CSC 103 - HW 9

Complete each of the following exercises. To receive full credit please ensure the following:

- Submission structure follows the following:

```
HW9
    -> data
           -> school.db
    -> init_db.py
    -> school.py
    -> Q1.txt (or any extension that makes sense if you decided to draw it)
    -> Q2.txt (or any extension that makes sense if you decided to draw it)
```

- Exercise artifacts are uploaded to your GitHub repository under a folder called `HW8` (i.e. https://github.com/<username>/carlow-me)
- Exercise artifacts are uploaded to Brightspace as a compressed `.zip` file called `<LastName>_HW9.zip`.

It is crucial that you are naming artifacts and storing files in the format mentioned above. All files that need to reference datasets should use a relative reference that refers to it's location in the `data` directory and not an absolute reference that would only work on your machine. Failure to meet any of these requirements will result in loss of points for that and any following problems.

Exercises

This week's exercise is a bit different. This week we are going to be building a collective application (API) that needs to meet a series of criteria. So what you'll need to do is provide the appropriate code in `school.py` that accomplishes those tasks and stores or reads data for any of those tasks from a database called `school.db` that should be stored in the `data` directory.

1. Provide a diagram that describes the design of the existing database. This should include the table names, the columns, and the data types for each of those columns. Also provide information regarding how each of these tables may related to one another.
2. Extending the diagram you provided in the previous exercise, extend this by providing a new table called `schools`. This should describe the universities that are related to all of

our existing entities. Describe columns that would be associated with this set of actors and how this could related to our existing tables.
3. Create a script called `init_db.py` at the same level as `app.py` that will initialize the `school.db` database and tables needed in the rest of the exercises.

4. Create an endpoint called `student` that accepts GET, POST, and DELETE actions
5. Create an endpoint called `teacher` that accepts GET, POST, and DELETE actions
6. Create an endpoint called `class` that accepts GET, POST, and DELETE actions

For any future actions that ask to store data, each endpoint should store that data in its respective table within the `school.db` database.

Teachers should have the following attributes: ID, Name, Email, Phone
Students should have the following attributes: ID, Name, Email, Phone, Year, Status
Class should have the following attributes: ID, Name, Department, TeacherID

7. Provide the ability to create a teacher via POST with data for the above fields.
8. Provide the ability to create a student via POST with data for the above fields.
9. Provide the ability to create a class via POST with data for the above fields. All fields except for the Department are required. If Department is not provided, it should default to the value of Misc.
10. For all required fields, if a field is not provided the response should return a `HTTP 500` error with the message `{'error': '<field name> is required'}' where `<field name>` is replaced with the field that is missing
11. When any of the above POST actions are taken, the appropriate values should be written to the database. Additionally, if any of the above is successfully created, it should return a JSON object containing the ID of the successfully created object.

When deleting any of the above entries, we should allow this to be done by passing the ID to the appropriate endpoint (i.e. `{'id':1234}`).

12. Provide the ability to delete a teacher via DELETE by passing the ID via data
13. Provide the ability to delete a student via DELETE by passing the ID via data
14. Provide the ability to delete a class via DELETE by passing the ID via query param.

When attempting to get information from any of the endpoints, we should be able to retrieve this data by providing the ID for that component via a url parameter. Responses for each respective object should return all available fields for that object.

15. Provide the ability to get a teacher information via GET via ID
16. Provide the ability to get a student information via GET via ID
17. Provide the ability to get class information via GET via ID

For the below exercises, you can get the count of rows from a database by doing `SELECT COUNT(*) FROM TABLE` (and additionally using any WHERE clause that may be appropriate).

18. Provide the ability to make a GET request to the class endpoint with a query parameter of `count=true` . If this parameter is present, it should return data that looks like the following: `{'count': <number>}` where number represents the number of classes that are present.
19. Provide the ability to make a GET request to the teacher endpoint with that teachers ID as a url parameter. If the query parameter `count=true` is present, it should return the number of classes that teacher teaches as part of the response. That is to say, the usual data should be returned with `count` as an additional field represented in a way similar to Q15.
20. Create two new endpoints that allow us to update the phone number for students and one that allows us to update the phone number for teachers. It should be built off of the existing `teacher` and `student` endpoints. This should be accomplished by providing the ID and the updated phone number via query param with the keys `id` and `phone`.