

Python 3 com Banco de Dados SQLite

Gerenciadores de bancos de dados

Anteriormente foi visto como gravar e ler arquivos texto para utilizá-los como forma de armazenamento permanente dos dados processados pelos programas. O uso de arquivos assim tem sua utilidade, porém, nem sempre é a melhor forma de implementar o armazenamento de dados.

O programador experiente já deve conhecer os conceitos que serão apresentados e pode pular diretamente para o próximo capítulo. Ao iniciante, convém ler atentamente e executar em seu computador as tarefas que serão sugeridas aqui.

Bancos de dados e bases de dados

Há muitas décadas tem-se desenvolvido a tecnologia de Sistemas Gerenciadores de Bancos de Dados (SGBD) ou, em inglês, *Data Base Management System* (DBMS), cujo objetivo maior é oferecer aos profissionais de desenvolvimento de sistemas uma base de armazenamento e recuperação de dados que seja ao mesmo tempo organizada, robusta, flexível, segura e escalável. Coloquialmente, um SGBD é conhecido como “o banco de dados”, e em uma conversa entre programadores é comum ouvir frases do tipo: “qual banco de dados vocês usam?” ou “onde está hospedado o seu banco de dados?”.

Ao usar frases assim, ocorre uma mistura de dois elementos distintos, que são simples e precisam ser compreendidos. O primeiro elemento é o SGBD em si, que é um software, ou seja, um conjunto de programas de computador que alguém – uma empresa, um consórcio ou uma comunidade virtual – desenvolve e mantém. O segundo elemento são os dados gerenciados por esse software.

Em um sistema computacional pode-se ter um software gerenciador de banco de dados, o qual é utilizado para gerenciar diversas **bases de dados**.

Assim, entenda-se este novo termo: base de dados é o conjunto de dados armazenados e utilizados pelos programas. Por exemplo, imagine-se um servidor de internet no qual estejam hospedados dez diferentes sites. Nesse servidor deve estar instalado um software de banco de dados, ou seja, um SGBD. Por outro lado, cada site tem sua base de dados, então, a conclusão imediata é que serão dez bases de dados nesse mesmo servidor. Em linhas gerais, é assim que



as coisas acontecem.

Existem muitos SGBDs no mercado de computação: Oracle, Microsoft SQL Server, MySQL, PostgreSQL, MongoDB, DB2 etc. A lista é bem grande, e o SQLite faz parte dela. Nessa lista, há sistemas que são proprietários e licenças precisam ser adquiridas para que se possa utilizá-los, assim como há os que são softwares livres e podem ser usados sob determinadas condições, a depender da licença de software livre sob a qual é disponibilizado.

Dada a importância atual do uso desses gerenciadores de banco de dados, tomou-se a decisão de incluir este capítulo neste livro sobre Python. Referente a banco de dados, nos cursos de Ciência da Computação e Desenvolvimento de Sistemas há disciplinas exclusivamente dedicadas ao assunto, de modo que aqui não se tem a pretensão nem espaço para cobrir e extinguir o assunto. Ao contrário, será uma abordagem totalmente de natureza prática, mostrando como a linguagem Python pode ser utilizada para escrever programas que se conectam a bases de dados e delas recebem, ou a elas enviam, seus dados.

Nos sistemas que entrarão em produção, escolher uma dentre tantas possibilidades de sistemas gerenciadores é tarefa para profissionais experientes e envolve a análise de muitos fatores.

A escolha por usar o SQLite foi em função de sua simplicidade e disponibilidade. Ele e a biblioteca necessária ao seu uso estão contidos nas instalações-padrão de Python, de modo que nenhum software adicional é necessário para que se utilize o par Python + SQLite.

Organização dos bancos de dados

Antes de iniciar o trabalho com o par Python + SQLite, é necessário apresentar alguns conceitos.

Em um SGBD, o ponto de partida conceitual é a forma de organização dos dados. A isso se dá o nome de **modelo** de banco de dados, e os quatro principais modelos são: hierárquico, rede, relacional e orientado a objetos. O modelo relacional é um dos mais utilizados, e toda a sua organização está fundamentada em tabelas, como a mostrada a seguir.



Nome da tabela: CADALUNOS

Colunas são Campos

	MATRICULA	NOME	DATANASC	ALTURA	PESO	CURSO	EMAIL
Linhas são Registros	1728	Carlos de Oliveira	12/04/1997	1,68	73,1	33	carlos@dominio.com
	1944	Diogo Sales	07/09/1996	1,82	86,3	29	diogo@dominio.com
	2093	Michele Sanches	14/08/1998	1,65	58,5	37	michele@provedor.com
	2155	Ana Paula Miranda	26/01/2997	1,73	69,6	33	ana.paula@algo.com.br

Chave primária

Elementos de uma tabela de banco de dados.

A Figura 8.1 contém os elementos essenciais de uma tabela de banco de dados do modelo relacional, a saber:

- **Nome da tabela:** toda tabela registrada na base de dados deve ter um nome de acordo com as regras de nomes adotadas no SGBD escolhido. Normalmente se utilizam letras, números e o caractere underline “_”. Na Figura 8.1 a tabela se chama CADALUNOS.
- **Registros:** cada linha da tabela é denominada registro e representa uma coleção heterogênea de dados interligados. Os registros são subdivisíveis em campos.
- **Campos:** é o elemento no qual um dado é armazenado. Um conjunto de campos forma um registro, ou linha da tabela. Os campos apresentam tipo – e, quando pertinente – tamanho definidos. Embora haja certa variação entre os diversos sistemas, os tipos básicos de campos são: texto, número inteiro, número real, data, hora, date e hora juntos, lógico (true/false), entre outros. Campos são identificados por nomes que seguem regras com as de nomes de tabelas. No exemplo, MATRICULA, NOME, DATANASC etc. são campos.
- **Chave primária:** é responsável pela identificação do registro no banco de dados. Pode ser constituída por um ou mais campos, e é obrigatório que seja única e não nula. No exemplo, a chave primária é a MATRICULA.

As bases de dados não têm apenas uma tabela, ao contrário, costumam ter muitas. Cada tabela pode ter centenas de campos com milhões de registros. E, além desses elementos citados no Exemplo 8.1, os bancos de dados também têm outros que não foram citados, como índices secundários, chaves estrangeiras, *views*, *stored procedures*, direitos de acesso (*grants*) etc. Ao modo como os dados são estruturados nas tabelas aplicam-se os conceitos que levam à



modelagem de dados, à organização das relações entre as tabelas, à normalização etc. E há, ainda, outras tantas coisas mais a se aprender. Como pôde perceber da leitura deste parágrafo, tal volume de conceitos tem mesmo de ser objeto de aprofundado estudo, o que justifica a existência das disciplinas de Banco de Dados citadas anteriormente.

O mais importante de tudo que foi exposto é que o básico contido no Exemplo 8.1 é o que basta, e com esse básico poderemos resolver muitos exercícios envolvendo a linguagem Python e o gerenciador de banco de dados SQLite.

A sigla SQL

Por que essa sigla está sempre presente?

Até existe uma tradução para o português desse termo, que é “linguagem de consulta estruturada”, mas ela não “pegou”, e os profissionais de computação brasileiros sempre usam “SQL”. Abreviação de *Structured Query Language*, o termo refere-se à linguagem utilizada para criar, armazenar, recuperar e atualizar dados em um banco de dados relacional. Foi desenvolvida no início dos anos 1970 pela IBM como parte do projeto que levou à criação do modelo relacional de bancos de dados (CHAMBERLIN, 1981).

SQL não é uma linguagem genérica, como C, Java ou Python. SQL é exclusivamente utilizada para interagir com bancos de dados relacionais e tem seus comandos divididos em grupos, segundo as operações que realizam:

- **DQL (*Data Query Language*):** este não é propriamente um grupo de comandos, pois aqui se conta apenas com o comando `select`. No entanto, é o mais usado e tem tantas variações que muitos programadores e gerentes de bancos de dados o classificam dessa maneira. O `select` é usado para buscar dados de tabelas, conforme mostrado nestes exemplos:

<code>select * from cadaluno</code>	Retorna todos os registros contidos na tabela CADALUNO.
<code>select nome, email from cadaluno where curso = 33</code>	Retorna os campos nome e email da tabela CADALUNO apenas para os registros em que o campo curso seja igual a 33.

- **DML (*Data Manipulation Language*):** são os comandos usados para realizar inclusões, alterações e exclusões de dados nas tabelas. As palavras-chave desses comandos



são: insert, update e delete. Exemplos de uso desses comandos serão vistos adiante.

- **DDL (*Data Definition Language*)**: são os comandos empregados para criar e excluir tabelas e seus elementos associados. As palavras-chave desses comandos são: create, alter e drop. Exemplos de uso desses comandos serão vistos adiante.

- **DCL (*Data Control Language*)**: são os comandos utilizados para controlar o acesso de usuários aos dados das tabelas, definindo “quem pode ver o quê”. Esses comandos não serão utilizados neste livro.

- **DTL (*Data Transaction Language*)**: são os comandos relacionados ao controle de transações no banco de dados, indicando quando e como os dados devem ser fisicamente salvos ou descartados. Existem com a finalidade de garantir a integridade de dados inter-relacionados. As palavras-chave desses comandos são: commit e rollback. Exemplos de uso desses comandos serão vistos adiante.

De tudo o que foi apresentado, espera-se que compreenda que há muito a ser aprendido sobre bancos de dados. No entanto, tal volume não impede que se compreenda e utilize o básico de BDs para a implementação de diversos programas escritos em Python que vão se conectar ao SQLite.

O Banco de dados SQLite

Há muitas opções de SGBDs, foi escolhido o SQLite para este livro simplesmente porque está disponível. O Python está instalado em seu computador? Se a resposta é sim, então, o SQLite está instalado também e o que é melhor: pronto para uso. Não requer download, não requer instalação, não requer configurações difíceis de entender, não requer um DBA (*Database Administrator*).

O SQLite é uma biblioteca que implementa as funções de gerenciamento de banco de dados de maneira autossuficiente, sem a necessidade de um computador servidor rodando um software servidor de banco de dados. Por isso, ele não requer qualquer configuração. Escrito em linguagem C, seu código foi colocado em domínio público por seus autores, ou seja, é um software livre e de código-fonte aberto. No entanto, existe uma versão melhorada que é paga e contém recursos para trabalhar com bancos de dados compactados e criptografados.

Estão disponíveis binários executáveis para diversos ambientes, como Windows, Linux, macOS, Android, iOS, entre outros.



O SQLite está distribuído em bilhões¹ de dispositivos diferentes e é muito leve, compacto e confiável, além de ser muito útil como repositório de dados para aplicações diversas. A versão mais recente disponível quando este livro foi escrito é a 3.21.0, de 24 de outubro de 2017.

Ao usá-lo neste texto, propicia-se ao iniciante a oportunidade de conhecer o mundo dos bancos de dados relacionais, e o programador experiente pode travar um primeiro contato com esse pequeno notável, caso ainda não o tenha utilizado em alguma outra oportunidade.

Sendo um produto de software tão bom e bem-aceito, ele tem aplicação em qualquer projeto de tecnologia da informação? Não necessariamente. Em sistemas cliente-servidor e aplicações web com muitos usuários simultâneos, muitos acessos concorrente e volumes de dados grandes o uso de SQLite não é recomendável. Em pequenos websites, aplicações que rodam em uma rede com poucos usuários ou em um único computador, aplicações para dispositivos móveis são o foco do SQLite.

Python + SQLite

Agora que se tem uma ambientação apropriada sobre bancos de dados e sobre o SQLite, está na hora de iniciar a parte prática.

Criação do primeiro banco de dados com Python + SQLite

Para os exemplos daqui por diante, suponha que está sendo construído um software para auxiliar no cadastro e no controle dos alunos de uma academia esportiva. Observe no Exemplo 8.1 o primeiro programa em que a dupla Python e SQLite é usada.

Na linha 1 é feita a importação da biblioteca `sqlite3`, para que o interpretador carregue os elementos que serão utilizados para conexão e interação com o SQLite. O próximo passo é estabelecer a conexão do Python com o banco de dados a ser aberto. Isso é feito com o método `connect` na linha 2. Esse método requer o nome do arquivo que contém o banco de dados. Caso esse banco de dados não exista, então, o método `connect` o criará como um banco de dados vazio. É exatamente isso que acontece na primeira vez em que o programa do Exemplo 8.1 for executado. O nome usado para o banco de dados é qualquer nome válido no sistema de arquivos do sistema operacional em que se está trabalhando. Como não foi especificada nenhuma pasta no nome do arquivo, então, ele será criado na mesma pasta em que está o programa. A extensão `.db` é uma mera escolha do programador, que pode escolher qualquer outra, ou nenhuma



extensão, conforme julgue apropriado.

```
Exemplo 8.1 Primeiro programa usando Python + SQLite import sqlite3 # linha 1
conector = sqlite3.connect("academia.db") # linha 2
cursor = conector.cursor() # linha 3 sql = """
create table cadastro

(codigo integer, nome text, idade integer) """
cursor.execute(sql) # linha 8 sql = """
insert into cadastro

(codigo, nome, idade) values (1284, 'Pedro de Oliveira', 32) """
cursor.execute(sql) # linha 13 sql = """
insert into cadastro

(codigo, nome, idade) values (1309, 'Maria Lúcia Machado', 37) """
cursor.execute(sql) # linha 18 conector.commit() # linha 19
cursor.close() # linha 20 conector.close() # linha 21
print("Abra a pasta do programa e veja se o arquivo está lá") print("Fim do programa")
```

Na linha 3 é criado o objeto que foi identificado como "cursor". Esse nome poderia ser qualquer outro válido, como x ou abc. No contexto de banco de dados, um cursor é um objeto utilizado pelo programador para se comunicar com o SGBD, enviando e recebendo comandos e dados. E o primeiro comando é enviado nas linhas 4 e 8. Nas linhas 4 a 7 é utilizado um *docstring* para preparar um comando (create table) que é atribuído ao objeto "SQL", e na linha 8 é utilizado o método cursor.execute para enviar o SQL ao banco de dados. Observe que nas linhas 8 a 13 e 14 a 18 esse processo é repetido, porém, com outros tipos de comandos (insert into). Você não deve se preocupar, por enquanto, com os detalhes dos comandos create table e insert into, pois eles são explicados no Quadro 8.1.

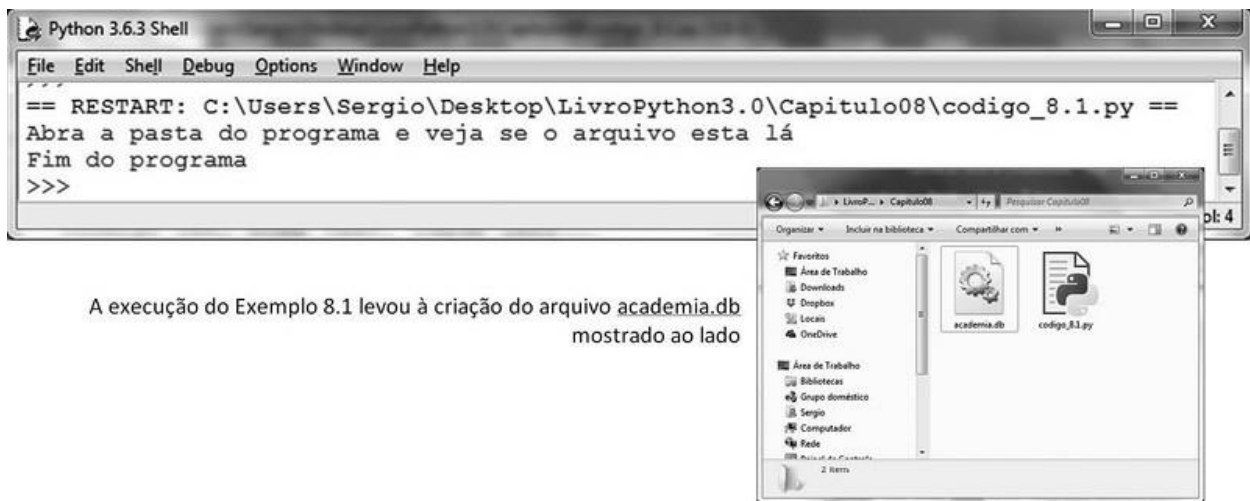
Na linha 19 é usado o método conector.commit, que pode ser entendido como o comando necessário para efetivamente salvar em disco os comandos enviados para o banco de dados. Na verdade, uma operação de commit encerra uma transação de banco de dados, e isso será abordado posteriormente. Antes de finalizar o programa, é preciso encerrar o cursor e a conexão



com o banco de dados. Isso é feito nas linhas 20 e 21 com os métodos close dos objetos cursor e conector.

A execução desse programa pode parecer estranha ao programador iniciante, pois não há interação do programa com o usuário, e seu resultado é apenas uma mensagem exibida no final, como mostrado na Figura 8.2.

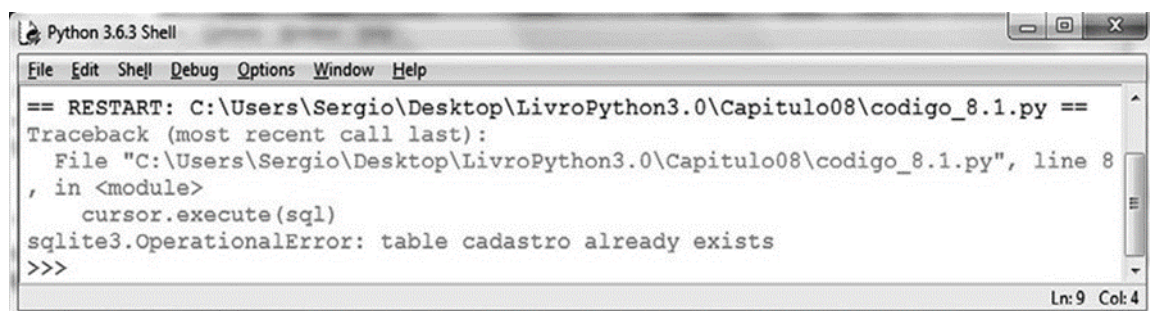
Porém, ao abrir a pasta onde o programa está salvo, constatará que ali está gravado o arquivo “academia.db” criado nessa execução.



A execução do Exemplo 8.1 levou à criação do arquivo `academia.db` mostrado ao lado

Resultado visível da execução do Exemplo 8.1.

Se executar esse programa novamente, o resultado será uma mensagem de erro, pois o programa se conectará ao banco de dados “academia.db” agora existente e nele tentará criar a tabela “cadastro”, também existente e levando à exibição do erro mostrado na Figura 8.3. Para rodar esse programa novamente sem obter a mensagem de erro, o arquivo do banco de dados deve ser excluído.



Resultado da segunda execução do Exemplo 8.1 – ocorre erro, a tabela cadastro já existe.



Comando	Descrição
<code>create table cadastro (código integer, nome text, idade integer)</code>	<p>Este é um comando SQL do grupo DDL (Data Definition Language) visto no Item 8.1.3. Ele é usado para criar uma tabela.</p> <p>Se a tabela já existir, ocorre um erro. Deve-se fornecer o nome da tabela: “cadastro”. Deve-se fornecer o nome e o tipo de dados de cada campo que essa tabela conterá. Neste exemplo, são três campos: “código” e “idade” são números inteiros (integer) e “nome” é texto (text).</p>
<code>insert into cadastro (codigo, nome, idade) values (1284, 'Pedro', 32)</code>	<p>Este é um comando SQL do grupo DML (Data Manipulation Language) visto no Item 8.1.3. Ele é utilizado para inserir dados em uma tabela. O nome da tabela deve ser fornecido seguido dos campos que receberão dado. Não é obrigatório que todos os campos da tabela estejam presentes. E na sequência da cláusula values colocam-se os dados que vão preencher os campos especificados.</p>

Quadro - Explicação dos comandos SQL usados no Exemplo 8.1.

Em síntese, o programa do Exemplo 8.1 faz as seguintes tarefas:

1. Carrega a biblioteca sqlite3 – linha 1.
2. Conecta-se com um BD existente ou cria o BD, caso não exista – linha 2.
3. Define um cursor para envio de comandos – linha 3.



4. Cria a tabela “Cadastro” – linhas 4 a 8.
5. Insere dois registros de dados na tabela “Cadastro” – linha 9 a 18.
6. Salva (commit) tudo no disco e encerra a transação – linha 19.
7. Encerra o cursor e a conexão – linha 20 e 21.

O resultado final é o arquivo do banco de dados gravado no disco com o nome especificado e dentro da pasta especificada.

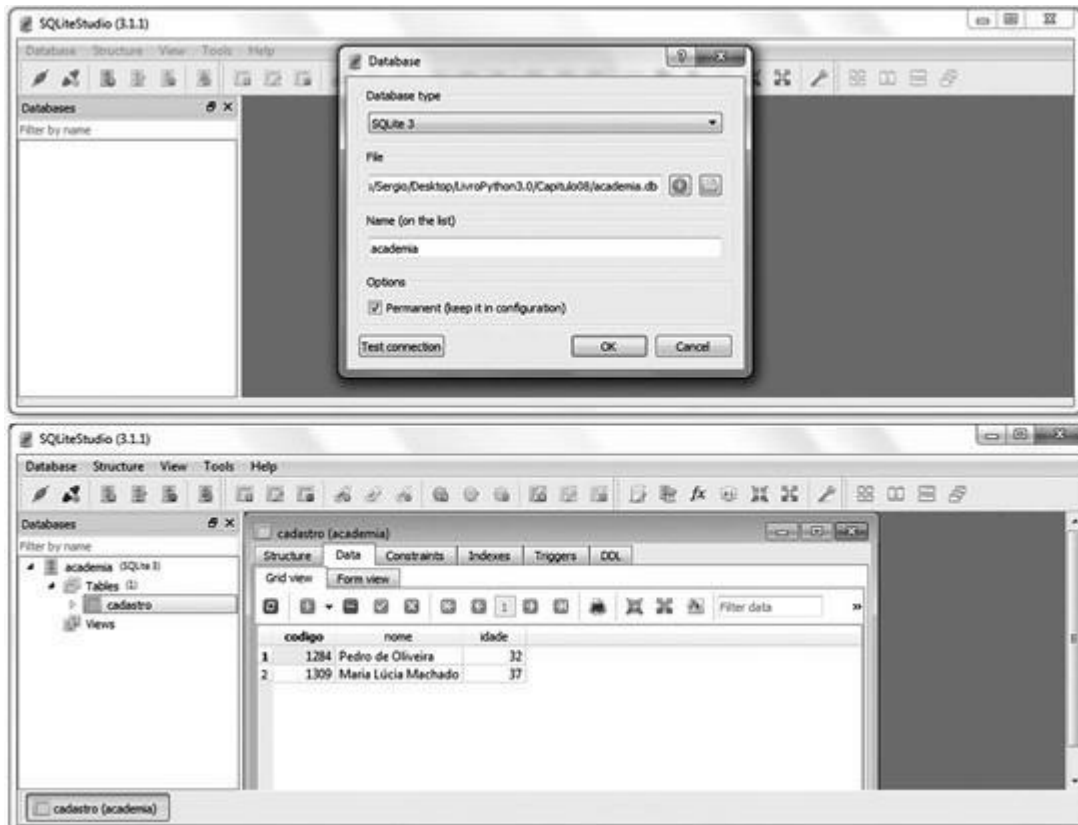
Inspecionando o banco de dados com o SQLite Studio

Qualquer banco de dados do SQLite é um arquivo gravado em disco. Como é um arquivo gravado em formato binário, não é possível abri-lo com um editor de textos, mas há várias opções para inspecionar seu conteúdo. Uma dessas opções é escrever um programa em Python para extrair os dados de uma tabela e exibí-los na tela. Isso será feito em seguida.

Existem diversos softwares que permitem acessar e manipular bancos de dados do SQLite. Muitos deles são gratuitos e de código livre, e alguns são pagos. Aqui será utilizado o SQLite Studio, que pode ser encontrado em <<https://sqlitestudio.pl>> e é um software livre, multiplataforma e que não requer instalação para ser usado. Basta acessar o endereço indicado, baixar a versão apropriada para seu computador, descompactar o arquivo e usar. A Figura mostra a aparência dele. Para começar a usá-lo, primeiro é preciso adicionar a base de dados. Para isso, clique no menu *Database* → *Add a Database*, selecione o banco de dados criado com o Exemplo 8.1 na caixa *File* e clique no botão *OK*. O nome do BD aparecerá no painel à esquerda. Dê um duplo clique sobre ele e pronto. A tabela criada estará disponível e poderá ser aberta, visualizada, editada etc.

Esse programa é uma ferramenta simples, porém, muito útil para verificar se os programas desenvolvidos em Python e SQLite estão gerando os resultados esperados.





SQLite Studio – utilitário para inspecionar um banco de dados do SQLite.

Como acessar os dados inseridos usando Python

Agora que o banco de dados “academia.db” já está criado, contém uma tabela e esta contém dois registros, o próximo passo é escrever um programa que acesse tais dados e os exiba na tela. O Exemplo 8.2 executa essa tarefa.

Exemplo 8.2 Exemplo de programa que acessa os dados presentes em uma tabela

```
import sqlite3 # linha 1
```

```
conector = sqlite3.connect("academia.db") # linha 2 cursor = conector.cursor() # linha 3
sql = "select * from cadastro" # linha 4 cursor.execute(sql) # linha 5
dados = cursor.fetchall() # linha 6 cursor.close() # linha 7 conector.close() # linha 8
print("\nConsulta ao Banco de Dados 'academia.db' \n") print("Dados da tabela 'cadastro'")
print("-" * 35)

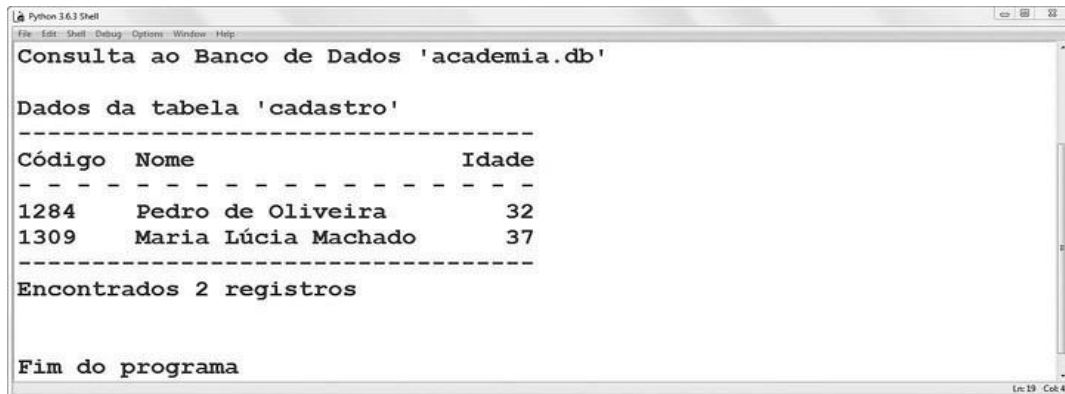
print("{:7} {:20} {:>6}".format("Código", "Nome", "Idade")) print("- " * 18)
for d in dados:
```



```
print("{:<7} {:>20} {:>6}".format(d[0], d[1], d[2]))
```

```
print("-" * 35)
```

```
print("Encontrados {} registros".format(len(dados))) print("\n\nFim do programa")
```



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Consulta ao Banco de Dados 'academia.db'

Dados da tabela 'cadastro'
-----
Código  Nome                Idade
-----
1284    Pedro de Oliveira         32
1309    Maria Lúcia Machado       37
-----
Encontrados 2 registros

Fim do programa
```

Resultado da execução do Exemplo 8.2.

Nesse exemplo, as linhas 1, 2 e 3 têm a mesma função de suas correspondentes explicadas no exemplo anterior. A linha 4 define o comando SQL select, explicado no Quadro 8.2, de acesso aos dados da tabela “Cadastro” e a linha 5 executa o comando. Após essa execução, tem-se a linha 6, que é a chave desse exemplo. Nela foi usado o método `cursor.fetchall`, cujo efeito é retornar todos os registros produzidos pelo select em um objeto lista do Python. Uma vez que a lista “dados” já esteja carregada, a conexão com o BD pode ser encerrada por meio das linhas 7 e 8. Após isso, o conteúdo lista pode ser utilizado da maneira que for necessária ao programa. No caso do exemplo, foi elaborada uma saída formatada.

Comand	Descrição
o	



<pre>select * from cadastro</pre>	<p>Este é um comando SQL do grupo DQL (<i>Data Query Language</i>), visto no Item 8.1.3. Ele é usado para acessar os dados existentes em uma tabela. Essa é a forma mais simples desse comando.</p> <p>O “select *” significa que todos os campos devem ser selecionados e retornados. Alternativamente ao asterisco, poderia ser colocada uma lista de campos da tabela e, nesse caso, apenas os campos listados são retornados.</p> <p>A cláusula from determina qual tabela será acessada.</p> <p>O comando select é o mais versátil e variado comando SQL. Outras formas deste serão vistas em outros exemplos.</p>
---	---



Quadro - Explicação do comando SQL usado no Exemplo 8.2.

Inserindo mais registros na tabela “Cadastro”

No próximo exemplo mais registros serão inseridos na tabela “Cadastro” desse banco de dados. Para isso, o programa permanecerá em laço até que seja digitada uma linha nula, ou seja, o usuário apertar *Enter* sem digitar nada. Os dados de Código, Nome e Idade deverão ser digitados pelo usuário separados por vírgulas. Para cada linha válida digitada os dados devem ser inseridos na tabela.

A solução para essa nova situação está apresentada no Exemplo 8.3. Nessa solução, parte do código foi separado na função `ExibeDados`. Basicamente, o código dessa função é a parte final do código presente no Exemplo 8.2. Ela recebe o parâmetro `L`, que será a lista produzida pelo retorno dos dados do BD.

Esse programa inicia sua execução na linha 15. Nessa linha e na seguinte é feita a conexão com o banco e a criação do cursor. Na linha 17 o objeto `sql` recebe o string com o comando `insert into`, que, neste caso, é diferente do Exemplo 8.1. Note que a cláusula `values` faz referência a um conjunto de interrogações (`?, ?, ?`). Tais interrogações são **parâmetros** que indicam ao SQLite que os dados serão passados à parte, e seu modo de funcionamento é ilustrado na Figura. Observe no código do programa que a linha 27 contém o método `cursor.execute` com dois parâmetros: o SQL e um objeto lista identificado como `D`. Essa abordagem garante flexibilidade ao programa, uma vez que o mesmo SQL poderá ser executado várias vezes, cada uma com um conjunto de dados diferente carregado no objeto `D`.

Regras quanto ao uso de parâmetros passados do Python para o SQLite:

1. O comando SQL deve ser escrito com o caractere interrogação “?” no local onde deve entrar o dado real. Podem ser usadas tantas interrogações quantas necessário.
2. No método `cursor.execute(SQL, Objeto)` devem estar presentes, além do comando SQL, um segundo parâmetro (Objeto), que fornecerá os dados para que seja feita a substituição das interrogações por dados reais.
3. O segundo parâmetro deve ser uma tupla ou uma lista, mesmo que só exista uma interrogação no SQL. Tipos simples, conjuntos não são aceitos. Dicionários são aceitos, mas isso será visto no item 8.2.8.
4. A quantidade de interrogações e a quantidade de elementos no Objeto



passado devem ser as mesmas. Caso contrário, ocorrerá erro.

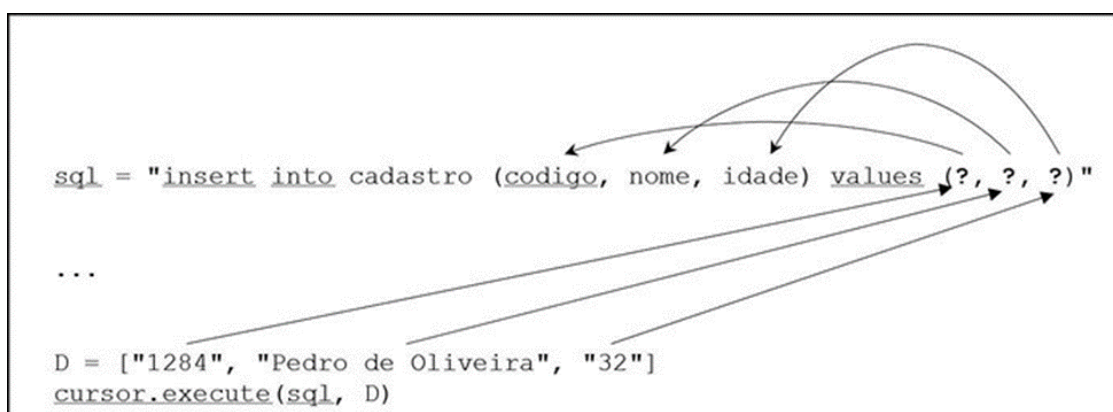


Ilustração de como são processados os comandos SQL com parâmetros.

Há, ainda, mais um aspecto importante relacionado à passagem de parâmetros do Python ao SQLite. É possível observar, na Figura 8.6, que o objeto D foi carregado com três elementos do tipo string. Porém, o primeiro e o terceiro, “1284” e “32”, serão, respectivamente, associados aos campos “codigo” e “idade”, que são campos numéricos inteiros. Assim sendo, o SQLite está recebendo do Python dados com formatos em desacordo com o tipo dos campos que devem ser preenchidos.

Essa é uma situação que deve ser evitada. Em outros gerenciadores de bancos de dados, uma situação assim geraria um erro e a execução do SQL falharia. No entanto, o SQLite é um SGBD mais flexível que a maioria dos outros, e quando isso acontece ele tenta acomodar a situação verificando se os dados passados como string podem ser convertidos para os tipos numéricos dos campos de destino. Se essa conversão for possível, ele a executa e não gera qualquer mensagem de erro. Para mostrar que isso funciona, no Exemplo 8.3 a lista D foi mantida com três elementos string, como poderá constatar. Em outros programas deste livro os dados sempre serão convertidos para o tipo correto no Python antes do envio ao SQLite.

Prosseguindo com a descrição do exemplo, a carga de dados no objeto D é feita a partir da leitura do teclado. Os três dados: Código, Nome e Idade, deverão ser digitados separados por vírgulas. O comando input na linha 23 carregará o objeto Ler com o string lido do teclado e o método split será usado para separar as três partes usando o caractere “,” como delimitador. Se for necessário relembrar o uso do método split.

Na parte final desse programa é utilizado um select para ler todos os dados da tabela



“Cadastro” e a função ExibeDados é chamada para executar a exibição. A Figura 8.7 mostra a execução desse programa e a Figura 8.8 mostra, por meio do SQLite Studio, que os novos dados realmente estão contidos na nova tabela.

Exemplo - Inserção de mais registros na tabela “Cadastro”

```
import sqlite3
def ExibeDados(L): # linha 3
```

```
    """Exibe uma saída formatada dos dados contidos em L"""
    print("\nConsulta ao Banco de
    Dados 'academia.db' \n")
    print("Dados da tabela 'cadastro")
```

```
    print("-" * 35)
```

```
    print("{:7} {:20} {:20>6}".format("Código", "Nome", "Idade"))
    print("-" * 18)
```

```
    for d in L:
```

```
        print("{:<7} {:20} {:>6}".format(d[0], d[1], d[2]))
```

```
    print("-" * 35)
```

```
    print("Encontrados {} registros".format(len(dados)))
    conector = sqlite3.connect("academia.db") # linha 15
```

```
    cursor = conector.cursor() # linha 16
    sql = """
```

```
    insert into cadastro
```

```
    (codigo, nome, idade) values (?, ?, ?) """
```

```
    print("Digite os dados separados por vírgulas")
    print("Codigo, Nome, Idade") # linha 22
```

```
    Ler = input() # linha 23
    while Ler != "": # linha 24
        D = Ler.split(",") # linha 25
        try: # linha 26
```

```
            cursor.execute(sql, D) # linha 27
            conector.commit() # linha 28
        except:
```

```
            print("{} Dados inválidos".format(D))
        else:
```

```
            print("{}*30, "...dados inseridos com sucesso")
```

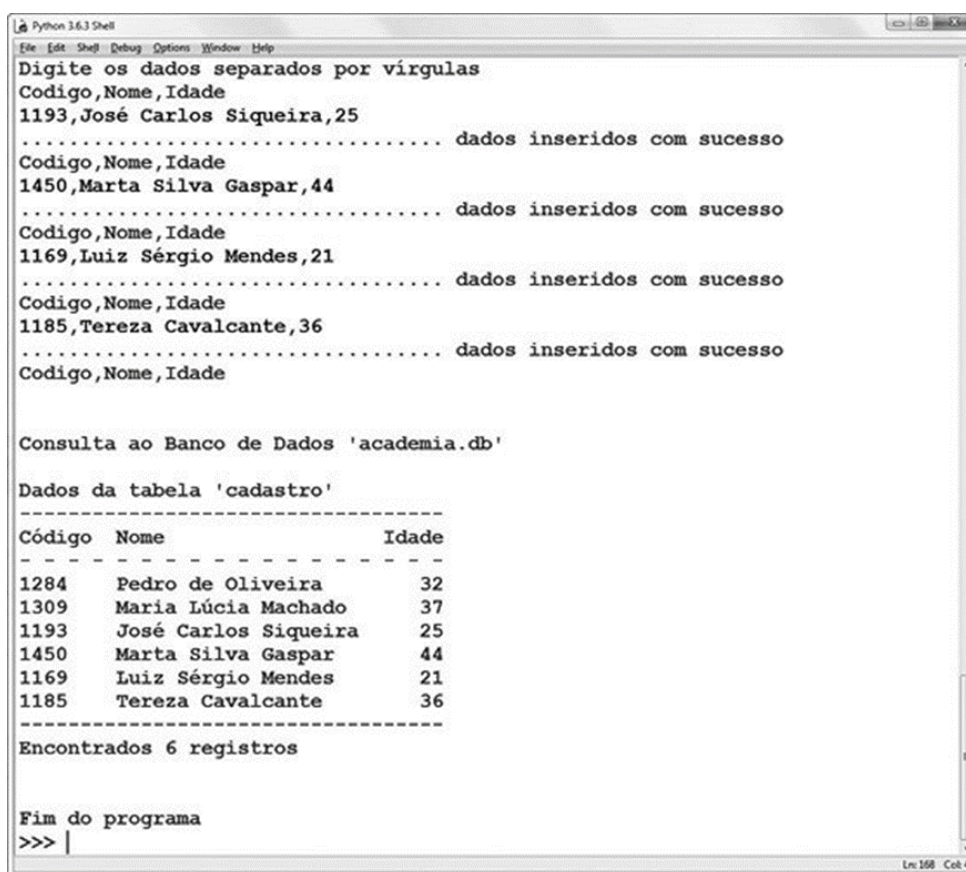
```
    finally:
        print("Codigo, Nome, Idade")
```

```
    Ler = input() # linha 35
```

```
    sql = "select * from cadastro" # linha 37
    cursor.execute(sql) # linha 38
```



dados = cursor.fetchall() # linha 39 cursor.close() # linha 40 conector.close() # linha 41
ExibeDados(dados) # linha 42 print("\n\nFim do programa")



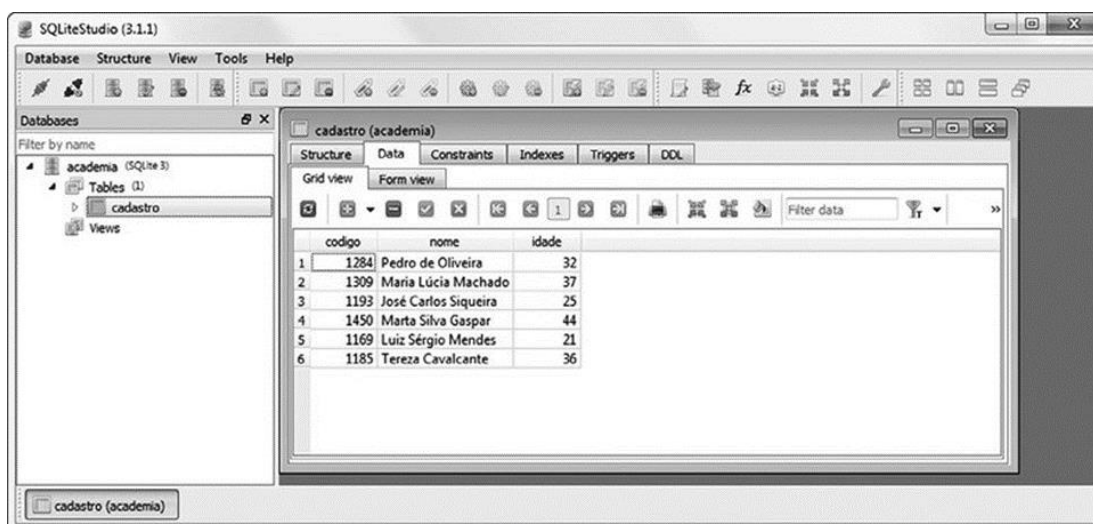
```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Digite os dados separados por vírgulas
Codigo,Nome,Idade
1193,José Carlos Siqueira,25
..... dados inseridos com sucesso
Codigo,Nome,Idade
1450,Marta Silva Gaspar,44
..... dados inseridos com sucesso
Codigo,Nome,Idade
1169,Luiz Sérgio Mendes,21
..... dados inseridos com sucesso
Codigo,Nome,Idade
1185,Tereza Cavalcante,36
..... dados inseridos com sucesso
Codigo,Nome,Idade

Consulta ao Banco de Dados 'academia.db'

Dados da tabela 'cadastro'
-----
Código Nome Idade
-----
1284 Pedro de Oliveira 32
1309 Maria Lúcia Machado 37
1193 José Carlos Siqueira 25
1450 Marta Silva Gaspar 44
1169 Luiz Sérgio Mendes 21
1185 Tereza Cavalcante 36
-----
Encontrados 6 registros

Fim do programa
>>> |
```

Figura -Resultado da execução do Exemplo 8.3.



Visualização do BD com SQLite Studio, após a execução do Exemplo 8.3.

