TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT HÀ NỘI

=======

BÁO CÁO HỌC THUẬT

Tìm hiểu thư viện Flask của Python

NGUYỄN THỊ HẢI YẾN

Hà Nội, T6/2023

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU FLASK FRAMEWORK

1.1 Tổng quan về flask framework

Flask là một web frameworks, nó thuộc loại micro-framework được xây dựng bằng ngôn ngữ lập trình Python. Flask cho phép ta xây dựng các ứng dụng web từ đơn giản tới phức tạp. Nó có thể xây dựng các api nhỏ, ứng dụng web chẳng hạn như các trang web, blog, trang wiki hoặc một website dựa theo thời gian hay thậm chí là một trang web thương mại. Flask cung cấp cho ta công cụ, các thư viện và các công nghệ hỗ trợ ta làm những công việc trên.

Flask là một micro-framework. Điều này có nghĩa Flask là một môi trường độc lập, ít sử dụng các thư viện khác bên ngoài. Do vậy, Flask có ưu điểm là nhẹ, có rất ít lỗi do ít bị phụ thuộc cũng như dễ dàng phát hiện và xử lý các lỗi bảo mật.

1.2 Cài đặt flask

1.2.1 Cài đặt Python

Nếu máy của ta không có sẵn Python, ta cần cài đặt nó. Tùy theo hệ điều hành ta đang sử dụng, ta sẽ dùng những trình cài đặt khác nhau (nếu là Windows, ta có thể download trực tiếp trình cài đặt từ trang chủ của Python – https://www.python.org/downloads/. Nếu là Mac OS X, ta cần dùng homebrew, nếu là Linux, ta phải dùng trình quản lý gói thích hợp tùy theo bản phân phối ta đang sử dụng – apt trên Ubuntu hoặc Debian, dnf trên Fedora, yum trên RHEL hoặc CentOS …). Tuy nhiên, dù dùng hệ điều hành nào đi nữa, hãy chắc rằng ta cài đặt Python 3 vì Python 2 sẽ kết thúc vòng đời và không được hỗ trợ kể từ tháng 1 năm 2020.

Để kiểm tra xem Python đã được cài đặt và hoạt động tốt trên máy của ta hay không, ta có thể dùng cửa sổ lệnh (Command Prompt hoặc Windows PowerShell trên Window, terminal trên Mac OS X hoặc Linux) và dùng lệnh python3 (nếu lệnh này không hoạt động, ta có thể thử chỉ dùng python).

Bước tiếp theo là cài đặt Flask, có một vài cách khác nhau để làm điều này. Tuy nhiên, chúng ta sẽ tìm hiểu cách tốt nhất để cài đặt các gói thư viện (package) mà Flask là một trong số chúng.

Trong Python, ta có thể tìm các gói thư viện như Flask từ các kho lưu trữ công cộng (public repository). Các kho lưu trữ này cho phép bất cứ người sử dụng nào có thể truy cập và tải về các gói thư viện Python. Kho lưu trữ công cộng chính thức của Python gọi là PyPI

(viết tắt của Python Package Index). Việc cài đặt các gói từ PyPI rất đơn giản với một công cụ có sẵn trong trình cài đặt Pyhon gọi là pip (pip được tích hợp sẵn trong các phiên bản Python 3, trong Python 2, chúng ta cần cài pip riêng. Tuy vậy, vì chúng ta đang dùng Python3, việc cài đặt pip là không cần thiết).

```
$ pip3 install <tên-gói>
```

Tuy nhiên trong thực tế, việc sử dụng lệnh pip như trên không phổ biến vì một số lý do. Nếu trình thông dịch Python được cài đặt trên máy của ta ở mức độ toàn phần (global) để mọi người sử dụng (user) đều có thể sử dụng được, có khả năng rất lớn là tài khoản mà ta dùng để đăng nhập (login) không có đủ thẩm quyền để sao chép các file cần thiết trong quá trình cài đặt vào đúng vị trí của chúng (dĩ nhiên đều này không luôn luôn đúng – đặc biệt nếu ta có quyền Admin với máy của ta. Nhưng đây là nguyên tắc chung được áp dụng cho bất kỳ hệ thống nào tại các môi trường làm việc chung – công ty, tổ chức …). Và cho dù là ta có thể chạy lệnh pip để cài đặt gói ở mức độ toàn phần, vẫn còn một rắc rối khác: nếu ta cài đặt một gói thành công ở mức toàn phần, pip sẽ tải gói được yêu cầu từ PyPI và cài đặt vào trình Python có sẵn trong máy của ta. Từ thời điểm này, tất cả các chương trình Python trên hệ thống của ta sẽ sử dụng gói này. Giả sử như trước đó ta sử dụng một phiên bản cũ của gói (ví dụ như Flask phiên bản 0.11) để viết một ứng dụng. Tại thời điểm ta viết ứng dụng đó, Flask 0.11 là phiên bản mới nhất. Nhưng về sau, ta cần viết một ứng dụng khác, và lúc đó, đã có một phiên bản Flask mới hơn, ví dụ như 0.12. Nếu ta dùng pip để cài đặt Flask 0.12 ở mức độ toàn phần, phiên bản này sẽ thay thế hoàn toàn phiên bản 0.11 trong hệ thống của ta. Điều gì sẽ xảy ra với ứng dụng cũ sử dụng Flask 0.11? Trong một số trường hợp nhất định, nó có thể sẽ không hoạt động nữa vì nó không tương thích với Flask phiên bản 0.12. Vì vậy, cách tốt nhất là tìm ra cách để cài đặt cả hai phiên bản Flask trong hệ thống của ta, một cho ứng dụng cũ và một cho ứng dụng mới.

1.2.2 Cài đặt virtual env

Để giải quyết vấn đề cài đặt song song nhiều phiên bản của cùng một gói thư viện cho các ứng dụng khác nhau, Python đưa ra khái niệm môi trường ảo (virtual environment). Một môi trường ảo là một bản sao hoàn chỉnh của trình thông dịch Python. Khi ta cài đặt một gói thư viện trong một môi trường ảo, trình thông dịch Python đã được cài đặt cho toàn hệ thống (hay ở mức toàn phần) không bị ảnh hưởng, gói mới được cài đặt chỉ có tác dụng trong phạm vi của môi trường ảo mà thôi. Vì vậy, chúng ta có thể tự do cài đặt các gói cần thiết trong phạm vi của môi trường ảo mà không sợ rằng chúng sẽ ảnh hưởng đến các ứng dụng trong các môi trường ảo khác. Ngoài ra, mỗi một môi trường ảo hoàn toàn thuộc về

người tạo ra nó, hay nói cách khác người tạo ra một môi trường ảo có toàn quyền thêm bớt các gói trong đó mà không cần đến quyền của quản trị hệ thống (Administrator).

Chúng ta có thể bắt đầu bằng cách tạo ra thư mục (directory) cho dự án nhỏ của chúng ta. Chúng ta có thể đặt tên cho nó là thư mục này First_Flask. Sau đó để tạo một virtual evn project, ta gõ lệnh "virtualenv tên_project" vào command line:

```
C:\Users\nhlon>virtualenv First_Flask
created virtual environment CPython3.8.2.final.0-64 in 770ms
  creator CPython3Windows(dest=C:\Users\nhlon\First_Flask, clear=False, global=False)
  seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, via=copy, app_data_dir=C:\Users\nhlon\
AppData\Local\pypa\virtualenv\seed-app-data\v1.0.1)
  activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator

C:\Users\nhlon>
```

Khởi động Virtual Environment bằng câu lệnh (phải vào trong thư mục chưa file activate – thông thướng nắm trong thư mục con Scripts): activate

```
C:\Users\nhlon\First_Flask\Scripts>activate

(First_Flask) C:\Users\nhlon\First_Flask\Scripts>
(First_Flask) C:\Users\nhlon\First_Flask\Scripts>
(First_Flask) C:\Users\nhlon\First_Flask\Scripts>
(First_Flask) C:\Users\nhlon\First_Flask\Scripts>
(First_Flask) C:\Users\nhlon\First_Flask\Scripts>pip install Flask
```

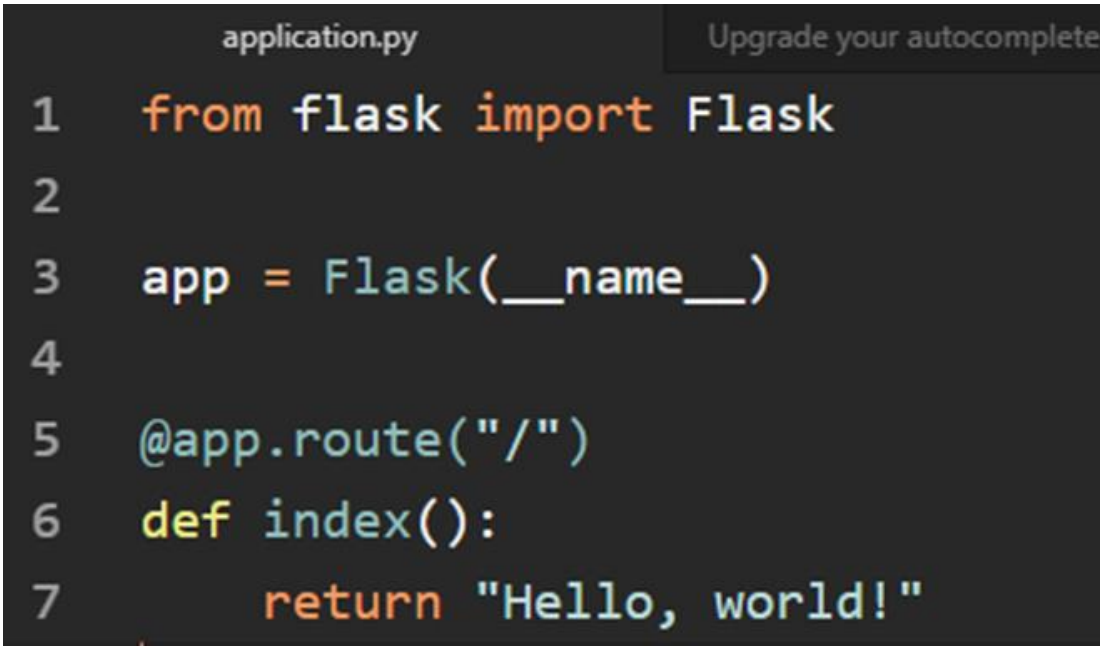
1.2.3 Cài đặt flask

Chạy câu lệnh pip install flask

```
(First_Flask) C:\Users\nhlon\First_Flask\Scripts>pip install flask
Collecting flask
  Using cached Flask-1.1.1-py2.py3-none-any.whl (94 kB)
Collecting Jinja2>=2.10.1
  Using cached Jinja2-2.11.1-py2.py3-none-any.whl (126 kB)
Collecting itsdangerous>=0.24
  Using cached itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
Collecting click>=5.1
  Using cached click-7.1.1-py2.py3-none-any.whl (82 kB)
Collecting Werkzeug>=0.15
  Using cached Werkzeug-1.0.0-py2.py3-none-any.whl (298 kB)
Collecting MarkupSafe>=0.23
  Downloading MarkupSafe-1.1.1-cp38-cp38-win_amd64.whl (16 kB)
Installing collected packages: MarkupSafe, Jinja2, itsdangerous, click, Werkzeug, flask
```

1.3 Tạo ứng dụng đầu tiên với Flask

Để tạo ứng dụng đầu tiên của flask ta gõ đoạn code sau và save vào file application.py



Và để thực thi chương trình ta cần thiết lập biến môi trường *set FLASK_APP=application.py* cho phiên làm việc và sau đó chạy lệnh *flask run*. Sau lệnh này, Flask sẽ khởi động một Web server và cho phép truy cập ứng dụng bằng cách nhập URL http://localhost:5000/ vào thanh địa chỉ của trình duyệt.

```
(First_Flask) C:\Users\nhlon\First_Flask>set FLASK_APP=application.py

(First_Flask) C:\Users\nhlon\First_Flask>Flask run
 * Serving Flask app "application.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [19/Mar/2020 16:30:17] "[37mGET / HTTP/1.1[0m" 200 -
127.0.0.1 - - [19/Mar/2020 16:30:17] "[33mGET /favicon.ico HTTP/1.1[0m" 404 -
```

Tạo ứng dụng thứ 2:

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello, world!"

@app.route("/<string:name>")
def hello(name):
    return "Hello, {}!".format(name)
```

1.4 Định tuyến(Routing)

Phần tiếp theo trong bài hướng dẫn flask python chúng ta sẽ đi tìm hiểu về cách định tuyến url tới các phần khác nhau của trang web. Hầu hết các website hiện tại đều có các url dễ nhớ, nó giúp người dùng dễ nhớ và truy cập trực tiếp khi cần tới.

Sử dụng route() để chỉ định mỗi url của người dùng sẽ trỏ tới một hàm nhất định, ví dụ như sau:

```
0
1  from flask import Flask
2
3  app = Flask(__name__)
4
5
6  @app.route('/')
7  def home():
8      return 'Home page'
9
10
11 @app.route('/user')
12 def user():
13     return 'User page'
14
15
16 @app.route('/about')
17 def about():
18     return 'About page'
19
20
21 if __name__ == '__main__':
22     app.run()
```

Khi đó, ta sẽ thu được các kết quả khác nhau cho mỗi url ta đã chỉ định.

- Ta sẽ nhận được dòng chữ Home page khi truy cập vào địa chỉ http://localhost:5000/
- Ta sẽ nhận được dòng chữ User page khi truy cập vào địa chỉ http://localhost:5000/user
- Ta sẽ nhận được dòng chữ About page khi truy cập vào địa chỉ http://localhost:5000/about

1.5 Render 1 file HTML

Hiện tại, hàm hiển thị trong ứng dụng trả về một chuỗi đơn giản. Điều đó chưa đủ, chúng ta muốn nâng cấp chuỗi trả về này thành một trang HTML hoàn chỉnh. Ta sẽ thực hiện như sau:

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")
```

Cách trả về chuỗi HTML cho trình duyệt như trên không phải là giải pháp tốt. Hãy thử tưởng tượng xem mã trong hàm hiển thị trên sẽ trở nên phức tạp như thế nào khi ứng dụng hoàn tất và liên tục nhận các bài viết với các nội dung khác nhau từ người dùng. Ứng dụng cũng sẽ phải có nhiều hàm hiển thị hơn để liên kết với các URL khác. Vì vậy, nếu lúc nào đó chúng ta cần thay đổi cách trình bày của trang Web, chúng ta phải cập nhật mã HTML trong từng hàm hiển thị. Rõ ràng đây không phải là cách hay khi ứng dụng của chúng ta trở nên phức tạp và có nhiều nội dung động hơn.

Vì vậy, sẽ tốt hơn nhiều nếu chúng ta chia phần trình bày (presentation) và phần logic của chương trình thành các thành phần tách biệt. Nếu làm được như vậy, bạn có thể nhờ người thiết kế Web để tạo ra các trang Web thật hấp dẫn, còn bạn thì có thể viết mã ứng dụng với Python hoàn toàn độc lập với họ.

Template cho phép chúng ta thực hiện việc phân tách giữa phần trình bày và logic của ứng dụng. Trong Flask, các template được viết trong các file khác nhau và lưu trong thư mục con templates của gói ứng dụng. Do đó, để bắt đầu với template, hãy đảm bảo rằng bạn đang ở trong thư mục First_Flask và tạo ra thư mục chứa các template. Dưới đây là template đầu tiên với chức năng tương tự như đoạn mã HTML được hàm hiển thị index() trả về ở trên. Bạn hãy lưu lại file này trong thư mục app/templates/index.html:

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Website!</title>
    </head>
    <body>
        <h1>{{ headline }}</h1>
    </body>
</html>
```

Đây hầu như là một trang Web với mã HTML tiêu chuẩn. Điểm khác nhau duy nhất giữa mã HTML thông thường và đoạn mã trên đây là vài vùng chứa (placeholder) cho các nội dung động nằm giữa hai dấu ngoặc nhọn {{..}}. Các placeholder này đại diện cho các nội dung có thể được thay đổi vào thời gian chạy (runtime). Jinja2 là ngôn ngữ hỗ trợ viết HTML theo kiểu này. Jinja2 là một templates engine. Jinja2 là một extensions hỗ trợ cho việc show dữ liệu từ app ra html. Jinja2 là 1 extension được kèm theo Flask nên sau khi cài đặt Flask, ta có thể sử dụng luôn Jinja2 mà không cần phải cài thêm.

Như vậy, vì chúng ta đã chuyển phần trình bày vào một HTML template, hàm hiển thị có thể được đơn giản hóa như sau:

```
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      headline = "Hello, world!"
8      return render_template("index.html", headline=headline)
```

Quá trình chuyển đổi template thành một trang HTML được gọi là quá trình kết xuất hay kiến tạo (render). Để kết xuất template, chúng ta phải tham chiếu đến một hàm trong Flask gọi là render_template(). Hàm này nhận vào tên file và một danh sách các tham số của template và trả về template đó, nhưng thay thế các vùng chứa (placeholder) bằng các giá trị tương ứng.

Hàm render_template() sẽ gọi đến Jinja2. Jinja2 sẽ thay thế các đoạn mã trong các khối {{ ... }} bằng các giá trị tương ứng từ các tham số được cung cấp khi gọi hàm render_template().

CHƯƠNG 2
HỆ QUẢN TRỊ CSDL POSTGRESQL

2.1 Giới thiệu về hệ quản trị CSDL PostgreSQL (PostGIS)

PostgreSQL là một hệ quản trị cơ sở dữ liệu quan hệ và đối tượng dựa trên POSTGRES, bản 4.2, được khoa điện toán của đại học California tại Berkeley phát triển. POSTGRES mở đường cho nhiều khái niệm quan trọng mà các hệ quản trị dữ liệu thương mại rất lâu sau mới có. Hệ quản trị cơ sở dữ liệu thuộc tính PostgreSQL đang phát triển mạnh mẽ và nhanh chóng trên thế giới và là đối thủ cạnh tranh với hệ quản trị cơ sở dữ liệu thuộc tính PostgreSQL. Đặc biệt trong lĩnh vực GIS thì hệ quản trị cơ sở dữ liệu thuộc tính PostgreSQL có lợi thế hơn hệ quản trị cơ sở dữ liệu thuộc tính MySQL vì nó có khả năng tích hợp với hệ quản trị cơ sở dữ liệu bản đồ PostGIS.

PostgreSQL là một chương trình mã nguồn mở xây dựng theo chuẩn SQL99 và có nhiều đặc điểm hiện đại:

- Câu truy vấn phức hợp (complex query)
- Khóa ngoại (foreign key)
- Thủ tục sự kiện (trigger)
- Các khung nhìn (view)
- Tính toàn vẹn của các giao dịch (integrity transactions)
- Việc kiểm tra truy cập đồng thời đa phiên bản (multiversion concurrency control)

PostgreSQL được phổ biến bằng giấy phép BSD cổ điển. Nó không quy định những hạn chế trong việc sử dụng mã nguồn của phần mềm. Bởi vậy PostgreSQL có thể được dùng, sửa đổi và phổ biến bởi bất kỳ ai cho bất kỳ mục đích nào.

Cơ sở dữ liệu Postgresql là một dạng cơ sở dữ liệu quan hệ. Nó rất mạnh về lưu trữ theo dạng bảng quan hệ, một điều khá thú vị nữa là nó chạy trên nền hệ điều hành linux và miễn phí, chạy rất ổn định và không thua kém những cơ sở dữ liệu thương mại, Nếu bạn lập trình mạng thì lựa chọn cơ sở dữ liệu Postgresql là khuyên dùng.

Hệ quản trị cơ sở dữ liệu PostgreSQL có đầy đủ tính năng của một cơ sở dữ liệu quan hệ như sử dụng các câu truy vấn SQL, các bảng quan hệ, các khóa,…và đặc biệt ở đây là tốc độ của Postgresql là rất cao và cho phép nhiều người truy cập cùng một lúc, nó thích ứng cho bạn xây dựng những ứng dụng trên mạng Internet.

2. 1 Giới thiệu về SQL

SQL là viết tắt của Structured Query Language có nghĩa là ngôn ngữ truy vấn có cấu trúc, là một ngôn ngữ máy tính để lưu trữ, thao tác và truy xuất dữ liệu được lưu trữ trong một cơ sở dữ liệu quan hệ.

SQL là ngôn ngữ chuẩn cho hệ cơ sở dữ liệu quan hệ. Tất cả các hệ thống quản lý cơ sở dữ liệu quan hệ (RDMS) như MySQL, MS Access, Oracle, Sybase, Informix, Postgres và SQL Server đều sử dụng SQL làm ngôn ngữ cơ sở dữ liệu chuẩn. SQL được sử dụng phổ biến vì nó có các ưu điểm sau:

- Cho phép người dùng truy cập dữ liệu trong các hệ thống quản lý cơ sở dữ liệu quan hệ.
- Cho phép người dùng mô tả dữ liệu.
- Cho phép người dùng xác định dữ liệu trong cơ sở dữ liệu và thao tác dữ liệu đó.
- Cho phép nhúng trong các ngôn ngữ khác sử dụng mô-đun SQL, thư viện và trình biên dịch trước.
- Cho phép người dùng tạo và thả các cơ sở dữ liệu và bảng.
- Cho phép người dùng tạo chế độ view, thủ tục lưu trữ, chức năng trong cơ sở dữ liệu.
- Cho phép người dùng thiết lập quyền trên các bảng, thủ tục và view.

Các lệnh SQL tiêu chuẩn để tương tác với cơ sở dữ liệu quan hệ là CREATE, SELECT, INSERT, UPDATE, DELETE và DROP. Các lệnh này có thể được phân thành các nhóm sau dựa trên bản chất của chúng

DDL - Ngôn ngữ định nghĩa dữ liệu(Data Definition Language)

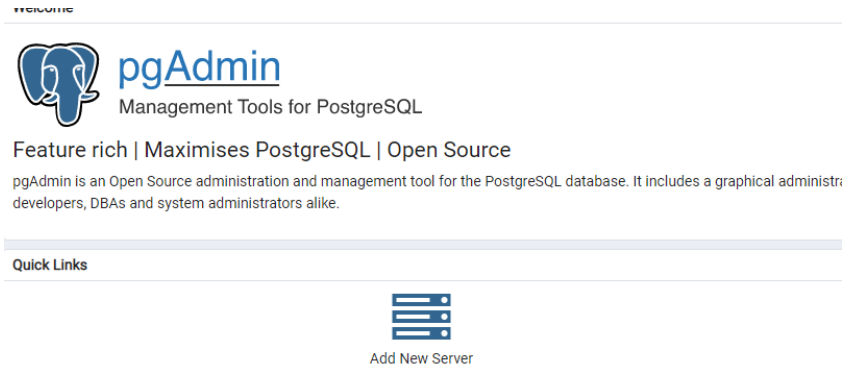
Lệnh	Mô tả
CREATE	Tạo ra một bảng mới, một view của một bảng, hoặc các đối tượng khác trong cơ sở dữ liệu.
ALTER	Sửa đổi một đối tượng cơ sở dữ liệu hiện có, chẳng hạn như một bảng.
DROP	Xoá toàn bộ một bảng, view của một bảng hoặc các đối tượng khác trong cơ sở dữ liệu.

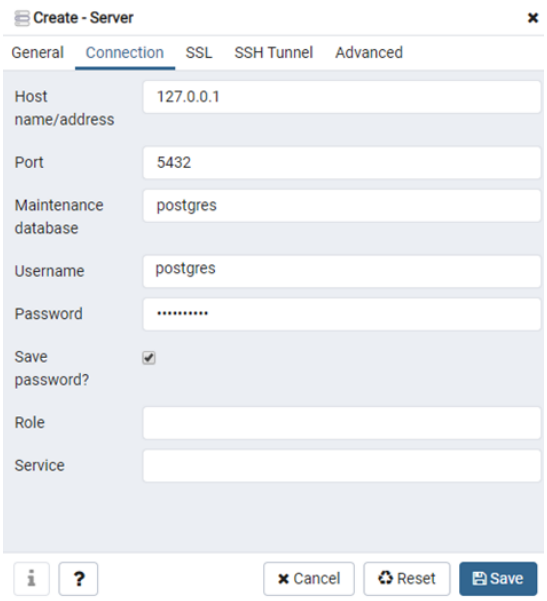
DML - Ngôn ngữ thao tác dữ liệu(Data Manipulation Language)

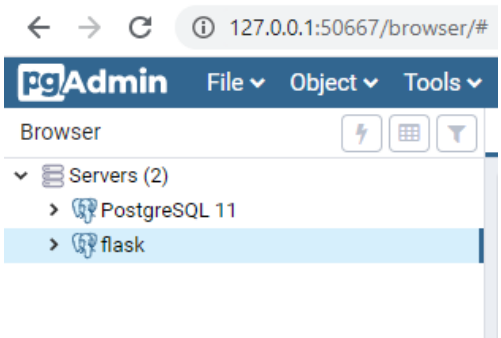
Lệnh	Mô tả
SELECT	Lấy ra các bản ghi nhất định từ một hoặc nhiều bảng.
INSERT	Tạo một bản ghi..
UPDATE	Chỉnh sủa bản nghi.
DELETE	Xóa bản ghi.

2. 2 Các câu lệnh SQL thông dụng

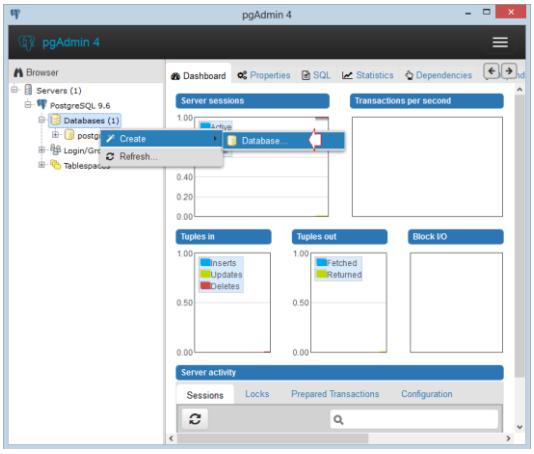
Tạo 1 Server

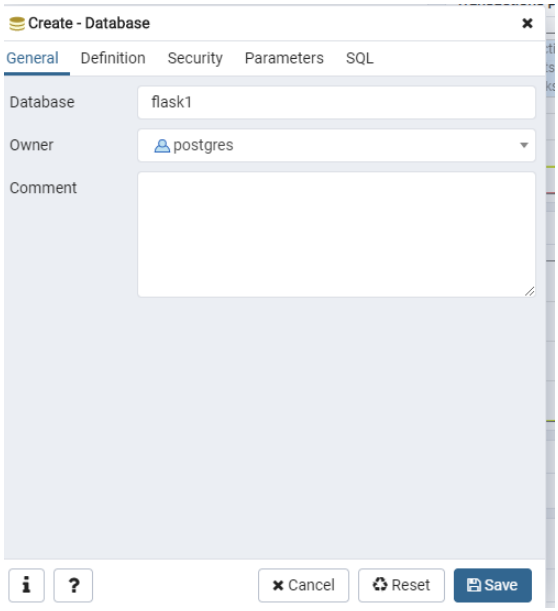






Tạo DataBase





Tạo bảng flights

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

Thêm dữ liệu vào bảng:

```
INSERT INTO flights (origin, destination, duration) VALUES ('Hà Nội', 'Hải Phòng', 415);
INSERT INTO flights (origin, destination, duration) VALUES ('Đã Nẵng', 'HCM', 455);
INSERT INTO flights (origin, destination, duration) VALUES ('Nghệ An', 'Quảng Ninh', 415);
INSERT INTO flights (origin, destination, duration) VALUES ('Hà Nội', 'HCM', 760);
INSERT INTO flights (origin, destination, duration) VALUES ('Hải Phòng', 'HCM', 700);
INSERT INTO flights (origin, destination, duration) VALUES ('HCM', 'Hà Nội', 760);
```

Khóa ngoại (foreign key)

- Tạo bảng passengers bao gồm các trường sau.

- Trường flight_id sẽ được tham chiếu đến trường id trong bảng fights đã tạo lúc trước thông qua từ khóa REFERENCES

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

Thêm dữ liệu vào bảng

Từ những dữ liệu đã thêm vào 2 bảng flights và passengers ta có thể có các câu hỏi sau:

- Hiện tại có những chuyến bay nào?
- Có những chuyến bay nào xuất phát từ Hà Nội?
- Có bao nhiêu chuyến bay xuất phát từ Hà Nội?
- Có những chuyến bay nào có thời gian bay lâu hơn 500 (phút)?
- Có những chuyến bay nào bay đến HCM và có thời gian bay lâu hơn 500 (phút)
- Có những chuyến bay nào bay đến HCM hoặc có thời gian bay lâu hơn 500 (phút)
- Có những chuyến bay nào xuất phát từ Hà Nội hoặc HCM
- Có những chuyên bay nào xuất phát từ địa phương có chữ ‘H’ (ví dụ: Hà Nội, HCM)
- Thử thách: liệt kê những chuyến bay có nhiều hơn 1 hành khách (+1 điểm)

Để trả lời cho các câu hỏi ở trên ta sử dụng Câu lệnh sql – SELECT

- SELECT * FROM flights
- SELECT origin, destination FROM flights
- SELECT * FROM flights WHERE id = 7
- SELECT * FROM flights WHERE origin = 'Hà Nội’
- SELECT COUNT(*) FROM flights WHERE origin = 'Hà Nội’
- SELECT * FROM flights WHERE duration > 500
- SELECT * FROM flights WHERE destination = 'HCM' AND duration > 500
- SELECT * FROM flights WHERE destination = 'HCM' OR duration > 500
- SELECT * FROM flights WHERE origin IN ('Hà Nội', 'HCM’)

- SELECT * FROM flights WHERE origin LIKE '%H%’

Để xóa dữ liệu trong bảng ta sử dụng câu lệnh sql – DELETE

DELETE FROM flights WHERE destination = ‘HCM'

Để cập nhật dữ liệu của một record nào đấy trong bảng ta sử dụng câu lệnh sql – UPDATE

UPDATE flights

SET duration = 400

WHERE origin = 'Hà Nội'

AND destination = 'Hải Phòng';

Câu lệnh sql – SELECT (lồng)

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

CHƯƠNG 3 ORM VÀ API

3. 1 Giới thiệu về ORM

Ánh xạ quan hệ đối tượng (ORM) là một kỹ thuật cho phép ta truy vấn và thao tác dữ liệu từ cơ sở dữ liệu bằng cách sử dụng mô hình hướng đối tượng. Khi nói về ORM, hầu hết mọi người đều đề cập đến thư viện thực hiện kỹ thuật Ánh xạ quan hệ đối tượng, do đó cụm từ "ORM".

Thư viện ORM là một thư viện hoàn toàn bình thường được viết bằng ngôn ngữ ta chọn, đóng gói mã cần thiết để thao tác dữ liệu, vì vậy ta không sử dụng SQL nữa; ta tương tác trực tiếp với một đối tượng trong cùng ngôn ngữ ta đang sử dụng.

3. 2 Thư viện SQLAlchemy

Một nhiệm vụ phổ biến khi bắt đầu xây dựng một web service bất kỳ là làm sao để xây dựng một cơ sở dữ liệu chắc chắn, bền vững. Trong quá khứ, lập trình viên sẽ phải viết những câu lệnh SQL, thông qua chúng, để thiết database hay lấy dữ liệu từ database. Ngày nay, lập trình viên có thể sử dụng object-relational mapping(ORM) thay vì các câu lệnh SQL phức tạp, và khó khăn trong việc bảo trì, đọc hiểu và chỉnh sửa.

ORM là một công nghệ lập trình, cho phép chuyển đổi (convert) từ những data trong database với những đối tượng trong các ngôn ngữ lập trình hướng đối tượng. Thông thường, những ngôn ngữ hướng đối tượng như python tồn tại những kiểu không thể thể hiện bằng các kiểu nguyên thủy(giống như là integer, string) .

Ví dụ, Một người có thể có các địa chỉ nhà ở khác nhau, như địa chỉ thường trú chỗ này , chỗ kia, có một, vài số điện thoại hay dùng, Một địa chỉ thì phải lưu trữ mã bưu điện của tỉnh/thành phố đó, hay lưu trữ tên đường, số nhà.v.v. Điều này thể hiện trong ngôn ngữ hướng đối tượng như sau:

Một Person object có thể có một danh sách các Address object, và một danh sách các PhoneNumber object. Một Address object có một PostCode object., một StreetName object và một StreetNumber object.

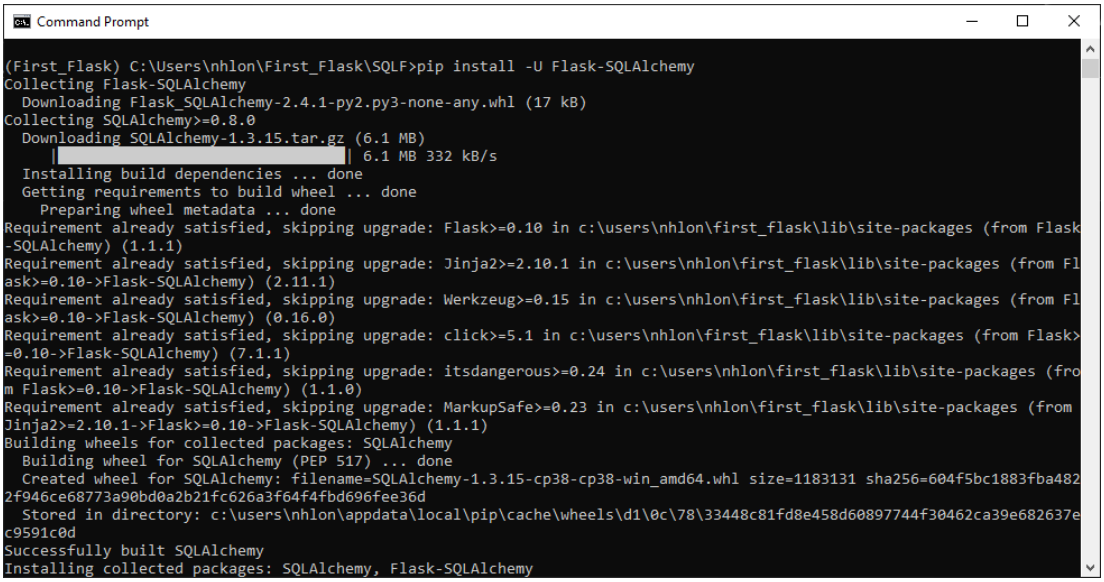
Mặc dù một object đơn giản như PostCode và StreetName có thể biểu diễn bằng các string, nhưng những object phức tạp hơn, giống như Address object hay Person, không thể chỉ sử dụng những kiểu nguyên thủy như String, Integer để thể hiện. Thêm vào đó, những

đối tượng phức tạp có thể bao gồm các thể hiện của một class, hoặc các phương thức trong class không thể sử dụng 1 kiểu cho tất cả các trường hợp.

Để quản lý những đối tượng phức tạp, con người đã phát triển một lớp hệ thống mới , gọi là ORM. Ở ví dụ trước của chúng ta, ORM có thể xây dựng với các thành phần: một class Person, một class Address , một class PhoneNumber và một class mapper để thực hiện việc map một python class với một bảng trong database. Thay vì phải viết những code SQL tẻ nhạt, một ORM có thể mang đến cho ta những tùy chọn có thể sử dụng trong khi lập trình những logic của hệ thống

3.2.1 Cài đặt SQLalchemy

Chạy lệnh pip install -U Flask-SQLAlchemy trong command line



3.2.2 Kết nối đến csdl sử dụng sqlalchemy

Có nhiều cách để kết nối đến CSDL sử dụng SQLAlchemy. Sau đây là 1 trong những cách để kết nối sử dụng create_engine

```
1  from sqlalchemy import create_engine
2  from sqlalchemy.orm import scoped_session, sessionmaker
3
4  engine = create_engine("postgresql://postgres:HMVuong216@localhost:5432/flask1")
5  db = scoped_session(sessionmaker(bind=engine))
6
7  def main():
8      flights = db.execute("SELECT origin, destination, duration FROM flights").fetchall()
9      for flight in flights:
10         print(f"{flight.origin} to {flight.destination}, {flight.duration} minutes.")
11
12 if __name__ == "__main__":
13     main()
```

Chạy file mới tạo sẽ cho ta kết quả

```
(First_Flask) C:\Users\nhlon\First_Flask\SQLF>list.py
Hà Nội to Hải Phòng, 415 minutes.
Đã Nẵng to HCM, 455 minutes.
Nghệ An to Quảng Ninh, 415 minutes.
Hà Nội to HCM, 760 minutes.
Hải Phòng to HCM, 700 minutes.
HCM to Hà Nội, 760 minutes.
```

3. 3 Sử dụng ORM để tương tác với CSDL

- Khai báo mới 1 class: class tên_đối_tượng
- Hàm __init__: Hàm khởi tạo đối tượng (khai báo các trường dữ liệu của đối tượng)

```
class Flight:

    def __init__(self, origin, destination, duration):
        self.origin = origin
        self.destination = destination
        self.duration = duration
```

```python
class Flight:

    def __init__(self, origin, destination, duration):
        self.origin = origin
        self.destination = destination
        self.duration = duration


def main():

    # Create flight.
    f = Flight(origin="New York", destination="Paris", duration=540)

    # Change the value of a variable.
    f.duration += 10

    # Print details about flight.
    print(f.origin)
    print(f.destination)
    print(f.duration)

if __name__ == "__main__":
    main()
```

```
(First_Flask) C:\Users\nhlon\First_Flask\ORM_API>python classes1.py
New York
Paris
```

```python
class Flight:

    def __init__(self, origin, destination, duration):
        self.origin = origin
        self.destination = destination
        self.duration = duration

    def print_info(self):
        print(f"Flight origin: {self.origin}")
        print(f"Flight destination: {self.destination}")
        print(f"Flight duration: {self.duration}")

def main():

    f1 = Flight(origin="New York", destination="Paris", duration=540)
    f1.print_info()

    f2 = Flight(origin="Tokyo", destination="Shanghai", duration=185)
    f2.print_info()

if __name__ == "__main__":
    main()
```

```python
class Flight:

    def __init__(self, origin, destination, duration):
        self.origin = origin
        self.destination = destination
        self.duration = duration

    def print_info(self):
        print(f"Flight origin: {self.origin}")
        print(f"Flight destination: {self.destination}")
        print(f"Flight duration: {self.duration}")

    def delay(self, amount):
        self.duration += amount

def main():

    f1 = Flight(origin="New York", destination="Paris", duration=540)
    f1.delay(10)
    f1.print_info()

if __name__ == "__main__":
    main()
```

```python
class Flight:

    counter = 1

    def __init__(self, origin, destination, duration):

        # Keep track of id number.
        self.id = Flight.counter
        Flight.counter += 1

        # Keep track of passengers.
        self.passengers = []

        # Details about flight.
        self.origin = origin
        self.destination = destination
        self.duration = duration
```

```python
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Flight(db.Model):
    __tablename__ = "flights"
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String, nullable=False)
    destination = db.Column(db.String, nullable=False)
    duration = db.Column(db.Integer, nullable=False)


class Passenger(db.Model):
    __tablename__ = "passengers"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False)
```

Object relation model

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

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

Create table from objects

```python
from flask import Flask, render_template, request
from models import * # import file models.py previously created

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "postgresql://postgres:HMVuong216@localhost:5432/ORM"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db.init_app(app)

def main():
    db.create_all()

if __name__ == "__main__":
    with app.app_context():
        main()
```

```
(First_Flask) C:\Users\nhlon\First_Flask\ORM_API>python create.py
```

Thêm dữ liệu sử dụng ORM:

flight = Flight(origin="New York", destination="Paris", duration=540)
db.session.add(flight)