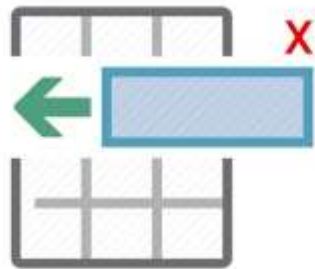


# APLICAR AS FUNCIONALIDADES PARA INSERÇÃO, REMOÇÃO E ATUALIZAÇÃO DE REGISTROS EM TABELAS

## INSERÇÃO DE DADOS EM TABELA

Neste módulo, vamos aprender a inserir, alterar e remover registros de tabelas. Para inserir registros em uma tabela, utilizamos o comando INSERT INTO, do SQL.



Para ilustrar a utilização desse comando, vamos inserir o seguinte registro na tabela Pessoa.

*CPF: 12345678900*

*Nome: João*

*Data de Nascimento: 31/01/2000*

*Usa óculos: Sim (True)*

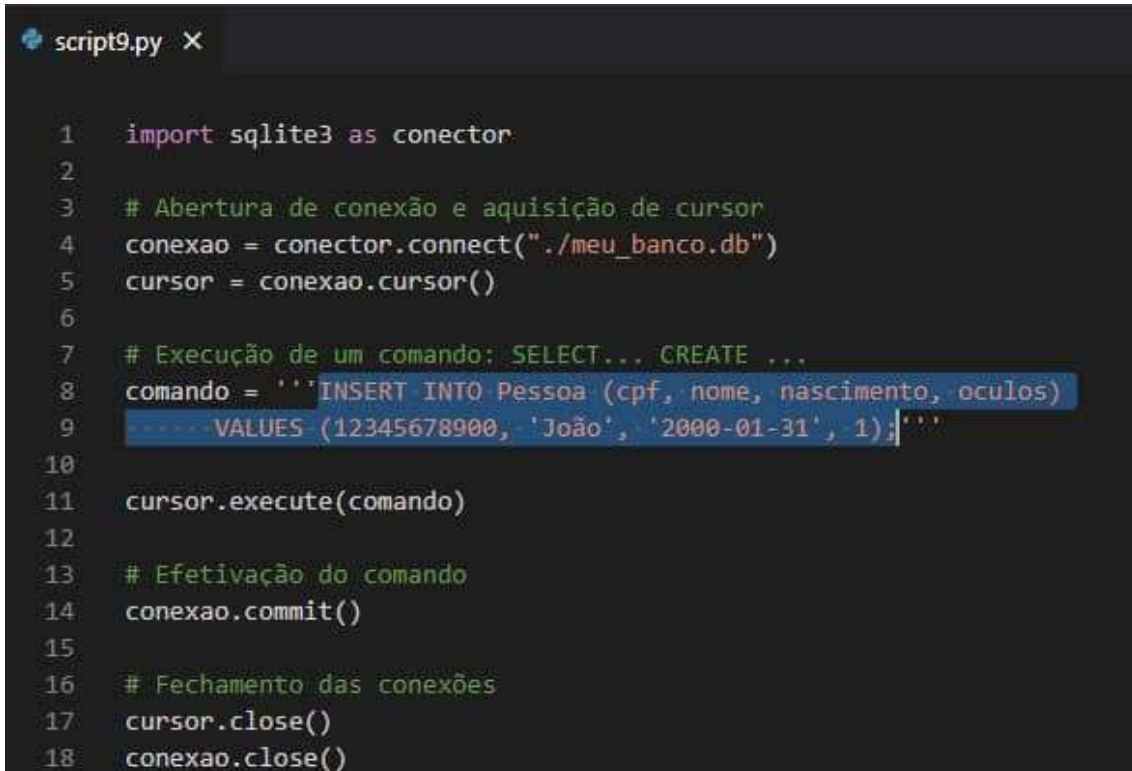
O Comando SQL para inserção desses dados é o seguinte:

```
INSERT INTO Pessoa (cpf, nome, nascimento, olhos)  
VALUES (12345678900, 'João', '2000-01-31', 1);
```

Observe que alteramos a formatação da data para se adequar ao padrão de alguns bancos de dados, como MySQL e PostgreSQL. Para o SQLite será indiferente, pois o tipo DATE será convertido por afinidade para NUMERIC, que pode ser de qualquer classe. Na prática, será convertido para classe TEXT.

Além disso, utilizamos o valor “1” para representar o booleano True. Assim como o DATE, o BOOLEAN será convertido para NUMERIC, porém, na prática, será convertido para classe INTEGER.

Confira como ficou o script para inserção dos dados, Figura 12.

The image shows a code editor window titled 'script9.py'. It contains 18 lines of Python code. Lines 1-5 set up the connection to a SQLite database named 'meu\_banco.db'. Line 7 is a comment about executing a command. Lines 8-9 define a SQL INSERT statement for a table named 'Pessoa'. Line 11 executes the command. Line 13 is a comment about committing the command. Line 14 calls 'commit()' on the connection object. Lines 16-18 close the cursor and the connection. The SQL command in lines 8-9 is highlighted with a blue background.

```
1 import sqlite3 as conector
2
3 # Abertura de conexão e aquisição de cursor
4 conexao = conector.connect("./meu_banco.db")
5 cursor = conexao.cursor()
6
7 # Execução de um comando: SELECT... CREATE ...
8 comando = '''INSERT INTO Pessoa (cpf, nome, nascimento, olhos)
9 .....VALUES (12345678900, 'João', '2000-01-31', 1);'''
10
11 cursor.execute(comando)
12
13 # Efetivação do comando
14 conexao.commit()
15
16 # Fechamento das conexões
17 cursor.close()
18 conexao.close()
```

Figura: 12.

Após se conectar ao banco e obter o cursor, criamos a string com o comando para inserir um registro na tabela Pessoa nas linhas 8 e 9.

Na linha 11, executamos esse comando com a execução do commit na linha 14.

Ao final do script, fechamos o cursor e a conexão.

Observe na figura 13 como ficou nossa tabela pelo DB Browser for SQLite.

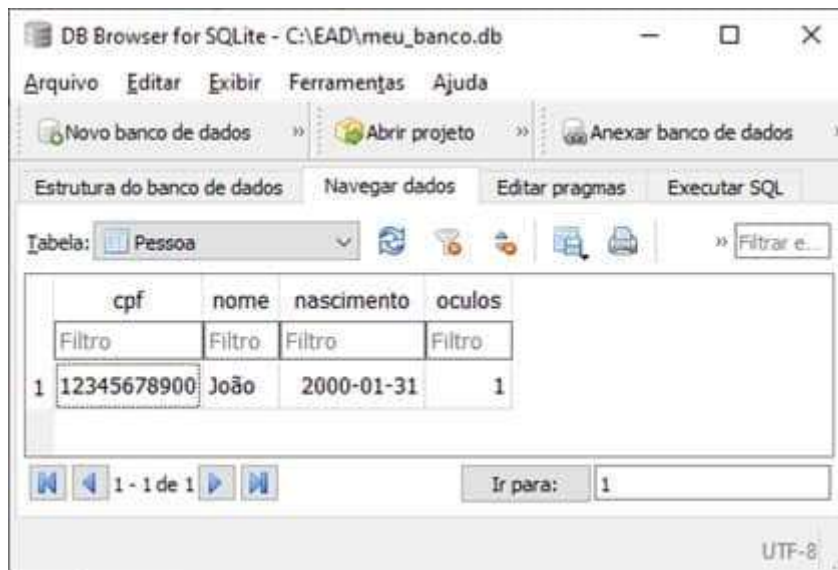


Figura: 13.

## INSERÇÃO DE DADOS EM TABELA COM QUERIES DINÂMICAS

É muito comum reutilizarmos uma mesma string para inserir diversos registros em uma tabela, alterando apenas os dados a serem inseridos.

Para realizar esse tipo de operação, o método `execute`, da classe `Cursor`, prevê a utilização de parâmetros de consulta, que é uma forma de criar comandos SQL dinamicamente.

### Comentário

De uma forma geral, as APIs **sqlite3**, **psycopg2** e **PyMySQL** fazem a concatenação da string e dos parâmetros antes de enviar ao banco de dados.

Essa concatenação é realizada de forma correta, evitando brechas de segurança, como SQL Injection, e convertendo os dados para os tipos e formatos esperados pelo banco de dados.

Como resultado final, temos um comando pronto para ser enviado ao banco de dados.

Para indicar que a string de um comando contém parâmetros que precisam ser substituídos antes da sua execução, utilizamos delimitadores. Esses delimitadores também estão previstos na PEP 249 e podem ser: "?", "%s", entre outros.

Na biblioteca do SQLite, utilizamos o delimitador "?".

Para ilustrar a utilização do delimitador “?” em SQLite, considere o comando a seguir:

```
>>> comando = "INSERT INTO tabela1 (atributo1, atributo2) VALUES (?, ?);"
```

Esse comando indica que, ao ser chamado pelo método `execute`, devemos passar dois parâmetros, um para cada interrogação. Esses parâmetros precisam estar em um iterável, como em uma tupla ou lista.

Veja a seguir como poderia ficar a chamada do método `execute` para esse comando:

```
>>> cursor.execute(comando, ("Teste", 123))
```

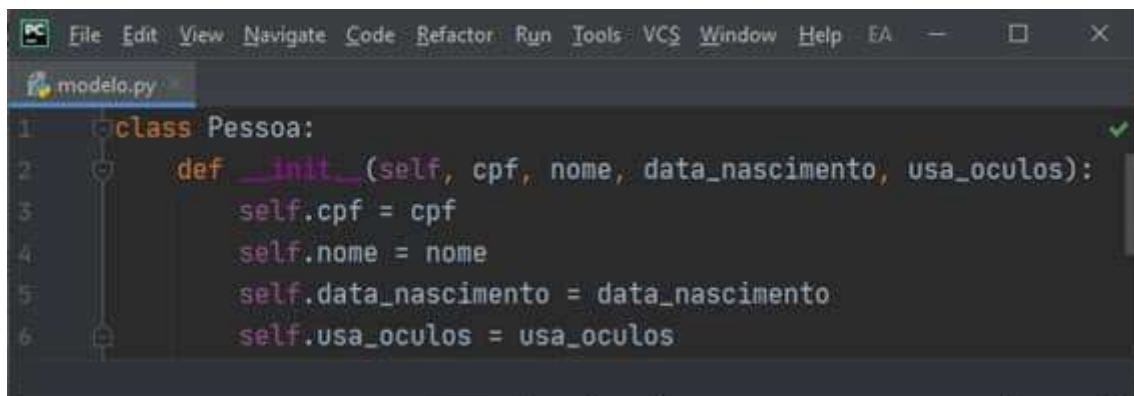
A partir da string e da tupla, é montado o comando final, que é traduzido para:

```
"INSERT INTO tabela1 (atributo1, atributo2) VALUES ('Teste', 123);"
```

A concatenação é feita da forma correta para o banco de dados em questão, aspas simples para textos e números sem aspas.

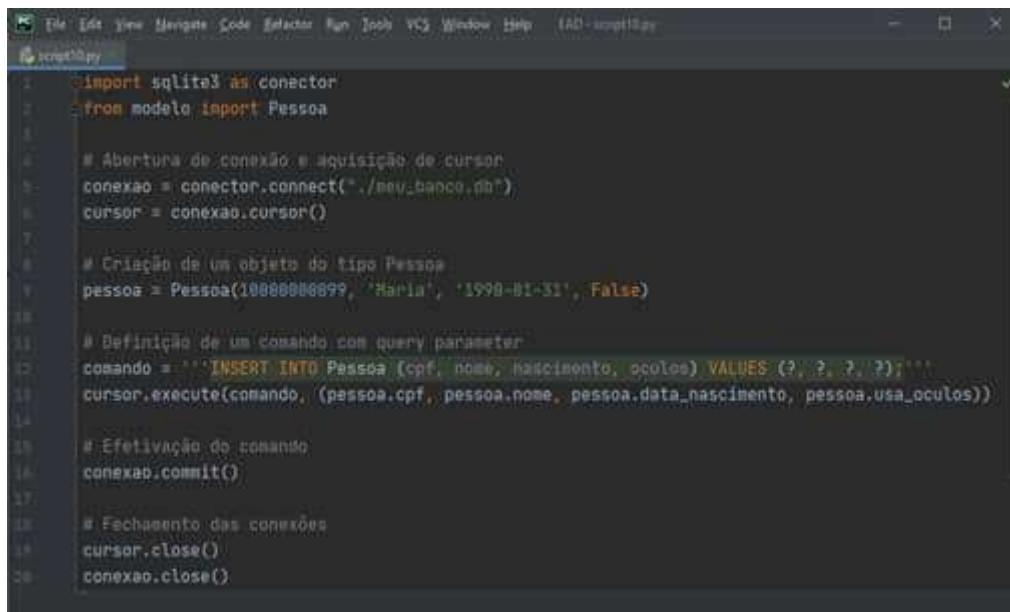
No exemplo a seguir, vamos detalhar a utilização de parâmetros dinâmicos, porém, antes, vamos definir uma classe chamada `Pessoa`, com os mesmos atributos da nossa entidade `Pessoa`.

Clique nas figuras abaixo.

A screenshot of a code editor window titled 'modelo.py'. The editor shows the definition of a Python class named 'Pessoa'. The class has an '\_\_init\_\_' method that takes four parameters: 'self', 'cpf', 'nome', 'data\_nascimento', and 'usa\_olhos'. Inside the method, each parameter is assigned to a corresponding attribute of 'self'. The code is as follows:

```
1 class Pessoa:
2     def __init__(self, cpf, nome, data_nascimento, usa_olhos):
3         self.cpf = cpf
4         self.nome = nome
5         self.data_nascimento = data_nascimento
6         self.usa_olhos = usa_olhos
```

A definição dessa classe pode ser vista no script `modelo.py`. Foi retirado a referência para a figura.

A screenshot of a code editor window titled 'script10.py'. The code is written in Python and demonstrates how to connect to a SQLite database, create a cursor, insert a record into a table named 'Pessoa', and then close the connection and cursor. The code is as follows:

```
1 import sqlite3 as conector
2 from modelo import Pessoa
3
4 # Abertura de conexão e aquisição de cursor
5 conexao = conector.connect("./meu_banco.db")
6 cursor = conexao.cursor()
7
8 # Criação de um objeto do tipo Pessoa
9 pessoa = Pessoa(10000000099, 'Maria', '1998-01-31', False)
10
11 # Definição de um comando com query parameter
12 comando = 'INSERT INTO Pessoa (cpf, nome, nascimento, oculos) VALUES (?, ?, ?, ?);'
13 cursor.execute(comando, (pessoa.cpf, pessoa.nome, pessoa.data_nascimento, pessoa.usa_olhos))
14
15 # Efetivação do comando
16 conexao.commit()
17
18 # Fechamento das conexões
19 cursor.close()
20 conexao.close()
```

Observe que temos os mesmos atributos que a entidade equivalente, mas com nomes diferentes. Fizemos isso para facilitar o entendimento dos exemplos deste módulo. Confira agora o script10.

Primeiro, importamos o conector na linha 1 e, na linha 2, importamos a classe Pessoa. Ela será utilizada para criar um objeto do tipo Pessoa.

Nas linhas 5 e 6, conectamos ao banco de dados e criamos um cursor.

Na linha 9, utilizamos o construtor da classe Pessoa para criar um objeto com os seguintes atributos: CPF: 10000000099, Nome: Maria, Data de Nascimento: 31/01/1990 e Usa óculos: Não (False). O valor False será convertido para 0 durante a execução do método execute.

Na linha 12, definimos a string que conterá o comando para inserir um registro na tabela Pessoa. Observe como estão representados os valores dos atributos! Estão todos com o delimitador representado pelo caractere interrogação (!)?

Como dito anteriormente, isso serve para indicar ao método execute que alguns parâmetros serão fornecidos, a fim de substituir essas interrogações por valores.

Na linha 13, chamamos o método execute utilizando, como segundo argumento, uma tupla com os atributos do objeto pessoa. Cada elemento dessa tupla irá ocupar o lugar de uma interrogação, respeitando a ordem com que aparecem na tupla.

O comando final enviado ao banco de dados pelo comando execute foi:

```
INSERT INTO Pessoa (cpf, nome, nascimento, olhos)
VALUES (10000000099, 'Maria', '1990-01-31', 0);
```

Nas linhas 16, efetivamos o comando utilizando o método commit.

Nas linhas 18 e 19, fechamos o cursor e a conexão.

## Dica

Nem todos os conectores utilizam o mesmo caractere como delimitador. Os conectores psycopg2, do PostgreSQL, e PyMySQL, do MySQL, utilizam o “%s”. É necessário ver a documentação de cada conector para verificar o delimitador correto!

# INSERÇÃO DE DADOS EM TABELA COM QUERIES DINÂMICAS E NOMES

Além da utilização do caractere “?” como delimitador de parâmetros, o **sqlite3** também possibilita a utilização de argumentos nomeados.

A utilização de argumentos nomeados funciona de forma similar à chamada de funções utilizando os nomes dos parâmetros.

Nessa forma de construção de queries dinâmicas, ao invés de passar uma tupla, devemos passar um dicionário para o método execute. Ele será utilizado para preencher corretamente os valores dos atributos.

A utilização de argumentos nomeados nos permite utilizar argumentos sem a necessidade de manter a ordem.

Para ilustrar a utilização dos argumentos nomeados em **SQLite**, considere o comando a seguir:

```
>>> comando = INSERT INTO tabela1 (atributo1, atributo2) VALUES (:atrib1, :atrib2);
```

Esse comando indica que ao ser chamado pelo método execute, devemos passar um dicionário com duas chaves, sendo uma “atrib1” e outra “atrib2”. Observe que há dois pontos (“:”) antes do argumento nomeado!

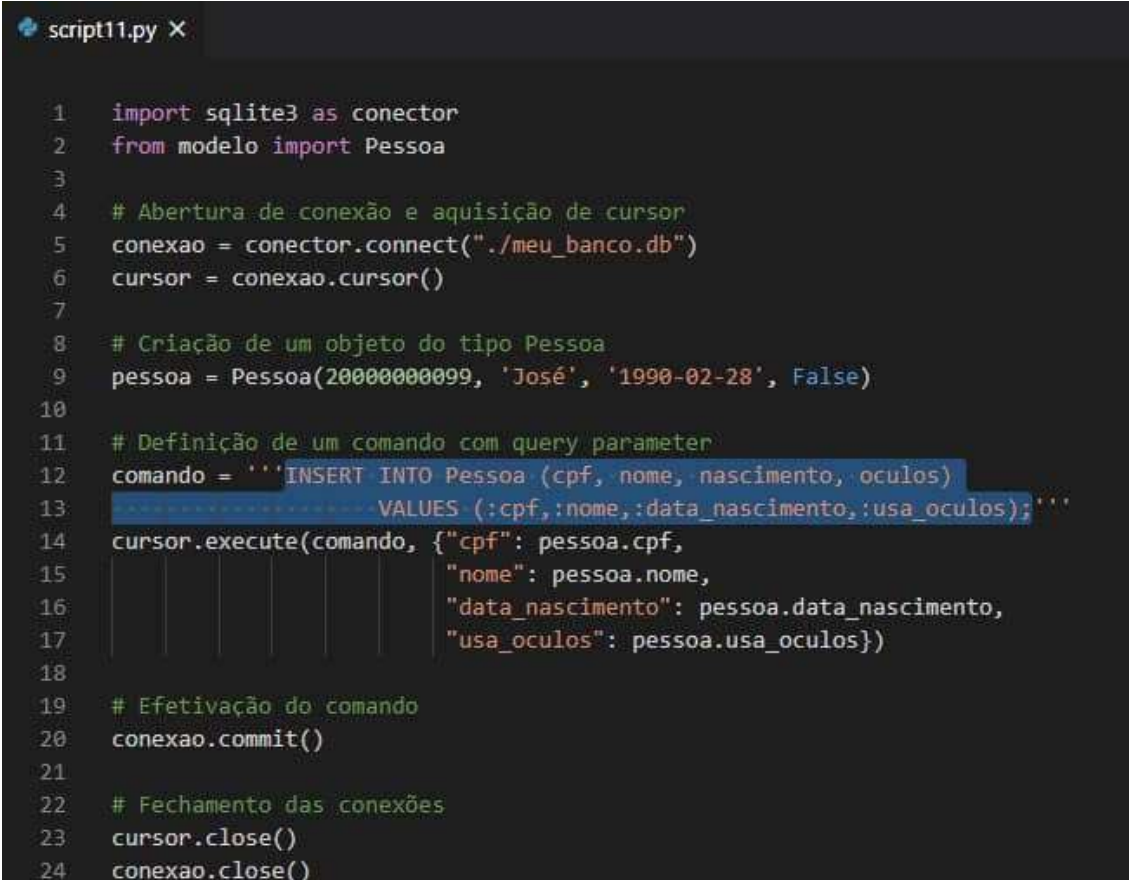
Veja a seguir como poderia ficar a chamada do método execute para esse comando:

```
>>> cursor.execute(comando, {"atrib1": "Teste", "atrib2": 123})
```

A partir da string e do dicionário é montado o comando final, que é traduzido para:

```
INSERT INTO tabela1 (atributo1, atributo2) VALUES ('Teste', 123);
```

Observe o exemplo da Figura 14, onde vamos criar um script similar ao anterior, no qual vamos utilizar novamente a classe Pessoa, porém o comando para inserir um registro no banco de dados utiliza os argumentos nomeados.



```
script11.py X

1  import sqlite3 as conector
2  from modelo import Pessoa
3
4  # Abertura de conexão e aquisição de cursor
5  conexao = conector.connect("./meu_banco.db")
6  cursor = conexao.cursor()
7
8  # Criação de um objeto do tipo Pessoa
9  pessoa = Pessoa(20000000099, 'José', '1990-02-28', False)
10
11 # Definição de um comando com query parameter
12 comando = '''INSERT INTO Pessoa (cpf, nome, nascimento, olhos)
13 .....VALUES (:cpf, :nome, :data_nascimento, :usa_olhos);'''
14 cursor.execute(comando, {"cpf": pessoa.cpf,
15                           "nome": pessoa.nome,
16                           "data_nascimento": pessoa.data_nascimento,
17                           "usa_olhos": pessoa.usa_olhos})
18
19 # Efetivação do comando
20 conexao.commit()
21
22 # Fechamento das conexões
23 cursor.close()
24 conexao.close()
```

Figura: 14.

Nas linhas 5 e 6, conectamos ao banco de dados e criamos um cursor.

Na linha 9, utilizamos o construtor da classe Pessoa para criar um objeto com os seguintes atributos: CPF: 20000000099, Nome: José, Data de Nascimento: 28/02/1990 e Usa óculos: Não (False).

Nas linhas 12 e 13, definimos a string que conterá o comando para inserir um registro na tabela Pessoa. Observe os nomes dos argumentos nomeados: cpf,



nome, data\_nascimento e usa\_olhos. Cada um desses nomes deve estar presente no dicionário passado para o método execute!

Na linha 14, chamamos o método execute utilizando, como segundo argumento, um dicionário com os atributos do objeto pessoa, definido na linha 10. Cada argumento nomeado do comando será substituído pelo valor da chave correspondente do dicionário.

O comando final enviado ao banco de dados pelo comando execute foi:

```
INSERT INTO Pessoa (cpf, nome, nascimento, olhos)  
VALUES (20000000099,'José','1990-02-28',0);
```

Na linha 20, efetivamos o comando utilizando o método commit.

Observe que nosso código está crescendo. Imagine se tivéssemos uma entidade com dezenas de atributos? A chamada do método execute da linha 14 cresceria proporcionalmente, comprometendo muito a leitura do nosso código.

A seguir, vamos simplificar nosso código, de forma que ele permaneça legível, independentemente do número de atributos de uma entidade.

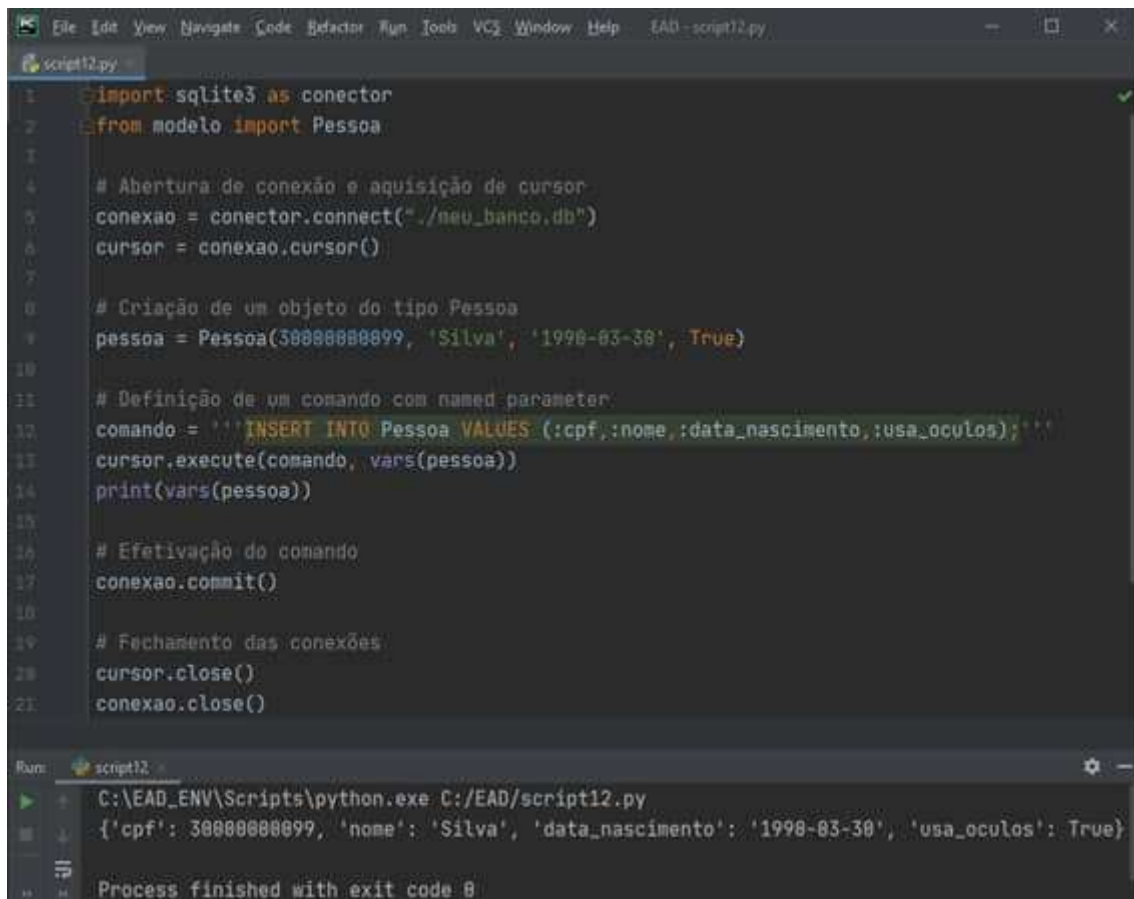
## Dica

Quando utilizamos o comando **INSERT INTO do SQL** para inserir um registro onde todos os atributos estão preenchidos, podemos suprimir o nome das colunas no comando.

Como vamos inserir uma pessoa com todos os atributos, vamos manter apenas os argumentos nomeados no comando **SQL**.

No próximo exemplo, vamos simplificar mais um pouco nosso código, removendo o nome das colunas no comando SQL e utilizando a função interna vars do Python que converte objetos em dicionários. Observe a Figura 15 a seguir.



The image shows a screenshot of a Python script editor window titled 'script12.py'. The script is written in Python and uses the sqlite3 module to connect to a database named 'neu\_banco.db'. It creates a cursor, then creates a 'Pessoa' object with attributes: cpf (30000000099), nome ('Silva'), data\_nascimento ('1990-03-30'), and usa\_olhos (True). A SQL 'INSERT INTO' command is constructed using these attributes and executed. The script then commits the transaction and closes the cursor and connection. Below the editor, a 'Run' console window shows the command executed: 'C:\EAD\_ENV\Scripts\python.exe C:/EAD/script12.py' and the output: {'cpf': 30000000099, 'nome': 'Silva', 'data\_nascimento': '1990-03-30', 'usa\_olhos': True}. The process finished with exit code 0.

```
1 import sqlite3 as conector
2 from modelo import Pessoa
3
4 # Abertura de conexão e aquisição de cursor
5 conexao = conector.connect("./neu_banco.db")
6 cursor = conexao.cursor()
7
8 # Criação de um objeto do tipo Pessoa
9 pessoa = Pessoa(30000000099, 'Silva', '1990-03-30', True)
10
11 # Definição de um comando com named parameter
12 comando = 'INSERT INTO Pessoa VALUES (:cpf,:nome,:data_nascimento,:usa_olhos);'
13 cursor.execute(comando, vars(pessoa))
14 print(vars(pessoa))
15
16 # Efetivação do comando
17 conexao.commit()
18
19 # Fechamento das conexões
20 cursor.close()
21 conexao.close()
```

Run: script12

C:\EAD\_ENV\Scripts\python.exe C:/EAD/script12.py

{'cpf': 30000000099, 'nome': 'Silva', 'data\_nascimento': '1990-03-30', 'usa\_olhos': True}

Process finished with exit code 0

Figura: 15.

Após a criação da conexão e do cursor, criamos um objeto do tipo Pessoa com todos os atributos preenchidos na linha 9. Observe que o valor True do atributo usa\_olhos será convertido para 1 durante a execução do método execute.

Na linha 12, criamos o comando SQL **“INSERT INTO”**, onde suprimimos o nome das colunas após o nome da tabela Pessoa.

Na linha 13, utilizamos o método execute passando como segundo argumento vars(pessoa).

A função interna vars retorna todos os atributos de um objeto na forma de dicionário, no qual cada chave é o nome de um atributo.

Observe na saída do console onde imprimimos o resultado de vars(pessoa), linha 14, e destacado a seguir:

**{'cpf': 30000000099, 'nome': 'Silva', 'data\_nascimento': '1990-03-30', 'usa\_olhos': True}**

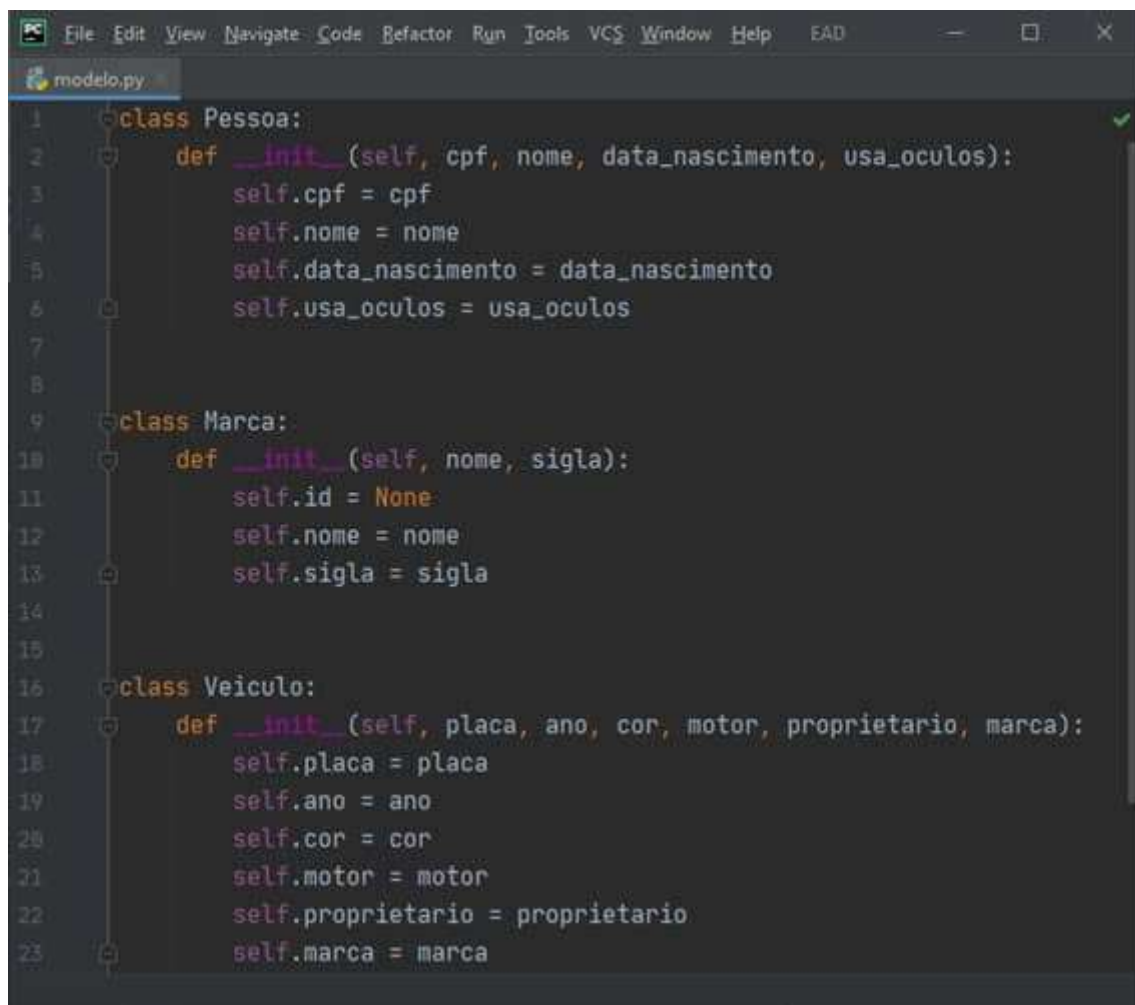
O comando final enviado ao banco de dados pelo comando execute foi:

**INSERT INTO Pessoa VALUES (300000000099,'Silva','1990-03-30',1);**

Na linha 17, efetivamos a transação e ao final do script fechamos o cursor e a conexão.

No exemplo a seguir, vamos inserir alguns registros nas outras tabelas, de forma a povoar nosso banco de dados. Vamos utilizar a mesma lógica do exemplo anterior, no qual utilizamos a função vars() e argumentos nomeados.

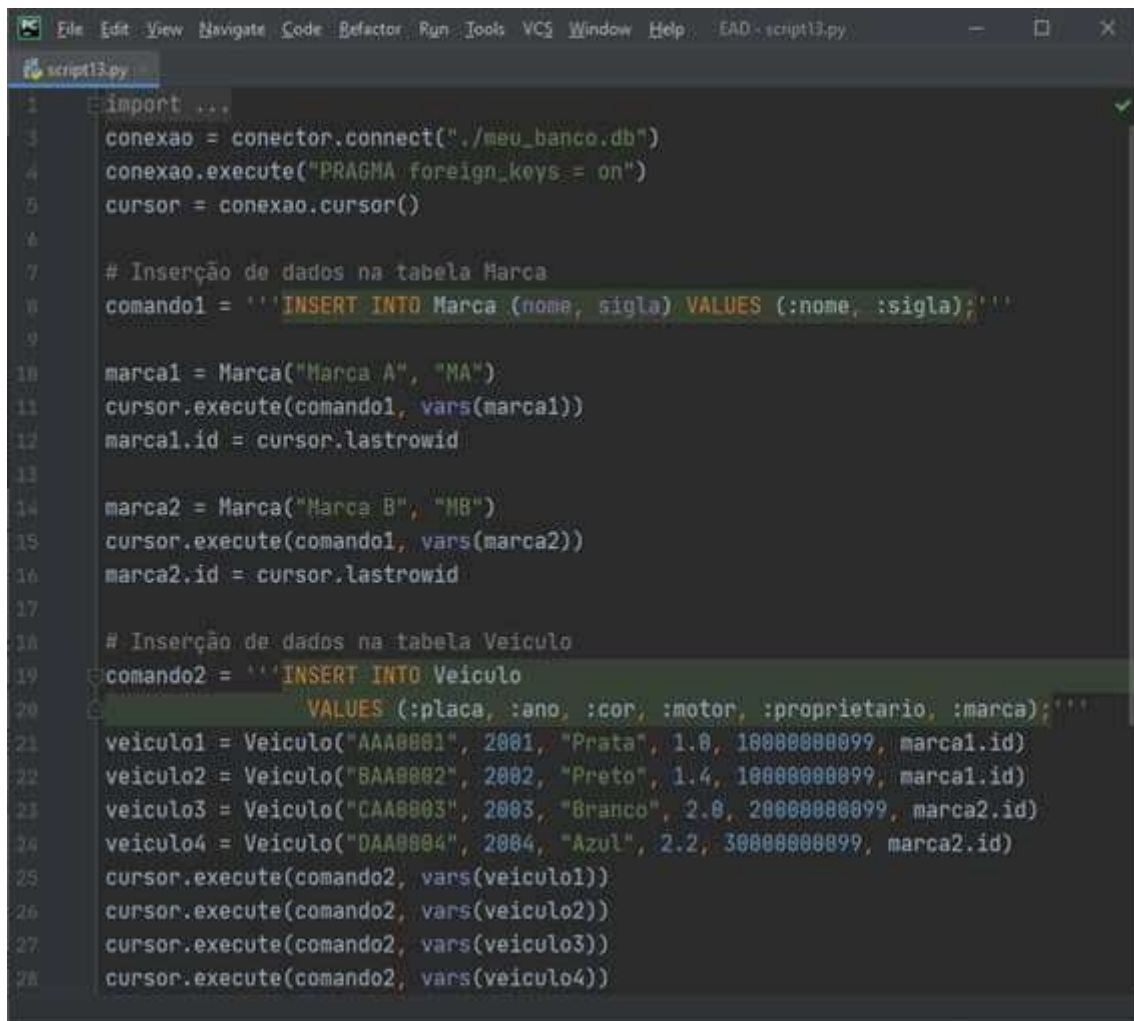
Primeiro, vamos criar mais duas classes no nosso módulo modelo.py para representar as entidades Marca e Veiculo. Confira essas classes no script da Figura 16.



```
1 class Pessoa:
2     def __init__(self, cpf, nome, data_nascimento, usa_olhos):
3         self.cpf = cpf
4         self.nome = nome
5         self.data_nascimento = data_nascimento
6         self.usa_olhos = usa_olhos
7
8
9 class Marca:
10     def __init__(self, nome, sigla):
11         self.id = None
12         self.nome = nome
13         self.sigla = sigla
14
15
16 class Veiculo:
17     def __init__(self, placa, ano, cor, motor, proprietario, marca):
18         self.placa = placa
19         self.ano = ano
20         self.cor = cor
21         self.motor = motor
22         self.proprietario = proprietario
23         self.marca = marca
```

Figura: 16.

Para inserir os registros, vamos criar o script definido na Figura 17 a seguir.

The image shows a screenshot of a code editor window titled 'script13.py'. The code is written in Python and interacts with a SQLite database. It starts with an import statement, then connects to a database file named 'meu\_banco.db'. A PRAGMA command is used to enable foreign key support. A cursor is created, and two INSERT statements are used to add records to a table named 'Marca'. These records are 'Marca A' and 'Marca B'. Then, four more INSERT statements are used to add records to a table named 'Veiculo', each referencing one of the 'Marca' records by their primary key (id). The script ends with the cursor being closed.

```
1 import ...
2
3 conexao = conector.connect("./meu_banco.db")
4 conexao.execute("PRAGMA foreign_keys = on")
5 cursor = conexao.cursor()
6
7 # Inserção de dados na tabela Marca
8 comando1 = '''INSERT INTO Marca (nome, sigla) VALUES (:nome, :sigla);'''
9
10 marca1 = Marca("Marca A", "MA")
11 cursor.execute(comando1, vars(marca1))
12 marca1.id = cursor.lastrowid
13
14 marca2 = Marca("Marca B", "MB")
15 cursor.execute(comando1, vars(marca2))
16 marca2.id = cursor.lastrowid
17
18 # Inserção de dados na tabela Veiculo
19 comando2 = '''INSERT INTO Veiculo
20                 VALUES (:placa, :ano, :cor, :motor, :proprietario, :marca);'''
21 veiculo1 = Veiculo("AAAB001", 2001, "Prata", 1.0, 100000000000, marca1.id)
22 veiculo2 = Veiculo("BAAB002", 2002, "Preto", 1.4, 100000000000, marca1.id)
23 veiculo3 = Veiculo("CAAB003", 2003, "Branco", 2.0, 200000000000, marca2.id)
24 veiculo4 = Veiculo("DAAB004", 2004, "Azul", 2.2, 300000000000, marca2.id)
25 cursor.execute(comando2, vars(veiculo1))
26 cursor.execute(comando2, vars(veiculo2))
27 cursor.execute(comando2, vars(veiculo3))
28 cursor.execute(comando2, vars(veiculo4))
```

Figura: 17.

No script13, após abrir conexão, vamos utilizar um comando especial do SQLite na linha 4. O comando PRAGMA.

## Atenção

O comando PRAGMA é uma extensão do SQL específica para o SQLite. Ela é utilizada para modificar alguns comportamentos internos do banco de dados.

O SQLite, por padrão, não força a checagem das restrições de chave estrangeira. Isso ocorre por motivos históricos, visto que versões mais antigas do SQLite não tinham suporte à chave estrangeira.

No comando da linha 4, habilitamos a flag `foreign_keys`, de forma a garantir que as restrições de chave estrangeiras sejam checadas antes de cada operação.

Após a utilização do comando PRAGMA e da criação de um cursor, vamos inserir registros relacionados à entidade `Marca` e `Veiculo` no banco.

Vamos começar pela entidade Marca, pois ela é um requisito para criação de registros na tabela Veiculo, visto que a tabela Veiculo contém uma chave estrangeira para a tabela Marca, por meio do atributo Veiculo.marca, que referencia Marca.id.

Na linha 8, escrevemos o comando de inserção de registro na tabela Marca utilizando argumentos nomeados.

Como não iremos passar um valor para o id da marca, que é autoincrementado, foi necessário explicitar o nome das colunas no comando INSERT INTO. Caso omitíssemos o nome das colunas, seria gerada uma exceção do tipo OperationalError, com a descrição indicando que a tabela tem 3 colunas, mas apenas dois valores foram fornecidos.

Na linha 10, criamos um objeto do tipo Marca, que foi inserido no banco pelo comando execute da linha 11.

Para criar uma referência da marca que acabamos de inserir em um veículo, precisamos do id autoincrementado gerado pelo banco no momento da inserção.

Para isso, vamos utilizar o atributo lastrowid do Cursor. Esse atributo armazena o id da linha do último registro inserido no banco e está disponível assim que chamamos o método execute do cursor. O id da linha é o mesmo utilizado para o campo autoincrementado.

Na linha 12, atribuímos o valor do lastrowid ao atributo id do objeto marca1, recém-criado.

Nas linhas 14 a 16, criamos mais um objeto do tipo Marca, inserimos no banco de dados e recuperamos seu novo id.

Nas linhas 19 e 20, temos o comando para inserir registros na tabela Veiculo, também utilizando argumentos nomeados. Como vamos inserir um veículo com todos os atributos, omitimos os nomes das colunas.

Nas linhas 21 a 24, criamos quatro objetos do tipo Veiculo. Observe que utilizamos o atributo id dos objetos marca1 e marca2 para fazer referência à marca do veículo. Os CPF dos proprietários foram escolhidos aleatoriamente, baseando-se nos cadastros anteriores.

Lembre-se de que como os atributos proprietario e marca são referências a chaves de outras tabelas, eles precisam existir no banco! Caso contrário, será

lançada uma exceção de erro de integridade (IntegrityError) com a mensagem Falha na Restrição de Chave Estrangeira (FOREIGN KEY constraint failed).

Na sequência, os veículos foram inseridos no banco pelos comandos execute das linhas 25 a 28.

Os comandos SQL gerados por esse script foram os seguintes:

```
INSERT INTO Marca (nome, sigla) VALUES ('Marca A', 'MA')
INSERT INTO Marca (nome, sigla) VALUES ('Marca B', 'MB')
INSERT INTO Veiculo VALUES ('AAA0001', 2001, 'Prata', 1.0, 10000000099, 1)
INSERT INTO Veiculo VALUES ('BAA0002', 2002, 'Preto', 1.4, 10000000099, 1)
INSERT INTO Veiculo VALUES ('CAA0003', 2003, 'Branco', 2.0, 20000000099, 2)
INSERT INTO Veiculo VALUES ('DAA0004', 2004, 'Azul', 2.2, 30000000099, 2)
```

Ao final do script, efetivamos as transações, fechamos a conexão e o cursor.

Neste momento, temos os seguintes dados nas nossas tabelas:

Tabela: Marca		
id	nome	sigla
Filtro	Filtro	Filtro
1	1 Marca A	MA
2	2 Marca B	MB

Tabela: Pessoa				
cpf	nome	nascimento	olhos	
Filtro	Filtro	Filtro	Filtro	
1	10000000099	Maria	1990-01-31	0
2	12345678900	João	2000-01-31	1
3	20000000099	José	1990-02-28	0
4	30000000099	Silva	1990-03-30	1

Tabela: Veiculo						
placa	ano	cor	motor	proprietario	marca	
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	
1	AAA0001	2001	Prata	1.0	10000000099	1
2	BAA0002	2002	Preto	1.4	10000000099	1
3	CAA0003	2003	Branco	2.0	20000000099	2
4	DAA0004	2004	Azul	2.2	30000000099	2

## ATUALIZAÇÃO DE DADOS EM UMA TABELA

Agora que já sabemos como inserir um registro em uma tabela, vamos aprender a atualizar os dados de um registro.

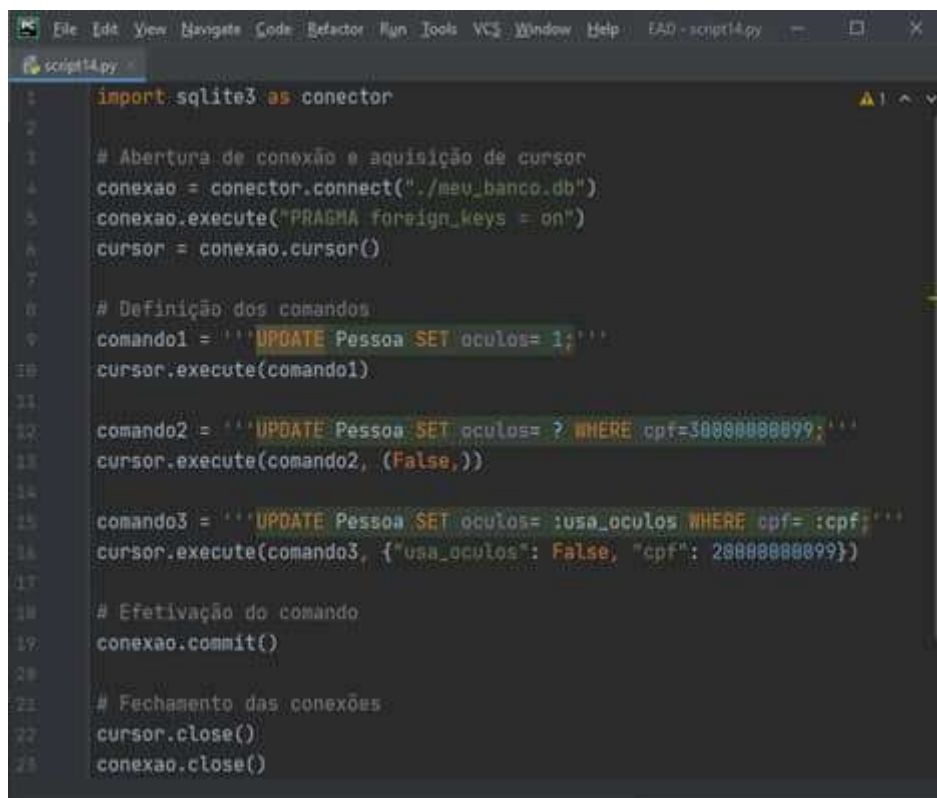
Para atualizar um registro, utilizamos o comando SQL UPDATE. Sua sintaxe é a seguinte:

```
UPDATE tabela1
SET coluna1 = valor1, coluna2 = valor2...
WHERE [condição];
```

Assim como no comando INSERT, podemos montar o comando UPDATE de três formas. Uma string sem delimitadores, uma string com o delimitador “?” ou uma string com argumentos nomeados.

Também podemos utilizar os delimitadores na condição da cláusula WHERE.

No exemplo a seguir, Figura 18, vamos mostrar como atualizar registros de uma tabela utilizando os três métodos.

A screenshot of a code editor window titled 'script14.py'. The code is in Python and uses the sqlite3 module to connect to a database and execute SQL commands. The code is as follows:

```
1 import sqlite3 as conector
2
3 # Abertura de conexão e aquisição de cursor
4 conexao = conector.connect("./meu_banco.db")
5 conexao.execute("PRAGMA foreign_keys = on")
6 cursor = conexao.cursor()
7
8 # Definição dos comandos
9 comando1 = 'UPDATE Pessoa SET olhos= 1;'
10 cursor.execute(comando1)
11
12 comando2 = 'UPDATE Pessoa SET olhos= ? WHERE cpf=30000000099;'
13 cursor.execute(comando2, (False,))
14
15 comando3 = 'UPDATE Pessoa SET olhos= :usa_olhos WHERE cpf= :cpf;'
16 cursor.execute(comando3, {"usa_olhos": False, "cpf": 20000000099})
17
18 # Efetivação do comando
19 conexao.commit()
20
21 # Fechamento das conexões
22 cursor.close()
23 conexao.close()
```

Figura: 18.

Após a abertura da conexão, habilitamos novamente a checagem de chave estrangeira, por meio da flag `foreign_keys`, ativada pelo comando PRAGMA.

Na sequência, criamos o cursor e definimos a string do nosso primeiro comando de atualização, onde passamos os valores de forma explícita, sem utilização de delimitadores. O string foi atribuído à variável `comando1`, linha 9.

No comando1, estamos atualizando o valor do atributo `olhos` para 1 (verdadeiro) para TODOS os registros da tabela. Isso ocorreu, pois a cláusula WHERE foi omitida.

Na linha 10, executamos o comando1.



Na linha 12, criamos um comando de UPDATE utilizando o delimitador “?” para o valor do atributo olhos e explicitamos o valor do cpf na cláusula WHERE. O comando executado pela linha 13 é:

```
UPDATE Pessoa SET olhos= 0 WHERE cpf=30000000099;
```

Ou seja, vamos alterar o valor do atributo olhos para zero (falso) apenas para quem tem cpf igual a 30000000099.

Na linha 15, criamos mais um comando de UPDATE, desta vez utilizando o argumento nomeado tanto para o atributo olhos quanto para o cpf da cláusula WHERE.

O comando final enviado pela linha 16 ao banco de dados é:

```
UPDATE Pessoa SET olhos= 0 WHERE cpf= 20000000099;
```

Onde vamos alterar o valor do atributo olhos para zero (falso) para quem tem cpf igual a 20000000099.

Na linha 19, efetivamos as transações e posteriormente fechamos o cursor e a conexão.

Se tentássemos alterar o CPF de uma pessoa referenciada pela tabela Veiculo, seria lançada uma exceção do tipo IntegrityError, devido à restrição da chave estrangeira.

## **Dica**

Cuidado ao executar um comando UPDATE sem a cláusula WHERE, pois, dependendo do tamanho do banco de dados, essa operação pode ser muito custosa.

# **REMOÇÃO DE DADOS DE UMA TABELA**

Nesta parte deste módulo, iremos aprender a remover registros de uma tabela.

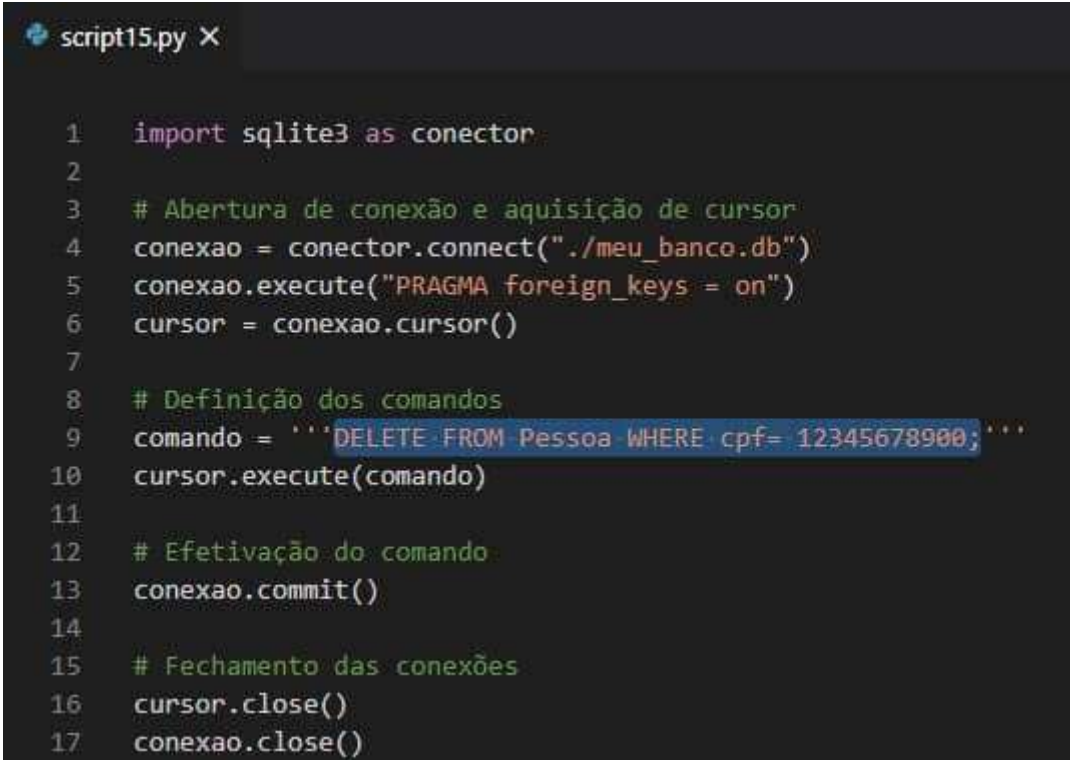
Para remover um registro, utilizamos o comando SQL DELETE. Sua sintaxe é a seguinte:



**DELETE FROM tabela1  
WHERE [condição];**

Assim como nos outros comandos, podemos montar o comando DELETE de três formas. Uma string sem delimitadores, uma string com o delimitador “?” ou uma string com argumentos nomeados. Todos para a condição da cláusula WHERE.

Veja o exemplo da utilização do comando DELETE a seguir, na Figura 19.



```
1  import sqlite3 as conector
2
3  # Abertura de conexão e aquisição de cursor
4  conexao = conector.connect("./meu_banco.db")
5  conexao.execute("PRAGMA foreign_keys = on")
6  cursor = conexao.cursor()
7
8  # Definição dos comandos
9  comando = 'DELETE FROM Pessoa WHERE cpf= 12345678900;'
10 cursor.execute(comando)
11
12 # Efetivação do comando
13 conexao.commit()
14
15 # Fechamento das conexões
16 cursor.close()
17 conexao.close()
```

Figura: 19.

Após a criação da conexão, habilitação da checagem de chave estrangeira e aquisição do cursor, criamos o comando SQL “DELETE” na linha 9, onde explicitamos o CPF da pessoa que desejamos remover do banco.

Na linha 10, utilizamos o método execute com o comando criado anteriormente.

O comando final enviado ao banco de dados pelo comando execute foi:

**DELETE FROM Pessoa WHERE cpf=12345678900;**

Na linha 13, efetivamos a transação e ao final do script fechamos o cursor e a conexão.

Observe que, como as outras pessoas cadastradas são referenciadas pela tabela Veiculo por meio de seu CPF, seria gerada uma exceção do tipo `IntegrityError` caso tentássemos removê-las.