CRUD no Backend

Usabilidade, desenvolvimento web, mobile e jogos

Prof. Me. Gustavo Torres Custódio
gustavo.custodio@anhembi.br

Conteúdo

CRUD

CRUD

CRUD

- CRUD é uma sigla que designa quatro operações:
  - Criação (`Create`);
  - Recuperação (`Read`);
  - Atualização (`Update`);
  - Remoção (`Delete`).

Criando um Banco

- Primeiro, precisamos criar um banco de dados para a aplicação.
  - Vamos criar um banco utilizando o `sqlite3`.
  - Ele permite criar um banco de dados dentro da pasta da própria instalação.

Criando um Banco

```
1   import sqlite3
2   from flask import Flask, request, render_template
3   from flask import g
4
5   BANCO_DE_DADOS = './banco.db'
6
7   app = Flask(__name__)
8   app.config["TEMPLATES_AUTO_RELOAD"] = True
9
10  def carregar_banco():
11      # Verifica se o banco já foi carregado
12      banco = getattr(g, '_database', None)
13
14      if banco is None:
15          g._database = sqlite3.connect(BANCO_DE_DADOS)
16          banco = g._database
17      return banco
```

SQL

- A função **carregar_banco** cria o banco de dados que vai conter todas as informações adicionadas na aplicação.
  - Caso ele já não tenha sido criado.
- O banco de dados utiliza a **linguagem SQL**.
- As operações *create, update, read* e *delete* são realizadas por meio de instruções SQL.

SQL

- O SQL é uma linguagem para manipular **bancos de dados relacionais**.
  - **Bancos de Dados Relacionais** armazenam dados em um esquema de **tabelas**.
- O SQL realiza operações no banco de dados por meio de consultas.
  - As consultas contêm a operação que será realizada no banco de dados e quais tabelas e registros serão afetados.

SQL - Tabelas

- **Bancos de Dados Relacionais** armazenam dados em um esquema de **tabelas**.
  - Cada tabela possui um conjunto de linhas e colunas.
    - As linhas são itens individuais da tabela.
    - As colunas representam atributos dos itens.
  - Suponha uma tabela chamada Funcionario:

SQL - Tabelas

ID	Nome	Idade	Endereço
1	Gustavo	30	Rua das Ruas 1
2	Guilherme	25	Rua das Ruas 3

- Cada linha representa um funcionário diferente.
- Cada coluna representa uma característica do funcionário: **nome**, **idade**, etc.

SQL - Criando uma tabela

- Vamos criar a tabela funcionário mostrada.
  - Para isso precisamos definir um arquivo com um **schema inicial** chamado `schema.sql`.
  - O arquivo vai conter uma instrução SQL para criar a tabela:

```sql
CREATE TABLE IF NOT EXISTS Funcionario (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome VARCHAR (20),
    idade INTEGER,
    endereco VARCHAR (100)
);
```

SQL - Criando uma tabela

- Agora vamos criar uma rota para gerar a tabela.

```
1    def iniciar_tabela():
2        banco = carregar_banco()
3        script_criacao = open('schema.sql').read()
4        banco.executescript(script_criacao)
5
6    @app.route('/')
7    def index():
8        banco = carregar_banco()
9        iniciar_tabela()
10       return "<h1>TABELA CRIADA</h1>"
```

- Após acessar essa rota, um arquivo chamado banco.db contendo a tabela Funcionario será criada.

SQL - Criando uma tabela

- A tabela criada chamada `Funcionario` possui 4 colunas:
  - Um id que é **incrementado automaticamente** (para cada item adicionado na tabela, atribua um id e some 1).
  - Um nome que é um campo que permite até 30 caracteres (VARCHAR).
  - Idade que é um campo numérico inteiro.
  - Endereço que é um campo que permite até 100 caracteres.

SQL - Criando uma tabela

- Podemos ver se a tabela foi criada com sucesso usando um leitor de arquivos db.
  - No caso, o arquivo banco.db
- O site https://inloop.github.io/sqlite-viewer/ permite ler arquivos db.

SQL - Operações

- O SQL possui quatro operações básicas:
  - **Insert** (Create);
  - **Select** (Read);
  - **Update**;
  - **Delete**.

INSERT

- A operação Insert cria um ou mais registros novos no banco de dados.
  - Ela possui a sintaxe:

```
1    INSERT INTO Tabela
2        (atributo1, atributo2, ...)
3    VALUES ('valor1', 'valor2', ...)
```

INSERT

- Vamos adicionar um formulário para cadastrar informações de usuário.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Inserir Funcionário</title>
</head>
<body>
    <h2>Inserir</h2>
    <form action="/add" method="post">
        <input type="text" name="nome" id="nome" placeholder="Nome">
            <br> <br>
        <input type="number" name="idade" id="idade" placeholder="Idade">
            <br> <br>
        <input type="text" name="endereco" id="endereco" placeholder="Endereço">
            <br> <br>
        <input type="submit">
    </form>
</body>
</html>
```

INSERT

- Adicionamos no arquivo Flask o código para atender requisições POST.

```
@app.route('/add', methods = ['POST',])
def salvar_info():
    banco = carregar_banco()
    cur = banco.cursor()

    nome = request.form["nome"]
    idade = request.form["idade"]
    endereco = request.form["endereco"]

    cur.execute("""
        INSERT INTO Funcionario
              (nome, idade, endereco)
        VALUES (?, ?, ?) """,
        (nome, idade, endereco)
    )
    banco.commit()
    banco.close()
    return "Nome: %s, Idade: %s, Endereço: %s" % (nome, idade, endereco)

@app.route('/create') # Abrir o form
def formulario_insercao():
    return render_template('formulario_insere.html')
```

SELECT

- Após inserir um elemento no banco de dados, precisamos consultá-lo para verificar se o mesmo foi inserido corretamente.
  - Para isso utilizamos a instrução SELECT.
  - Sintaxe:

```
1    SELECT atributo1, atributo2, ...
2    FROM Tabela
3    WHERE condição;
```

SELECT

- A condição do `WHERE` pode, por exemplo, indicar um funcionário com um ID específico.
- Se desejarmos selecionar todos os atributos, podemos fazer isso com o asterisco (*).
- Exemplo:
  - `SELECT * ... FROM Tabela` seleciona todas as colunas.

SELECT

- Vamos criar um formulário para buscar o usuário pelo seu id.

```
<form action="/ver" method="POST">
    <fieldset id="dados">
        <legend>Busca</legend>
        <p>
            <label for="id">Identificação:</label>
            <input type="number" name="id" id="id">
        </p>
    </fieldset>
    <button type="submit">OK</button>
</form>
```

SELECT

- Adicionamos uma rota POST para receber o número de usuário buscado.

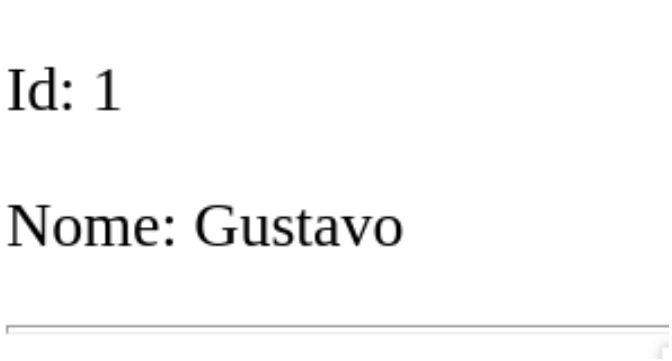
```
@app.route('/ver', methods = ['POST',])
def ver_info():
    banco = carregar_banco()
    cur = banco.cursor()

    id = request.form["id"]
    resultados = cur.execute("""
        SELECT id, nome
        FROM Funcionario
        WHERE id = ?
        """, (id)
    ).fetchall()
    banco.commit()
    banco.close()
    return render_template("resultados_busca.html",
                           resultados=resultados)

@app.route('/read')
def formulario_busca():
    return render_template('formulario_busca.html')
```

Busca

- Agora criamos o template para mostrar os resultados da busca.
  - `templates/resultados_busca.html`.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width">
        <title>Resultados busca</title>
    </head>
    <body>
        {% for res in resultados %}
        <div>
            <p>Id: {{res[0] }}</p>
            <p>Nome: {{res[1] }}</p>
        </div>
        <hr>
        {% endfor %}
    </body>
</html>
```

Busca



- O `for` é um recurso do template para renderizar múltiplos elementos

Busca

- Vamos criar agora uma rota que lista todos os funcionários no banco de dados na tela

```
1    @app.route('/listarfuncionarios')
```

Busca

Id: 1

Nome: Gustavo

Id: 2

Nome: Joao

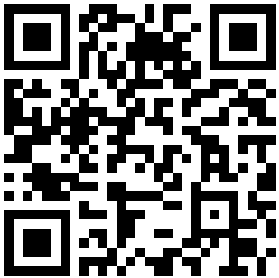
Id: 3

Nome: Gustavo

Resumo

- Criamos um servidor em Flask.
  - O servidor espera o acesso na respectiva porta alocada.
  - Conseguimos inserir e consultar informações de usuário.
  - As próximas operações a serem vistas são o UPDATE e o DELETE.

Conteúdo



https://gustavotcustodio.github.io/usabilidade.html

Obrigado

gustavo.custodio@anhembi.br