

EMPREGAR AS FUNCIONALIDADES PARA CONEXÃO, ACESSO E CRIAÇÃO DE BANCOS DE DADOS E TABELAS

CONECTANDO A UM BANCO DE DADOS

Neste módulo, vamos aprender a criar e a conectar-se a um banco de dados, criar e editar tabelas e seus relacionamentos.



Como o SQLite trabalha com arquivo e não tem suporte à autenticação, para se conectar a um banco de dados *SQLite*, basta chamar a função **connect** do módulo **sqlite3**, passando como argumento o caminho para o arquivo que contém o banco de dados.

Veja a sintaxe a seguir:

```
>>> import sqlite3
>>> conexao = sqlite3.connect('meu_banco.db')
```

Pronto! Isso é o suficiente para termos uma conexão com o banco de dados `meu_banco.db` e iniciar o envio de comandos SQL para criar tabelas e inserir registros.

Caso o arquivo não exista, ele será criado automaticamente! O arquivo criado pode ser copiado e compartilhado.

Se quisermos criar um banco de dados em memória, que será criado para toda execução do programa, basta utilizar o comando `conexao = sqlite3.connect(':memory:')`.

CRIANDO TABELAS

Agora que já sabemos como criar e se conectar a um banco de dados SQLite, vamos começar a criar nossas tabelas.

Antes de colocarmos a mão na massa, vamos verificar o nosso modelo entidade relacionamento (ER) que utilizaremos para criar nossas tabelas neste primeiro momento. Veja o modelo na Figura 3.



Figura 3.

Nosso modelo é composto por três entidades: Pessoa, Veiculo e Marca.

Nossa primeira entidade se chama Pessoa e contém os atributos:

- **cpf**, como chave primária;
- **nome**;
- **nascimento**, para armazenar a data de nascimento da pessoa; e
- **oculos**, que indica se a pessoa precisa usar óculos.

A segunda entidade se chama Marca, que contém os atributos:

- **id**, como chave primária,
- **nome**; e
- **sigla**.

Nossa terceira e última entidade se chama Veiculo, que contém os atributos:

- **placa**, como chave primária;
- **cor**;
- **proprietario**, que é uma referência à pessoa dona do veículo; e
- **marca**, referência à marca do veículo.

Para os relacionamentos do nosso modelo, uma pessoa pode ter zero, um ou mais veículos e um veículo só pode ter um proprietário. Uma marca pode estar em zero, um ou mais veículos e um veículo só pode ter uma marca.

Agora que já temos nosso modelo ER, precisamos definir os tipos de cada atributo das nossas entidades. Como estamos trabalhando com SQLite, precisamos ter em mente a tabela de afinidades disponibilizada no módulo anterior.

Vamos definir a nossa entidade Pessoa, de forma que os atributos tenham os seguintes tipos e restrições:

CPF	INTEGER (chave primária, não nulo)
NOME	TEXT (não nulo)
NASCIMENTO	DATE (não nulo)
OCULOS	BOOLEAN (não nulo)

Para criar uma tabela que represente essa entidade, vamos utilizar o comando SQL:

```
CREATE TABLE Pessoa (  
  cpf INTEGER NOT NULL,  
  nome TEXT NOT NULL,  
  nascimento DATE NOT NULL,  
  olhos BOOLEAN NOT NULL,  
  PRIMARY KEY (cpf)  
);
```

Observe pela nossa tabela de afinidades, que os tipos DATE e BOOLEAN serão convertidos por afinidade para NUMERIC. Na prática, os valores do tipo DATE serão da classe TEXT e os do tipo BOOLEAN da classe INTEGER, pois armazenaremos os valores True como 1 e False como 0.

Definido o comando SQL, vamos ver como criar essa tabela em Python no exemplo da Figura 4 a seguir.

```

script4.py X

1  import sqlite3 as conector
2
3  try:
4      # Abertura de conexão e aquisição de cursor
5      conexao = conector.connect("./meu_banco.db")
6      cursor = conexao.cursor()
7
8      # Execução de um comando: SELECT... CREATE ...
9      comando = '''CREATE TABLE Pessoa (
10         ..... cpf INTEGER NOT NULL,
11         ..... nome TEXT NOT NULL,
12         ..... nascimento DATE NOT NULL,
13         ..... olhos BOOLEAN NOT NULL,
14         ..... PRIMARY KEY (cpf)
15         ..... );'''
16
17      cursor.execute(comando)
18
19      # Efetivação do comando
20      conexao.commit()
21
22  except conector.DatabaseError as err:
23      print("Erro de banco de dados", err)
24
25  finally:
26      # Fechamento das conexões
27      if conexao:
28          cursor.close()
29          conexao.close()

```

Figura 4.

Na linha 1, importamos o módulo sqlite3 e atribuímos o alias conector.

Observe que envolvemos todo o nosso código com a cláusula try/catch, capturando as exceções do tipo DatabaseError.

Na linha 5, criamos uma conexão com o banco de dados meu_banco.db. Caso esse arquivo não exista, será criado um arquivo no mesmo diretório do script sendo executado.

Na linha 6, criamos um cursor para executar as operações no nosso banco.

Nas linhas 9 a 15, definimos a variável comando, que é uma string contendo o comando SQL para criação da nossa tabela Pessoa.

Na linha 17, utilizamos o método `execute` do cursor para executar o comando SQL passado como argumento.

Na linha 19, efetivamos a transação pendente utilizando o método `commit` da variável conexão. Nesse caso, a transação pendente é a criação da tabela.

Nas linhas 22 e 23, capturamos e tratamos a exceção `DatabaseError`.

Nas linhas 28 e 29 fechamos o cursor e a conexão na cláusula `finally`, para garantir que nenhuma conexão fique aberta em caso de erro.

Observe como ficou a árvore de diretórios à esquerda da figura após a execução do programa. Veja que agora temos o arquivo `meu_banco.db`.

Atenção

Nos exemplos ao longo deste tema, vamos utilizar a mesma estrutura de código do script anterior, porém, vamos omitir as cláusulas `try/catch` para fins de brevidade.

Agora vamos tratar de nossa próxima entidade, `Marca`. Vamos defini-la de forma que os atributos tenham os seguintes tipos e restrições:

<code>id</code>	<code>INTEGER</code> (chave primária, não nulo)
<code>nome</code>	<code>TEXT</code> (não nulo)
<code>sigla</code>	<code>CHARACTER</code> (não nulo, tamanho 2)

A codificação latin-1, muito utilizada no Brasil, utiliza um byte por caractere. Como a sigla da marca será composta por 2 caracteres, nosso atributo `sigla` terá tamanho 2 (`CHAR(2)` ou `CHARACTER(2)`).

Para criar uma tabela que represente essa entidade, vamos utilizar o comando SQL:

```
CREATE TABLE Marca (  
  id INTEGER NOT NULL,  
  nome TEXT NOT NULL,  
  sigla CHARACTER(2) NOT NULL,  
  PRIMARY KEY (id)  
);
```

Pela tabela de afinidades, o tipo `CHARACTER(2)` será convertido para `TEXT`.

Confira a imagem a seguir, Figura 5, para verificar como ficou o script de criação dessa tabela.

```

script5.py X
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 = '''CREATE TABLE Marca (
9      ..... id INTEGER NOT NULL,
10     ..... nome TEXT NOT NULL,
11     ..... sigla CHARACTER(2) NOT NULL,
12     ..... PRIMARY KEY (id)
13     ..... );'''
14
15  cursor.execute(comando)
16
17  # Efetivação do comando
18  conexao.commit()
19
20  # Fechamento das conexões
21  cursor.close()
22  conexao.close()

```

Figura: 5.

Após a criação da conexão e do cursor, linhas 4 e 5, definimos uma string com o comando SQL para criação da tabela Marca, linhas 8 a 13.

O comando foi executado na linha 15 e efetivado na linha 18.

Nas linhas 21 e 22, fechamos o cursor e a conexão.

A próxima entidade a ser criada será a entidade Veiculo. Os seus atributos terão os seguintes tipos e restrições.

PLACA	CHARACTER (chave primária, não nulo, tamanho 7)
ANO	INTEGER (não nulo)
COR	TEXT (não nulo)
PROPRIETÁRIO	INTEGER (chave estrangeira, não nulo)
MARCA	INTEGER (chave estrangeira, não nulo)

Como nossas placas são compostas por 7 caracteres, nosso atributo placa terá tamanho 7 (CHAR(7) ou CHARACTER(7)). Por afinidade, ele será convertido para TEXT.

O atributo proprietario será utilizado para indicar um relacionamento da entidade Veiculo com a entidade Pessoa. O atributo da tabela Pessoa utilizado no

relacionamento será o CPF. Como o CPF é um INTEGER, o atributo relacionado proprietario também precisa ser INTEGER.

Analogamente, o atributo marca será utilizado para indicar um relacionamento da entidade Veiculo com a entidade Marca, por meio do atributo id da tabela Marca, que também é um INTEGER.

Veja o comando SQL a seguir para criar a tabela Veiculo e o relacionamento com as tabelas Pessoa e Marca.

```
CREATE TABLE Veiculo (  
placa CHARACTER(7) NOT NULL,  
ano INTEGER NOT NULL,  
cor TEXT NOT NULL,  
proprietario INTEGER NOT NULL,  
marca INTEGER NOT NULL,  
PRIMARY KEY (placa),  
FOREIGN KEY(proprietario) REFERENCES Pessoa(cpf),  
FOREIGN KEY(marca) REFERENCES Marca(id)  
);
```

Definido o comando SQL, vamos ver como criar essa tabela em Python no exemplo da Figura 6 a seguir.


```
script6.py X

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 = '''CREATE TABLE Veiculo (
9      .....placa CHARACTER(7) NOT NULL,
10     .....ano INTEGER NOT NULL,
11     .....cor TEXT NOT NULL,
12     .....proprietario INTEGER NOT NULL,
13     .....marca INTEGER NOT NULL,
14     .....PRIMARY KEY (placa),
15     .....FOREIGN KEY(proprietario) REFERENCES Pessoa(cpf),
16     .....FOREIGN KEY(marca) REFERENCES Marca(id)
17     .....);'''
18
19  cursor.execute(comando)
20
21  # Efetivação do comando
22  conexao.commit()
23
24  # Fechamento das conexões
25  cursor.close()
26  conexao.close()
```

Figura: 6.

Este script também segue os mesmos passos do script anterior até a linha 8, onde definimos a string com o comando SQL para criação da tabela Veiculo.

Esse comando foi executado na linha 19 e efetivado na linha 22.

Dica

Caso a referência da chave estrangeira seja feita a um atributo inexistente, será lançado um erro de programação: `ProgrammingError`.

Para visualizar como ficaram nossas tabelas, vamos utilizar o programa DB Browser for SQLite.

Após abrir o programa DB Browser for SQLite, basta clicar em Abrir banco de dados e selecionar o arquivo `meu_banco.db`. Observe a Figura 7 a seguir.

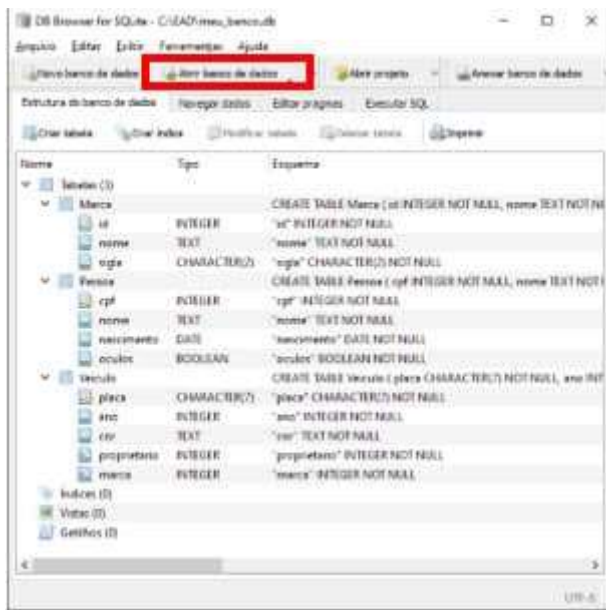


Figura: 7.

Observe que está tudo conforme o previsto, inclusive a ordem dos atributos obedece a sequência na qual foram criados.

ALTERAÇÃO E REMOÇÃO DE TABELA

Neste momento, temos o nosso banco com as três tabelas exibidas no modelo ER da Figura 3.

Durante o desenvolvimento, pode ser necessário realizar alterações no nosso modelo e, conseqüentemente, nas nossas tabelas. Nesta parte do módulo, vamos ver como podemos fazer para adicionar um novo atributo e como fazemos para remover uma tabela.

Para alterar uma tabela e adicionar um novo atributo, precisamos utilizar o comando ALTER TABLE do SQL.

Para ilustrar, vamos adicionar mais um atributo à entidade Veículo.

O atributo se chama motor e corresponde à motorização do carro: 1.0, 1.4, 2.0 etc. Esse atributo deverá conter pontos flutuantes e, por isso, vamos defini-lo como do tipo REAL.

Para alterar a tabela Veículo e adicionar a coluna motor, utilizamos o seguinte comando SQL.

**ALTER TABLE Veiculo
ADD motor REAL;**

Confira o script da Figura 8, onde realizamos a alteração da tabela Veículo.

```

script7.py X

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 = '''ALTER TABLE Veículo
9             ..... ADD motor REAL;'''
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: 8.

Após se conectar ao banco e obter um cursor, linhas 4 e 5, construímos uma string com o comando ALTER TABLE nas linhas 8 e 9.

Na linha 11, executamos o comando e na linha 14 efetivamos a modificação.

Nas linhas 17 e 18, fechamos o cursor e a conexão.

Observe como ficou nossa tabela após a criação da nova coluna.

Nome	Tipo	Esquema
Tabelas (3)		
Marca		CREATE TABLE Marca (id INTEGER NOT NULL
Pessoa		CREATE TABLE Pessoa (cpf INTEGER NOT
Veiculo		CREATE TABLE "Veiculo" ("placa" CHARA
placa	CHARACTER(7)	"placa" CHARACTER(7) NOT NULL
ano	INTEGER	"ano" INTEGER NOT NULL
cor	TEXT	"cor" TEXT NOT NULL
proprietario	INTEGER	"proprietario" INTEGER NOT NULL
marca	INTEGER	"marca" INTEGER NOT NULL
motor	REAL	"motor" REAL
Índices (0)		
Vistas (0)		
Gatilhos (0)		

Figura: 9.

Veja, pela Figura 9, que o atributo motor foi adicionado ao final da entidade, obedecendo a ordem de criação.

Exemplo

Em algumas situações, pode ser necessário que as colunas sigam uma ordem predeterminada. Um exemplo é quando precisamos carregar os dados de uma planilha diretamente para um banco de dados, para realizarmos o chamado bulk insert (inserção em massa). Nesses casos, as colunas da planilha precisam estar na mesma sequência das colunas do banco.

Como nem todos os bancos de dados, incluindo o SQLite, dão suporte à criação de colunas em posição determinada, vamos precisar remover nossa tabela para recriá-la com os atributos na posição desejada.

Para o nosso exemplo, desejamos a seguinte sequência: placa, ano, cor, motor, proprietario e marca.

No exemplo da Figura 10, vamos remover a tabela Veiculo, utilizando o comando DROP TABLE do SQL e, posteriormente, vamos criá-la novamente com a sequência desejada.

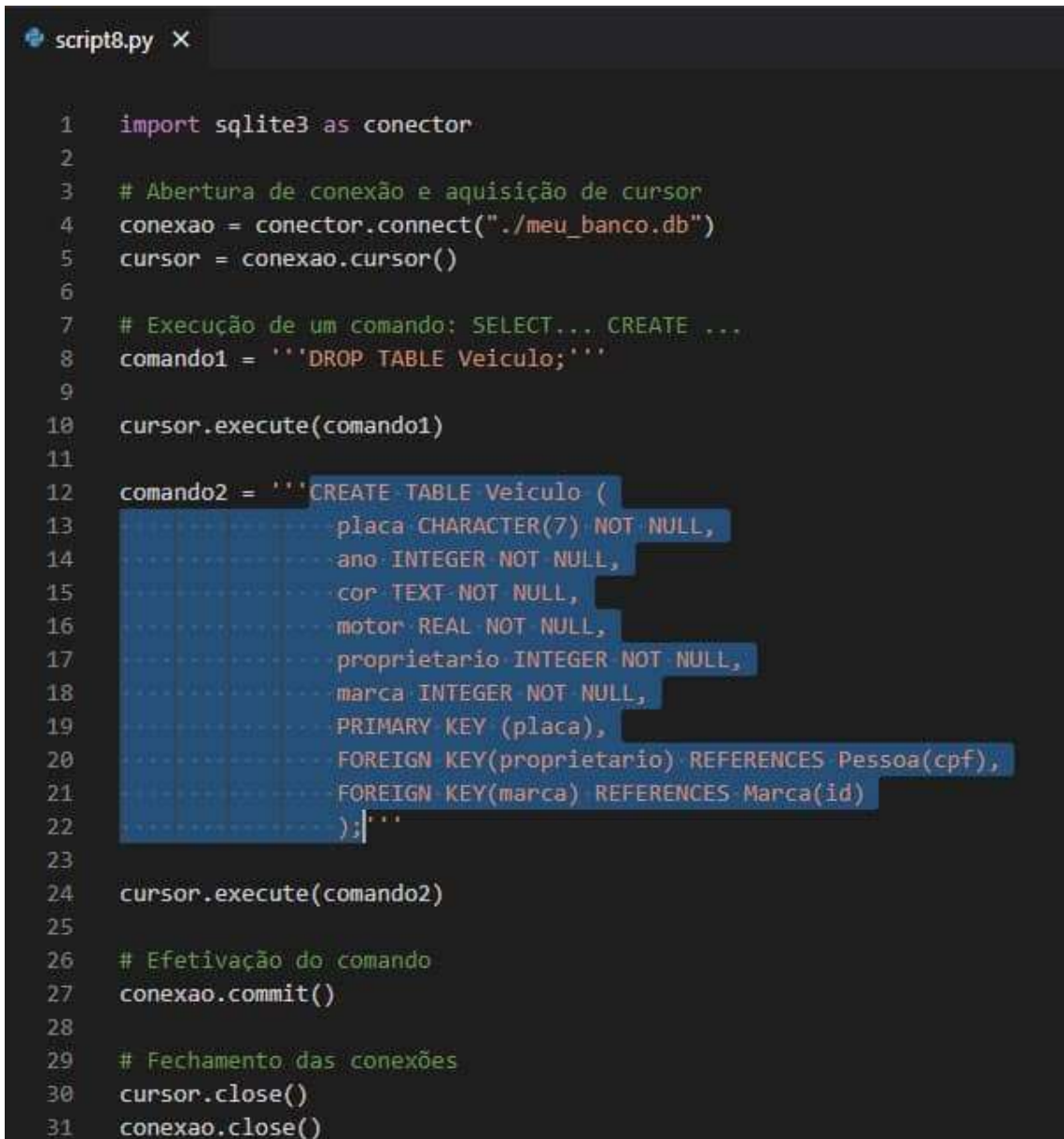
Para remover a tabela Veiculo, utilizamos o seguinte comando SQL.

```
DROP TABLE Veiculo;
```

Para recriar a tabela, utilizamos o seguinte comando SQL:

```
CREATE TABLE Veiculo (  
placa CHARACTER(7) NOT NULL,  
ano INTEGER NOT NULL,  
cor TEXT NOT NULL,  
motor REAL NOT NULL,  
proprietario INTEGER NOT NULL,  
marca INTEGER NOT NULL,  
PRIMARY KEY (placa),  
FOREIGN KEY(proprietario) REFERENCES Pessoa(cpf),  
FOREIGN KEY(marca) REFERENCES Marca(id)  
);
```

Confira como ficou nosso script na Figura 10 a seguir:



```

script8.py X

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  comando1 = '''DROP TABLE Veiculo;'''
9
10 cursor.execute(comando1)
11
12 comando2 = '''CREATE TABLE Veiculo (
13     ..... placa CHARACTER(7) NOT NULL,
14     ..... ano INTEGER NOT NULL,
15     ..... cor TEXT NOT NULL,
16     ..... motor REAL NOT NULL,
17     ..... proprietario INTEGER NOT NULL,
18     ..... marca INTEGER NOT NULL,
19     ..... PRIMARY KEY (placa),
20     ..... FOREIGN KEY(proprietario) REFERENCES Pessoa(cpf),
21     ..... FOREIGN KEY(marca) REFERENCES Marca(id)
22     ..... );'''
23
24 cursor.execute(comando2)
25
26 # Efetivação do comando
27 conexao.commit()
28
29 # Fechamento das conexões
30 cursor.close()
31 conexao.close()

```

Figura: 10.

Após se conectar ao banco e obter o cursor, criamos a string comando1 com o comando para remover a tabela Veiculo, na linha 8. Na linha 10, executamos esse comando.

Nas linhas 12 a 22, criamos o comando para criar novamente a tabela Veiculo com os atributos na ordem mostrada anteriormente e, na linha 24, executamos esse comando.

Na linha 27, efetivamos todas as modificações pendentes, tanto a remoção da tabela, quanto a criação. Observe que não é preciso efetuar um commit para cada comando!

Nas linhas 30 e 31, liberamos o cursor e fechamos a conexão.

Chegamos ao final do módulo 2 e agora nosso modelo ER está conforme a Figura 11 a seguir:



Figura: 11.

Saiba mais

Algumas bibliotecas de acesso a banco de dados oferecem uma funcionalidade chamada mapeamento objeto-relacional, do inglês object-relational mapping (**ORM**).

Esse mapeamento permite associar classes definidas em Python com tabelas em banco de dados, onde cada objeto dessas classes corresponde a um registro da tabela.

Os comandos SQL de inserções e consultas são todos realizados por meio de métodos, não sendo necessário escrever o comando SQL em si. Os ORM nos permitem trabalhar com um nível mais alto de abstração.

Como exemplo dessas bibliotecas, podemos citar **SQLAlchemy** e **Peewee**.

Visite as páginas dessas bibliotecas e veja como elas facilitam a manipulação de registros em banco de dados.