Lab 5: Flask + SQL

GW CS 2541: Database Systems and Team Projects - 2024

Prof. Gabe Parmer, Kate Halushka, Sameen Ahmad, and Dania Abdalla

Connecting Python with your Database

Load the SQLite library

Open a connection to your database file

```
import sqlite3

connection = sqlite3.connect('/path/to/database.db')
cur = connection.cursor()

cur.execute("{{SQL STATEMENT}}")

connection.commit()
connection.close()
```

Create a **Cursor object** that allows you to execute queries!

Commit your changes and close the connection

Fetching Data

```
import sqlite3


connection = sqlite3.connect('student.db')
connection.row_factory = sqlite3.Row
cursor = connection.cursor()


cursor.execute("SELECT * FROM students")
data = cursor.fetchone()
print(data.keys())
     # ['name', 'id', 'email']
print(data['name'])
     # 'Kate Halushka'


connection.commit()
connection.close()
```

Fetching results returns row(s) as a list of tuples

- cursor.fetchall( ) → fetches all rows of a query result
- cursor.fetchmany(n) → fetches *n* rows of a query result
- cursor.fetchone( ) → fetches a single row

What if we want to fetch data into a dictionary?

- Assigning our connection with the row_factory() helper class makes our cursor return 'dictionary' rows instead of tuples!
- Column names can be treated as a dictionary

Fetching Lots of Data

```
import sqlite3


connection = sqlite3.connect('student.db')
connection.row_factory = sqlite3.Row
cursor = connection.cursor()


cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()


# Let's print all the rows that were returned
for row in rows:
    print(f"{row['name']}, {row['id']}, {row['email']}")



connection.commit()
connection.close()
```

How would we display our student info on the front end instead of printing to the console?

Inserting Data into the DB

```
import sqlite3

connection = sqlite3.connect(‘student.db’)
cursor = connection.cursor()

# Insert new student into the students table
Sameen_name = “Sameen Ahmad”
Sameen_id = “G00000000”
cursor.execute(“INSERT INTO students (name, id) VALUES (?,?)”, (sameen_name, sameen_id) )


 connection.commit()
 connection.close()
```

Why do you think we use (?) placeholders for input data when we interact with our db?

Whenever we want to make changes to the DB, we must **commit** our changes

If only providing one value, put a "," to ensure Python treats this as a tuple, eg `(ethan_name,)`

Updating Data in the DB

```
import sqlite3

connection = sqlite3.connect(‘student.db’)
cursor = connection.cursor()

# Update a student’s email
new_email = “new.email@yahoo.com”
sameen_id = “G00000000”
cursor.execute(“UPDATE students SET email = ? WHERE id = ?”, (new_email, sameen_id) )


 connection.commit()
 connection.close()
```

Python + SQL Exercise

- Let’s try out some queries with a simple student database…

Rebuilding the DB

Table details are in `create.sql`

To rebuild database in VSCode:

Right click in the file, and select

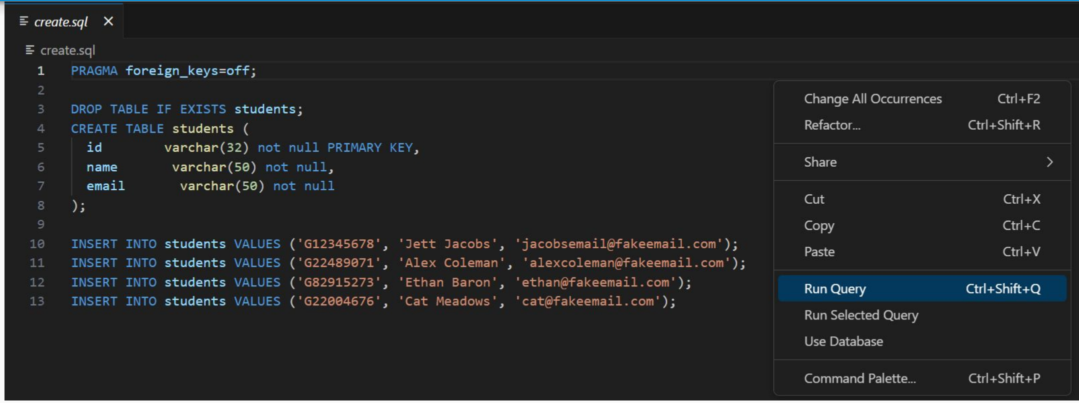
`*Run Query*`

Then, select your database

(This can be changed by selecting the `*Use Database option*` after a right click)

Alternatively, you can choose to use the command palette at the top of the screen (or using ctrl-shift-p) to run commands

To rebuild database outside VSCode:
Run `sqlite3 myDatabase.db ".read create.sql"`



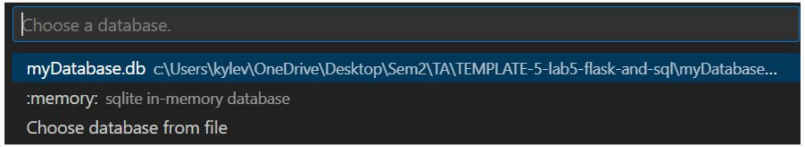
create.sql

```sql
PRAGMA foreign_keys=off;

DROP TABLE IF EXISTS students;
CREATE TABLE students (
  id        varchar(32) not null PRIMARY KEY,
  name       varchar(50) not null,
  email       varchar(50) not null
);

INSERT INTO students VALUES ('G12345678', 'Jett Jacobs', 'jacobsemail@fakeemail.com');
INSERT INTO students VALUES ('G22489071', 'Alex Coleman', 'alexcoleman@fakeemail.com');
INSERT INTO students VALUES ('G82915273', 'Ethan Baron', 'ethan@fakeemail.com');
INSERT INTO students VALUES ('G22004676', 'Cat Meadows', 'cat@fakeemail.com');
```

Change All Occurrences Ctrl+F2
Refactor... Ctrl+Shift+R
Share
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Run Query Ctrl+Shift+Q
Run Selected Query
Use Database
Command Palette... Ctrl+Shift+P



Choose a database.
myDatabase.db c:\Users\kylev\OneDrive\Desktop\Sem2\TA\TEMPLATE-5-lab5-flask-and-sql\myDatabase...
:memory: sqlite in-memory database
Choose database from file

Activity 1

Retrieve a list of student information from the sqlite database and print to a route (‘/’) using a for loop in a flask template

You can structure the template however you like, just make sure it prints ALL the information from the database.

What information will you need to pass to the template?

If you need to verify, you can always run a query in Python!

How can I take in User Input?

- Data is exchanged from client side to server side using **post requests**
- Data can be accessed by variables sent from a **form**

```
from flask import Flask, render_template, request
app = Flask('app')
@app.route('/', methods=['GET', 'POST'])
 def print_name():
     if request.method == 'POST':
          print (request.form["field_name"])
     return render_template('simple_form.html')
 app.run(host='0.0.0.0', port=8080)
```

```
<body>
 <form action="/" method="POST">
   <input type="text" name="field_name" ><br>
   <input type="submit" name="submit">
 </form>
</body>
```

Forms

```
from flask import Flask, render_template, request

app = Flask(‘app’)

@app.route(‘/’, methods=[‘GET’, ‘POST’])
 def print_name():
     if request.method == ‘POST’:
         name = request.form[“field_name”])
         print(name)
     return render_template(‘simple_form.html’)

 app.run(host=’0.0.0.0’, port=8080)
```

```
<html>
<head>
<title> My Form </title>
</head>
<body>
 <form action="/" method="POST">
   <input type="text" name="field_name" ><br>
   <input type="submit" name="submit">
 </form>
</body>
</html>
```

Use the **form** attribute to **post** input data to our Flask server

Specify which route to post data to using “action”

Activity 2

1. Extend Activity 1 to create a new route (‘/addstudent’) that displays a simple form for “registering” a new student for the class.
    a. This form should take in a name, email, and ID for a new student and insert to the database

1. Once you submit the form, you should be able to verify that it worked by going back to the default (‘/’) route to see the new student being displayed