



Curso Programação Backend com Python

Exercício REST e Database

© Cleuton Sampaio 2021

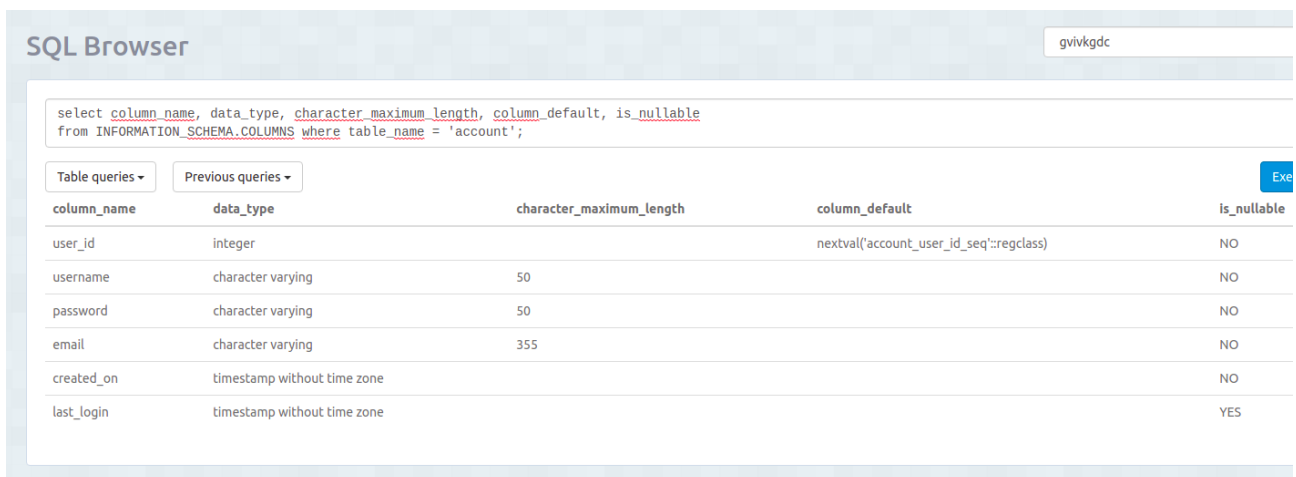
Sumário

Exercício	3
Dicas	4
Coloque a URL em uma variável de ambiente	4

Exercício

Vamos fazer um exercício completo: Um servidor REST que faz todas as operações de CRUD + listagem dos registros existentes em uma tabela no ElephantSQL.

Eu tenho uma tabela no ElephantSQL chamada “account”, cujo layout é:



The screenshot shows the SQL Browser interface. At the top, there's a search bar with 'gvivkgdc'. Below it, a SQL query is entered: `select column_name, data_type, character_maximum_length, column_default, is_nullable from INFORMATION_SCHEMA.COLUMNS where table_name = 'account';`. Below the query, there are two tabs: 'Table queries' and 'Previous queries'. The 'Table queries' tab is active, showing a table with 5 columns: `column_name`, `data_type`, `character_maximum_length`, `column_default`, and `is_nullable`. The table contains 7 rows of data for the 'account' table.

column_name	data_type	character_maximum_length	column_default	is_nullable
user_id	integer		nextval('account_user_id_seq'::regclass)	NO
username	character varying	50		NO
password	character varying	50		NO
email	character varying	355		NO
created_on	timestamp without time zone			NO
last_login	timestamp without time zone			YES

Ela tem os campos:

- user_id integer
- username character varying 50
- password character varying 50
- email character varying 355
- created_on timestamp without time zone
- last_login timestamp without time zone

Se quiser, crie uma tabela igual, ou então crie sua própria tabela com seus próprios campos. No meu caso, “user_id” é a chave primária.

Crie um servidor e um cliente REST.

O servidor deverá atender:

- **GET:** Uma só conta identificada, exemplo: `GET /account/1`
- **GET:** Listar todas as contas, exemplo: `GET /accounts`
- **POST:** Criar uma conta, exemplo: `POST /account`
- **PUT:** Modificar uma conta, exemplo: `PUT /account/1`
- **DELETE:** Apagar uma conta, exemplo: `DELETE /account/1`

O Cliente deverá exercitar todos esses métodos.

Dicas

Coloque a URL em uma variável de ambiente

Proteja suas informações evitando colocá-las dentro do código-fonte. Temos o pacote “os” que permite ler variáveis de ambiente. Crie uma variável de ambiente com a URL do ElephantSQL e use dentro do código. Veja este meu exemplo:

- 1) Criando a variável de ambiente:
 - o MS Windows: `set URL=postgres://...`
 - o Linux: `export URL=postgres://...`
- 2) Importe o pacote “os”: `import os`
- 3) Obtenha o valor: `variavel=os.getenv("URL")`

O método **getenv()** retorna **None**, caso a variável não exista. Se quiser, pode informar um valor default, caso a variável não exista no ambiente:

```
os.getenv('variável', valor_default)
```

Separe as funções REST das funções de banco de dados. Vejamos o meu exemplo de modificar a conta (update):

Servidor:

```
# Função de banco de dados:
def modify_account(account):
    # Ordem: username, password, email. O último é o user_id
    sql = """
        update account
            set username = %s,
              password = %s,
              email = %s
            where user_id = %s
    """
    cursor = db.cursor()
    try:
        cursor.execute(sql, (account['username'], \
                             account['password'], \
                             account['email'], \
                             account['user_id'],)
                        )
    except:
        raise
    finally:
        db.commit()
        cursor.close()
...

```

```
# Método REST que a invoca:
@app.route('/account/<id>', methods=['PUT'])
def replace_usuario(id):
    try:
        account = request.json
        existing = get_one_account(account['user_id'])
        if existing:
            modify_account(account)
            return '', 204
        response = make_response(jsonify({'status': 'Not found'}), 404)
        return response
    except Exception as e:
        print(e)
        raise
    db.close()
```

É importante:

- Sempre utilizar blocos **try**;
- Usar **raise** para levantar a exceção, caso não tenha como tratá-la;
- Sempre fechar o cursor no **finally**;
- Em caso de atualizações do banco, sempre usar **commit** no banco, após a atualização bem sucedida. Caso contrário, você não encerra a transação no SGBD;
- A relação entre o **SQL** e os **parâmetros** é sequencial. Você passa uma **tupla** com parâmetros no método **execute()**, e eles devem corresponder à ordem expressa no string do comando SQL.

Há um arquivo “**exercicioREST.zip**” com a minha versão deste exercício. Sugiro que tente fazer. Tente fazer a consulta simples e múltipla, depois a inserção e deleção.

