Conectividade do MySQL com Python

Tradutor: Douglas Soares de Andrade – dsa em unilestemg ponto br
Site Original: http://www.devshed.com/c/a/Python/MySQL-Connectivity-With-Python/

Uma das coisas mais legais sobre Python é que, como uma linguagem orientada a objeto, ela obriga o desenvolvedor a pensar em termos de abstração e modularização de código enquanto está escrevendo código, o que resulta em aplicações limpas e melhor empacotadas. Entretanto, empacotar seu código em funções é simplemente o começo; Python também te permite criar coleções de fragmentos de código compartilhado, ou "módulos", que podem ser importados e usados por qualquer programa Python. Poderosos, flexíveis e muito interessantes, os módulos podem ser uma das mais potentes armas no arsenal do desenvolvedor Python... desde que eles sejam usados corretamente.

No curso deste artigo, eu estarei explorando um dos módulos mais úteis de Python, o módulo MySQLdb. O módulo MySQLdb permite aos desenvolvedores conectar seu código Python a um servidor de banco de dados MySQL, executar comandos SQL nele, e transformar os dados resultantes em um formato utilizável...e isso vem a calhar quando você começar a desenvolver aplicações Python com banco de dados. Mas não acredite nas minhas palavras - veja por você mesmo!

Nas palavras do seu autor, MySQLdb é "uma interface compatível com threads para o popular servidor de banco de dados MySQL que fornece a API de banco de dados Python." Desenvolvido por Andy Dustman, este módulo é necessário para a conectividade com MySQL através do Python.

A primeira coisa a fazer é ter certeza que você tem o módulo MySQLdb instalado em seu sistema de desenvolvimento Python. A maneira mais fácil para checar isso é através do interpletador interativo de linha de comando Python – como o exemplo seguinte demonstra:

```
Python 2.3.3 (#1, May 12 2004, 22:56:21)
[GCC 3.3.3] on linux2
Type "copyright", "credits" or "license()" for more information.

>>> import MySQLdb
Traceback (innermost last):
  File "<stdin>", line 1, in ?
ImportError: No module named MySQLdb
>>>
```

Se você está vendo alguma coisa parecida com a mensagem de erro acima, você pode assumir seguramente que o módulo não está instalado no seu sistema. Vá até http://sourceforge.net/projects/mysql-python, baixe a última cópia do módulo MySQLdb, extraia e instale no seu sistema.

```
$ tar -xzvf MySQL-python-0.9.2.tar.gz

$ cd MySQL-python-0.9.2

$ python setup.py build

$ python setup.py install
```

Agora, você pode verificar se o módulo foi instalado tentando importá-lo novamente pela linha de comando.

```
Python 2.3.3 (#1, May 12 2004, 22:56:21)
[GCC 3.3.3] on linux2
Type "copyright", "credits" or "license()" for more information.

>>> import MySQLdb
>>>
```

Nenhum problema desta vez – o que significa que nós podemos continuar para escrever um pouco de código.

Com isso fora do caminho, aqui está um simples exemplo que demonstra um pouco da funcionalidade do DBI. Considere a tabela seguinte do banco de dados,

```
mysql> SELECT * FROM animals;
+---------+----------+
| name    | species  |
+---------+----------+
| Wallace | Walrus   |
| Polly   | Parrot   |
| Freddie | Frog     |
| Tusker  | Elephant |
| Sammy   | Skunk    |
+---------+----------+
5 rows in set (0.01 sec)
```

e então considere este pequeno script Python, que conecta ao banco de dados e imprime os dados da tabela.

```
#!/usr/bin/python

# Importa o módulo do MySQL

import MySQLdb

# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")
```

```
# Cria um cursor

cursor = db.cursor()

# Executa o SQL
cursor.execute("SELECT * FROM animals")


# Pega o resultset como uma tupla

result = cursor.fetchall()

# Itera (navega) pelo resultset
for record in result:
        print record[0] , "-->", record[1]
```

A maioria deste código é fácil de entender (self-explanatory), mas de qualquer forma, deixe-me ir através dele com você.

O primeiro passo é importar o módulo MySQLdb, através da função "import" do Python.

```
# Importa o módulo do MySQL
import MySQLdb
```

Assim que acabar, você pode abrir uma conexão com o MySQL passando para o método connect() do módulo uma série de parâmetros de conexão – o nome do servidor, o nome do usuário do banco de dados, a senha e o nome do banco de dados.

```
# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")
```

Uma conexão efetuada com sucesso retorna um objeto Connection, que você pode usar para criar um cursor.

```
# Cria um cursor
cursor = db.cursor()
```

Este cursor é necessário para executar uma consulta SQL e para recuperar o resultset gerado.

```
# Executa o SQL
cursor.execute("SELECT * FROM animals")
# pega o resultset como uma tupla
result = cursor.fetchall()
```

Alguns métodos estão disponíveis para recuperar o resultset da consulta – o método utilizado aqui é o fetchall(), que retorna uma tupla de tuplas, cada tupla interna representa uma linha do resultset. Esta tupla pode então ser iterada usando um simples loop "for", e os seus elementos impressos para a saída padrão.

```
# Itera (navega) pelo resultset
for record in result:
        print record[0] , "-->", record[1]
```

Você também pode usar o método fetchone() do objeto cursor para pegar e listar uma linha de cada vez, ao invés de tudo de uma vez. Considere o seguinte exemplo, que demonstra como isso deve funcionar:

```
#!/usr/bin/python

# Importa o módulo do MySQL
import MySQLdb

# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")

# Cria um cursor

cursor = db.cursor()

# Executa o SQL

cursor.execute("SELECT * FROM animals")

# Pega o número de linhas no resultset

numrows = int(cursor.rowcount)

# Lista uma linha de cada vez
for x in range(0,numrows):
        row = cursor.fetchone()
        print row[0], "-->", row[1]
```

Neste caso, o método rowcount() do objeto cursor é primeiramente usado para encontrar o número de linhas no resultset. Este número é então usado em um loop "for" para iterar o resultset, com o método fetchone() usado para mover através do resultset e sequencialmente listar o conteúdo de cada linha.

Você pode fazer com que o MySQLdb lhe de um subset específico de um resultset com o método fetchmany() do objeto cursor, que lhe permite especificar o tamanho do resultset retornado. Considere o seguinte exemplo, que demonstra:

```
#!/usr/bin/python

# Importa o módulo do MySQL
import MySQLdb


# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")
```

```
# Cria um cursor
cursor = db.cursor()

# Executa o SQL
cursor.execute("SELECT * FROM animals")

# Limita o resultset para 3 itens
result = cursor.fetchmany(3)

# Itera (navega) pelo resultset
for record in result:
        print record[0] , "-->", record[1]
```

Obviamente, você também pode executar consultas de inserção, atualização e exclusão através do módulo MySQLdb. Considere o exemplo seguinte:

```
#!/usr/bin/python

# Importa o módulo do MySQL

import MySQLdb

# conecta

db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")

# Cria um cursor

cursor = db.cursor()

# Executa o SQL

cursor.execute("""INSERT INTO animals (name, species) VALUES ("Harry",
               "Hamster")""")
```

Você pode modificar isto para que os valores da string de consulta sejam inseridos pelo usuário – dê uma olhada nesta variante do exemplo acima:

```
#!/usr/bin/python


# Importa o módulo do MySQL
import MySQLdb

# Pega a entrada do usuário
name = raw_input("Please enter a name: ")
species = raw_input("Please enter a species: ")

# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")
```

```
# Cria um cursor
cursor = db.cursor()

# Executa o SQL
cursor.execute("INSERT INTO animals (name, species) VALUES (%s, %s)",
               (name, species))
```

Desta vez, quando você executar o script, você será perguntado pelos valores para serem inseridos no banco de dados.

```
Please enter a name: Rollo
Please enter a species: Rat
```

Note a maneira com a qual as variáveis foram integradas na consulta SQL do exemplo acima. O símbolo "%s" é usado para representar cada variável na string de consulta, com os valores atuais guardados em uma tupla e passados como um segundo argumento.

No caso de você ter campos auto-incremento no seu banco de dados, você pode usar o método insert_id() do objeto cursor para obter o ID do último registro inserido – isso vem a calhar quando você está lidando com tabelas ligadas em um banco de dados relacional (RDBMS), pois os IDs recém inseridos de uma tabela frequentemente servem como chaves para outras tabelas. O seguinte snippet (script Python - hehehe) deve demonstrar como isso funciona:

```
#!/usr/bin/python

# Importa o módulo do MySQL

import MySQLdb

# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")

# Cria um cursor
cursor = db.cursor()

# Executa o SQL
cursor.execute("""INSERT INTO test (field1, field2) VALUES ("val1",
               "val2")""")

# Pega o ID do último registro inserido
print "ID of inserted record is ", int(cursor.insert_id())
```

Muitos scripts de banco de dados envolvem preparar uma única consulta (de INSERÇÃO, por exemplo) e executá-la várias vezes com valores diferentes. MySQLdb vem com o método executemany(), que simplifica este trabalho e também pode reduzir despesas de performance.

Para entender como isso funciona, considere o seguinte exemplo, que demonstra:

```
#!/usr/bin/python

# Importa o módulo do MySQL
import MySQLdb

# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")

# Cria um cursor
cursor = db.cursor()

# Gera consultas SQL dinâmicas a partir da lista
cursor.executemany("INSERT INTO animals (name, species) VALUES (%s,%s)",
[  ('Rollo', 'Rat'),  ('Dudley', 'Dolphin'),  ('Mark', 'Marmoset')
])
```

Neste caso, a mesma consulta é repedita várias vezes, com um grupo diferente de valores de cada vez. Os valores de cada iteração são passados para o método executemany() como uma lista Python; cada elemento da lista é uma tupla contendo os valores para cada iteração.

Usando esta técnica, é possível escrever um script que pede para o usuário para entrar uma série de valores de dados, e então inserir todos eles no banco de dados de um golpe só usando o método executemany(). Isso é feito aqui em baixo:

```
#!/usr/bin/python

# Importa o módulo do MySQL
import MySQLdb

# Inicializa algumas variáveis
name = ""
data = []

# Entra em um loop e pede a entrada do usuário
while (1):
        name = raw_input("Please enter a name (EOF to end): ")

        if name == "EOF":
                break
        species = raw_input("Please enter a species: ")


        # Colocar a entrada do usuário em uma tupla
        tuple = (name, species)


        # E insere na lista de dados
```

```
        data.append(tuple)


# Conecta
db = MySQLdb.connect(host="localhost", user="joe", passwd="secret",
     db="db56a")

# Cria um cursor
cursor = db.cursor()

# Gera consultas SQL dinÂmicas a partir da lista de dados
cursor.executemany("INSERT INTO animals (name, species) VALUES (%s,%s)",
                   data)
```

Neste caso, um laço "while" é usado continuamente para pegar a entrada do usuário, com cada grupo de valores inseridos sendo empacotados em uma tupla e sendo adicionados para a lista de dados. Depois que o usuário terminar de inserir todos os dados, o método executemany() é usado na combinação com os vários valores inseridos pelo usuário, para INSERIR os valores dentro do banco de dados.

Alguns outros métodos que vem com a classe MySQLdb – aqui está uma pequena lista dos métodos mais úteis:

connection.begin() - Inicia uma conexão

connection.apilevel() - Retorna o nível da DB API corrente

connection.conv() - Configure as opções de conversão de tipos MySQL e Python

connection.commit() - Dá um commit na transação

connection.rollback() - Dá um roll back na transação

E isso é tudo por agora. Neste artigo, eu lhe mostrei como configurar e instalar o módulo MySQLdb do Python, e utilizá-lo para ligar seus scripts Python com um servidor de banco de dados MySQL. Eu demonstrei as diferentes técnicas para iterar em um resultset, mostrei o básico de como usar variáveis e consultas preparadas, e ilustrei alguns dos métodos contidos neste módulo.

Mesmo que este tutorial consiga fazer com que você faça scripts com Python e MySQL, você não deveria parar de ler agora. Aqui estão alguns links que merecem ser checados:

A página do projeto MySQLdb no SourceForge, em http://sourceforge.net/projects/mysql-python

A página do Python, em http://www.python.org/

A página do MySQL, em http://www.mysql.com/

Até a próxima vez...seja bonzinho!

Nota: Todos os exemplo deste artigo foram testados em um Linux/i686 com Python 2.3.3, MySQL 4.1.1a e MySQLdb 0.9.2. Examplos são meramente ilustrativos, e não foram projetados para um ambiente de produção. Melonfire não dá nenhuma garantia ou suporte ao fonte distribuído neste artigo. YMMV!