Lab Homework 7: Initializing a Database & SQL Joins

In this lab you will learn how to initialize a database with only using SQL. You'll also practice joining tables using SQL's `JOIN` clause.

1. Learning Objectives

- Learn how to create a database with just SQL (no *DB Browser for SQLite*).
- Practice using the SQL reference documentation.
- Practice joining DB tables using SQL `JOIN`.

2. Deadline

Lab Homework	Deadline	Slip Days	Credit	Solution
All Parts	Sun 4/12, 11:59pm ET	Max: 2 days	20 points (completion)	Provided

3. Instructions

1. **Clone** your lab repository.

   Clone the following URL:
   `git@github.coecis.cornell.edu:info2300-2020sp/YOUR_GITHUB_USERNAME-lab07.git`
   Replace **YOUR_GITHUB_USERNAME** in the URL with **your Cornell GitHub username**.

2. **Work together.**

   Feel free to work with your peers to complete this lab. Use your section specific chat rooms. Organize a Zoom hangout to work on the assignment together. Take this as an opportunity for some *virtual* human contact!

   Note: You are encouraged to work together so long as you do your own work and you don't give away answers.

3. **Ask questions** or **say hi** during your registered section live Zoom Q & A.

   Your section leaders will hold a Zoom Q & A during your registered section time. Feel free to pop in and **say hi** or **ask a question**! Again, use this as another opportunity to keep up with your fellow Cornell community members!

4. **Submit.**

   When you're finished, follow the instructions in **submit.md** to submit your assignment.

Part I: Initialize a Database for 2300 Plop Box

We're going to code up our own Dropbox-like clone, 2300 Plop Box. In this lab, you'll get the database ready and prepare some seed data to test with. In the next lab, we'll learn how to implement file uploads to finish our Plop Box.

In this lab, we will create our database using SQL rather than using *DB Browser for SQLite*.

1. Scripting Initializing a Database with SQL

In your repository, please open **secure/init.sql**. This SQL code initializes all the tables and seed data for our 2300 site.

We will need to execute the queries in **secure/init.sql** to create our database. Fortunately, we've provided a user-defined function in *init.php* that does the work for you: `open_or_init_sqlite_db()`. Observe that we have also removed the `open_sqlite_db()` function since we will no longer create databases using *DB Browser for SQLite*.

Objective: Take a moment with your peers and try and figure out what `open_or_init_sqlite_db()` does:

```php
function open_or_init_sqlite_db($db_filename, $init_sql_filename)
{
  if (!file_exists($db_filename)) {
    $db = new PDO('sqlite:' . $db_filename);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    if (file_exists($init_sql_filename)) {
      $db_init_sql = file_get_contents($init_sql_filename);
      try {
        $result = $db->exec($db_init_sql);
        if ($result) {
          return $db;
        }
      } catch (PDOException $exception) {
        // If we had an error, then the DB did not initialize properly,
        // so let's delete it!
        unlink($db_filename);
        throw $exception;
      }
    } else {
      unlink($db_filename);
    }
  } else {
    $db = new PDO('sqlite:' . $db_filename);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    return $db;
  }
  return null;
}
```

Note: The code in *init.php* is a bit more complicated in order to help students remember to regenerate their database if *init.sql* changes.

Spoiler: This function checks to see if your SQLite database exists, if it does, it opens it. If it doesn't exist, then it will create a new database by reading the SQL init script, and then executing the SQL from the init script. This creates your database and initializes it with your tables and *seed* data.

IMPORTANT!: You must reinitialize your database if you change *init.sql*! Simply delete the .sqlite database file (*secure/site.sqlite*) and the next time you load a PHP page that calls `open_or_init_sqlite_db()` it will reinitialize the database for you!

Note: SQLite is a development database. In most circumstances you should not check-in (commit and push) a development database in your Git repository. Observe that there is a **.gitignore** in the root of your repository. Using this file, we've told Git not to allow you to check-in your **.sqlite** files. **This is intentional.** You should not commit your databases for any assignments the remainder of the semester.

2. Initialize Database

Objective: Look for the **TODO** in *init.php*. Initialize and create a connection to the database. The database should named **secure/site.sqlite**. Store the database connection in a `$db` variable.

3. Create a Database using SQL

Open **secure/init.sql**.

Observe here how it uses the `CREATE TABLE` statement to initialize the database's tables. Take a moment and review the reference documentation for `CREATE TABLE`. Here are two links that might help:

- https://www.sqlite.org/lang_createtable.html
- https://www.w3schools.com/sql/sql_create_table.asp

For our *Plop Box* we want a `documents` table that includes the following information:

1. A field to be a primary key to identify each individual entry.
2. A field to store the original name of the uploaded file.
3. A field to store the file extension of the uploaded file_ext.
4. A field that takes in an optional description that users may want to provide about their file.

Objective: Using the reference documentation above, write the SQL code in **secure/init.sql** to create our *documents* table with the following database schema:

Field	Type	Not Null	Primary Key	Auto Increment	Unique
id	INTEGER	Yes	Yes	Yes	Yes
file_name	TEXT	No	No	No	No
file_ext	TEXT	No	No	No	No
description	TEXT	No	No	No	No

Tip: If you want to check if you query is valid, create a test database in *DB Browser for SQLite* and execute your `CREATE TABLE` query. When you're done testing, simply throw away the test database and copy and paste the SQL query into your *init.sql* file.

Objective: Test that your `CREATE TABLE` query in *init.sql* properly initializes the database:

1. Uncomment the 2 seed data SQL queries in *init.sql*.
2. Delete **secure/site.sqlite**
3. Refresh the 2300 site in the browser.
4. Visit the Plop Box page and see if you see two uploads (gregory.jpg, cornell-seal.svg).

   If you don't see the two uploads, you probably have an error. Fix your error and try again.

4. INSERT seed data using SQL

Seed data creates entries that populate the database when it is first created. Seed data is written as SQL queries in our database initialization script: **secure/init.sql**. The *uploaded* files that correspond to the seed records are already stored in the **documents** folder (same name as `documents` table) under **uploads**.

Objective: Check out the existing seed data for the `documents` table in **secure/init.sql** and in **uploads/documents**.

Objective: Add 3 new seed data records to the `documents` table. **Make sure that you also provide the corresponding *uploaded* file in the *uploads/documents* folder.** Each *uploaded* file should be named with the primary key and have the same file extension as the value of the `file_ext` field. See the existing seed uploaded files as an example.

When you are ready to test your seed data, simply delete the **secure/site.sqlite** file and then open or refresh the **Plop Box** in your web browser. This will recreate the database with your new seed data! Remember to do this **EVERYTIME** you change *init.sql*!

Part II: JOIN Queries

When designing a good database schema, you will make each table about *one thing* and the define relationships between tables using *foreign keys*. If you want to gather information from multiple tables, you will need to SQL's `JOIN` clause.

There are two `JOIN` type supported by SQLite:

1. `INNER JOIN`
2. `LEFT OUTER JOIN`

Below are examples of using these joins with the following data:

Our *left* table: **classes**

id	class_name
1	Rabbit
2	Elephant
3	Flower
4	Tiger
5	Lion
6	Sunshine

and our *right* table: **students**

id	name	age	class_id
1	Phoebe	3	6
2	Ross	3	2
3	Monica	2	1
4	Rachel	2	3
5	Chandler	3	2
6	Joey	2	1
7	Janice	5	*NULL*

1. INNER JOIN

We use an `INNER JOIN` when we don't want any NULL values from either table being included in our results. We only want matched entries.

For example, take the following query.

```
SELECT students.name, classes.class_name FROM classes INNER JOIN students ON classes.id =
students.class_id;
```

This query would `INNER JOIN` our left and right table and return the following records:

	students.name	classes.class_name
1	Phoebe	Sunshine
2	Ross	Elephant
3	Monica	Rabbit
4	Rachel	Flower
5	Chandler	Elephant
6	Joey	Rabbit

You should notice that there is no NULL in the results whether in the `students.name` or `classes.class_name` field. You can see with the INNER JOIN we have the least results as we leave out any un-matched meaning any entries that would have a missing or NULL value. Therefore, Janice, Tiger, and Lion are excluded from the results.

2. LEFT OUTER JOIN

We use a `LEFT OUTER JOIN` when we don't mind having NULL values from our right table being included in our results.

For example, take the following query.

```
SELECT classes.class_name, students.name FROM classes LEFT OUTER JOIN students ON classes.id =
students.class_id;
```

This would `LEFT OUTER JOIN` our right table **to** our left table when we want a result such as the following:

	classes.class_name	students.name
1	Rabbit	Monica
2	Rabbit	Joey
3	Elephant	Ross
4	Elephant	Chandler
5	Flower	Rachel
6	Tiger	*NULL*
7	Lion	*NULL*
8	Sunshine	Phoebe

We start off with the left table information and add the information provided from the right table, thus why Tiger and Lion are included and Janice is excluded. Because the entry for Janice is NULL for class_name, there is no existing class_name value it pairs with and thus is left out.

3. Multiple JOINS

You can JOIN multiple tables together, not just two. If you want to JOIN three tables, JOIN the first two, and then JOIN the third, treating the newly *joined table* as one new "table" (it's not really a table).

For example: **parents**

id	parent_name	student_id
1	Jack	2
2	Judy	3

```
SELECT students.name, classes.class_name, parents.parent_name FROM classes INNER JOIN students ON
classes.id = student.class_id LEFT OUTER JOIN parents ON students.id = parents.student_id;
```

	students.name	classes.class_name	parents.parent_name
1	Ross	Elephant	Jack
2	Monica	Rabbit	Judy

4. JOINS Practice

Now that we've reviewed the JOIN types, it's time for you to get some practice using them. You will answer questions in **joins.md**. Open and look over the file named **joins.md** and the **joins.sqlite** database.

The database **joins.sqlite** has multiples table where some tables share some information via foreign keys. In **joins.md**, we have written the information we would like to obtain from the tables; however, they require looking at multiple tables, not just one.

Objective: In the **joins.md** file, we have created questions for you to answer to guide you in forming the queries. Write all your responses and final query directly in the **joins.md** file for submission.

To test your queries, open **joins.sqlite** with *DB Browser for SQLite* and go to the tab "Execute SQL". Run your queries and check that your results match what we have given you. All of the queries can be done with just `LEFT OUTER JOIN` and `INNER JOIN`.

Contributors

The following individuals made contributions to this assignment.

- Kyle Harms
- Sharon Jeong