Dauphine | PSL ★

→ COMPUTER SCIENCE, DECISION-MAKING, AND DATA

Algorithmic and advanced Programming in Python

Eric Benhamou eric.benhamou@dauphine.eu
Chien-Chung.Huang chien-chung.huang@ens.fr
Sofía Vázquez sofia.Vazquez@dauphine.eu

Master class 6

Outline

1. Generic concept about API

2. Introduction to virtual environment + Flask

3. Data structure

4. Start Playing with data structure and linked list

Reminder of the objective of this course

- People often learn about data structures out of context
- But in this course you will learn foundational concepts by building a real application with python and Flask

- To learn the ins and outs of the essential data structure, experiencing in practice has proved to be a much more powerful way to learn data structures

Reminder of previous session

- In Master class 5, we discuss about stacks and queues
- Question: can you summarize the pros and cons of stacks and queues?

What is an API?

- Web APIs (Application Programming Interfaces) are tools for making information and functionalities accessible via the Internet.

- In this lecture, we will see
  - What are AI and how to use them
  - How to build an API that makes data available to users
  - What should be good practices in terms of API writing and how to apply them on a concrete example

  Question: do you know what an API is?

What is exactly an API?

- A web API allows information and functionality to be manipulated by computer programs over the internet.

- For example, with the Twitter web API, you can write a program in a language like Python or JavaScript that collects metadata about tweets.

- More generally, in programming, the term API (short for Application Programming Interface) refers to a part of a computer program designed to be used or manipulated by another program, unlike interfaces which are designed to be used or manipulated by humans.

Question: what APIs are good for and when to use it?

When to create an API?

- Computer programs often need to communicate with each other or with the underlying operating system, and APIs are one way to do this.

- In general, we create an API when:
  - Our dataset is large, which makes uploading by FTP cumbersome or resource intensive.
  - Users need to access data in real time, for example for viewing on a website or as part of an application.
  - Our data is changed or updated frequently.
  - Users only need to access part of the data at a time.
  - Users have to do more than just view the data, such as contributing, updating, or deleting.

Question: what are alternatives to API?

What are alternative to API?

- If you have data that you want to share with the world, creating an API is one way to do it.

- However, APIs are not always the best way to share data with users.

- If the volume of data you want to share is relatively small, it is better to provide a "data dump" in the form of a JSON, XML, CSV or Sqlite file. Depending on the resources available, this approach may be viable up to a volume of a few gigabytes.

- Note that we can provide both a data dump and an API; it's up to the users to choose what suits them best.

Question: what are the common words in web API world?

API terminology 1/3

- When building or using an API, the following terms are frequently encountered:

  - **HTTP** (HyperText Transfer Protocol): This is the primary means of communicating information on the Internet. HTTP implements a number of "methods" that tell which direction data should move and what should be done with it. The two most common are GET, which retrieves data from a server, and POST, which sends new data to a server.

    -> Do you know any other method?

  - **HTTPS**:

    -> Do you know the difference with HTTP?

API terminology 2/3

URL (Uniform Resource Locator): Address of a resource on the web, such as https://dauphine.psl.eu/en/. A URL consists of a protocol (https : //), a domain (dauphine.psl.eu), an optional path (/ en) .

It describes the location of a specific resource, such as a web page. In the field of APIs, the terms URL, request, URI, endpoint refer to similar ideas. In the following, we will only use the terms URL and request, to be clearer.

To make a GET request or follow a link, all you need is a web browser.

API terminology 3/3

- **JSON** (JavaScript Object Notation): This is a text data storage format designed to be readable by both humans and machines. JSON is the most common format for data retrieved by API, the second most common being XML.

- **REST** (REpresentational State Transfer): this is a methodology that brings together best practices in terms of API design and implementation. APIs designed according to the principles of the REST methodology are called REST APIs. There are, however, many controversies surrounding the exact meaning of the term. In the following, we will speak more simply of Web API or HTTP API.

We will now implement an API

- In the following, we will see how to create an API using Python and the Flask framework + SQLite

- There are alternatives to this solutions:
  - Flask + PostGre
  - Python Django + SQLite/PostGre
  - PhP + MySQL/PostGre
  - Node JS + SQLite/PostGre/MongoDB

  Question: Do you know the differences?
  Question: Do you have experience with FLASK?

What is flask?

- Flask is a micro web framework written in Python whose initial release was in 2010

- It is a microframework because it does not require particular tools or libraries.

- In particular it has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

- It relies on the Jinja template engine and the Werkzeug WSGI toolkit

- https://flask.palletsprojects.com/en/2.0.x/

Why Flask?

- Python has several development frameworks for producing web pages and APIs.

- Apart from Flask, the other best known is Django, that is a very rich and heavy framework. Django can be overwhelming for inexperienced users, however.

- Flask applications are built from very simple frameworks and are therefore more suitable for prototyping APIs.
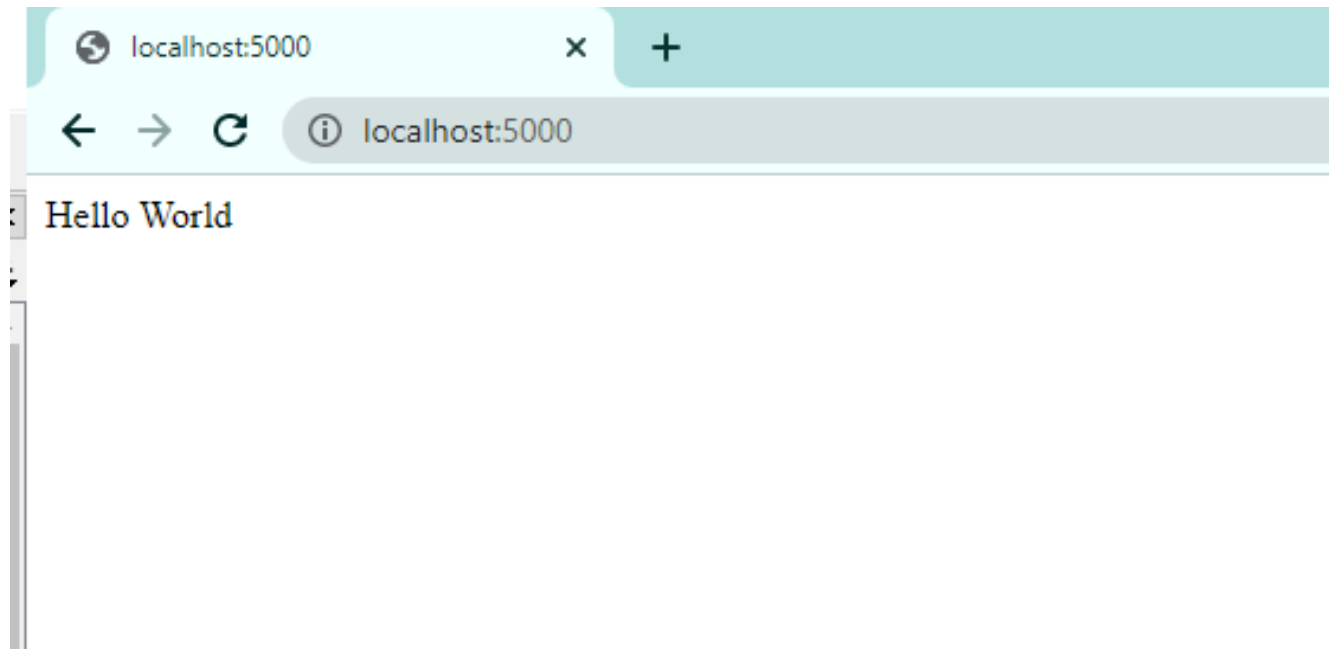
- Flask in addition is faster lighter and more modern

A simple Flask example

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


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

if __name__ == '__main__':
    app.run(host="localhost", port=int("5000"))
```

If you run the python code, you get

It is good practice to name main index home as follows

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


@app.route("/", methods=['GET'])
def home():
    return """
<h1>Distant Reading Archive</h1>
<p>This site is a prototype API for distant
reading of science fiction novels.</p>
"""


if __name__ == '__main__':
    app.run(host="localhost", port=int("5000"))
```

Question: What should I get in terms of output?

Here we go!

How works Flask?

- Flask sends HTTP requests to Python functions.
- In our case, we applied a URL path ('/') to a function: home.
- Flask executes the function code and displays the result in the browser.
- In our case, the result is a welcome HTML code on the site hosting our API.

Routes!

- The process of applying URLs to functions is called routing.
- The instruction:

  @app.route('/', methods=['GET'])

appearing in the program tells Flask that the home function matches the path /.

The methods list (methods = ['GET']) is a keyword argument that tells Flask what type of HTTP requests are allowed.

We will use mostly GET requests in the following, but many web applications use both GET (to send data from the application to users) and POST (to receive data from users).

Other requests

- The most popular API design methodology is called REST.

- The most important aspect of REST is that it is based on four methods defined by the HTTP protocol: GET, POST, PUT and DELETE.

- These correspond to the four standard operations performed on a database: READ, CREATE, UPDATE and DELETE.

Important Flask concepts

- import Flask:

This instruction imports the Flask library, which is available by default in Anaconda. If not you need to install the package

-> Do you know how?

app = flask.Flask(__name__) :

""""Creates the Flask application object, which contains application data and methods for actions that can be performed on the object. The last app.run () statement is an example of method usage.
""""

app.run( )

""""allows you to run the application. You can also specify host=None, port=None, debug=None"""""

Programming concept

- It is good practice to create a virtual environment

Question: Do you know what is a virtual environment in python?

First use a virtual environment

- A virtual environment essentially just compartmentalizes our project and its dependencies so that there won't be any clashes with additional project that we might make in the future.

- So it is highly recommended to use a virtual environment for large project so that we have an independent setup.

- When we use **pythonanywhere**

this is precisely what we have already put in place

Create a virtual environment

• Create a project folder and a venv folder within it:

$ mkdir FlaskAPI

$ cd FlaskAPI

$ python –m venv venv

You can check that the venv has been created easily with ls

$ ls

-> venv/

Checking virtual environment

If you change your directory to venv, you should be able to see what is inside

$ cd venv

$ ls

You should see it contains various subfolders

📁 bin/ 📁 include/ 📁 lib/ 📄 pyvenv.cfg

You do not need to worry about these folders but you can check that within bin folder, you have various libraries

📄 activate 📄 activate.fish 📄 easy_install-3-9 📄 pip 📄 pip3….

How to check your current python version and libraries?

- Just type

$ python –version

Once we have installed the virtual environment, we need to activate the virtual environment

$. venv/bin/activate   In windows: > venv/Scripts/activate

You can see the virtual environment is activated as we see it in parenthesis (venv)

You can check also the virtual environment is activated with this command

$ env | grep VIRTUAL_ENV

You should get VIRTUAL_ENV=/Users/…./FlaspAPI/venv/

If Flask is not installed

- pip install flask
- If you go to virtual environment you can check that flask is installed as you would get
- $ ls venv/bin

You should see now

▤ Flask

One dot pifile

- To educe the complexity of our project, we will implement all databse concepts using one dot pifile. Keep in mind that in real production environment this will be more complicated

- Let us start and create a file called server.py

$ vim server.py

Throught the class, I will be using vim but feel free to use your own editor

Create a server.py file

from flask import Flask

app = Flask(__name__)

""" What it does behind the scene is that the Flask object implements a WSGI application and acts as the central object. It is passed the name of the module or package of the application

Once it is created it will act as a central registry for the view functions the url rules template configuration and much more

You do not really need to know. Flask is a middleware that sits between our python server (on the backend) and the front end """

Add a database

Once we have the app variable which is a Flask object, we can associate a database

app = Flask(__name__)
app.config["SQALCHEMY_DATABASE_URI"]

""" App.config is a standard python dictionary
 we will use a file to keep this simple """
app.config["SQALCHEMY_DATABASE_URI"] = "sqlite://sqlitedb.file"
app.config["SQL_TRACK_MODIFICATIONS"] = 0

What does it do behind the scene?

The instruction

app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite://sqlitedb.file"

tells the Flask app that there is an SQL Lite database whose data are stored on a file called sqlitedb.file

You can disable for the time being SQL track modification to ensure this is quite light

app.config["SQL_TRACK_MODIFICATIONS"] = 0

SqlAlchemy

- SqlAlchemy is an object relational mapper

- https://www.sqlalchemy.org/

- It gives application developers the full power and flexibility of SQL with an an object relational mapper (ORM)

Question: Do you know what is a virtual environment in python?

SQLAlchemy

- In short, it allows you to interact with an SQL database in an object-oriented way . Behind the scenes it's converting your object-oriented code into SQL statements and queries

- So Flask sql alchemy just makes sql alchemy more compatible with Flask applications and the way this essentially is going to work is we're going to create classes that represent the tables in our database

A typical example

- so for example if we have a table for users with an orm we're going to create models for each table and a user table in class form would look something like this

```python
# models
class User(db.Model):
    __tablename__ = "user"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    email = db.Column(db.String(50))
    address = db.Column(db.String(200))
    phone = db.Column(db.String(50))
    posts = db.relationship("BlogPost", cascade="all, delete")
```

The class object

class User(db.Model):

Tells python that this is a table in an SQL databse

__tablename__ = "user"

Gives the name the table

Add sql table fields or columns

id = db.Column(db.Integer, primary_key=True)

name = db.Column(db.String(50))

email = db.Column(db.String(50))

address = db.Column(db.String(200))

phone = db.Column(db.String(50))

posts = db.relationship("BlogPost", cascade="all, delete")

Question: Can you tell what each line is doing?

db.Column

- db.Column creates a column in the table

- db.Column(type either db.String, db.Date, db.Integer
- For string, you can specify the max length,

- You can say what is the primary_key for the Table

Question: What is a primary key?

Relating two tables together

posts = db.relationship("BlogPost", cascade="all, delete")

so now that we understand what an object relational mapper is and we understand what sql alchemy is we can move back up to our necessary imports

we're going to need to add one more configuration because by default sqlite doesn't enforce foreign key constraints

Enforce foreign key constraint

- from sqlalchemy import event
- from sqlalchemy.engine import Engine

```python
# configure sqlite3 to enforce foreign key contraints
@event.listens_for(Engine, "connect")
def _set_sqlite_pragma(dbapi_connection, connection_record):
    if isinstance(dbapi_connection, SQLite3Connection):
        cursor = dbapi_connection.cursor()
        cursor.execute("PRAGMA foreign_keys=ON;")
        cursor.close()
```

Question: What is a foreign key constraint?

foreign key constraint

- A foreign key constraint **specifies that the key can only contain values that are in the referenced primary key**, and thus ensures the referential integrity of data that is joined on the two keys. You can identify a table's foreign key when you create the table, or in an existing table with ALTER TABLE .

foreign key constraint

- A Foreign Key is a database key that is used to link two tables together.
- The FOREIGN KEY constraint identifies the relationships between the database tables by referencing a column, or set of columns, in the Child table that contains the foreign key, to the PRIMARY KEY column or set of columns, in the Parent table.

- The relationship between the child and the parent tables is maintained by checking the existence of the child table FOREIGN KEY values in the referenced parent table's PRIMARY KEY before inserting these values into the child table.
- In this way, the FOREIGN KEY constraint, in the child table that references the PRIMARY KEY in the parent table, will enforce database referential integrity.

- Referential integrity ensures that the relationship between the database tables is preserved during the data insertion process. Recall that the PRIMARY KEY constraint guarantees that no NULL or duplicate values for the selected column or columns will be inserted into that table, enforcing the entity integrity for that table.
- The entity integrity enforced by the PRIMARY KEY and the referential integrity enforced by the FOREIGN KEY together form the key integrity.

Difference between FOREIGN KEY and PRIMARY KEY

- The FOREIGN KEY constraint differs from the PRIMARY KEY constraint in that, you can create only one PRIMARY KEY per each table, with the ability to create multiple FOREIGN KEY constraints in each table by referencing multiple parent table. Another difference is that the FOREIGN KEY allows inserting NULL values if there is no NOT NULL constraint defined on this key, but the PRIMARY KEY does not accept NULLs.

- The FOREIGN KEY constraint provides you also with the ability to control what action will be taken when the referenced value in the parent table is updated or deleted, using the ON UPDATE and ON DELETE clauses.

Hence here are our two tables

```python
# models
class User(db.Model):
    __tablename__ = "user"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    email = db.Column(db.String(50))
    address = db.Column(db.String(200))
    phone = db.Column(db.String(50))
    posts = db.relationship("BlogPost", cascade="all, delete")


class BlogPost(db.Model):
    __tablename__ = "blog_post"
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(50))
    body = db.Column(db.String(200))
    date = db.Column(db.Date)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
```

Create the database

- From server import db

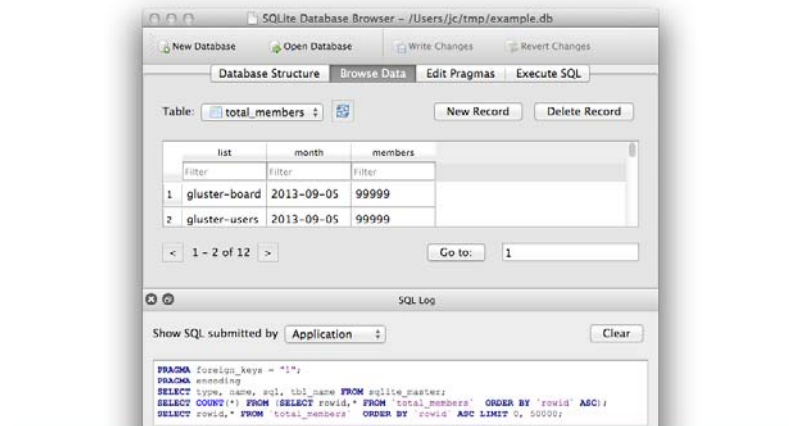- db.create_all()

- This will create the database

Check the database
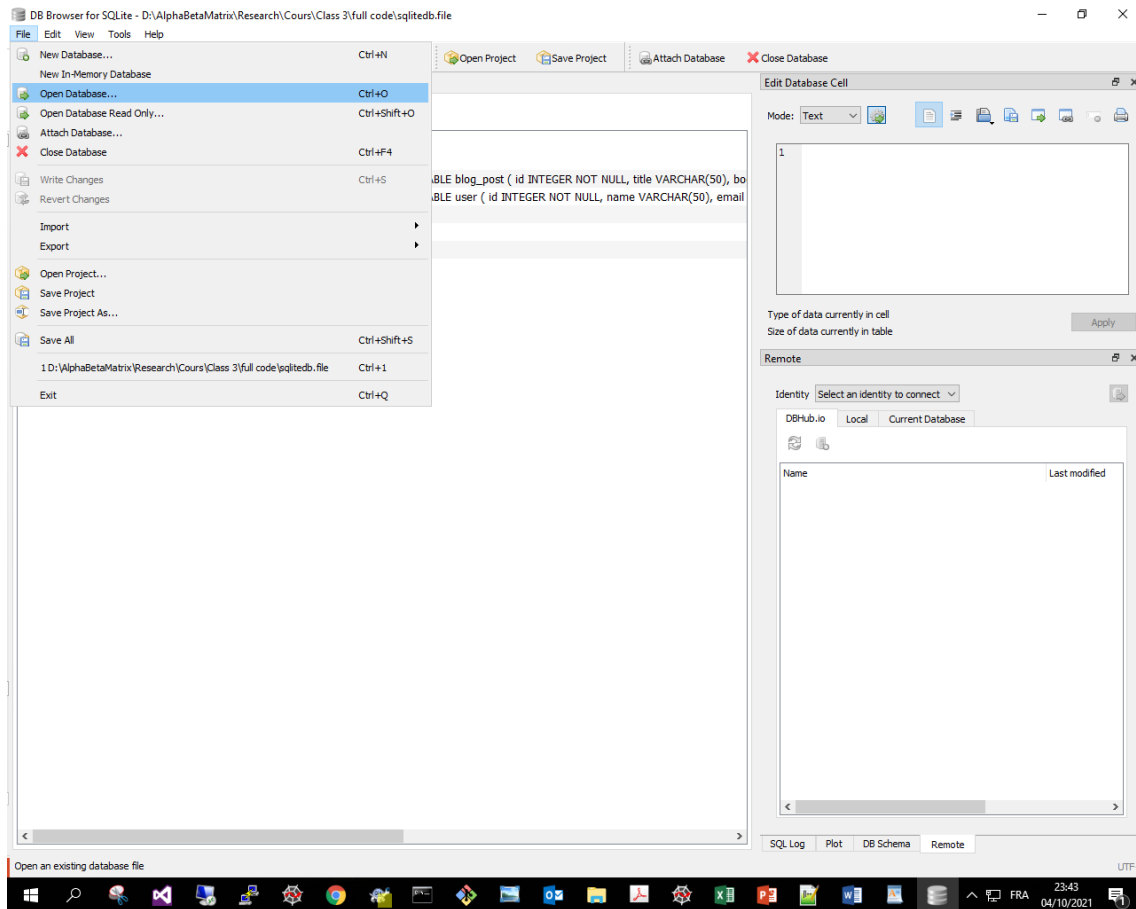
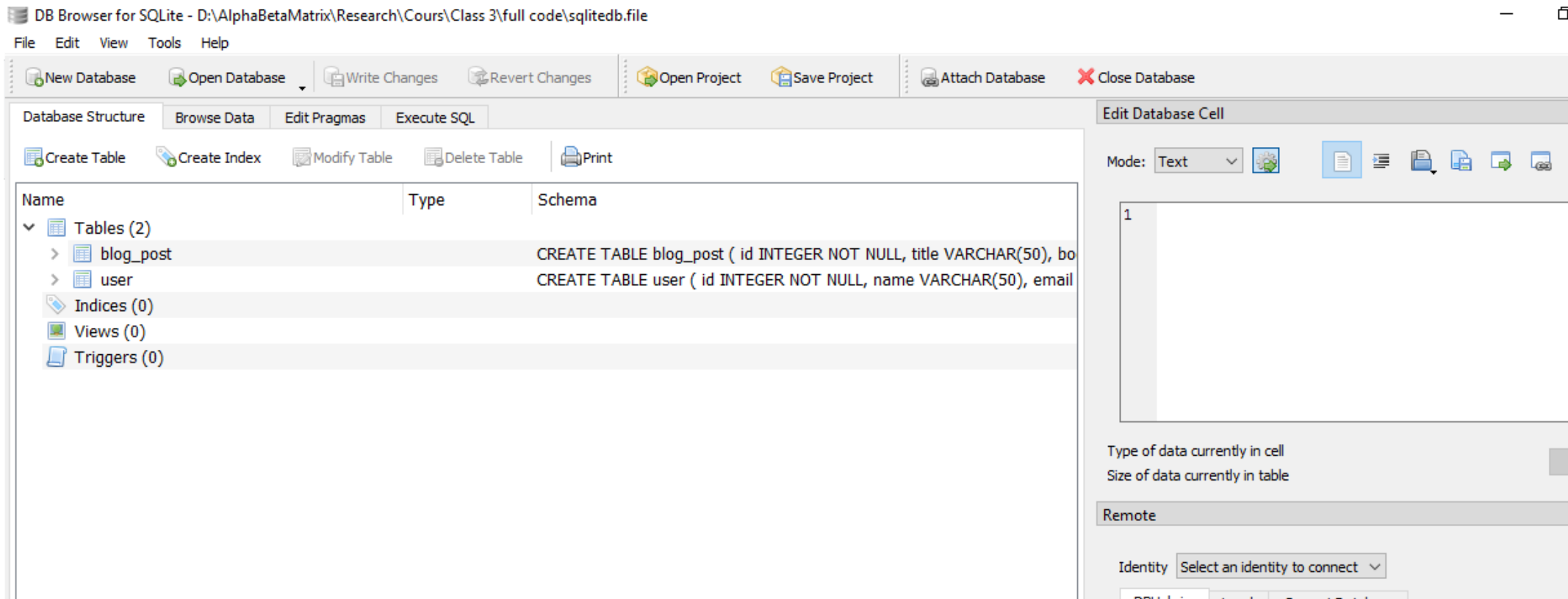- We can browse the sqlite database with https://sqlitebrowser.org/

Open BD Browser for SQLite

Open file

Last but not least create routes for the Flask api

- Routes are functions with decorator for the backend
- So to create the first route for creating user is quite simple

```
# routes
@app.route("/user", methods=["POST"])
def create_user():
    data = request.get_json()
    new_user = User(
        name=data["name"],
        email=data["email"],
        address=data["address"],
        phone=data["phone"],
    )
    db.session.add(new_user)
    db.session.commit()
    return jsonify({"message": "User created"}), 200
```

Let us comment this code

```python
# routes
@app.route("/user", methods=["POST"])
def create_user():
    data = request.get_json()
    new_user = User(
        name=data["name"],
        email=data["email"],
        address=data["address"],
        phone=data["phone"],
    )
    db.session.add(new_user)
    db.session.commit()
    return jsonify({"message": "User created"}), 200
```

In Lab session

- You will play with the concepts and starts getting more and more familiar with the database

- This will be useful for your project

- Lab is done by Sofia Vasquez