

## Alteração de tabelas

Muitas vezes é necessário alterar a estrutura de tabelas já existentes, incluindo ou excluindo campos, acrescentando índices ou até mesmo alterando sua chave primária. Em um projeto de um sistema novo benfeito isso não deve ocorrer. Porém, ao longo da vida útil de um sistema mudanças assim podem acontecer em virtude de alterações nas regras de negócio impostas por fatores externos ao sistema, tais como mudanças no negócio, implantação de novas funcionalidades, mudanças de legislação, adoção de novas tecnologias, entre outras motivações. Serão vistos a seguir três exemplos em que isso é realizado.

No Exemplo 8.4 são incluídos quatro novos campos na tabela “Cadastro”. Um campo inteiro que será utilizado para armazenar o número do curso que a pessoa frequenta na academia, a data de ingresso, que é uma data, e o peso e a altura do aluno, que são números reais. Depois de incluídos, esses campos precisarão ser carregados com conteúdo. Os campos curso e data de ingresso serão respectivamente 10 (musculação) e 01/07/2017 (quando a academia iniciou as atividades e os alunos já cadastrados a frequentam desde o início).

Para efetuar a inclusão de novos campos, é usado um comando SQL do grupo DDL alter table, e para atualizar o conteúdo dos campos é utilizado update, um comando do grupo DML. Ambos são explicados no Quadro 8.3.

Nesse programa não há nenhum aspecto novo no que diz respeito ao Python. São os mesmos comandos utilizados nos exemplos anteriores para: conexão com o banco de dados, criação do cursor, execução dos comandos DDL e DML, commit (necessário apenas por causa do update) e encerramento da conexão. Esse programa não interage com o usuário, e ao seu término uma mensagem é mostrada na tela indicando que o BD foi atualizado.

Comando	Descrição
alter table cadastro	Este é um comando SQL do grupo DDL ( <i>Data Definition Language</i> ) visto no Item 8.1.3. Ele é usado para acrescentar campos a uma tabela já existente. Os novos campos



add curso integer	inseridos não terão valor e estarão com conteúdo nulo ( <i>NULL</i> ). Deve-se usar um comando deste para cada campo a ser incluído.
update cadastro set curso = 16, dtingr = '01/07/2017'	Este é um comando SQL do grupo DML ( <i>Data Manipulation Language</i> ) visto no Item 8.1.3. Este comando atualiza todos os registros da tabela colocando em cada campo os valores que lhes foram atribuídos por meio da cláusula set.

**Quadro 8.3** Explicação do comando SQL utilizado no Exemplo 8.2.

**Exemplo 8.4** Inclusão de campos em tabelas import sqlite3

```
conector = sqlite3.connect("academia.db") cursor = conector.cursor()
sql = "alter table cadastro add curso integer" cursor.execute(sql)
sql = "alter table cadastro add dtingr date" cursor.execute(sql)
sql = "alter table cadastro add peso double"
cursor.execute(sql)

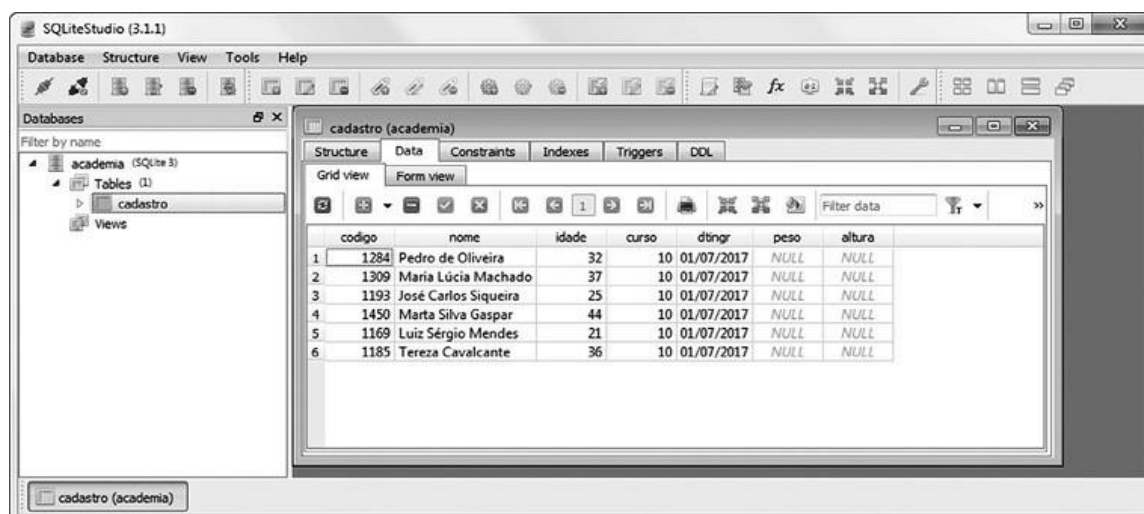
sql = "alter table cadastro add altura double" cursor.execute(sql)
sql = "update cadastro set curso = 10, dtingr = '01/07/2017'" cursor.execute(sql)
conector.commit() cursor.close() conector.close()
print("\n\nBanco de dados atualizado com sucesso")

print("\n\nFim do programa")
```



Na Figura 8.9 pode constatar que a tabela está alterada.





Resultado da execução do Exemplo 8.4 com os novos campos incluídos.

O Exemplo 8.5 é utilizado para atualizar os campos de peso e altura. Como esses dados variam para cada pessoa, é preciso fazer um update em separado usando a cláusula where para limitar os registros que serão afetados. O Quadro 8.4 mostra o formato deste SQL.

Comando	Descrição
update cadastro set peso = ?, altura = ? where codigo = ?	<p>Este é um comando SQL do grupo DML (<i>Data Manipulation Language</i>) visto no Item 8.1.3. Este comando atualiza apenas os registros que tiverem o código que foi usado na cláusula where.</p> <p>Se nenhum registro satisfizer o critério, nada acontece.</p> <p>Neste caso, são utilizadas as interrogações porque serão feitas várias atualizações passando os dados uma pessoa por vez.</p>

**Quadro 8.4** Explicação do comando SQL usado no Exemplo 8.2

Para escrever esse programa, os dados de peso e altura de cada pessoa devem estar disponíveis. Seria possível fazer como no Exemplo 8.3 e pedir para o usuário fazer a



digitação. Porém, aqui será lido um arquivo texto no qual cada linha contém código, peso e altura de uma pessoa separados com “;”, como no exemplo a seguir:

```
1284;93.5;1.74
```

```
1309;53.6;1.62 ... etc.
```

Como o arquivo é pequeno, utilizou-se o método `readlines` para carregar uma lista com o arquivo inteiro e, em seguida, a lista é processada, fazendo-se o processamento de cada um de seus elementos.

**Exemplo 8.5** Atualização do banco de dados a partir da leitura de um arquivo em disco

```
import sqlite3
```

```
arq = open("PesoAltura.txt", "r") L = arq.readlines()
arq.close()
```

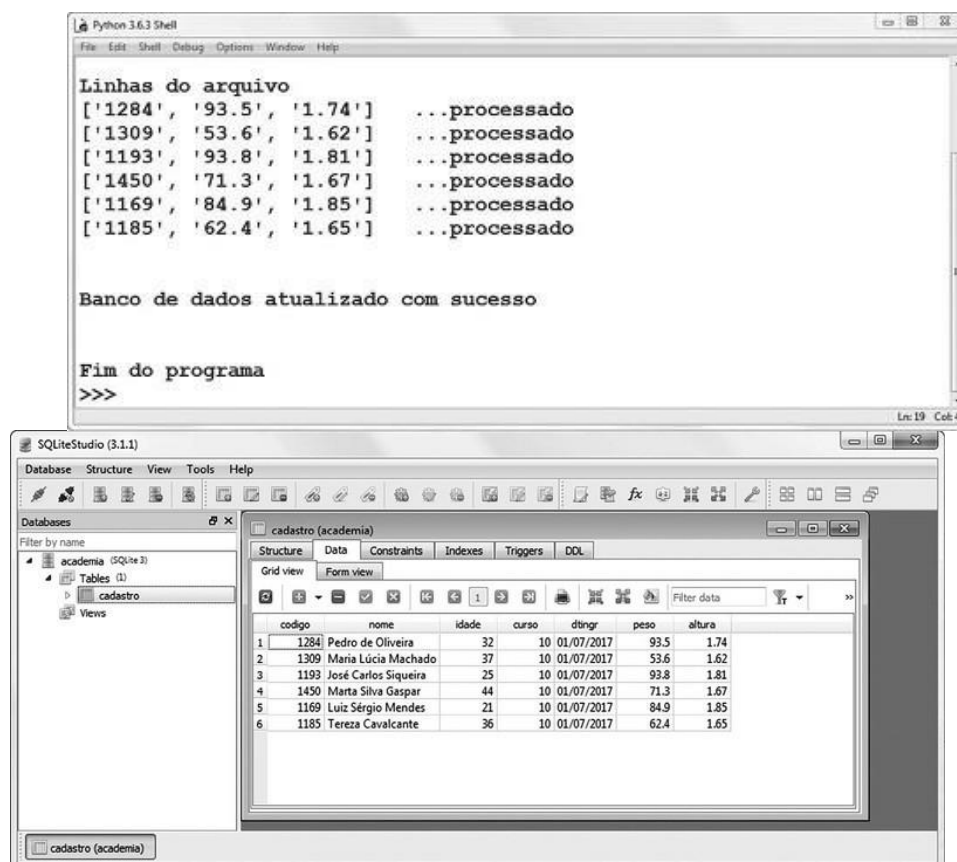
```
print("\nLinhas do arquivo")
```

```
sql = """update cadastro set peso = ?, altura = ? where codigo = ?"""
conector = sqlite3.connect("academia.db") cursor = conector.cursor()
for s in L:
    d = s.rstrip()

    d = d.split(";")
```

```
cursor.execute(sql, (d[1], d[2], d[0])) conector.commit()
print(d, "...processado") cursor.close() conector.close()
print("\n\nBanco de dados atualizado com sucesso")
```

```
print("\n\nFim do programa")
```



The image shows two windows. The top window is a Python 3.6.3 Shell displaying the output of a script. It lists six lines of data from a file, each followed by "...processado". Below this, it says "Banco de dados atualizado com sucesso" and "Fim do programa". The bottom window is SQLiteStudio 3.1.1, showing a database named "academia" with a table named "cadastro". The table has columns: codigo, nome, idade, curso, dtinsc, peso, and altura. The data is displayed in a grid view.

	codigo	nome	idade	curso	dtinsc	peso	altura
1	1284	Pedro de Oliveira	32	10	01/07/2017	93.5	1.74
2	1309	Maria Lúcia Machado	37	10	01/07/2017	53.6	1.62
3	1193	José Carlos Siqueira	25	10	01/07/2017	93.8	1.81
4	1450	Marta Silva Gaspar	44	10	01/07/2017	71.3	1.67
5	1169	Luiz Sérgio Mendes	21	10	01/07/2017	84.9	1.85
6	1185	Tereza Cavalcante	36	10	01/07/2017	62.4	1.65

Resultado da execução do Exemplo 8.5 com os campos peso e altura atualizados.

Na Figura 8.10, pode-se verificar que de fato cada registro está atualizado com os valores corretos de peso e altura. No caso desse exemplo, o campo “codigo” foi usado como critério de seleção para a atualização dos registros. É frequente em bancos de dados que isso aconteça. Na verdade, um campo como esse é tão importante que recebe tratamento especial.

Na tabela “Cadastro” o campo “codigo” é uma **chave primária**. Porém, quando a tabela foi criada isso não foi especificado. Campos-chave são indexados e, por isso, permitem acesso muito mais rápido aos dados. Eles também devem obrigatoriamente ser preenchidos, não podendo ser *NULL*, e não pode haver repetição, ou seja, cada registro tem sua chave primária com valor único.

O que será feito no Exemplo 8.6 é transformar o campo “código” em chave primária. No entanto, há um problema: o SQLite não permite que essa transformação seja feita



diretamente. Para realizar essa transformação, é preciso fazer os seguintes passos:

1. Clonar a tabela “cadastro” para uma tabela temporária.
2. Eliminar a tabela “cadastro”.
3. Criar uma nova tabela “cadastro”, dessa vez, com chave primária e todos os demais campos.
4. Copiar os dados da tabela temporária para a tabela “cadastro”.
5. Eliminar a tabela temporária.

É isso que está implementado no código a seguir.

```
Exemplo 8.6 Transformação do campo “codigo” em chave primária
import sqlite3
print("\nCriação de Chave Primária na tabela 'cadastro'")
conector = sqlite3.connect("academia.db")
cursor = conector.cursor()

sql = "create table temp as select * from cadastro" # passo 1
cursor.execute(sql)
sql = "drop table cadastro" # passo 2
cursor.execute(sql)
sql = ""

create table cadastro (
codigo integer NOT NULL PRIMARY KEY, nome text,
idade integer, curso integer, dtingr date, peso double,
altura double)
cursor.execute(sql) # passo 3
sql = ""

insert into cadastro
(codigo, nome, idade, curso, dtingr, peso, altura)
select codigo, nome, idade, curso,
dtingr, peso, altura from temp

"""
```



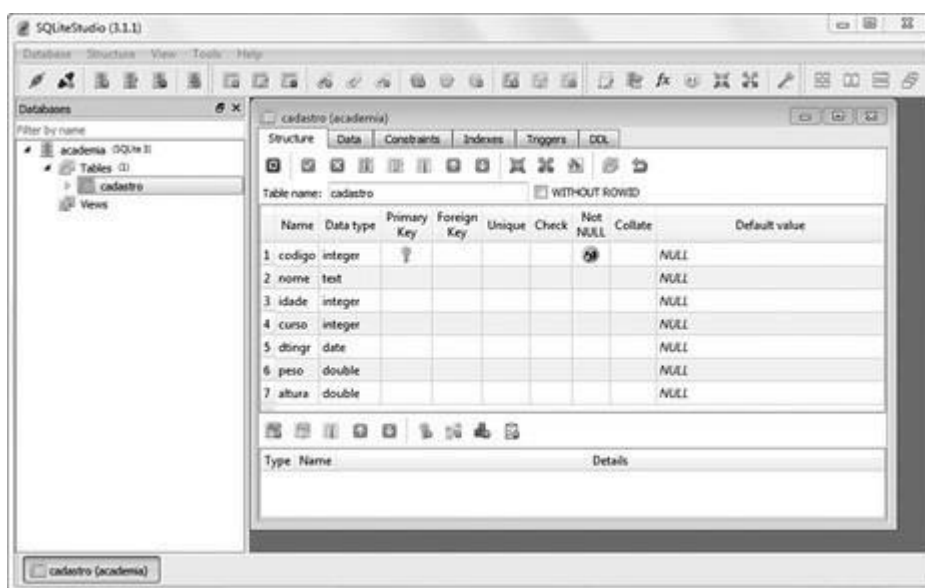
```
cursor.execute(sql) # passo 4
```

```
conector.commit() # commit necessário para o insert into sql = "drop table temp"
```

```
cursor.execute(sql) # passo 5 cursor.close() conector.close()
```

```
print("\n\nBanco de dados atualizado com sucesso") print("\n\nFim do programa")
```

O resultado do processamento desse exemplo pode ser visto em seguida. Usando o SQLite Studio, clicando na aba *Structure*, pode-se verificar que agora o campo “codigo” está identificado como Not NULL e Primary Key.



Resultado do Exemplo 8.6 que transforma o campo “codigo” em chave primária.

### Alteração dos Exemplos 8.2 e 8.3 Tarefa adicional

Agora, a tabela “Cadastro” tem diversos campos a mais do que tinha quando foi feito o Exemplo 8.2, que exibe em tela todos os seus registros. Adapte aquele exemplo para exibir a tabela ampliada com a formatação mostrada na Figura 8.12.





```
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  Curso  Ingresso      Peso      Altura
-----
1284    Pedro de Oliveira          32     10    01/07/2017    93.5      1.74
1309    Maria Lúcia Machado        37     10    01/07/2017    53.6      1.62
1193    José Carlos Siqueira       25     10    01/07/2017    93.8      1.81
1450    Marta Silva Gaspar         44     10    01/07/2017    71.3      1.67
1169    Luiz Sérgio Mendes         21     10    01/07/2017    84.9      1.85
1185    Tereza Cavalcante          36     10    01/07/2017    62.4      1.65
-----

Encontrados 6 registros

Fim do programa
```

Exibição ampliada dos dados da tabela “Cadastro”.

### Tarefa adicional

O Exemplo 8.3 também pode ser adaptado para ler as novas informações para o cadastro do aluno da academia (todas em uma linha e separadas por vírgulas) e, após a leitura, fazer a inserção no banco de dados. Se desejar, você pode fazer essa alteração. Ao fazer a inserção de novos alunos, forneça variados cursos e diferentes datas de ingresso.

### Criação e preenchimento de nova tabela

O banco de dados da academia já conta com a tabela “Cadastro”. Agora, será criada uma nova tabela para conter dados dos cursos oferecidos. Essa tabela terá os campos descritos no Quadro 8.5 e os dados do Quadro 8.6.

Campo	Tipo	Observações
codcurso	Número inteiro (integer)	Não pode ser nulo e é chave primária.
nomecurso	Texto	Será usado para conter o nome do curso.
valores	Número real (double)	Será usado para conter o valor da mensalidade do curso.





**Quadro - Campos da tabela "Cursos".**

Código	Nomecurso	Valores
10	Musculação	110,00
11	Treino aeróbico	110,00
12	Combo 1: musculação + aeróbico	180,00
15	Natação	180,00
22	Pilates	165,00
25	Combo 2: pilates + aeróbico	214,00
30	Crossfit	180,00
41	Muay Thai	140,00
42	Jiu Jitsu	140,00
43	Boxe	140,00

**Quadro 8.6** Campos da tabela "Cursos".

Esse exemplo tem por objetivo demonstrar o uso do método `cursor.executemany`, que pode ser utilizado no lugar do método `cursor.execute` para os casos em que se necessita trabalhar com múltiplos conjuntos de dados. O objeto `DadosCursos` é uma lista e foi carregado com diversas tuplas, cada uma contendo os três dados referentes a um curso. Essa lista de tuplas é passada para o método `cursor.executemany` em conjunto com o comando SQL definido nas linhas 14 a 17, o qual se encarregará de executar o SQL uma vez para cada tupla contida na lista `DadosCursos`.

**Exemplo 8.7** Criação de nova tabela e carga de dados usando `executemany` import `sqlite3`  
`print("\n\nCria e carrega tabela 'cursos'") conector = sqlite3.connect("academia.db")`

```
cursor = conector.cursor() #Cria a nova tabela
```

```
sql = ""
```

```
create table cursos
```

```
(codcurso integer not NULL Primary Key, nomecurso text, valormes double)
```



```
cursor.execute(sql)
```

```
print("\n ...tabela cursos criada") #Carrega com dados dos cursos # linha 13 sql = ""
```

```
insert into cursos (codcurso, nomecurso, valores) values(?, ?, ?)
```

```
""" # linha 17 DadosCursos = [ # linha 18
```

```
(10, "Musculação", 110.00),
```

```
(11, "Treino Aeróbico", 110.00),
```

```
(12, "Combo 1: Musculação + Aeróbico", 180.00),
```

```
(15, "Natação", 180.00),
```

```
(22, "Pilates", 165.00),
```

```
(25, "Combo 2: Pilates + Aeróbico", 240.00),
```

```
(30, "Crossfit", 180.00),
```

```
(41, "Muay Thai" , 140.00),
```

```
(42, "Jiu Jitsu", 140.00),
```

```
(43, "Boxe", 140.00)]
```

```
print("\n ...dados a serem carregados") for dados in DadosCursos:
```

```
print(" ", dados)
```

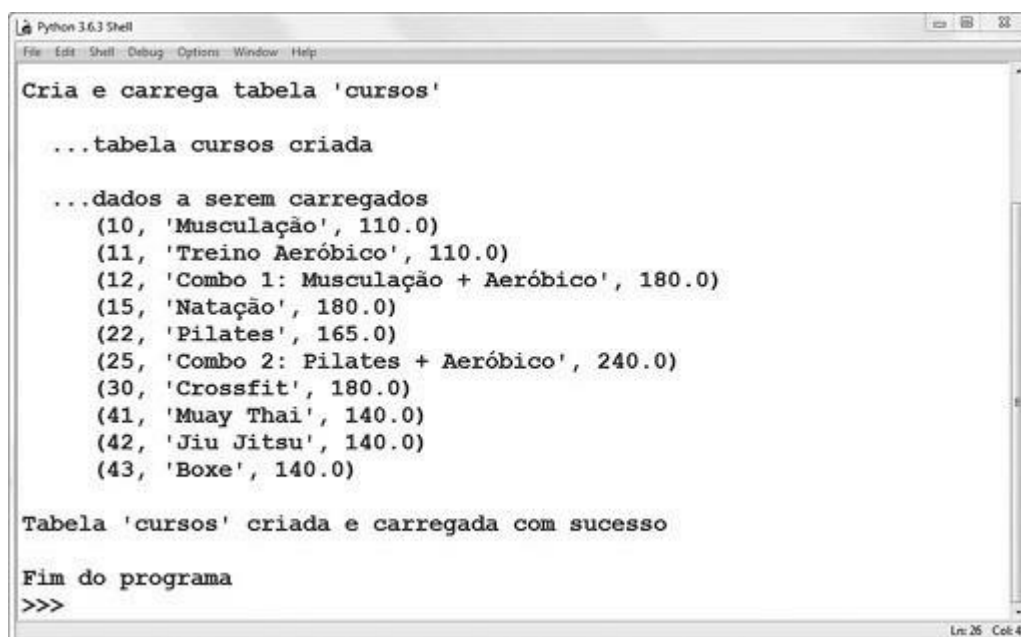
```
cursor.executemany(sql, DadosCursos) # linha 32 conector.commit()
```

```
cursor.close() conector.close()
```



```
print("\nTabela 'cursos' criada e carregada com sucesso") print("\nFim do programa")
```

Como resultado da execução desse exemplo, a tabela “cursos” será criada no banco de dados da academia. As Figuras 8.13 e 8.14 mostram esse resultado. Uma observação sobre o SQLite Studio: caso tenha executado esse programa com o Studio aberto e conectado ao banco de dados, ao alternar do programa Python para o Studio, a nova tabela não aparecerá automaticamente. Para que apareça, deve-se acionar o comando de menu *Database* → *Refresh selected database scheme* ou pressionar a tecla F5.



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help

Cria e carrega tabela 'cursos'

...tabela cursos criada

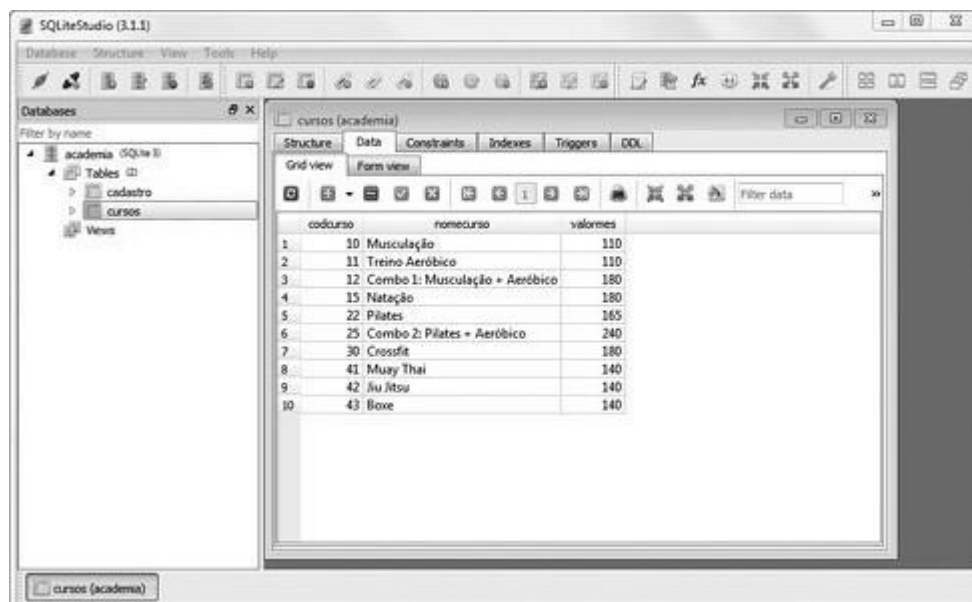
...dados a serem carregados
(10, 'Musculação', 110.0)
(11, 'Treino Aeróbico', 110.0)
(12, 'Combo 1: Musculação + Aeróbico', 180.0)
(15, 'Natação', 180.0)
(22, 'Pilates', 165.0)
(25, 'Combo 2: Pilates + Aeróbico', 240.0)
(30, 'Crossfit', 180.0)
(41, 'Muay Thai', 140.0)
(42, 'Jiu Jitsu', 140.0)
(43, 'Boxe', 140.0)

Tabela 'cursos' criada e carregada com sucesso

Fim do programa
>>>
```

Resultado da execução do Exemplo 8.7 que cria e carrega a tabela “Cursos”.





Como fica nova tabela “Cursos” no SQLite Studio.

### Tarefa adicional

Escreva um programa para inserir novos alunos na tabela cadastro, usando o método executemany. Faça esse programa de modo que os dados dos alunos a serem inseridos no cadastro sejam lidos de um arquivo em disco, a exemplo do que foi feito no Exemplo 8.5. Será necessário criar esse arquivo para testar o programa. O Quadro 8.5 contém dados sugeridos para realizar os testes.

```
1227;Maria do Carmo Sila;54;12;16/07/2017;88.2;1.59
1377;Onofre Vilasboas;52;15;14/10/2017;108.7;1.78
1453;Takeshi Yamazaki;19;12;09/08/2017;71.1;1.75
1073;Lucas Marcolino;43;15;01/09/2017;84.9;1.80
1487;João Silva Ponte;23;30;10/07/2017;66.7;1.77
1002;Kaique Guimarães;19;30;10/07/2017;70.7;1.83
1230;Luciana Cardal Pó;46;41;14/10/2017;84.8;1.67
1103;Giulliana Trovatto;22;25;15/09/2017;85.1;1.69 1233;Piera
Trovatto;46;25;15/09/2017;84.8;1.62
1046;Vinícius Catagallo;31;41;16/07/2017;98.0;1.79
```

**Quadro 8.7** Dados de novos alunos para a tarefa adicional referente ao Item 8.2.7.



## Exclusão de registros em uma tabela

No próximo exemplo serão apresentados alguns novos aspectos da programação para banco de dados com Python e SQLite. Esse programa contém duas funções, além da parte principal:

- A função `ExibeCursos` é usada para produzir uma saída em tela formatada com aspecto de tabela que exibe todos os cursos cadastrados.
- A função `ExcluiCurso(Codigo)` recebe o parâmetro `Codigo` e verifica se existe um curso com o código que é passado. Caso não exista, ela retorna uma mensagem de texto informando que não existe. Caso exista, é feita a exclusão e retorna uma mensagem de texto informando que a exclusão ocorreu.
- A parte principal do programa contém um laço que só terminará quando for digitado zero (a estratégia utilizada foi laço infinito interrompido por `break` quando a opção digitada for igual a zero). Dentro desse laço, é feita uma chamada da função `ExibeCursos`, é apresentado um texto solicitando a entrada do usuário e, uma vez que algo tenha sido digitado, é feita uma chamada à função `ExcluiCurso` para processar a entrada do usuário.

Antes que o laço termine, a instrução

```
dummy = input("Pressione Enter para prosseguir...")
```

foi acrescentada para que o programa dê uma parada nesse ponto e o usuário tenha condição de ler a mensagem exibida. Após pressionar *Enter*, a tela é redesenhada (rola para cima) e o laço inicia uma nova repetição.

### Exemplo 8.8 Exclusão de registro em tabela import sqlite3

```
def ExibeCursos():
```

```
    """Exibe os cursos existentes em uma saída formatada""" sql = "select * from cursos"
    cursor.execute(sql) dados = cursor.fetchall()
    print("\nConsulta ao Banco de Dados 'academia.db' \n") print("Dados da tabela 'cursos'")
    print("-" * 49)
```



```
print("{:7} {:30}{:>11}".format(

"Código", "Nome do Curso", "Val./Mês")) print("- " * 25)
for d in dados:

print("{:<7} {:30} {:>10.2f}".format(d[0], d[1], d[2]))

print("- " * 49)
print("Encontrados {} registros\n".format(len(dados))) def ExcluiCurso(Codigo):
"""Verifica se o curso existe e o exclui.""" sql = """select Count(codcurso) from cursos where
codcurso = :param""" cursor.execute(sql, {'param' : Codigo})
x = cursor.fetchone() print(x[0])
if x[0] == 0:

return "Curso {} não Existe".format(Codigo) else:
sql = "delete from cursos where codcurso = :param" cursor.execute(sql, {'param' : Codigo})
conector.commit()

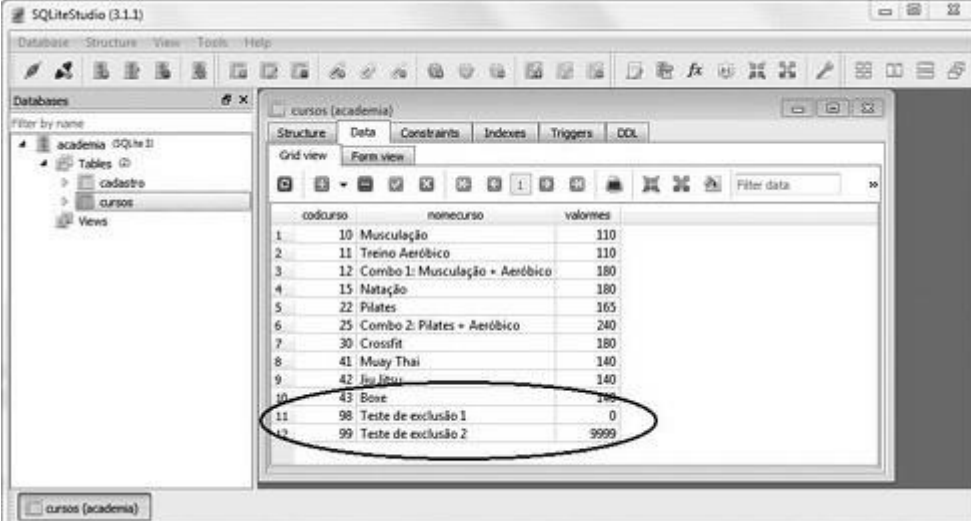
return "Curso {} Excluído".format(Codigo) # O programa começa a executar por aqui
conector = sqlite3.connect("academia.db")
cursor = conector.cursor() while True:
ExibeCursos()

print("Para excluir um curso digite o Código") print("Para sair do programa digite 0 (zero)")
Opc = int(input("sua escolha >> "))
if Opc == 0:

break else:
Msg = ExcluiCurso(Opc) print(Msg)
dummy = input("Pressione Enter para prosseguir...") cursor.close()
conector.close() print("\n\nFim do programa")
```



Os cursos 98 e 99 mostrados na Figura 8.15 serão utilizados como exemplo para exclusão. Para testar esse programa, esses dois registros adicionais foram incluídos usando o SQLite Studio, e a Figura 8.16 mostra que esses dois cursos aparecem na tela de execução do Python. Isso mostra que não importa qual software seja utilizado para manipular os dados. Por estarem em um local de armazenamento centralizado (o arquivo “academia.db”), todos os programas que acessarem o banco de dados terão acesso aos registros das tabelas.



	codcurso	nomecurso	valormes
1	10	Musculação	110
2	11	Treino Aeróbico	110
3	12	Combo 1: Musculação + Aeróbico	180
4	15	Natação	180
5	22	Pilates	165
6	25	Combo 2: Pilates + Aeróbico	240
7	30	Crossfit	180
8	41	Muay Thai	140
9	42	Jiu Jitsu	140
10	43	Boxe	140
11	98	Teste de exclusão 1	0
12	99	Teste de exclusão 2	9999

Tabela “cursos” contendo mais dois cursos inseridos diretamente no SQLite Studio.





```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help

Consulta ao Banco de Dados 'academia.db'

Dados da tabela 'cursos'
-----
Código Nome do Curso Val./Mês
-----
10 Musculação 110.00
11 Treino Aeróbico 110.00
12 Combo 1: Musculação + Aeróbico 180.00
15 Nataação 180.00
22 Pilates 165.00
25 Combo 2: Pilates + Aeróbico 240.00
30 Crossfit 180.00
41 Muay Thai 140.00
42 Jiu Jitsu 140.00
43 Boxe 140.00
98 Teste de exclusão 1 0.00
99 Teste de exclusão 2 9999.00
-----

Encontrados 12 registros

Para excluir um curso digite o Código
Para sair do programa digite 0 (zero)
sua escolha >> |
```

Os dados digitados no SQLite Studio estão disponíveis para o Python.

Ao digitar 98 no programa, o registro correspondente acaba sendo excluído.

```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help

Consulta ao Banco de Dados 'academia.db'

Dados da tabela 'cursos'
-----
Código Nome do Curso Val./Mês
-----
10 Musculação 110.00
11 Treino Aeróbico 110.00
12 Combo 1: Musculação + Aeróbico 180.00
15 Nataação 180.00
22 Pilates 165.00
25 Combo 2: Pilates + Aeróbico 240.00
30 Crossfit 180.00
41 Muay Thai 140.00
42 Jiu Jitsu 140.00
43 Boxe 140.00
99 Teste de exclusão 2 9999.00
-----

Encontrados 11 registros

Para excluir um curso digite o Código
Para sair do programa digite 0 (zero)
sua escolha >> |
```



A tabela “cursos” após a exclusão do registro 98.

Por fim, nesse exemplo foi usada uma forma diferente de passagem de parâmetro para o comando SQL, a passagem com parâmetro nomeado. Nos dois SQLs utilizados dentro da função ExcluiCurso e destacados a seguir, aparece a construção `codcurso = :param`, em que `:param` é um parâmetro nomeado e a palavra usada poderia ser qualquer uma no lugar do “param”. Veja em seguida como seria a alternativa no caso de ser utilizado um parâmetro posicional.

# foi usado assim – parâmetro nomeado

`sql = “select Count(codcurso) from cursos where codcurso = :param”` # mas poderia ter sido assim – parâmetro posicional

`sql = “select Count(codcurso) from cursos where codcurso = ?”`

# foi usado assim – parâmetro nomeado

`sql = “delete from cursos where codcurso = :param”` # mas poderia ter sido assim – parâmetro posicional  
`sql = “delete from cursos where codcurso = ?”`  
`cursor.execute(sql, {“param” :Codigo})`

Quando é utilizado o parâmetro nomeado, os dados devem ser fornecidos por meio de um dicionário do Python. No caso desse exemplo, o dicionário foi construído da seguinte maneira: `{“param” :Codigo}`, em que a chave “param” deve coincidir com o parâmetro nomeado existente no comando SQL, e o valor associado a essa chave será usado em sua execução.

