



- O que é CRUD?
- Operações CRUD no BD
- Criando WebServices com CRUD
- E agora?

Python

CRUDs e WebServices com Python

Usando Flask

01 O que é CRUD?

- Um dos termos bastante utilizados na área de Desenvolvimento de *Software* é CRUD. Mas o que é CRUD?
- Antes de “traduzir” o que significa, observe a tela abaixo.
 - Basicamente, temos uma listagem de usuários (são 5 linhas, uma com Pablo e outras 4 linhas)
 - Em cada uma das linhas, temos um botão **Edit** que altera os dados de um usuário e um botão **Delete** que exclui o usuário.

Add Record **OK**

ID ▲	Username	Fullname	Date		
1	Pablo	Pablo Picasso	1881-1973	Edit	Delete
2	Johannes	Johannes Vermeer	1632-1675	Edit	Delete
3	Claude	Claude Monet	1840-1926	Edit	Delete
4	Rembrandt	Rembrandt van Rijn	1606-1669	Edit	Delete
10	Leonardo	Leonardo da Vinci	1452-1519	Edit	Delete

Previous **1** **2** **Next**

- E, caso seja necessário adicionar um novo usuário, clicamos no botão **Add Record** e a tela a seguir é apresentada. Nesta segunda tela, preenchemos os dados de um novo usuário e ao clicar no botão **Save**, que um novo usuário é criado e voltamos a tela anterior que, ao invés de 5 usuários, agora terá 6.

Save **Cancel**

ID	1
Username	Pablo
Fullname	Pablo Picasso
Date	1881-1973
Active	<input checked="" type="checkbox"/>
Country	Spain
Role	admin

- Esta funcionalidade inteira (listar, inserir, excluir e editar), chamamos de CRUD. Ou seja, o que acabamos de explicar foi a funcionalidade de CRUD de usuários.
- Mas, afinal, o que é CRUD? CRUD é um acrônimo para C (Create, criar, inserir), R (read, ler), U (Update, atualizar) e D (Delete, Excluir).



02 Operações CRUD no BD

- O pré-requisito é termos o Python e o PIP instalados, após isto, execute o seguinte comando no terminal do PyCharm:
 - `python3 -m pip install mysql-connector-python`
- Para isso, no PyCharm, clique em **Terminal** e informe o comando acima, como apresentado a seguir.

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject1 - main.py
pythonProject1 main.py
Project main.py
Terminal: Local +
PS C:\Users\teste\config\PycharmProjects\pythonProject1
> python3 -m pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.0.29-cp310-cp310-win_amd64.whl (7.7 MB)
Collecting protobuf>=3.0.0
  Downloading protobuf-3.20.1-cp310-cp310-win_amd64.whl
Version Control Python Packages TODO Python Console Problems Terminal Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built shared indexes (1 hour ago) 1:1 CRLF UTF-8 6 spaces Python 3.9 (pythonProject1)
```

- O próximo passo (e bem sugerido) é criar um módulo para guardar as operações no banco de dados. Criaremos o arquivo operacoesbd.py com os seguintes métodos:

- Método para abrir uma conexão no BD.

```
1 import mysql.connector
2
3 def abrirBancoDados(host,user,password,database):
4     return mysql.connector.connect(host,user,
5                                     password,database)
```

- Do mesmo jeito que abrimos uma conexão, precisamos encerrá-la após o uso com o método a seguir.

```
7 def encerrarBancoDados(connection):
8     connection.close()
```

- Note que, no método que abre a conexão, este retorna uma conexão. Enquanto no método que encerra o banco de dados, recebe a conexão como parâmetro e a encerra.
- Na sequência, abaixo temos o método que realiza inserção de dados no banco de dados. Note que, precisamos especificar a conexão, a consulta (SQL) e os dados a serem inseridos.

```
10 def insertNoBancoDados(connection, sql, dados):
11     cursor = connection.cursor()
12     cursor.execute(sql, dados)
13     connection.commit()
14     id = cursor.lastrowid
15     cursor.close()
16     return id
```

- Para que o exemplo funcione, vamos criar uma tabela chamada produtos

```
1 • create table produtos (
2     id int not null auto_increment primary key,
3     nome varchar(100) not null,
4     descricao varchar(100) not null,
5     preco float not null
6 );
```

- Uma vez criada a tabela e o método acima, podemos usá-los dentro do nosso main.py, conforme exemplo a seguir.
 - Linha 1: Import do módulo criado antes
 - Linha 3: Criamos a conexão a ser usada
 - Linha 5: Informamos a consulta de INSERT
 - Linha 6: Informamos os dados
 - Linha 7: Chamamos o SQL e os dados usando o método criado
 - Linha 9: Encerramos a conexão criada na linha 3

```
1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "INSERT INTO produtos(nome,descricao,preco) VALUES (%s, %s, %s)"
6 dados = ('Banana','Bana Pacovan',8)
7 operacoesbd.insertNoBancoDados(conn,sql,dados)
8
9 operacoesbd.encerrarBancoDados(conn)
```

- No método a seguir, apresentamos todos os registros no banco de dados.
 - Devemos informar a conexão e o SQL da consulta

```
18 def listarBancoDados(connection,sql):
19     cursor = connection.cursor()
20     cursor.execute(sql)
21     results = cursor.fetchall()
22     cursor.close()
23     return results
```

- Como faço para chamar o método acima? O exemplo está a seguir.
 - Linha 5: Consulta SQL
 - Linha 6: Chamamos o método criado informando a conexão e o SQL (linha 5)
 - Pegamos o resultado e atribuímos a variável resultado
 - Linha 7: Imprimimos o resultado
 - Linha 11: Encerramos a conexão da linha 3

```
1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "SELECT * FROM PRODUTOS"
6 resultado = operacoesbd.listarBancoDados(conn,sql)
7
8 for elemento in resultado:
9     print(elemento)
10
11 operacoesbd.encerrarBancoDados(conn)
```

- Seguindo, agora vamos fazer a atualização do BD com o método a seguir.
- Note que, este método retorna a quantidade de linhas que foram atualizadas por meio da consulta.

```
25 def atualizarBancoDados(connection,sql, dados):
26     cursor = connection.cursor()
27     cursor.execute(sql, dados)
28     connection.commit()
29     linhasAfetadas = cursor.rowcount
30     cursor.close()
31     return linhasAfetadas
```

- Complementarmente, o código a seguir realiza o uso do método acima.
 - Linha 7: Chamamos o método criado informando a consulta (linha 5) e os dados (linha 6)

```
1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "UPDATE produtos SET nome = %s WHERE id = %s"
6 dados = ('Banana de Cozinhar',1)
7 linhasAfetadas = operacoesbd.atualizarBancoDados(conn,sql,dados)
8 print('Linhas afetadas: ',linhasAfetadas)
9
10 operacoesbd.encerrarBancoDados(conn)
```

- Por fim, vamos excluir dados do BD, né? O método a seguir faz isto. Note que, é bem parecido com o método de atualizar.

```

33 def excluirBancoDados(connection,sql,dados):
34     cursor = connection.cursor()
35     cursor.execute(sql, dados)
36     connection.commit()
37     linhasAfetadas = cursor.rowcount
38     cursor.close()
39     return linhasAfetadas

```

- Complementarmente, o código a seguir mostra como chamar o método acima.

```

1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "DELETE FROM produtos WHERE id = %s"
6 data = (2,)
7 linhasAfetadas = operacoesbd.excluirBancoDados(conn,sql,dados)
8 print('Linhas afetadas: ',linhasAfetadas)
9
10 operacoesbd.encerrarBancoDados(conn)

```

03 Criando WebServices que Façam as Operações CRUD

- Na seção anterior, lembram que criamos um módulo com as operações de um banco de dados? Enfim, agora nesta seção, vamos criar o nosso webservice que usa este módulo e possui as seguintes operações:

- **Linha 10:** Método que recebe requisições GET
 - Como todo método dentro de classe, o primeiro parâmetro é um self
 - Note que, na linha 13 colocamos a consulta, que é enviada na linha 14. Na linha 15 encerramos a conexão.
 - Por fim, na linha 16, convertemos o resultado da linha 14 em formato JSON
- **Linha 18:** Método que recebe requisições POST
 - Linhas 21,22 e 23: Pegamos os parâmetros enviados no body (corpo) da requisição
 - Na linha 25 temos a consulta, que é enviada na linha 27.
- **Os dois métodos GET e POST (linhas 10 e 17) estão dentro de uma classe chamada Produtos**
 - Usamos esta classe para encapsular os 2 métodos
 - Posteriormente, na linha 62, informamos: “pegue os métodos da classe Produtos e faça eles atenderem a requisições em /produtos”
- De maneira análoga, na linha 43 criamos uma classe para abrigar métodos que recebem parâmetros por URL (tipo, GET ou DELETE passando o ID)

```

1 from flask import Flask, request, jsonify
2 from flask_restful import Resource, Api
3 import operacoesbd
4
5 app = Flask(__name__)
6 api = Api(app)
7
8 class Produtos(Resource):
9
10     def get(self):
11         conn = operacoesbd.abrirBancoDados('localhost',
12                                             'root', '12345', 'testes')
13         sql = 'SELECT * FROM PRODUTOS'
14         resultado = operacoesbd.listarBancoDados(conn,sql)
15         operacoesbd.encerrarBancoDados(conn)
16         return jsonify(resultado)
17
18     def post(self):
19         conn = operacoesbd.abrirBancoDados('localhost',
20                                             'root', '12345', 'testes')
21         nome = request.json['nome']
22         descricao = request.json['descricao']
23         preco = request.json['preco']
24
25         sql = 'INSERT INTO produtos(nome,descricao,preco) values (%s,%s,%s)'
26         dados = (nome,descricao,preco)
27         operacoesbd.insertNoBancoDados(conn,sql,dados)
28         operacoesbd.encerrarBancoDados(conn)
29         return {"status": "success"}
30
31 class ProdutosPorId(Resource):
32
33     def delete(self, id):
34         conn = operacoesbd.abrirBancoDados('localhost',
35                                             'root', '12345', 'testes')
36         sql = 'DELETE PRODUTOS where id = %s'
37         dados = (id)
38         operacoesbd.excluirBancoDados(conn,sql,dados)
39         operacoesbd.encerrarBancoDados(conn)
40         return {"status": "success"}
41
42     def get(self, id):
43         conn = operacoesbd.abrirBancoDados('localhost',
44                                             'root', '12345', 'testes')
45         sql = 'SELECT * FROM PRODUTOS where id = ' + id
46         resultado = operacoesbd.listarBancoDados(conn,sql)[0]
47         operacoesbd.encerrarBancoDados(conn)
48         return jsonify(resultado)
49
50 api.add_resource(Produtos, '/produtos')
51 api.add_resource(ProdutosPorId, '/produtos/<id>')
52
53 if __name__ == '__main__':
54     app.run()
55
56

```

- Por fim, os métodos get (por ID) e delete (por ID) estão nas linhas 45 e 54. Na linha 63, adicionamos esta nova classe, que vai atender a requisições em '/produtos/<id>'.

04 E agora? Acabou?

- Pior que sim ☹ Entretanto você pode seguir estudando, concordas?
- Uma recomendação final é que leiam o **PEP 8 - Style Guide for Python Code**, pois foi um assunto negligenciado durante esta série, pois o objetivo era “startar” sua carreira de desenvolvedor usando a linguagem Python.
- Espero que esta série de conteúdos tenham sido de bastante utilidade para você!
- Abraços cordiais, Daniel Abella.