Acceso a Bases de Datos

Antonio Sarasa Cabezuelo

MySQL

- Es una base de datos relacional que no está integrada en Python por lo que requiere instalarse un conector propio denominado mysql-connector

Conexión a la base de datos

- Una vez instalado el conector, se puede realizar la conexión desde Python. A diferencia de SQLite , en MySQL es necesario indicar algunos argumentos tales como: usuario, password, host, puerto, base de datos,…, al me´todo que genera la conexión. Este método devuelve un objeto de tipo conexión.

Conexión a la base de datos

- Existen varias formas de conectarse:
  - Usando el método connect de mysql.connector.

```
import mysql.connector

cnx = mysql.connector.connect(user='pepito', password='45666',
                              host='127.0.0.1',port=3307,
                              database='ejemplo')
cnx.close()
```

Conexión a la base de datos

- Cuando hay que pasar muchos argumentos a la conexión se pueden introducir en un diccionario y pasar el diccionario como argumento del método connect

```
import mysql.connector

config = {
  'user': 'pepito',
  'password': '45666',
  'host': '127.0.0.1',
  'port':3307,
  'database': 'ejemplo',
  'raise_on_warnings': True,
}

cnx = mysql.connector.connect(**config)

cnx.close()
```

Conexión a la base de datos

- Otra posibilidad consiste en usar la clase connection.MySQLConnection()

```
from mysql.connector import (connection)

cnx = connection.MySQLConnection(user='pepito', password='45666',
                                 host='127.0.0.1',port=3307,
                                 database='ejemplo')
cnx.close()
```

Conexión a la base de datos

- Para tratar los posibles errores que se producen en una conexión se puede utilizar una estructura try/except como en el ejemplo siguiente:

```
import mysql.connector
from mysql.connector import errorcode

try:
  cnx = mysql.connector.connect(user='pepito',
                                database='ejemplo')
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print("Existe un problema en tu usuario o password")
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print("La base de datos no existe")
  else:
    print(err)
else:
  cnx.close()
```

```
>>>
Existe un problema en tu usuario o password
>>>
```

Cursores

- Para ejecutar cualquier acción sobre el servidor de bases de datos es necesario crear un objeto de tipo cursor asociado a la conexión actual. Para ello se usa el método cursor() del objeto conexión.

```
import mysql.connector

cnx = mysql.connector.connect(user='pepito', password='45666',
                              host='127.0.0.1', port=3307)
cursor=cnx.cursor()
cnx.close()
```

Cursores

- Una vez que se dispone de un objeto cursor, se pueden ejecutar acciones sobre la base de datos usando el método execute del objeto cursor. Este método toma como argumento una sentencia SQL.

Creación de una base de datos

- En MySQL es necesario crear las bases de datos (no se crean automáticamente al crear una conexión como ocurría en SQLite). Para ello se pueden crear manualmente en MySQL o bien crearlas desde Python.

Creación de una base de datos

- En el siguiente ejemplo se define una función para crear una base de datos denominada “Biblioteca”:

```
import mysql.connector
from mysql.connector import errorcode

def create_database(cursor,DB_NAME):
    try:
        cursor.execute(
            "CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format(DB_NAME))
    except mysql.connector.Error as err:
        print("Failed creating database: {}".format(err))
        exit(1)

cnx = mysql.connector.connect(user='pepito', password='45666',
                              host='127.0.0.1', port=3307)
cursor=cnx.cursor()
create_database(cursor, 'Biblioteca')
cnx.database = 'Biblioteca'
cursor.close()
cnx.commit()
```

Creación de una base de datos

- En el ejemplo se define una función que toma como argumento un objeto cursor y el nombre de una base de datos que es utilizado para crear una base de datos usando una sentencia SQL.
- A continuación se llama a la función, y una vez creada se actualiza la conexión asociando como base de datos la creada.

Creación de tablas

- Se va a crear una tabla llamada Libros con una columna de texto llamada "Título" y otra columna de enteros llamada "prestamos". Además antes de crear la tabla la vamos a eliminar para asegurarse que no existe ya en la base de datos.

```
import mysql.connector
from mysql.connector import errorcode

cnx = mysql.connector.connect(user='pepito', password='45666',
                              host='127.0.0.1', port=3307, database='Biblioteca')
cursor=cnx.cursor()
cursor.execute('DROP TABLE IF EXISTS Libros')
cursor.execute("CREATE TABLE Libros (titulo varchar(14),ejemplares int(11))")
cursor.close()
cnx.commit()
```

Inserción de datos

- Una vez creada la tabla Libros se pueden guardar datos usando una llamada a execute() con el comando SQL INSERT.

```
from mysql.connector import MySQLConnection, Error
query = "INSERT INTO Libros(titulo,ejemplares) VALUES(%s,%s)"
args1 = ("El quijote", 21)
args2 = ("El escarabajo de oro", 15)
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query, args1)
    cursor.execute(query, args2)
    cursor.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

Inserción de datos

- Observar que para insertar se especifica con porcentajes (%s,%s) los valores para indicar que serán pasados como una tupla en el segundo parámetro de la llamada a execute().

Inserción de datos

- Cuando se quieren insertar múltiples filas en una sola operación se puede usar el método executemany y proporcionar como argumento la secuencia de filas que se quieren insertar como una lista. Como resultado se llama al método execute una vez por cada fila.

```
from mysql.connector import MySQLConnection, Error
argu=[("Hamlet", 31),("Dracula", 17)]
query = "INSERT INTO Libros(titulo,ejemplares) VALUES(%s,%s)"
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.executemany(query, argu)
    cursor.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

Inserción de datos

- De forma similar se podrían haber insertado las filas usando un bucle for que actuará sobre la lista y el método execute():

```
from mysql.connector import MySQLConnection, Error
argu=[("Romeo y Julieta", 31),("La Colmena", 17)]
query = "INSERT INTO Libros(titulo,ejemplares) VALUES(%s,%s)"
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    for fila in argu:
        cursor.execute(query, fila)
    cursor.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

Consultas

- Para realizar consultas a la base de datos también se utiliza el método execute tomando como argumento una cadena que represente una sentencia SELECT de SQL.

Consultas

- Cuando se realiza una consulta, el cursor no lee todos los datos de la base de datos cuando se ejecuta la sentencia SELECT sino que los datos serán leídos a medida que se pidan las filas.

Consultas

- Para consultar las tuplas resultantes de la sentencia SQL se puede llamar a los métodos de cursor fetchone, fetchmany o fetchall o usar el objeto cursor como un iterador. El significado de los métodos es el mismo que había en SQLite.

Consultas

- En este ejemplo se usa fetchall que recupera una lista de las filas que hay en la tabla.

```
from mysql.connector import MySQLConnection, Error
query = "SELECT * FROM biblioteca.Libros"
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query)
    filas=cursor.fetchall()
    for fila in filas:
        print fila
    cursor.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

```
(u'Dracula', 17)
(u'El escarabajo de oro', 15)
(u'El quijote', 21)
(u'Hamlet', 31)
(u'La Colmena', 17)
(u'Romeo y Julieta', 31)
```

Consultas

- De la misma forma que ocurría en SQLite es posible formatear la salida de las filas recuperadas, recuperar sólo determinadas columnas,…

Consultas

- En el siguiente ejemplo se utiliza el método fetchone() que devuelve la siguiente tupla del conjunto resultado o None si no existen más

```
from mysql.connector import MySQLConnection, Error
query = "SELECT * FROM biblioteca.Libros"
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query)
    fila=cursor.fetchone()
    while fila is not None:
        print fila
        fila=cursor.fetchone()
    cursor.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

```
(u'Dracula', 17)
(u'El escarabajo de oro', 15)
(u'El quijote', 21)
(u'Hamlet', 31)
(u'La Colmena', 17)
(u'Romeo y Julieta', 31)
```

Consultas

- Usando el método fetchmany se pasa como parámetro el número de tuplas que se quieren recuperar.

```
from mysql.connector import MySQLConnection, Error
query = "SELECT * FROM biblioteca.Libros"
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query)
    filas=cursor.fetchmany(4)
    for fila in filas:
        print fila
    cursor.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

```
(u'Dracula', 17)
(u'El escarabajo de oro', 15)
(u'El quijote', 21)
(u'Hamlet', 31)
```

Consultas

- Observar que una vez que se han recuperado todas las filas con fetchall, fetchone o fetchmany, si se quieren volver a recuperar las filas sería necesario realizar una nueva llamada a execute con la sentencia SELECT dado que se pierden una vez recuperadas.

Consultas

- Alternativamente a los métodos anteriores, también es posible iterar sobre el cursor con el que se ha realizado la consulta.

```
from mysql.connector import MySQLConnection, Error
query = "SELECT * FROM biblioteca.Libros"
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query)
    for fila in cursor:
        print fila
    cursor.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

```
(u'Dracula', 17)
(u'El escarabajo de oro', 15)
(u'El quijote', 21)
(u'Hamlet', 31)
```

Actualizaciones

- Para realizar actualizaciones o borrados también se usa el método execute del objeto cursor. En el siguiente ejemplo se va actualizar la columna "ejemplares" de la fila correspondiente al libro con título "Dracula" y se va a rellenar con el valor 45.

```
from mysql.connector import MySQLConnection, Error
query1 = "UPDATE Libros SET ejemplares=%s WHERE titulo=%s"
arg1=(45,'Dracula')
query2 = "SELECT * FROM Libros WHERE titulo=%s"
arg2=('Dracula',)
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query1,arg1)
    conn.commit()
    cursor.execute(query2,arg2)
    print cursor.fetchall()
    conn.close()
except Error as error:
    print(error)
```

```
[(u'Dracula', 45)]
```

Actualizaciones

- En el siguiente ejemplo se va eliminar la fila correspondiente al libro con título “Romeo y Julieta”.

```
from mysql.connector import MySQLConnection, Error
query1 = "DELETE FROM Libros where titulo=%s"
arg1=('Romeo y Julieta',)
query2 = "SELECT * FROM Libros"
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query1,arg1)
    conn.commit()
    cursor.execute(query2)
    filas=cursor.fetchall()
    for fila in filas:
        print fila
    conn.close()
except Error as error:
    print(error)
```

```
(u'Dracula', 45)
(u'El escarabajo de oro', 15)
(u'El quijote', 21)
(u'Hamlet', 31)
(u'La Colmena', 17)
```

Llamada a procedimientos almacenados

- Para poder llamar a un procedimiento almacenado en primer lugar hay que almacenar el procedimiento en la base de datos. En el siguiente ejemplo se almacena una simple consulta que recupera todos los libros de la base de datos.

```
CREATE PROCEDURE `Recuperar` ()
BEGIN
SELECT * FROM Libros;
END
```

Llamada a procedimientos almacenados

- A continuación desde Python se puede ejecutar un procedimiento almacenado usando el método callproc de un objeto cursor. Este método recibe como argumento el nombre del procedimiento almacenado y genera un conjunto de resultados
- Una vez llamado para recuperar las filas resultantes, es necesario usar el método stored_results() del objeto cursor que devuelve de un iterador sobre el conjunto de resultados, requiriendo iterar sobre la misma usando alguno de los métodos vistos:fetchall, fetchone,…

Llamada a procedimientos almacenados

```
from mysql.connector import MySQLConnection, Error
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca
    cursor = conn.cursor()
    cursor.callproc('Recuperar')
    for fila in cursor.stored_results():
        print fila.fetchall()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

```
>>>
[(u'Dracula', 45), (u'El escarabajo de oro', 15), (u'El qu
ijote', 21), (u'Hamlet', 31), (u'La Colmena', 17)]
>>>
```

Llamada a procedimientos almacenados

- En el caso de que el procedimiento almacenado tuviera argumentos entonces hay que pasarlos al método callproc como segundo argumento del mismo en forma de una lista.

Llamadas a procedimientos almacenados

- En el siguiente ejemplo se llama a un procedimiento que recupera los ejemplares de un título de libro pasado como argumento.

```
CREATE PROCEDURE `RecuperaTitulo` (IN tit VARCHAR(54), OUT ejemp INT(25) )
BEGIN
SELECT ejemplares FROM Libro WHERE titulo=tit;
END
```

Llamadas a procedimientos almacenados

```
from mysql.connector import MySQLConnection, Error
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    arg=['Dracula',0]
    cursor.callproc('RecuperaTitulo',arg)
    for fila in cursor.stored_results():
        print fila.fetchall()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

```
>>>
[(45,)]
```

Manipulación de Blob

- Un caso particular de tipo de datos son los datos blob pensados para almacenar datos de gran tamaño como por ejemplo una foto. Se tratan de la misma forma que los datos normales.

Manipulación de datos Blob

- En el siguiente ejemplo se carga una foto desde un archivo y se almacena en una tabla que tiene dos columnas formadas por un identificador y una columna de tipo blob que albergará la foto.

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
idFotos	int(11)		NO			select,insert,update,referen
Foto	blob		YES			select,insert,update,referen

Manipulación de datos Blob

- En el siguiente programa en Python se lee una foto “Ejemplo.jpg” y se inserta dentro de la tabla.

```
from mysql.connector import MySQLConnection, Error
f=open('Ejemplo.jpg','rb')
foto=f.read()
query="INSERT INTO Fotos(Foto) VALUES(%s)"
arg=(foto,)
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query,arg)
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

	idFotos	Foto
▶	1	BLOB
*	NULL	NULL

Manipulación de datos Blob

- De manera similar se pueden recuperar datos de este tipo para almacenarlos en un archivo. En el siguiente ejemplo se va a realizar la operación inversa, y se recuperarán la foto y se escribirá a un archivo de salida.

```
from mysql.connector import MySQLConnection, Error
f=open('Salida.jpg','wb')
query="SELECT * FROM Fotos WHERE idFotos=%s"
arg=(1,)
try:
    conn = MySQLConnection(user='pepito', password='45666',host='127.0.0.1', port=3307, database='Biblioteca')
    cursor = conn.cursor()
    cursor.execute(query,arg)
    foto=cursor.fetchone()[1]
    f.write(foto)
    f.close()
    conn.commit()
    conn.close()
except Error as error:
    print(error)
```

Ejemplo-MySQL

- En siguiente ejemplo ha sido obtenido de la guía de desarrollo de MySQL con el conector Python, y en el mismo se crea una base de datos para mantener información de empleados de una empresa.

Ejemplo-MySQL

- En primer lugar se crea un diccionario TABLES con todas las sentencias de creación de las tablas de la base de datos, y se define una variable DB_NAME que contiene el nombre de la base de datos.

Ejemplo-MySQL

```
DB_NAME = 'employees'

TABLES = {}
TABLES['employees'] = (
    "CREATE TABLE `employees` ("
    "  `emp_no` int(11) NOT NULL AUTO_INCREMENT,"
    "  `birth_date` date NOT NULL,"
    "  `first_name` varchar(14) NOT NULL,"
    "  `last_name` varchar(16) NOT NULL,"
    "  `gender` enum('M','F') NOT NULL,"
    "  `hire_date` date NOT NULL,"
    "  PRIMARY KEY (`emp_no`)"
    ") ENGINE=InnoDB")

TABLES['departments'] = (
    "CREATE TABLE `departments` ("
    "  `dept_no` char(4) NOT NULL,"
    "  `dept_name` varchar(40) NOT NULL,"
    "  PRIMARY KEY (`dept_no`), UNIQUE KEY `dept_name` (`dept_name`)"
    ") ENGINE=InnoDB")
```

Ejemplo-MySQL

```
TABLES['salaries'] = (
    "CREATE TABLE `salaries` ("
    "  `emp_no` int(11) NOT NULL,"
    "  `salary` int(11) NOT NULL,"
    "  `from_date` date NOT NULL,"
    "  `to_date` date NOT NULL,"
    "  PRIMARY KEY (`emp_no`,`from_date`), KEY `emp_no` (`emp_no`),"
    "  CONSTRAINT `salaries_ibfk_1` FOREIGN KEY (`emp_no`) "
    "     REFERENCES `employees` (`emp_no`) ON DELETE CASCADE"
    ") ENGINE=InnoDB")

TABLES['dept_emp'] = (
    "CREATE TABLE `dept_emp` ("
    "  `emp_no` int(11) NOT NULL,"
    "  `dept_no` char(4) NOT NULL,"
    "  `from_date` date NOT NULL,"
    "  `to_date` date NOT NULL,"
    "  PRIMARY KEY (`emp_no`,`dept_no`), KEY `emp_no` (`emp_no`),"
    "  KEY `dept_no` (`dept_no`),"
    "  CONSTRAINT `dept_emp_ibfk_1` FOREIGN KEY (`emp_no`) "
    "     REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,"
    "  CONSTRAINT `dept_emp_ibfk_2` FOREIGN KEY (`dept_no`) "
    "     REFERENCES `departments` (`dept_no`) ON DELETE CASCADE"
    ") ENGINE=InnoDB")
```

Ejemplo-MySQL

```
TABLES['dept_manager'] = (
    "  CREATE TABLE `dept_manager` ("
    "  `dept_no` char(4) NOT NULL,"
    "  `emp_no` int(11) NOT NULL,"
    "  `from_date` date NOT NULL,"
    "  `to_date` date NOT NULL,"
    "  PRIMARY KEY (`emp_no`,`dept_no`),"
    "  KEY `emp_no` (`emp_no`),"
    "  KEY `dept_no` (`dept_no`),"
    "  CONSTRAINT `dept_manager_ibfk_1` FOREIGN KEY (`emp_no`) "
    "     REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,"
    "  CONSTRAINT `dept_manager_ibfk_2` FOREIGN KEY (`dept_no`) "
    "     REFERENCES `departments` (`dept_no`) ON DELETE CASCADE"
    ") ENGINE=InnoDB")

TABLES['titles'] = (
    "CREATE TABLE `titles` ("
    "  `emp_no` int(11) NOT NULL,"
    "  `title` varchar(50) NOT NULL,"
    "  `from_date` date NOT NULL,"
    "  `to_date` date DEFAULT NULL,"
    "  PRIMARY KEY (`emp_no`,`title`,`from_date`), KEY `emp_no` (`emp_no`),"
    "  CONSTRAINT `titles_ibfk_1` FOREIGN KEY (`emp_no`)"
    "     REFERENCES `employees` (`emp_no`) ON DELETE CASCADE"
    ") ENGINE=InnoDB")
```

Ejemplo-MySQL

- A continuación se crea una conexión a MySQL en el que no se indica aún el nombre de la base de datos, la cual se va a crear desde cero.

```
cnx = mysql.connector.connect(user='scott')
cursor = cnx.cursor()
```

Ejemplo-MySQL

- Para crear la base de datos se define una función en Python que crea una base de datos si se le pasa como argumento un objeto de tipo cursor:

```
def create_database(cursor):
    try:
        cursor.execute(
            "CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format(DB_NAME))
    except mysql.connector.Error as err:
        print("Failed creating database: {}".format(err))
        exit(1)

try:
    cnx.database = DB_NAME
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_BAD_DB_ERROR:
        create_database(cursor)
        cnx.database = DB_NAME
    else:
        print(err)
        exit(1)
```

Ejemplo-MySQL

- Una vez que se ha creado la base de datos, se crean las tablas iterando sobre el diccionario que contiene las sentencias de creación de las mismas.

```
for name, ddl in TABLES.iteritems():
    try:
        print("Creating table {}: ".format(name), end='')
        cursor.execute(ddl)
    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
            print("already exists.")
        else:
            print(err.msg)
    else:
        print("OK")

cursor.close()
cnx.close()
```

Ejemplo-MySQL

- Una vez creadas las tablas, ya se pueden insertar datos como por ejemplo en el siguiente script dónde se inserta información personal de un nuevo empleado y sobre su salario.

```
from datetime import date, datetime, timedelta
import mysql.connector

cnx = mysql.connector.connect(user='scott', database='employees')
cursor = cnx.cursor()

tomorrow = datetime.now().date() + timedelta(days=1)

add_employee = ("INSERT INTO employees "
                "(first_name, last_name, hire_date, gender, birth_date) "
                "VALUES (%s, %s, %s, %s, %s)")
add_salary = ("INSERT INTO salaries "
              "(emp_no, salary, from_date, to_date) "
              "VALUES (%(emp_no)s, %(salary)s, %(from_date)s, %(to_date)s)")

data_employee = ('Geert', 'Vanderkelen', tomorrow, 'M', date(1977, 6, 14))
```

Ejemplo-MySQL

```
# Insert new employee
cursor.execute(add_employee, data_employee)
emp_no = cursor.lastrowid

# Insert salary information
data_salary = {
  'emp_no': emp_no,
  'salary': 50000,
  'from_date': tomorrow,
  'to_date': date(9999, 1, 1),
}
cursor.execute(add_salary, data_salary)

# Make sure data is committed to the database
cnx.commit()

cursor.close()
cnx.close()
```

Ejemplo-MySQL

- También se pueden hacer consultas como por ejemplo recuperar todos los empleados contratados en el año 1999.

```
import datetime
import mysql.connector

cnx = mysql.connector.connect(user='scott', database='employees')
cursor = cnx.cursor()

query = ("SELECT first_name, last_name, hire_date FROM employees "
         "WHERE hire_date BETWEEN %s AND %s")

hire_start = datetime.date(1999, 1, 1)
hire_end = datetime.date(1999, 12, 31)

cursor.execute(query, (hire_start, hire_end))

for (first_name, last_name, hire_date) in cursor:
  print("{}, {} was hired on {:%d %b %Y}".format(
    last_name, first_name, hire_date))

cursor.close()
cnx.close()
```