Web Applications

Back-End : SQL Database

Overview

	Web Applications
Front-End	Part 1a : Basic HTML Part 1a : Basic CSS Part 2 : HTML - Forms
Back-End	Part 3 : Form with Flask Part 4 : SQLite with Flask

Part 4 : SQL with Flask

Managing with database

Part 4 : SQL with Flask

4.1	Recap SQLite
4.2	SQLite with Python Query with Python
4.3	SQLite with Flask

4.1 Recap SQLite

SQLite

- Create Database and Table
- Insert a single row and multiple rows of data into table
- View all the data and some fields in the table
- View particular row(s) in the table
- Aggregate Functions and others
- Change the data in the table
- Delete a row from the table
- Grouping some data
- Combining multiple queries
- Connect multiple tables together
- Nested Query

Create Datatbase and Table

- Create `airline.db`

```
CREATE DATABASE airline.db
```

- Create a `flights` table in `airline.db`

```
CREATE TABLE flights (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        origin VARCHAR(20) NOT NULL,
        destination VARCHAR(20) NOT NULL,
        duration INTEGER NOT NULL
);
```

Insert one row of data into table

```
INSERT INTO flights (origin, destination,
duration) VALUES ('New York', 'London', 415);
```

id	origin	destination	duration
1	New York	London	415

How to insert the remaining rows of data into table?

id	origin	destination	duration
1	New York	London	415
2	Shanghai	Paris	760
3	Istanbul	Tokyo	700
4	New York	Paris	435
5	Moscow	Paris	245
6	Lima	New York	455

Insert multiple rows of data into table

```
INSERT INTO flights
      (origin, destination, duration)
VALUES
      ('Shanghai', 'Paris', 760),
      ('Istanbul', 'Tokyo', 700),
      ('New York', 'Paris', 435),
      ('Moscow', 'Paris', 245),
      ('Lima', 'New York', 455);
```

View data in the table

- View all the data in the table

```
SELECT * FROM flights;
```

- View some fields in the table

```
SELECT origin, destination FROM flights;
```

View data in the table

- View particular row(s) in the table

```
SELECT * FROM flights WHERE id = 3;

SELECT * FROM flights WHERE origin = 'New York';

SELECT * FROM flights WHERE duration > 500;

SELECT * FROM flights WHERE destination = 'Paris'
AND duration > 500;

SELECT * FROM flights WHERE destination = 'Paris'
OR duration > 500;
```

Aggregate Functions

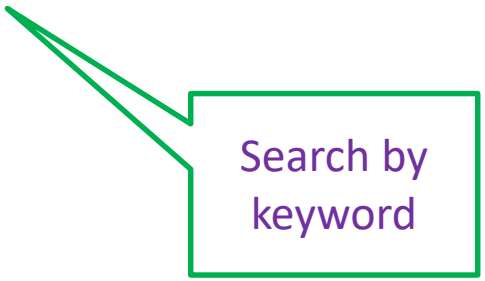
- AVG, COUNT, SUM, MAX, MIN

```
SELECT AVG(duration) FROM flights;

SELECT AVG(duration) FROM flights WHERE
origin = 'New York';

SELECT COUNT(*) FROM flights;
```

Other commands

- IN and LIKE

```
SELECT * FROM flights WHERE origin IN ('New
York', 'Lima');

SELECT * FROM flights WHERE origin LIKE
'%sha%';
```



Change data in the table

- Update info for some rows of data

```
UPDATE flights
      SET duration = 430
      WHERE origin = 'New York'
      AND destination = 'London';
```

- Delete a row from the table

```
DELETE FROM flights
      WHERE destination = 'Tokyo';
```

We have delete the row with `id` = 3, what if we add a new row now, what will the `id` be?

Is it 3 or 7?

```
INSERT INTO flights
      (origin, destination, duration)
VALUES ('Istanbul', 'Tokyo', 700);
```

Why do you think this value is assigned as the `id` for the new row ?

Query

- Sorting the data

```
SELECT * FROM flights LIMIT 3;

SELECT * FROM flights ORDER BY duration ASC
LIMIT 3;

SELECT * FROM flights ORDER BY duration DESC
LIMIT 3;
```

Query

- Group some data

```
SELECT origin, COUNT(*) FROM flights
GROUP BY origin;

SELECT origin, COUNT(*) FROM flights
GROUP BY origin HAVING COUNT(*) > 1;
```

Connecting multiple tables

- Create the `passengers` table

```
CREATE TABLE passengers (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name VARCHAR(20) NOT NULL,
      flight_id INTEGER REFERENCES flights(id)
);
```

Insert multiple rows of data into table

id	name	flight_id
1	Alice	1
2	Bob	1
3	Charlie	2
4	Dave	2
5	Erin	4
6	Frank	6
7	Grace	6

Insert multiple rows of data into table

```
INSERT INTO passengers (name, flight_id)
VALUES
      ('Alice', 1),
      ('Bob', 1),
      ('Charlie', 2),
      ('Dave', 2),
      ('Erin', 4),
      ('Frank', 6),
      ('Grace', 6);
```

Combining multiple queries

```
SELECT * FROM passengers WHERE name = 'Alice';
SELECT * FROM flights WHERE id = 1;
```

we can combine them into :

```
SELECT origin, destination, duration FROM
passengers p, flights f WHERE
p.flight_id = f.id AND p.name = 'Alice';
```

* not covering JOIN, INNER/OUTER JOIN, LEFT/RIGHT JOIN

Nested Query

```
SELECT flight_id FROM passengers GROUP BY
flight_id HAVING COUNT(*) > 1;
```

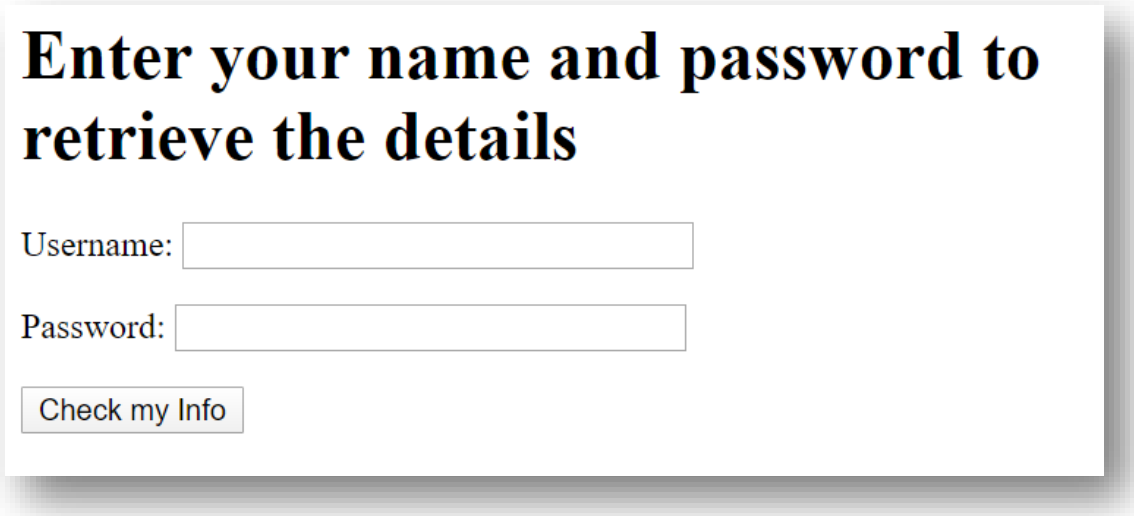
To select all the flights with more than 1 passenger:

```
SELECT * FROM flights WHERE id IN
(SELECT flight_id FROM passengers GROUP BY
flight_id HAVING COUNT(*) > 1);
```

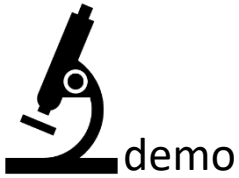
Issue #1 : SQL Injection

Issue#1 : SQL Injection

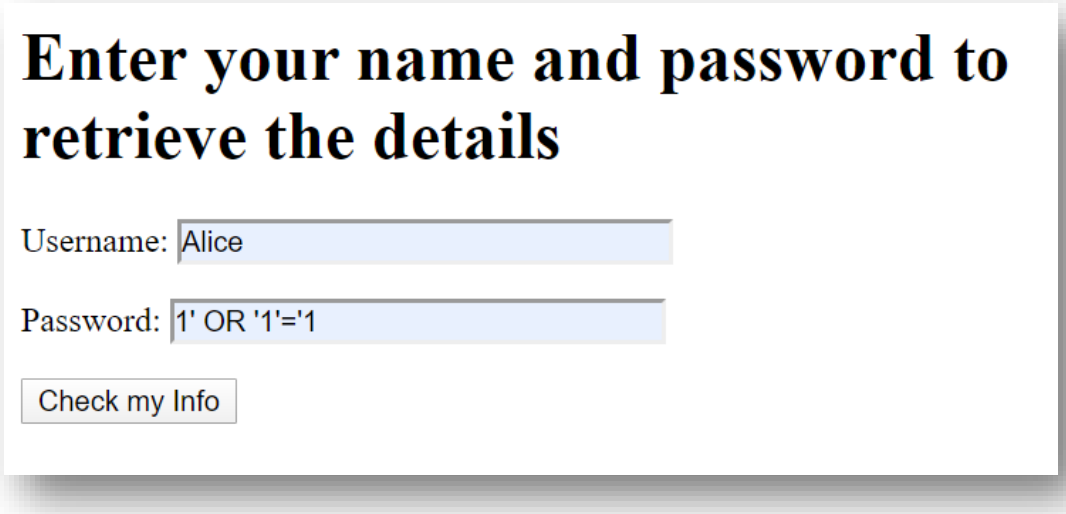
SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.



Hacker #1




```
txtPassword = '1' OR '1'='1'

SELECT * FROM Some_Table WHERE name='Alice'
      AND (password = '1' OR '1'='1')
```



Hacker #2

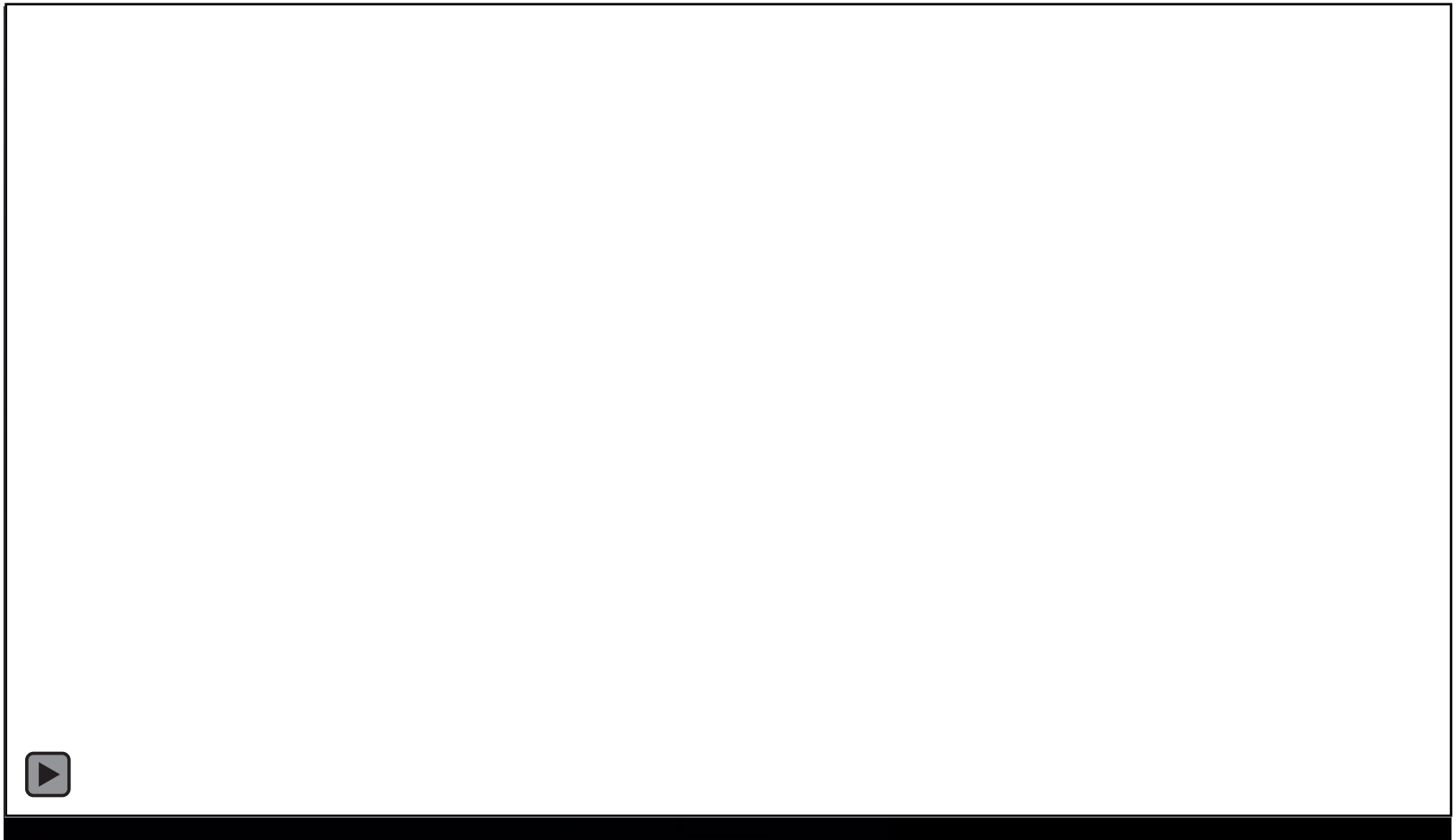

```
txtUsername = ' '; DROP TABLE Some_Table; COMMIT; '

SELECT * FROM Some_Table WHERE name=' ';
      DROP TABLE Some_Table; COMMIT;
      ' AND (password = ' ')
```

SYNTAX ERROR

Issue #2 : Race Condition

Issue#2 : Race Condition

A race condition may occur if two users attempt to access the same database at the same instant, and neither computer receives notification that the database is occupied.



4.2 SQLite with Python

A Summary

Creating/Opening a database file

```
import sqlite3
db = sqlite3.connect("example.db")
```

Opens SQLite file named `example.db` or automatically creates it if it doesn't exist

```
c = db.cursor()
c.execute('''SELECT * FROM table''')
```

Store the return value of `execute()` in variable named `cursor c`

Fetching a result

```
c.execute('''SELECT * FROM table WHERE id=0''')
item = c.fetchone()
print(item)
```

Ask `cursor c` to fetch just one result and store it in the variable named `item`

Fetching many results

```
c.execute('''SELECT * FROM table''')

items = c.fetchall()

for item in items:
    print(item)
```

Ask `cursor c` to fetch all the results and store them in the variable named `items`

Insert one row of data into table

```
INSERT INTO flights (origin, destination,
duration) VALUES ('New York', 'London', 415);
```

id	origin	destination	duration
1	New York	London	415

Inserting a data into the table using a Tuple

```
import sqlite3

db = sqlite3.connect("example.db")


c = db.cursor()

c.execute('''INSERT INTO some_table (field1, field2)
             VALUES(?, ?)''', (x, y))
```

Important: If ANY part of the SQL comes from user input, then it cannot be trusted. Put any untrusted data in a tuple and use this "?" syntax to safely include them in the SQL command.

Inserting a data into the table using a Dictionary

```
import sqlite3

db = sqlite3.connect("example.db")

c = db.cursor()

c.execute('''INSERT INTO some_table (field1, field2)
             VALUES(:a, :b)''',
             {"a":'first data', "b":'second data'})
```

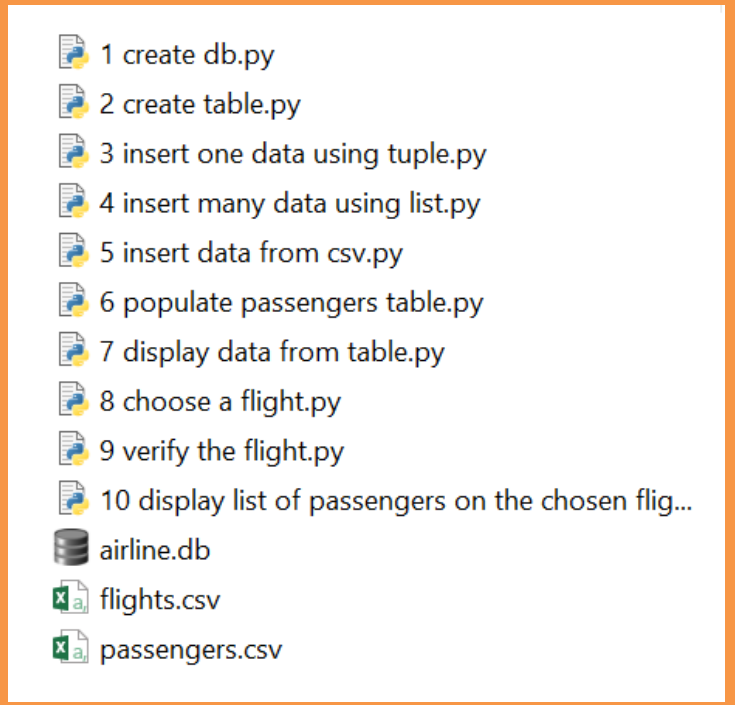
Important: Watch out on the syntax when using a dictionary to insert a data

Saving changes

```
import sqlite3
db = sqlite3.connect("example.db")

c = db.cursor()
c.execute('''INSERT INTO some_table (field1,
          field2) VALUES(?, ?)''', (x, y))
db.commit()

db.close()
```

For INSERT, UPDATE and DELETE, the changes are not actually saved until you **commit** the changes using `commit()`.

We should always close the file by calling `close()`; however, `close()` itself doesn't commit any of the changes!



Learn with an Example

SQLite with Python

- Create database
- Create table
- Insert data into table
  - Using parameter
  - Using dictionary
  - Using a list
  - Import from csv file
- List the data in the table

id	origin	destination	duration
1	New York	London	415
2	Shanghai	Paris	760
3	Istanbul	Tokyo	700
4	New York	Paris	435
5	Moscow	Paris	245
6	Lima	New York	455

Other new data provided

Remember to use DB Browser to check your database/table(s).

1 create db.py

Create database

```
import sqlite3
db = sqlite3.connect('airline.db')
```

Opens SQLite database file named `airline.db` or automatically creates it if it doesn't exist

```
db.close()
```

Close the database file

2 create table.py

Create table

```
import sqlite3

db = sqlite3.connect('airline.db')

try:
    c = db.cursor()
    c.execute('''CREATE TABLE flights (\
            id INTEGER PRIMARY KEY AUTOINCREMENT,\
            origin VARCHAR(20) NOT NULL,\
            destination VARCHAR(20) NOT NULL,\
            duration INTEGER NOT NULL);''')
except:
    print('Table already exist, cannot create.')

db.commit()
db.close()
```

Save the change

3a insert one data using tuple.py

Insert one data into the table using Tuple

```
import sqlite3

db = sqlite3.connect('airline.db')

c = db.cursor()
c.execute('''INSERT INTO flights(origin,
                destination, duration) \
          VALUES(?,?,?)''',
                ('New York', 'London', 415))

db.commit()
db.close()
```

Important: If ANY part of the SQL comes from user input, then it cannot be trusted. Put any untrusted data in a tuple and use this "?" syntax to safely include them in the SQL command.

3b insert one data using dictionary.py

Insert another row of data

```
import sqlite3
db = sqlite3.connect('airline.db')
c = db.cursor()

c.execute('''INSERT INTO flights\
    (origin, destination, duration) \
    VALUES(:o, :dest, :dur)''',\
    {'o':'Shanghai', 'dest':'Paris', 'dur':760})

db.commit()
db.close()
```

Another way: Using dictionary to insert user supplied data.

4 insert many data using list.py

Insert many rows of data

```
datalist = [('Istanbul', 'Tokyo', 700),
            ('New York', 'Paris', 435),
            ('Moscow', 'Paris', 245),
            ('Lima', 'New York', 455)]


for data in datalist:
    c.execute('''INSERT INTO flights(origin,
                 destination, duration) \
    VALUES(:origin, :destination, :duration)''',
                 data)
```

Insert data from csv file

5 insert data from csv.py

```
import sqlite3
import csv
db = sqlite3.connect('airline.db')
c = db.cursor()

f = open("flights.csv")
reader = csv.reader(f)
for o, dest, dur in reader:
    db.execute('''INSERT INTO flights \
        (origin, destination, duration) \
        VALUES (:origin, :destination,
                :duration)''',
        {"origin":o, "destination":dest,
            "duration":dur})

db.commit()
db.close()
```

6 populate passengers table.py

Create table and insert data

Create the `passengers` table in `airline.db` using the following SQL command :

```
CREATE TABLE passengers (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(10) NOT NULL,
    flight_id INTEGER NOT NULL REFERENCES flights(id));
```

Insert the data from `passengers.csv` into the table

Query with Python

- List all the flights
- Prompt the user to choose a flight
- Verify chosen flight is valid
- Display list of passengers on the chosen flight

7 display data from table.py

List all the flights

```
import sqlite3
db = sqlite3.connect('airline.db')
c = db.cursor()
c.execute('''SELECT origin, destination,
duration FROM flights''')

flights = c.fetchall()

for flight in flights:
    print(flight[0], 'to', flight[1], ',',
flight[2], 'minutes.')

db.close()
```

8 choose a flight.py

Prompt user to choose a flight

```
# Prompt user to choose a flight.
flight_id = int(input("\nChoose a Flight ID: "))

import sqlite3
db = sqlite3.connect('airline.db')
c = db.cursor()

c.execute('''SELECT origin, destination,
          duration FROM flights
          WHERE id = :id''', {"id": flight_id})

flight = c.fetchone()
print(flight)
```

9 verify the flight.py

Verify the flight is valid

```
# Make sure flight is valid.

if flight == None:
    print("Error: No such flight.")
    #return None
else:
    print(flight)
```

10 display list of passengers on the chosen flight.py

Display list of passengers on chosen flight

```
# Make sure flight is valid.
if flight == None:
    print("Error: No such flight.")
    #return None
else:
    #print(flight)
    # List passengers.
    c.execute('''SELECT name FROM passengers
                 WHERE flight_id = :flight_id''',
                      {"flight_id": flight_id})
    passengers = c.fetchall()
    print(passengers)
```

Display list of passengers on chosen flight – what if no passenger?

```
passengers = c.fetchall()

if len(passengers) == 0:
    print("No passenger.")
else:
    print("\nPassengers:")

    for passenger in passengers:
        print(passenger[0])
```

4.3 SQLite with Flask

SQLite with Flask

- Booking a flight
- Display Flight Info

#1 Display a list of available flights

server.py

```
from flask import Flask, render_template,
request
app = Flask(__name__)

import sqlite3
db = sqlite3.connect('airline.db')

@app.route("/")
def index():
    db = sqlite3.connect('airline.db')
    c = db.cursor()
    c.execute('''SELECT id, origin, destination
                 FROM flights''')
    flights = c.fetchall()
    return render_template("index.html",
                           flights=flights)
db.close()
app.run(debug=True, port=5000)
```

#1 Display a list of available flights

layout.html

```
<!DOCTYPE html>
<html>

<head><title>
                {% block title %}
                {% endblock %}
</title></head>

<body>
    {% block body %}
    {% endblock %}
</body>

</html>
```

#1 Display a list of available flights

Index.html

```
{% extends "layout.html" %}

{% block title %} Flights {% endblock %}

{% block body %} <h1> Available Flights </h1>
<ul>
{% for flight in flights %}
    <li>
    {{flight[0]}}.{{flight[1]}} to {{flight[2]}}
    </li>
{% endfor %}
</ul>
{% endblock %}
```

#2a Select a flight

Index.html

```
{% extends "layout.html" %}

{% block title %} Flights {% endblock %}

{% block body %} <h1> Book a Flight 2</h1>
<form action={{url_for('book')}} method="POST">
<select name="flight_id">
    {% for flight in flights %}
    <option value="{{ flight[0] }}">
    {{flight[0]}}.{{flight[1]}} to {{flight[2]}}
    </option>
    {% endfor %}
</select> </form>
{% endblock %}
```

#2b Capture user name

index.html

```
{% block body %}
...

<br><input type="text" name="name"
placeholder="Passenger Name">

<br><button type="submit">Book Flight</button>

</form>
{% endblock %}
```

#2 Select a flight and capture user name

server.py

```
...

@app.route("/book", methods=["POST"])
def book():

    # Get form information.
    username = request.form.get("name")
    user_flight_id = int(request.form.get(
                                "flight_id"))

    return username + user_flight_id
```

#3 Store flight and name into database

server.py

```
... #return username + user_flight_id
    # Assume valid username provided
    db = sqlite3.connect('airline.db')
    c = db.cursor()
    c.execute('''INSERT INTO passengers
                 (name, flight_id)
                 VALUES (:name, :flight_id)''',
                 {"name": username,
                      "flight_id":
                                  user_flight_id})
    db.commit()

    return render_template("success.html",
    message="You have successfully booked your
                                       flight.")
db.close()
```

#3 Display success

success.html

```
{% extends "layout.html" %}

{% block title %} Success! {% endblock %}

{% block body %}  <h1> Success! </h1>

{{message}}

{% endblock %}
```

#4 Check for flights info

server.py

```
@app.route("/check")
def check():
    db = sqlite3.connect('airline.db')
    c = db.cursor()
    c.execute('''SELECT id, origin,
                 destination FROM flights''')
    flights = c.fetchall()
    return render_template("flights.html",
                           all_data=flights)
```

#4 Display a list of available flights

flights.html

```
{% extends "layout.html" %}

{% block title %} Flights Info{% endblock %}

{% block body %} <h1> All Flights </h1>

<ul>
{% for data in all_data %}
    <li> {{data[0]}}, {{data[1]}}, {{data[2]}} </li>
{% endfor %}
</ul>

{% endblock %}
```

#5 Hyperlink for detailed flight info

flights.html

```
{% block body %} <h1> All Flights </h1>

<ul>
{% for data in all_data %}
<a href=
"{{ url_for('check1', flight_id=data[0]) }}">
<li> {{ data[1] }} to {{ data[2] }} </li>
</a>
{% endfor %}
</ul>

{% endblock %}
```

#5 Check for detailed flight info

server.py

```
@app.route("/check/<int:flight_id>")
def check1(flight_id):
    return str(flight_id)
```

#5a Retrieve flight info

server.py

```
@app.route("/check/<int:flight_id>")
def check1(flight_id):
    db = sqlite3.connect('airline.db')
    c = db.cursor()

    c.execute('''SELECT * FROM flights
                 WHERE id = :id''',
                 {"id": flight_id})
    flight = c.fetchone()

    return render_template("flight.html",
                           flight=flight)
```

#5a Display flight info

flight.html

```
{% extends "layout.html" %}

{% block title %} Flight Details{% endblock %}

{% block body %} <h1> Flight Details </h1>

<ul>
    <li>Origin:  {{ flight[1] }}</li>
    <li>Destination:  {{ flight[2] }}</li>
    <li>Duration:  {{ flight[3]  }}minutes</li>
</ul>
{% endblock %}
```

#5b Retrieve passenger info

server.py

```
@app.route("/check/<int:flight_id>")
def check1(flight_id):
    db = sqlite3.connect('airline.db')
    c = db.cursor()
    c.execute('''SELECT * FROM flights
         WHERE id = :id''', {"id": flight_id})
    flight = c.fetchone()
    c.execute('''SELECT name FROM passengers
              WHERE flight_id = :flight_id''',
              {"flight_id": flight_id})
    passengers = c.fetchall()
    return render_template("flight.html",
         flight=flight, passengers=passengers)
```

#5b Display passengers info

flight.html

```
{% block body %} <h1> Flight Details </h1>
<ul>
    ...
</ul>
          <h2>Passengers</h2>
<ul>
    {% for passenger in passengers %}
        <li>{{ passenger[0] }}</li>
    {% else %}
        <li>No passenger.</li>
    {% endfor %}
</ul>
{% endblock %}
```

Assignment

Web Application (Part 4)

Assignment

Design a Web Application "**NowPay**" include but not limiting to the following:

- A database table to record all the items purchased
- A database table for the bank account balance
- The front and back-end transaction to deduct from the bank account for purchasing the items.

Files to submit :

- `Database files`
- `server.py`
- `all html files in templates folders`

Assignment – a suggestion

- A website contain a list of items for user to select for purchase;
- After user's selection, the total amount for the items is computed and displayed;
- Prompt the user to select a payment mode, eg using "**NowPay**" ;
- The system will check to ensure sufficient balance in the user's bank account for the purchase;
- Deduct the purchase amount and update the account balance;
- Display the account balance after the purchase

The End

Web Application (Part 4)