Lab 4: Databases

February 2023

In the practical section of this lab you will follow a tutorial on how to set up a database and connect it to a Flask application using SQL. In this case we will use an DBMS (Data Base Management System) called SQLite. To keep track of all the contents of the database you may use DB Browse for SQLite. Starting from the base template we have done previously (on a new LAB_4 folder).

- Once you have Flask installed, create a new directory for your project and within that directory, create a file called app.py and add the following code:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

This will create a basic Flask app that will display "Hello World!" when you run it.

Next, let's add SQLite to our project. SQLite is a lightweight, file-based database that doesn't require a separate server to run. It's a great choice for small projects like this one. You can install SQLite with pip:

```
pip install sqlite3
```

Now that we have SQLite installed, let's create a new database for our project. We'll create a file called db.sqlite in the root of our project. You can create this file using the SQLite command-line interface:

```
sqlite3 db.sqlite
```

This will open the SQLite command-line interface and you can start creating tables and inserting data.

To connect to our SQLite database from our Flask app, we'll use the sqlite3 module that we installed earlier. Add the following code to your app.py file:

```
import sqlite3

DATABASE = 'db.sqlite'

def connect_db():
    return sqlite3.connect(DATABASE)
```

This will create a function that connects to our db.sqlite file and returns a connection object.

Now, we can use this function to execute SQL statements in our app. Let's create a table for storing user information. Add the following code to your app.py file:

```
def init_db():
    with connect_db() as con:
        con.execute('CREATE TABLE users (id INTEGER
        PRIMARY KEY, name TEXT)')
```

This function will create a table called users with two columns: id and name. The id column is an integer primary key, which means it's unique and cannot be null.

You can initialize your table by running the init_db() function once.

Now that we have our database set up, let's create functions to perform CRUD (Create, Read, Update, and Delete) operations on our users.

To create a new user, we can execute an INSERT statement using the execute() method on the connection object.

```
def create_user(name):
    with connect_db() as con:
        con.execute("INSERT INTO users (name) VALUES
        (?)", (name,))
```

To query for all users, we can execute a SELECT statement and fetch all rows using the fetchall() method on the cursor object.

```
def get_users():
    with connect_db() as con:
    cursor = con.execute("SELECT * FROM users")
    return cursor.fetchall()
```

To query for a specific user by id, we can execute a SELECT statement with a WHERE clause and fetch the first row using the 'fetchone()' method on the cursor object.

```
def get_user(id):
    with connect_db() as con:
        cursor = con.execute("SELECT * FROM users
        WHERE id = ?", (id,))
        return cursor.fetchone()
```

To update a user, we can execute an UPDATE statement with a WHERE clause to specify the user to update.

```
def update_user(id, name):
    with connect_db() as con:
        con.execute("UPDATE users SET name = ? WHERE
        id = ?", (name, id))
```

To delete a user, we can execute a DELETE statement with a WHERE clause to specify the user to delete.

```
def delete_user(id):
    with connect_db() as con:
        con.execute("DELETE FROM users WHERE id = ?", (id,))
```

And that's it! You now have a fully functional SQLite database in your Flask app, and you can use the provided functions to perform CRUD operations on your users.

1 Optional

Another option is to use Flask-SQLAlchemy, with a ORM prespective, handling database migrations with Flask-Migrate:

- Step 1: Set up a Database

  Install a Postgres DBMS on your computer, and create a new database for your application. Make note of the database's connection details, such as the hostname, port, username, and password.

- Step 2: Install Flask-SQLAlchemy

  Open the terminal and activate your virtual environment by typing conda activate flask_environment. Install Flask-SQLAlchemy by typing pip install Flask-SQLAlchemy in the terminal. In your app.py file, import and initialize Flask-SQLAlchemy by adding the following lines of code at the top of the file:

```
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
```

```
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://username:password@hostname:port/databasename'
db = SQLAlchemy(app)
```

- Step 3: Create and Query Models

  In the app.py file, create a model for a table in your database by defining a class that inherits from db.Model. For example, to create a model for a table called "users":

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    email = db.Column(db.String(120), unique=True)

    def __init__(self, username, email):
        self.username = username
        self.email = email
```

  To create a new record in the "users" table, use the db.session.add() method:

```
user1 = User(username='John', email='john@example.com')
db.session.add(user1)
db.session.commit()
```

  To query the "users" table, use the query attribute of the model:

```
users = User.query.all()
```

- Step 4: Installing and Using DB Browser

  Download and install DB Browser for SQLite (https://sqlitebrowser.org/) or any other DB browser that supports your DBMS. Open DB Browser and connect to your database using the connection details you noted in step 1. Use DB Browser to create and modify tables, insert and query data, and perform other database tasks.

- Step 5: Handling Database Migrations with Flask-Migrate

  Install Flask-Migrate by typing pip install Flask-Migrate in the terminal. In your app.py file, import and initialize Flask-Migrate by adding the following lines of code at the top of the file:

```
from flask_migrate import Migrate
migrate = Migrate(app, db)
```

  In the terminal, run the following command to create a new migration script:

```
flask db init
```

To create the tables in the database, run the following command:

```
flask db migrate
```

To apply the changes to the database, run the following command:

```
flask db upgrade
```

To rollback the changes to the database, run the following command:

```
flask db downgrade
```

You should now have a basic understanding of how to set up a database and connect it to a Flask application using Flask-SQLAlchemy, as well as creating and querying models, handling database migrations with Flask-Migrate, routing and URL handling, request and response objects, and templates and rendering views. However, there are other aspects of database management such as performance, security and scalability that are not covered in this course. Additionally, in real-world scenarios, you will have to handle various other aspects such as security, validation, error handling, and so on.