# this will fetch the next record, or None if
# there are no more results.
second_record = cursor.fetchone()
# this loops through the results of the cursor one by one
for row in cursor:
  print (list(row))
```

The above description is a way to directly write and run SQL queries as strings, and directly manipulate the result relations. SQLAlchemy also includes an Object Relational Mapper that provides an interface that hides SQL query strings and result sets from you. **In this project, you will directly write and run SQL queries, and cannot use any ORM functionality.**

(Optional) Using a toolkit for application front-end

You are welcome to use the Bootstrap framework to style the front-end of your application. Note that this is by no means necessary, and styling your application **will not impact grading in any way**; see "Grading for Web Front-End Option" below. You are also welcome to use AJAX to create more complex interactions with your site. However, if you do so, you must use JQuery and you must add a section to your README file describing where each call occurs, what triggers the call, what endpoint it hits, and how it's processed. Furthermore, any JavaScript you write must be well-commented, and cannot rely on frameworks or libraries not mentioned here. Particularly, you cannot use any tools that alter the way you interact with the database (e.g., GraphQL). If you have any questions regarding specific frameworks, please contact your project mentor.

Working with a version control system

Since you are working collaboratively on your teammate, we **strongly encourage** you to use a version control system for your code such as `git` on GitHub. You should use a **private repository** on GitHub for this project.

We cover below the basics of `git` and `GitHub`. With this setup, you can code on your own desktop, commit and push your changes to the `GitHub` repository, and then pull the updated changes on your Google Cloud Compute Engine. You can also code without your teammate being in the same room, and your teammate will be aware of the updates that you push to or pull from `GitHub`. Conversely, your teammate will also be able to contribute to the same code repository, so you can both work collaboratively. Please follow these steps to get started with `git` and `GitHub`:

1. Register an account on `GitHub`. You will need to provide/verify your email address and provide a username. There is no requirement on what email or username you use here. Your teammate should also register for a different account, so that you both have access to the code repository.
2. Once your account is created, log into `GitHub`. Create a repository by clicking on Create a new repository.
3. Give your repository a name, such as "w4111-proj1". From here on we will use <projectname> to denote the name that you chose in this step for your project. Make sure that you make this repository **Private** by choosing the correct radio button; also be **not** check the README box.
4. Give your teammate access to the repository, as follows: if your repository is at URL <REPOURL>, go to webpage <REPOURL>/settings/collaboration by manually adding "/settings/collaboration" to the URL (e.g., if your repository is at `https://github.com/c13483c1/db4111`, then go to `https://github.com/c13483c1/db4111/settings/collaboration`). Enter your teammate's `GitHub` username or email, and click "Add collaborator." Your teammate will receive an invitation by email, which your teammate should accept. After this, we are done with the `GitHub` web interface.
5. Now start your Google Cloud Compute Engine (recall our instructions for Part 2). `ssh` into it, and run the following commands:

   ```
   # download and extract our skeleton Flask web application
   cd ~
   wget http://www.cs.columbia.edu/~kar/cs4111/spring21/webserver.tar
   # webserver.tar can also be downloaded from Courseworks
   tar xf webserver.tar
   mv webserver <projectname>
   cd <projectname>
   chmod -R 777 .
   # configure your git environment
   git config --global user.name "<your full name>"
   git config --global user.email "<email you used to register for GitHub>"
   # initialize git repository, and push the skeleton files up to the server
   # here <username> denotes your GitHub username
   git init
   git remote add origin <REPOURL>.git
   git add *
   git commit -m "Initial commit"
   git push -u origin master # you will be prompted for your username and password here
   ```

   You have now created a local `git` repository, made your first commit, and pushed the files contained in webserver.tar onto `GitHub`. Your usual workflow after these initial steps will be a bit different than this, as described in the next step.
6. Your workflow with `git` will typically comprise the following steps:
   a. Pull your changes from the `GitHub` repository so that your code is up-to-date: `git pull`. Note that this step is necessary because, when working on a project with a teammate, your teammate may have updated and pushed the code while you were not working on the project, so you should always pull the changes before beginning to work on a new revision.
   b. Work on the project, make some changes locally, and test them.
   c. See what `git` thinks you have changed: `git status`
   d. Add any modified files to the next "commit." For example, if you modified files `server.py` and `README`, then run `git add server.py README`
   e. Commit your changes (locally) with an appropriate commit message: `git commit -m "added feature X"`
   f. Push your changes to the `GitHub` server: `git push`
7. Step 5 is more or less all you need to get started if you are working on this project alone. However, if you are working as a two-person team, you might even be using different computers to do your coding. (It's cumbersome to develop your code directly on Google Cloud, and it's much more convenient to use your desktop or laptop for writing the code and then "pull" the code into your Google Cloud account using `git`, as discussed in the "Running your application..." section below.) So if you carried out Step 5 above for your team, then your teammate will need to perform the following steps to get a copy of the code on their desktop or laptop. Similarly, you will have to perform the steps below for any new computer that you want to use to continue developing your code:

   ```
   # configure your git environment, as in Step 5
   git config --global user.name "<your full name>"
   git config --global user.email "<email you used to register for GitHub>"
   # clone (i.e., download) the repository onto your local machine
   # use the username of the owner of the repository here
   git clone <REPOURL>.git
   # you're done setting up; you can now continue as in Step 6
   ```

`git` provides many other powerful functions that we haven't talked about. For more information, you can refer to this tutorial, and documentation on that website. If you encounter a problem with `git`, you can typically do a quick search with the error message and get **a lot** of helpful information online. Feel free also to come to office hours with any of your questions about `git` and `GitHub`.

Running your application on your Google Cloud compute engine

Once you have developed (a preliminary version of) your web application, you will deploy it to your Google Cloud Compute Engine, as follows

1. (One time setup) Follow our directions to make a port accessible to the internet so anyone can access your application.
2. Write down the IP of your virtual machine.
3. `ssh` to your virtual machine and enter the virtual environment you created in Part 2 of the project.
4. Make sure you have committed and pushed all the latest changes to your code to your `GitHub` repository. (See Step 6 above.)
5. Copy the latest version of your code to the Google Cloud Compute Engine by running `git pull` on your virtual machine.
6. Run the Python server with the defaults parameters, which will listen for requests on port 8111. Run with `--help` if you need help:

   ```
   # <projectname> is the name of your repository and directory that you created in Step 5 above
   cd <projectname>
   python3 server.py
   ```

7. Go to `http://<IP ADDRESS>:8111/` in your browser to check that it worked.

You will need this URL when presenting the project to your mentor. Please do not turn off your virtual machine after you are done modifying your code and when you are ready to submit, so that your IP address does not change and the URL that you include with your project submission works.

Keeping your application running while meeting with your mentor: To keep your application running "in the background" (so that it is available when you meet with your mentor), you can use the `screen` command. To install `screen`, run on your VM: `sudo apt-get install screen`. Run `screen` in the terminal, then switch to the correct environment (recall our instructions for doing so), and finally execute your server application normally (i.e., by running `python3 server.py`). Now, you can detach from `screen` by pressing CTRL + a, and then d. The application will be running in the background now and it is safe for you to log out of the `ssh` session. Run `screen -r` to bring back the detached screen (and your process) to the foreground, and to stop your application (after we are done grading Part 3). (For more information and details on `screen`, please refer to https://www.mattcutts.com/blog/a-quick-tutorial-on-screen/.)

What to submit and when for Web Front-End Option

If you are following the Web Front-End Option, you will need to submit your code for your application and a README file on CourseWorks by March 22nd. Here are the instructions for your electronic submission:

- Create a directory named <group>-proj1-3, where you should replace <group> with your Project 1 Group as specified on CourseWorks (for example, if your group is "Project 1 Group 9," then the directory should be named group9-proj1-3).
- Copy all the Python source code files into the <group>-proj1-3 directory, and include all the other files that are necessary for your program to run.
- Tar and gzip the <group>-proj1-3 directory, to generate a single file <group>-proj1-3.tar.gz, which is the first file that you will submit.
- Login to CourseWorks and select the site for our class. To submit this file, you need to be in the Class view (not the Group view) and then upload your file to the "Part 3" assignment under Assignments. Submit file <group>-proj1-3.tar.gz.
- Separately, submit a compressed README file with the following information:
  - The PostgreSQL account where your database on our server resides. (This should be the same database that you used for Part 2, but we need you to confirm that we should check that database.)
  - The URL of your web application. Once again, please do not turn off your virtual machine after you are done modifying your code and when you are ready to submit, so that your IP address does not change and the URL that you include with your project submission works.
  - A description of the parts of your original proposal in Part 1 that you implemented, the parts you did not, which hopefully is nothing or something very small, and possibly new features that were not included in the proposal and that you implemented anyway. If you did not implement some part of the proposal in Part 1, explain why.
  - Briefly describe two of the web pages that require (what you consider) the most interesting database operations in terms of what the pages are used for, how the page is related to the database operations (e.g., inputs on the page are used in such and such way to produce database operations that do such and such), and why you think they are interesting.

In summary, you need to submit on CourseWorks exactly two files: (1) your <group>-proj1-3.tar.gz file with your code and (2) your uncompressed README file. You need to submit these two files by March 22nd.

Additionally, both teammates in each team will meet together with their project mentor between March 23rd and 26th. Your project mentor will email you shortly to schedule a 15 minute meeting for either day. (If you haven't received an email from your project mentor by March 19th please contact your mentor immediately after that day.) During the meeting with your project mentor, you will show your mentor your application using a regular web browser:

- You should have your application up and running so that you and your project mentor can access it over the web simply by typing a URL in a regular browser. Your project mentor will be running Chrome.
- Your project mentor should be able to interact with your application and access the functionality that you specified in Part 1, over the database that you created for Part 2.
- The project mentor might ask to look at your code during your meeting.
- Your web interface does not need to be fancy. (See Grading below.) However, you should not force users to type SQL. The less your web site looks like it is interacting with a relational database, the better. At the very least, the user should be completely shielded from having to type anything resembling SQL. Most interactions should involve some sort of input values in addition to the user pressing a "submit" button. Whenever possible, input values should be specified using menus, radio buttons, checkboxes, scrollers, etc. Text input boxes may also be appropriate.
- Sophisticated error handling is not necessary; however your web site definitely should not "lock up" (i.e., crash or become blocked) regardless of how the user chooses to interact with it.
- Your database should contain (at least) the data that you entered for Part 2. You can, of course, add extra tuples to your tables if you want to make interaction with your application more interesting and revealing.
- Your grade will suffer considerably if your application is not running properly when you access it from your project mentor's machine. It is your responsibility to ensure that your application is up and running when you meet with your project mentor.
- You should have a number of example interactions prepared so that you can use your meeting time efficiently. The more you can impress your project mentor during the 15-minute meeting, the better your grade is likely to be, so come to the meeting prepared.

Grading for Web Front-End Option

Your grade for Part 3 of Project 1 will be a function of how well your application (which should be up and running) matches your specification that you submitted as Part 1, of how well you have incorporated any feedback that your project mentor has given you, and of how well you have followed the guidelines above. Your grade will not be influenced by how fancy the web-based user interface to your application is. It is sufficient and perfectly fine for this interface to be plain and simple as long as it supports the functionality that you indicated earlier, following the guidelines above about not having to type SQL commands, not "locking up" on unexpected input, etc.

Expanded-Design Option

If you are following the Expanded-Design Option, you need to follow the expansion plans that you outlined in Part 1, and:

1. Extend your E/R diagram from Part 1 to include the entity sets and relationship sets—and all associated real-world constraints—for your expanded design.
2. Extend your SQL schema from Part 2 of your database on our PostgreSQL server to include the mapping of all new entity sets and relationship sets, following the procedure from Part 2 of the project on how to specify constraints in your expanded SQL design.
3. Add tuples to your new tables on our PostgreSQL server, following the guidelines on the number of tuples from Part 2 of the project.

What to submit and when for Expanded-Design Option

You will submit this part of the project electronically on CourseWorks directly, along the lines of what you did for Part 2. The deadline is **Monday March 22nd.** just as for Parts 1 and 2, your submission should be your project exactly once per team, rather than once per student. To submit your project, you need to be in the Class view (not the Group view) on CourseWorks and then upload your file to the "Part 3" assignment under Assignments. You should submit one or more (uncompressed) files containing:

1. The name and UNI of both teammates.
2. The PostgreSQL account name for your database on our server (i.e., specify which teammate's UNI we should use to identify the database for your team.) This will normally be the same database that you used for Part 2, but we need you to confirm that we should check that database.
3. A textual description of your extensions on the database design, explaining which entity sets and relationship sets are new, and how you mapped them to SQL statements.
4. Your new, complete E/R diagram, including all of your entity sets and relationship sets, both from Part 1 and the new ones from Part 3.
5. The CREATE TABLE statements and any other elements of the full database that you created on your PostgreSQL database. (We will of course also check the schema directly on the database server, but we need all the statements as part of your submission file.) You should include all of your tables, not just the new ones for Part 3.
6. Three "interesting" SQL queries over your expanded database, with a sentence or two per query explaining what the query is supposed to compute. Each of these queries should involve at least one of the new tables that you added for Part 3. The goal of these queries is to help us better understand your application and your additions for Part 3. You will not be graded on these queries, but we strongly suggest that you submit well formed queries that run without problems, so please make sure that you have tested your queries by running them on your database exactly as submitted (use copy and paste).

Grading for Expanded Design Option

Your grade for Part 3 of Project 1 will be a function of how well you have incorporated any feedback that your project mentor has given you, and the following factors:

1. Quality of your expanded E/R diagram: We will evaluate how well your expanded E/R diagram implements your plans for the Expanded-Design Option from Part 1, and how well your expanded E/R diagram models your application, including how well you modeled any relevant real-world constraints.
2. Quality of your expanded SQL schema and implementation on PostgreSQL: We will evaluate how well you mapped your expanded E/R diagram, including constraints, into a SQL schema on PostgreSQL, using the techniques that we covered in class.
3. Quality of your expanded constraint handling: We will evaluate how well you managed to capture real-world constraints of your expanded design through primary key, foreign key, unique, and good-style attribute- and tuple-based CHECK constraints.
4. Quality of the expanded real-world (or at least realistic) data that you loaded into the expanded database on PostgreSQL.