Lab 5: Flask + SQL

GW CS 2541: Database Systems and Team Projects - 2026
Prof. Gabe Parmer, Emil Abbasov, Bella Dayrit, Sydney Berritt, Max Eichholz, Aditya Arjun

SQLite Command Line

You should often run sqlite using the command line:

```
$ sqlite3 database.db < main.sql
```

Your columns will need to **exactly** match the specifications (e.g. name). Test with:

```
$ sqlite3 database.db < main.sql > output
$ diff output README
```

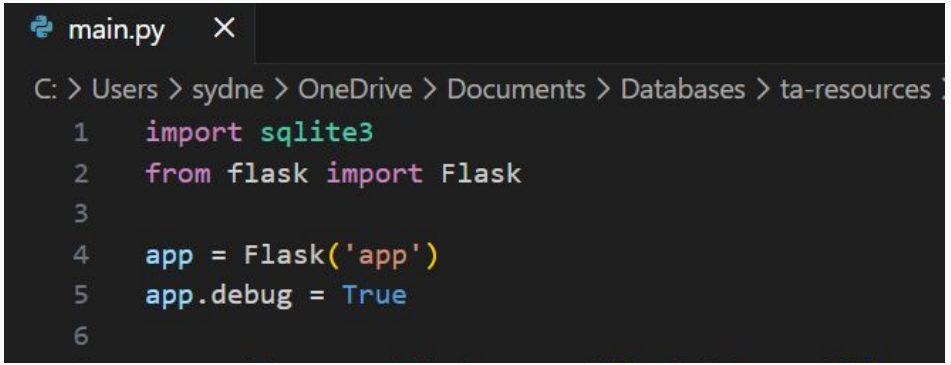
You should make sure that the columns (names, spacing, etc…) are **exactly** the same as those under the corresponding task number in the README file.

Or, you may create a temporary file “expected_output.txt” and copy paste the line under each corresponding task from the README file, and then just do a **`diff`** with that file:

```
$ sqlite3 database.db < main.sql > output
$ diff output expected_output.txt
```

Debugging Tip – Always Do This When Dev!

- If there is an error/bug in your code you'll see an "Internal Server Error" message when you try to run your application
- Set app.debug = True
  - Will provide you with detailed error messages to help with debugging your code



Connecting Python with your Database

Load the SQLite library

Open a connection to your database file

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

```
import sqlite3

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

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

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

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

Are there any issues from this structure?

Better way to do this? (recall: what does LIMIT do?)

```
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()
```

Inserting Data into the DB

```
import sqlite3

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

# Insert new student into the students table
name = "Sameen Ahmad"
id = "G00000000"
cursor.execute("INSERT INTO students (name, id) VALUES (?,?)", (name, 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 `(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"
id = "G00000000"
cursor.execute("UPDATE students SET email = ? WHERE id = ?", (new_email, id) )


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

Rebuilding the DB

Table details are in `create.sql`

To rebuild database in VSCode:

Right click in the file, and select

`Run Query`

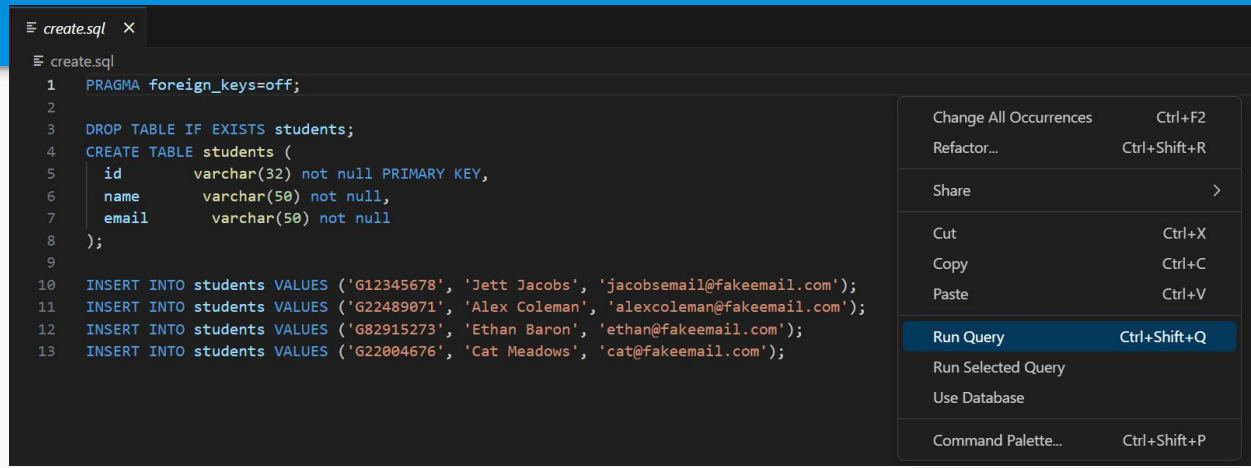
Then, select your database

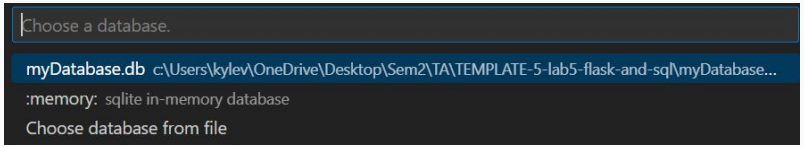
(This can be changed by selecting the *`Use Database option`* after a right click)

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"`





Activity 1

See the README.md in the classroom link which focuses on integrating SQL queries with a flask route.

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)
```

Verify the request is **POST** before accessing the input data

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

```
<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>
```

Specify which route to post data to using "action"

Activity 2

See the README.md in the classroom link to extend Activity 1 to include a form.