SQLite - Python

Neste capítulo, você aprenderá como usar o SQLite em programas Python.

Instalação

O SQLite3 pode ser integrado ao Python usando o módulo sqlite3, que foi escrito por Gerhard Haring. Ele fornece uma interface SQL compatível com a especificação DB-API 2.0 descrita pelo PEP 249. Você não precisa instalar este módulo separadamente porque ele é enviado por padrão junto com a versão 2.5.x do Python em diante.

Para usar o módulo sqlite3, você deve primeiro criar um objeto de conexão que represente o banco de dados e, opcionalmente, criar um objeto de cursor, que o ajudará na execução de todas as instruções SQL.

APIs do módulo sqlite3 do Python

A seguir, são apresentadas importantes rotinas do módulo sqlite3, que podem ser suficientes para você trabalhar com o banco de dados SQLite do seu programa Python. Se você está procurando uma aplicação mais sofisticada, pode procurar na documentação oficial do módulo Python sqlite3.

Sr. Não.	API e descrição
1	**sqlite3.connect (banco de dados [, tempo limite, outros argumentos opcionais])** Essa API abre uma conexão com o arquivo de banco de dados SQLite. Você pode usar ": memory:" para abrir uma conexão com um banco de dados que reside na RAM e não no disco. Se o banco de dados for aberto com sucesso, ele retornará um objeto de conexão. Quando um banco de dados é acessado por várias conexões e um dos processos modifica o banco de dados, o banco de dados SQLite fica bloqueado até que a transação seja confirmada. O parâmetro timeout especifica por quanto tempo a conexão deve esperar o bloqueio desaparecer até gerar uma exceção. O padrão para o parâmetro timeout é 5.0 (cinco segundos). Se o nome do banco de dados fornecido não existir, essa chamada criará o banco de dados. Você também pode especificar o nome do arquivo com o caminho necessário se desejar criar um banco de dados em qualquer outro lugar, exceto no diretório atual.
2	**connection.cursor ([cursorClass])** Essa rotina cria um **cursor** que será usado em toda a programação do banco de dados com o Python. Este método aceita um único parâmetro opcional cursorClass. Se fornecida, deve ser uma classe de cursor personalizada que estenda o sqlite3.Cursor.
3	**cursor.execute (sql [, parâmetros opcionais])** Essa rotina executa uma instrução SQL. A instrução SQL pode ser parametrizada (ou seja, espaços reservados em vez de literais SQL). O módulo sqlite3 suporta dois tipos de espaços reservados: pontos de interrogação e espaços reservados nomeados (estilo nomeado). **Por exemplo** - cursor.execute ("insira valores de pessoas (?,?)", (Quem, idade))
4	**connection.execute (sql [, parâmetros opcionais])** Essa rotina é um atalho do método de execução acima fornecido pelo objeto cursor e cria um objeto de cursor intermediário chamando o método cursor, depois chama o método execute do cursor com os parâmetros fornecidos.
5	**cursor.executemany (sql, seq_of_parameters)** Essa rotina executa um comando SQL em todas as sequências de parâmetros ou mapeamentos encontrados na sequência sql.
6	

	connection.executemany (sql [, parâmetros]) Essa rotina é um atalho que cria um objeto cursor intermediário chamando o método cursor e, em seguida, chama o método cursor.s executemany com os parâmetros fornecidos.
7	**cursor.executescript (sql_script)** Essa rotina executa várias instruções SQL de uma vez fornecidas na forma de script. Ele emite uma instrução COMMIT primeiro e depois executa o script SQL que obtém como parâmetro. Todas as instruções SQL devem ser separadas por ponto e vírgula (;).
8	**connection.executescript (sql_script)** Essa rotina é um atalho que cria um objeto de cursor intermediário chamando o método cursor e, em seguida, chama o método executescript do cursor com os parâmetros fornecidos.
9	**connection.total_changes ()** Essa rotina retorna o número total de linhas do banco de dados que foram modificadas, inseridas ou excluídas desde que a conexão com o banco de dados foi aberta.
10	**connection.commit ()** Este método confirma a transação atual. Se você não chamar esse método, tudo o que você fez desde a última chamada para commit () não será visível em outras conexões com o banco de dados.
11	**connection.rollback ()** Este método reverte quaisquer alterações no banco de dados desde a última chamada para commit ().
12	**connection.close ()** Este método fecha a conexão com o banco de dados. Observe que isso não chama automaticamente commit (). Se você apenas fechar sua conexão com o banco de dados sem chamar commit () primeiro, suas alterações serão perdidas!
13	**cursor.fetchone ()** Esse método busca a próxima linha de um conjunto de resultados da consulta, retornando uma única sequência ou Nenhuma quando não houver mais dados disponíveis.

14	**cursor.fetchmany ([tamanho = cursor.arraysize])** Essa rotina busca o próximo conjunto de linhas de um resultado da consulta, retornando uma lista. Uma lista vazia é retornada quando não há mais linhas disponíveis. O método tenta buscar quantas linhas forem indicadas pelo parâmetro size.
15	**cursor.fetchall ()** Essa rotina busca todas as linhas (restantes) de um resultado da consulta, retornando uma lista. Uma lista vazia é retornada quando nenhuma linha está disponível.

Conectar ao banco de dados

O código Python a seguir mostra como se conectar a um banco de dados existente. Se o banco de dados não existir, ele será criado e, finalmente, um objeto de banco de dados será retornado.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print "Opened database successfully";
```

Aqui, você também pode fornecer o nome do banco de dados como o nome especial **:memory:** para criar um banco de dados na RAM. Agora, vamos executar o programa acima para criar nosso banco de dados **test.db** no diretório atual. Você pode mudar seu caminho conforme sua exigência. Mantenha o código acima no arquivo sqlite.py e execute-o como mostrado abaixo. Se o banco de dados for criado com sucesso, ele exibirá a seguinte mensagem.

```
$chmod +x sqlite.py
$./sqlite.py
Open database successfully
```

Criar uma tabela

O programa Python a seguir será usado para criar uma tabela no banco de dados criado anteriormente.

```
#!/usr/bin/python

import sqlite3
```

```
conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute('''CREATE TABLE COMPANY
         (ID INT PRIMARY KEY     NOT NULL,
         NAME           TEXT    NOT NULL,
         AGE            INT     NOT NULL,
         ADDRESS        CHAR(50),
         SALARY         REAL);''')
print "Table created successfully";

conn.close()
```

Quando o programa acima for executado, ele criará a tabela COMPANY no seu **test.db** e exibirá as seguintes mensagens -

```
Opened database successfully
Table created successfully
```

INSERIR Operação

O programa Python a seguir mostra como criar registros na tabela COMPANY criada no exemplo acima.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print "Records created successfully";
conn.close()
```

Quando o programa acima é executado, ele cria os registros fornecidos na tabela COMPANY e exibe as duas linhas a seguir -

```
Opened database successfully
Records created successfully
```

Operação SELECT

O programa Python a seguir mostra como buscar e exibir registros da tabela COMPANY criada no exemplo acima.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

Quando o programa acima é executado, ele produz o seguinte resultado.

```
Opened database successfully
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
```

```
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Operação UPDATE

O código Python a seguir mostra como usar a instrução UPDATE para atualizar qualquer registro e, em seguida, buscar e exibir os registros atualizados da tabela COMPANY.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit()
print "Total number of rows updated :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

Quando o programa acima é executado, ele produz o seguinte resultado.

```
Opened database successfully
Total number of rows updated : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 25000.0

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
```

```
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

Operação DELETE

O código Python a seguir mostra como usar a instrução DELETE para excluir qualquer registro e, em seguida, buscar e exibir os registros restantes da tabela COMPANY.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()
print "Total number of rows deleted :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

Quando o programa acima é executado, ele produz o seguinte resultado.

```
Opened database successfully
Total number of rows deleted : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0
```

```
ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```