Python
Acessando o Banco de Dados MySQL

ANTONIO SÉRGIO NOGUEIRA

PRESIDENTE PRUDENTE – SP
2009

Sumário

Python

1. Introdução: O Python permite a você acessar os bancos de dados mais comuns do mercado, para isto temos um padrão de interface que é DB-API, a maioria das interfaces de banco de dados adere a este padrão.

Podemos escolher o Banco de Dados que mais nos agrada já que a Python Database API possui uma grande quantidade de servidores de banco de dados são eles:

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

Você deve carregar o módulo DB API de cada banco de dado que você necessita acessar, no nosso caso vamos a carregar o módulo do Database Mysql. Este Módulo nos permitirá acessar o banco de dados através de estruturas Python. Neste API encontramos comandos para abrir uma conexão, emitir comandos SQL, acessar procedimentos armazenados e fechar conexão.

2.Interface MySQLdb – é a interface API para conexão com o servidor de banco de dados MySQL no Python, ela é construída em cima do MySQL API C e implementa a versão Python Database API v2.0. O programa MySQL (R) é um servidor robusto de bancos de dados SQL (Structured Query Language – Linguagem Estruturada para Pesquisas) muito rápido, multi-tarefa e multi-usuário. O Servidor MySQL pode ser usado em sistemas de produção com alta carga e missão crítica bem como pode ser embutido em programa de uso em massa. MySQL é uma marca registrada da MySQL AB. O programa MySQL é de Licença Dupla. Os usuários podem escolher entre usar o programa MySQL como um produto Open Source/Free Software sob os termos da GNU General Public License (http://www.fsf.org/licenses/) ou podem comprar uma licença comercial padrão da MySQL AB. O site web do MySQL (http://www.mysql.com/) dispõe das últimas informações sobre o programa MySQL.

3. Instalando o MySQLdb – para instalar o módulo adote o seguinte procedimento:

- Instale o MySql (Servidor de Banco de Dados), você tem que ter privilégio de administrador.
- Vá até o seguinte link: http://sourceforge.net/projects/mysql-python
- Escolha a opção download.
- Escolha o pacote mysql-python.
- Agora escolha o pacote MySQL-python-1.2.2.win32-py2.5.exe. (Vamos usar o Python 2.5

- Pronto agora é só fazer o download.
- Feito o download clique em cima do arquivo é instale-o no diretório do Python 2.5.

4. Verificando se MySQLdb está instalado – Abra o IDLE e no shell digite o seguinte comando:

>>>import MySQLdb

>>>

Se houver erro o shell emite uma informação, senão aparece o cursor >>>.

5. Vamos nos conectar ao Banco de Dados – antes de fazermos a conexão vamos executar alguns comandos no mysql para criarmos:

- Base de dados
- A tabela de dados
- Para entender mais sobre banco de dados veja: Manual de Referência do MySQL : http://downloads.mysql.com/docs/refman-4.1-pt.pdf

6. Criando o banco de dados e as tabelas no MySQL: mostraremos passo a passo a criação de um banco de dados no MySQL.

- **Acesse o MySQL:** acesse o **MySQL Command Line Client**
- **Forneça a Senha:** aparecerá **Enter Password:** SENHA (você vai acessá-lo como root – administrador para não ter restrições de acesso).
- **Criando um banco de dados:** No MySQL os bancos de dados são implementados como diretórios contendo arquivos que correspondem a tabelas do banco de dados. Como ainda não existem tabelas no banco de dados criado, este comando somente cria um subdiretório no diretório de dados do MySQL. Ocorrerá um erro se o banco de dados já existir ou se você não tiver o privilégio apropriado.
  CREATE DATABASE testedb;
- Acessando o Banco de Dados: USE Identifica o banco de dados default, o qual será utilizado quando as referências a uma tabela não especificarem explicitamente o nome do banco de dados.
  USE testedb;
- **Criando as tabelas de dados do banco:** Criaremos a tabela Empregado com os campos Nome, Idade, Sexo e Salário.

  CREATE TABLE empregado (nome varchar(30) not null primary key, idade int, sexo char(1), salario int);
- **Verifique o que criou:** digite os comandos para verificar as tabelas e as colunas: **SHOW TABLES;** (tabelas do banco de dados)

  SHOW COLUMNS FROM empregado; (colunas da tabela)

7. Conectando-se ao banco de dados com o Python: Vamos primeiro nos conectar ao banco de dados para executarmos os comandos.

Exemplo:

```
# file: conectardb.py
#!\python25\python
import MySQLdb
# Abre o banco de dados
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# prepara um objeto cursor usando método cursor()
cursor = db.cursor()
# executa SQL usando método execute()
cursor.execute("SELECT VERSION()")
# Busca uma linha usando método fetchone()
data = cursor.fetchone()
print "Database version : %s " % data
# desconectando
db.close()
```

Você verá a seguinte mensagem:

```
Database version : 5.1.32-community
```

Ao estabelecer a conexão com a fonte de dados um objeto de Conexão é retornado e salvo em **db** ou senão ele é setado para **None.** O objeto **db** é usado para criar um objeto tipo **cursor** o qual é usado para executar queries. Finalmente encerramos a conexão e liberamos os recursos.

8. Criando as tabelas do banco de dados: Uma vez que estabelecemos a conexão, estamos preparados para criar e gravar tabelas através do método **execute** do **cursor** criado.

Exemplo:

```
# file:Criatabela.py
#!\python25\python
import MySQLdb
```

```
# Abre conexão
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# prepara o objeto cursor
cursor = db.cursor()
# Elimina tabela se ela existir usando o comando SQL através do método execute.
method.
cursor.execute("DROP TABLE IF EXISTS EMPREGADO")
# Cria a tabela requerida
sql = """CREATE TABLE EMPREGADO (
        NOME  CHAR(30) NOT NULL,
        IDADE INT,
        SEXO CHAR(1),
        SALARIO FLOAT )"""
cursor.execute(sql)
# desconectando
db.close()
```

Esta mensagem aparece se a tabela não existir.

```
Warning (from warnings module):
  File "C:/Python25/criatabela.py", line 13
    cursor.execute("DROP TABLE IF EXISTS EMPREGADOS")
Warning: Unknown table 'empregados'
```

9. Inserindo dados nas tabelas: uma vez criada a tabela podemos agora inserir dados nela através do comando SQL **INSERT**, que criará um registro na tabela empregado.

Exemplo:

```
#file:inseredado.py
#!\python25\python
import MySQLdb
# ABRE CONEXÃO
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# prepara metodo cursor
cursor = db.cursor()
```

```
# Prepara comando SQL
sql = """INSERT INTO EMPREGADO(NOME,
        IDADE, SALARIO, SEXO)
        VALUES ('Sergio', 22, 2000,'M')"""
try:
   # Executa  comando SQL
    cursor.execute(sql)
   # Faz as mudanças no database
    db.commit()
except:
#Desfaz mudanças na tabela em caso de erro
    db.rollback()
# desconecta
db.close()
```

Outro exemplo usando agora comando SQL **Insert** dinâmico.

```
#file: inseredadod.py
#!\python25\python
import MySQLdb
# Abre conexão com o banco de dados
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# prepare o objeto curso com o método cursor.
cursor = db.cursor()
# Prepare comando SQL para inserir registro no BD.
sql = "INSERT INTO EMPREGADO(NOME, \
       IDADE, SEXO, SALARIO) \
       VALUES ('%s', '%d', '%c', '%d' )" % \
       ('MARCOS', 30, 'M', 1000)
try:
   # Execute comando SQL
   cursor.execute(sql)
   # EfetuarPerpetar as mudanças no database
   db.commit()
```

```
except:
   # desfazer em caso de erro
   db.rollback()
# desconecta
db.close()
```

10. Executando passagem de parâmetro diretamente – o exemplo a seguir é uma outra forma de executar o programa e passar parâmetros diretamente.

Exemplo:

```
#file: inseredadod1.py
#!\python25\python
import MySQLdb

# Abre conexão com o banco de dados
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# prepare o objeto curso com o método cursor.
cursor = db.cursor()
nome='Antonio'
idade=34
sexo='M'
salario=1350
try:
# executa sql
    cursor.execute('insert into empregado values("%s","%d","%c","%d")'%
                (nome,idade,sexo,salario))
# Efetuar as mudanças no database
    db.commit()
except:
   # desfazer em caso de erro
   db.rollback()
# desconecta
db.close()
```

11. Operação de Leitua (READ): ler um banco de dados significa buscar alguma informação útil no banco de dados. Após a conexão ser estabelecida, nós estamos prontos para pesquisar no banco de dados. Nós podemos procurar um único registro através do método fetchone() ou múltiplos registros através do método fetchall().

fetchone() - este método encontra a próxima linha do conjunto de uma busca. Um objeto é retornado quando o objeto cursor é usado na busca em uma tabela.

fetchall() - este método busca todas as linha de um conjunto de uma busca. Se algumas linhas já foram extraídas, o método retorna as linhas restantes.

rowcount() - método de apenas leitura e retorna o número de linhas afetadas pelo método execute().

Exemplo: este exemplo retorna todos os registros da tabela **empregado** que tenham salário maior que 1000.

```
# file:ledadosdb.py
#!\python25\python
import MySQLdb
# abre conexão
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# objeto cursor
cursor = db.cursor()
#  SQL de busca de dados.
sql = "SELECT * FROM EMPREGADO \
      WHERE SALARIO > '%d'" % (1000)
try:
  # Executa SQL
  cursor.execute(sql)
  # Busca todas as linhas de uma tabela
  results = cursor.fetchall()
  for row in results:
    fname = row[0]
    age = row[1]
    sex = row[2]
    income = row[3]
    # Imprime resultado
    print "Nome=%s,Idade=%d,Sexo=%s,Salario=%d" % \
```

```
        (fname, age, sex, income )
except:
   print "Erro na busca."
# desconecta
db.close()
```

Resultado da pesquisa na tabela cadastrada:

```
Nome=Sergio,Idade=22,Sexo=M,Salario=2000
Nome=Antonio,Idade=34,Sexo=M,Salario=1350
```

12. Operação de atualização ou alteração de dados: a operação UPDATE sobre um banco de dado significa atualizar ou modificar um ou mais registros já disponíveis no banco de dados. O procedimento seguinte atualizará todos os registros que tenham sexo 'M'. Nós iremos incrementar a idade de todos as pessoas de sexo M.

Exemplo:

```
# file:ledadosdb.py
#!\python25\python
import MySQLdb
# abre conexão
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# objeto cursor
cursor = db.cursor()
#  SQL de busca de dados.
sql = "SELECT * FROM EMPREGADO \
      WHERE SALARIO > '%d'" % (1000)
try:
   # Executa SQL
   cursor.execute(sql)
   # Busca todas as linhas de uma tabela
   results = cursor.fetchall()
   for row in results:
      fname = row[0]
```

```
      age = row[1]
      sex = row[2]
      income = row[3]
      # Imprime resultado
      print "Nome=%s,Idade=%d,Sexo=%s,Salario=%d" % \
            (fname, age, sex, income )
except:
   print "Erro na busca."

# desconecta
db.close()
```

Resultado:

Nome=Sergio,Idade=23,Sexo=M,Salario=2000

Nome=Antonio,Idade=35,Sexo=M,Salario=1350

13. Operação de exclusão de registros: quando queremos apagar alguns registros usamos o operação **DELETE**. O programa abaixo apaga todos os registros de empregado onde a idade é menor do que 25.

Exemplo:

```
# file: apagadb.py
#!\python25\python
import MySQLdb
# abre conexão
db = MySQLdb.connect("localhost","root","senha","TESTEDB" )
# prepare o objeto cursor
cursor = db.cursor()
# Prepare o comando SQL
sql = "DELETE FROM EMPREGADO WHERE IDADE < '%d'" % (25)
try:
   # Executa comando SQL
   cursor.execute(sql)
```

```
    # Efetua modificação do banco
    db.commit()
except:
    # Desfaz a modificação
    db.rollback()
# desconecta
db.close()
```

Quando falamos em banco de dados, falamos em consistência dos dados e para que isso aconteça temos que ter as seguintes propriedades: atomicidade, consistência, isolação, durabilidade. Para que isto aconteça temos dois métodos no Python DB API 2.0 committ e rollback. Committ dá o sinal verde para efetuar a transação e após este comando nenhum erro pode atrapalhar esta modificação. Já o comando Rollback desfaz a transação se um erro ocorrer antes do committ. O comando de desconexão serve para encerrar a transação e desfazer as mudanças(committ) não efetuadas.

14. Manipulando Erros: algumas fontes de erro são falha de conexão como por exemplo chamar um método de busca para um comando cancelado, etc. Em seguida temos a tabela de erros que deve ser confirmada para cada banco utilizado.

Exceção	Descrição
Warning	Usado para emitir mensagens não fatais. Subclasse de StandardError.
Error	Classe Base para erros. Subclasse de StandardError.
InterfaceError	Usado para erros no módulo de banco de dados, não do database. Suclasse de Error.
DatabaseError	Usado em erro da base de dados. Subclasse de Error.
DataError	Subclass of DatabaseError that refers to errors in the data.
OperationalError	Subclasse de DatabaseError que refere-se aos erros como perda de conexão. Esses erros são normalmente externos ao controle do script Python.
IntegrityError	Subclasse de DatabaseError para situações que danificariam a integridade relacional.
InternalError	Subclasse de DatabaseError que refere-se aos erros internos do módulo de database, como cursor não ativo.
ProgrammingError	Subclasse de DatabaseError que refere-se a erros como nome errado da tabela ou coisas que podem seguramente ser reprovados.
NotSupportedError	Subclasse de DatabaseError que refere-se a tentativa de chamadas não suportadas funcionalmente.

Referência Bibliográfica:

Prates Rubens, Guia de Consulta Rápida MySQL, 2000 - Novatec Editora Ltda.

Python - MySQL Database Access, maio 2009,
http://www.tutorialspoint.com/python/python_database_access.htm

Manual de Referência do MySQL 4.1, 2009, MySQL AB

HOWTO Use Python in the web, http://www.python.org/doc/2.6.2/howto/webservers.html