Curso	Engenharia da Computação	Período	9A
Disciplina	Sistemas Computacionais Distribuídos	Data	18/04/2024
Professor	Prof. Marcílio F. Oliveira Neto		
EXERCÍCIO WebAPIs com SQLite3			

Crie os scripts necessários e capture um print da tela durante a execução para anexar na atividade.

1) Valide se os seguintes módulos Python estão instalados na máquina:
   a. Flask - https://flask.palletsprojects.com/en/3.0.x/quickstart/
   b. SQLite3 (já incluso desde o python 2.5) - https://docs.python.org/3/library/sqlite3.html#tutorial

   Os links acima são para consulta ao longo da atividade.

2) Criar um script em python e seguir os passos abaio:
   1. Importar os seguintes módulos:

```
from flask import Flask, request
import sqlite3
from sqlite3 import Error
```

   2. Construir uma função capaz de se conectar com um banco de dados chamado "**primeiro.db**" e retornar essa conexão. Abaixo, você encontra o fragmento de código que mostra como se conectar com um banco de dados no sqlite3.

```
    con = sqlite3.connect("sqlite_teste.db", check_same_thread=False)
```

   3. Crie uma variável global no seu script e atribua a ela a chamada da função criada no passo anterior.

   4. Construa uma outra função que será responsável por criar uma tabela no seu banco de dados. Essa função deve conter uma variável do tipo ***str*** com o seguinte conteúdo:

```
create table if not exists Aluno (
    id integer primary key autoincrement,
    nome text not null,
    email text not null,
    ra integer not null,
    media real not null
)
```

   Além disso, sua função deverá ter as seguintes linhas de código. Porém, substitua a variável **"contexto" pela variável global definida por você no passo 3**.

```
cursor = contexto.cursor()
cursor.execute("<sua variável str aqui>")
contexto.commit()
```

   Pesquise o que essas linhas de código fazem e crie um comentário no seu script com uma breve explicação.

5. No escopo global do seu script, chame a função criada no passo anterior e **EXECUTE O SEU CÓDIGO!** Essa função será executada e criará a tabela no banco de dados. Logo após a execução, remova a chamada dessa função para que não seja mais executada.

   Isso é necessário apenas para que o sqlite3 não tente recriar toda a estrutura do banco.

6. Logo após, no mesmo script, construa uma classe chamada ALUNO que contém os seguintes atributos: id, nome, email, ra e média.
   Essa classe deverá ter apenas o método construtor, sendo que os atributos deverá ser setados via parâmetro.

   Se necessário, pesquise como criar uma classe em Python com o método construtor recebendo variáveis por parâmetro.

7. Agora, logo abaixo da sua classe, no escopo global, crie a variável que será o seu servidor flask, como exemplo:

```
servidor = Flask(__name__)
```

8. Construa um endpoint chamado home, que deverá retornar a frase “Servidor em execução”.

9. Construa um endpoint chamado **cadastrar_aluno**. Esse endpoint receberá os dados de um aluno (nome, email, ra e média) através do corpo da requisição e será responsável por instanciar um objeto da classe Aluno com as informações recebidas.

   Um exemplo disso pode ser visto a seguir:

```
dados = request.get_json()
aluno = Aluno(dados['nome'], dados['email'], dados['ra'], dados['media'])
```

   Note que a variável **request** é do próprio flask, a qual permite acessar os dados vindos no corpo da requisição. Como os dados são tratados em *json*, podemos utilizar a função get_json() e acessá-los como se fossem dicionários.

   A variável **aluno** é um objeto da classe **Aluno** e possui seus atributos setados.

   a. Ainda no mesmo endpoind **cadastrar_aluno**, precisamos definir o script SQL responsável por inserir as informações no banco de dados. Para isso, veja um exemplo abaixo e reproduza, ajustando o que for necessário o seu código.

```
#inserindo o aluno com as informações da requisição
sql = "insert into Aluno (nome, email, ra, media) values (?, ?, ?, ?)"
cursor = contexto.cursor()
cursor.execute(sql, (aluno.nome, aluno.email, aluno.ra, aluno.media))
contexto.commit()
```

Novamente, a variável **contexto** nesse exemplo deve ser substituída pela sua variável criada no passo 3.

O que os códigos acima fazem? Inclua um comentário no código explicando.

b. Por fim, seu endpoint deverá retornar que um novo aluno foi incluído na base de dados.

10. Construa um novo endpoint, chamado **deletar_aluno**. Esse endpoint deverá receber via URL o ID do aluno a ser deletado, veja como podemos definir uma rota que faz isso:

```
@app.route("/aluno/<int:id>", methods=["DELETE"])
```

Lembre-se, precisamos que a variável id exista no parâmetro do endpoint. Após isso, você deverá criar um script SQL para deletar o aluno e executar esse script na base de dados, conforme exemplo abaixo.

```
sql = "delete from Aluno where id = (?)"
cursor = contexto.cursor()
cursor.execute(sql, (id,))
```

Por fim, você deverá retornar que o aluno foi deletado, com o status code 200.

11. **DESAFIO -** Construa um endpoint chamado **listar_aluno** que é responsável por executar um select na base de dados – na tabela Aluno, criar um dicionário com todos os alunos e suas respectivas informações e retornar esse objeto, com o status code 200.

Dica: a partir do script SQL que faz a seleção de todos os dados na tabela Aluno, podemos executar os seguintes códigos para acessar seus valores:

```
cursor = contexto.cursor()
alunos = cursor.execute(sql).fetchall() ##recupera todos os registros do banco
```

Agora, a variável alunos é uma lista de objetos aluno, sendo que cada propriedade (nome, email etc), é acessado através de um índice. Exemplo: alunos[0][0] -> Id; alunos[0][1] -> nome.

12. **DESAFIO –** Construa um endpoint chamado **filtrar_aluno**. Esse endpoint terá que receber um ID de aluno e executar uma seleção no banco de dados buscando todas as informações desse aluno (*select* com *where*).

Dica: a forma de executar uma seleção na base de dados e resgatar as informações de apenas um único aluno é:

```
cursor = contexto.cursor()
aluno = cursor.execute(sql, (id,)).fetchone() #precisamos deixar uma vírgula
após uma única propriedade (TUPLA)
```

Pesquise mais afundo o que o código *fetchone* faz e insira uma explicação no seu script.

13. **DESAFIO –** Baseado em tudo o que você fez até agora, construa um endpoint, chamado **atualizar_aluno** que seja capaz de receber dados atualizados de um aluno e que reflita essa atualização na base de dados.

    Dica: independente da informação a ser atualizada, sempre envie o Id, assim você conseguirá executar um comando de *update* na base de dados.

14. Construa um cliente, utilizando o módulo requests, para fazer uso dos endpoints definidos por você.