N03c_Web_App_with_Sqlite3_DB

In this chapter, students will learn and understand:

- Use `sqlite3.connect()` to open or create a `SQLite` file.
- Use `sqlite3.Connection.execute()` to run SQL commands.
- Use `sqlite3.Cursor.fetchone()` and `sqlite3.Cursor.fetchall()` to retrieve database rows.
- Set `sqlite3.Connection.row_factory` to `sqlite3.Row` in order to simplify the reading of values from retrieved database rows.
- Use `sqlite3.Connection.commit()` to save changes and `sqlite3.Connection.close()` to close SQLite files.
- Use the `CREATE TABLE`, `SELECT`, `INSERT`, `UPDATE`, `DELETE` and `DROP TABLE` SQL commands.

Section 1: Sqlite3 in Python

1.1 Creating Database Connection and Table

1. Open **app.py.** Import `sqlite3` as you will be using it later.

```
import os.path, sqlite3
```

2. In **app.py**, add the following code to create a database connection and table as shown below.

```
def get_db():
    db = sqlite3.connect('db.sqlite3')
    print("Opened database successfully");
    return db

def create_db():
    db = get_db()
    query = """
    CREATE TABLE IF NOT EXISTS posting (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT,
    email TEXT,
    message TEXT
    )
    """
    db.execute(query)
    print("Table created successfully")
    db.close()

#creates table only if database file does not already exist
if not os.path.isfile('db.sqlite3'):
    create_db()
```

3. Run **app.py**

You should see the following on Python shell if your database and table are created successfully.

```
Opened database successfully
Table created successfully
 * Restarting with stat
```

Section 2: CRUD Operations in Flask Web Application with Sqlite3

2.1 Inserting Data Into Table Using A Form

1. Open **app.py**.

2. Comment out the original `form` function.

3. Create a new `form` function as shown below.

We will rewrite the form function such that data entered through the web form on **message_form.html** will be added to the table, `posting`, instead of being displayed.

We will use an `INSERT INTO` statement to add the form data into the `posting` table. `'?'` is a placeholder for data to be entered by the user. After the insertion, the user will be redirected to the URL defined in the `success` function.

Take note that you will need to write the `success` function later.

```
@app.route('/form/', methods=['GET', 'POST'])
def form():
    if request.method == 'POST':
        db = get_db()
        query = """
        INSERT INTO posting
        (username, email, message)
        VALUES
        (?,?,?)
        """
        username = request.form['username']
        email = request.form['email']
        message = request.form['message']
        tpl = (username, email, message)

        db.execute(query, tpl)
        db.commit()
        db.close()
        return redirect(url_for('success'))
    else:
        return render_template('message_form.html')
```

4. Run **app.py**.

5. Enter `http://localhost:5000/form` on your browser.

The form page should load as previously done.

2.2.1 Selecting Data From Table

1. Create a new function called **success**.

Inside this function, write a `SELECT` statement to retrieve records from the `posting` table. The results will be stored in a data object, called `records`. This data object will then be passed as an argument into the `render_template` function with a new webpage, **message_board.html**, being rendered to display all records in the `posting` table.

After the user submits the data through the form, the form's `POST` method will pass the data for insertion into the table. After insertion, the `success` function is triggered. This function will retrieve records from the table and display them on **message_board.html**.

```
@app.route('/success/')
def success():
    db = get_db()
    query = "SELECT * FROM posting"
    cursor = db.execute(query)
    records = cursor.fetchall()
    cursor.close()
    db.close()
    return render_template('message_board.html',
                           records=records)
```

2. Create a new webpage and save it as **message_board.html** under templates. This webpage will be used to display the multi-dictionary data object called `records`, which actually stores the results of the `SELECT` statement executed in the database.

This webpage should display all fields of each data record in the `posting` table.

- `id`
- `username`
- `email`
- `message`

You may use a table to display the table headers and values.

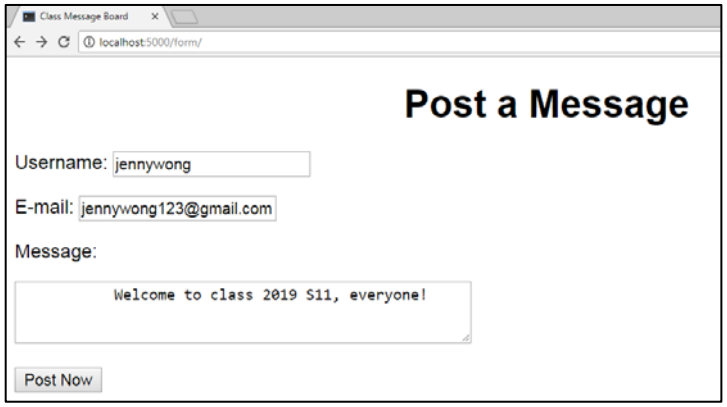
```
<!doctype html>
<html>
<head>
    <title>Message Board</title>
    <link rel="stylesheet" href="/static/css/style.css">
</head>
<h1>Message Board</h1>
<body>
    <table border="1">
        <tr>
            <th>Posting ID</th>
            <th>Username</th>
            <th>Email</th>
            <th>Message</th>
        </tr>
        {% for record in records %}
        <tr>
            <td>{{ record[0] }}</td>
            <td>{{ record[1] }}</td>
```

```
            <td>{{ record[2] }}</td>
            <td>{{ record[3] }}</td>
        </tr>
        {% endfor %}
    </table>
    <p><a href="{{ url_for('form') }}">Post again</a></p>
</body>
</html>
```

3. Run **app.py**.

4. Enter `http://localhost:5000/form` on your browser.

5. Enter some test data into the form. Click **Post Now**.

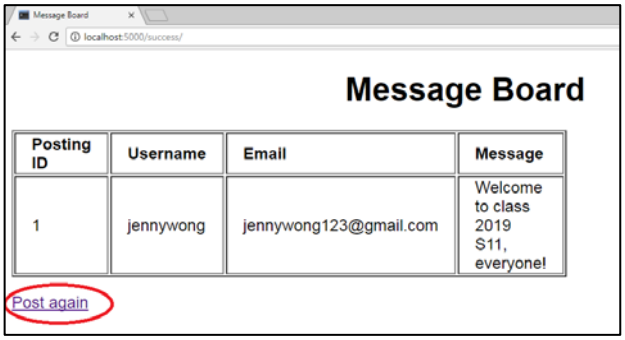


You should be redirected to a new webpage `http://localhost:5000/success/` with the entered data being shown.

2.2.2 Using url_for to Create A Dynamic URL

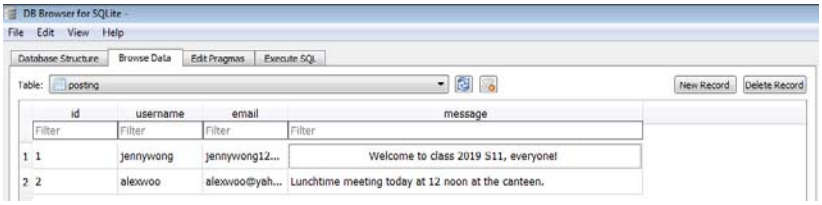
1. In **message_board.html**, add a link to allow the user to add a new posting.

The link should appear like this.



2.2.3 Add Records Using DB Browser for SQLite

1. Open SQLite3 editor such as **DB Browser for SQLite**.
2. Add more records to the **posting** table.

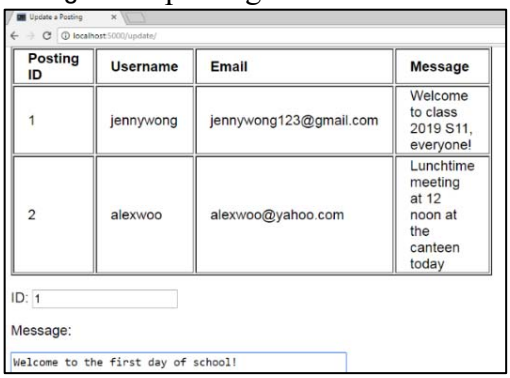


2.3 Update Record In Table

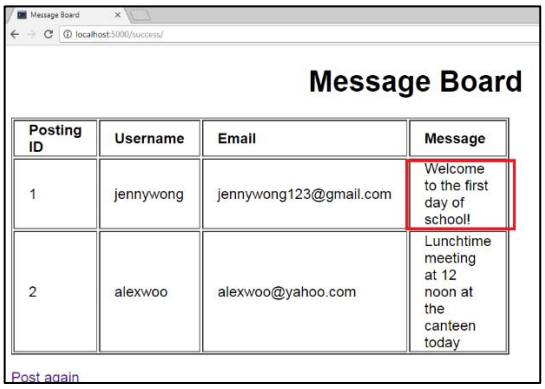
Now you will create a new webpage and add a new function to allow updating of the message field of a posting.

1. Open **app.py**.
2. Add a new function called `update` to perform an Update operation where the user is allowed to update the message.

The **update.html** is provided for you in **templates** subdirectory. The webpage will look like as shown below. It will display all fields of each record in the `posting` table and allow the user to key in the `id` and `message` for updating.



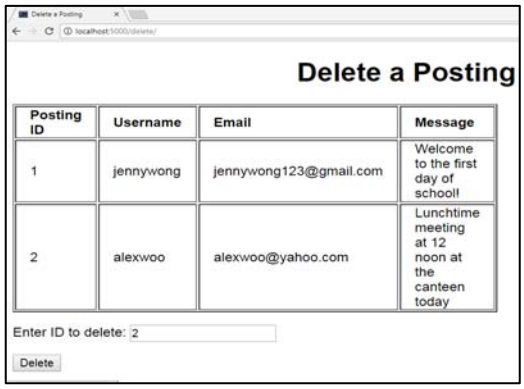
After clicking on **Update**, an `UPDATE` operation is performed on the selected record in the `posting` table. The user is redirected to a new webpage where the selected posting's message is reflected as updated.



2.4 Delete Record In Table

You will now create a function to delete a record based on the `id`.

1. Create a new template called **delete.html** and save it inside the templates directory. It should allow the user to enter the `ID` of the posting for deletion.



2. In **app.py**, add a function to delete a posting. If the deletion is successful, the user should be redirected to 'success'.

References:

- MOE Python Flask Web Application Starter Kit
- MOE Teacher Training 2018 - HTML and CSS
- Udemy: Python and Flask Bootcamp: Create Websites using Flask!