Acceso a Bases de Datos

Antonio Sarasa Cabezuelo

Acceso a Bases de Datos

- En Python existe una propuesta de API estándar para el manejo de bases de datos, de forma que el código sea prácticamente igual independientemente de la base de datos que se está utilizando por debajo. Esta especificación recibe el nombre de Python Database API o DB-API.

Acceso a Base de Datos

- Se van a estudiar dos casos:
  - SQLite.
  - MySQL

SQLite

- Python tiene integrada una base de datos relacional denominada SQLite

Creación de una base de datos

- En primer lugar hay que realizar una conexión con el servidor de la base de datos. Esto se hace mediante la función connect, cuyos parámetros no están estandarizados y dependen de la base de datos.
- En el caso de sqlite3 sólo se necesita pasar como parámetro una cadena con la ruta al archivo en el que guardar los datos de la base de datos, o bien la cadena ":memory:" para utilizar la memoria RAM en lugar de un fichero en disco.

Creación de una base de datos

- Vamos a crear una base de datos denominada “Biblioteca”

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
```

- La función connect devuelve un objeto de tipo Connection que representa la conexión con conexión con la base de datos almacenada en el archivo biblioteca.sqlite3 del directorio actual. Si el archivo no existe, se creará nuevo.

Creación de una base de datos

- Las distintas operaciones que se pueden realizar con la base de datos se realizan a través de un objeto Cursor. Para crear este objeto se utiliza el método cursor() del objeto Connection.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
```

Creación de una base de datos

- Una vez que tenemos el cursor, se pueden ejecutar comandos sobre el contenido de la base de datos, usando el método execute() que toma como argumento una cadena con el código SQL a ejecutar.

Creación de una base de datos

- Por ejemplo 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 sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute("DROP TABLE IF EXISTS Libros")
cur.execute("CREATE TABLE Libros (titulo TEXT, ejemplares INTEGER)")
cur.close()
```

Inserción de datos

- Una vez creada la tabla Libros se pueden guardar datos usando una llamada a execute() con el comando SQL INSERT. Este comando indica qué tabla se va a utilizar y luego define una fila nueva, enumerando los campos a incluir y seguidos por los valores (VALUES) que se desean colocar en esa fila.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( "El quijote", 20 )' )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ("El escarabajo de oro", 15)' )
cur.close()
```

Inserción de datos

- Otra forma de insertar consiste en especificar como signos de interrogación (?,?) los valores para indicar que serán pasados como una tupla en el segundo parámetro de la llamada a execute().

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
cur.close()
```

Inserción de datos

- Si la base de datos soporta transacciones y están activadas, y además la característica de *auto-commit* está desactivada, será necesario llamar al método commit de la conexion para que se lleven a cabo las operaciones definidas en la transacción.

Inserción de datos

- Si en estas circunstancias se usara una herramienta externa para comprobar el contenido de la base de datos sin hacer primero el commit aparecería entonces con una base de datos vacía. Sin embargo si se consulta desde Python parecería que se han llevado a cabo los cambios, aunque no es así.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
cur.close()
conn.commit()
```

Inserción de datos

- En el ejemplo primero se insertan dos filas en la tabla con INSERT y luego se usa commit() para forzar que los datos sean escritos en el archivo de la base de datos.

Inserción de datos

- Si la base de datos soporta la característica de *rollback* entonces se puede cancelar la transacción actual con el método rollback de la conexión. Si la base de datos no soporta rollback, entonces al llamar a este método producirá una excepción.

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.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.executemany('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
[( 'El quijote', 20 ),( 'El escarabajo de oro', 15 ) ])
print cur.rowcount
cur.close()
conn.commit()
```

Inserción de datos

- El mismo efecto se podría haber conseguido utilizando un bucle sobre una lista y el método execute().

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
entradas=[( 'El quijote', 20 ),( 'El escarabajo de oro', 15 ) ]
for fila in entradas:
    cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',fila)
cur.close()
conn.commit()
```

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.

Consultas

- En los siguientes ejemplos primero se insertaran dos filas en la tabla con INSERT y luego se usará commit() para forzar que los datos sean escritos en el archivo de la base de datos.
- Después se usará el comando SELECT para recuperar las filas que se acaban de insertar en la tabla, y en cada ejemplo se usará una forma distinta de pedir las filas recuperadas.

Consultas

- Al final del programa se ejecuta el comando DELETE para borrar las filas que se acaban de crear, y por último se ejecuta un commit() para forzar a los datos a ser eliminados de la base de datos.

Consultas

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

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT titulo, ejemplares FROM Libros')
print cur.fetchall()
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
...
Libros:
[(u'El quijote', 20), (u'El escarabajo de oro', 15)]
```

Consultas

- También es posible iterar sobre las tuplas obtenidas con fetchall:

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT titulo, ejemplares FROM Libros')
for fila in cur.fetchall():
    print fila
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
(u'El quijote', 20)
(u'El escarabajo de oro', 15)
```

Consultas

- También es posible recuperar los valores de las tuplas utilizando tuplas en el bucle.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT titulo, ejemplares FROM Libros')
for (titulo,ejemplar) in cur.fetchall():
    print "Titulo:", titulo
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
Titulo: El quijote
Titulo: El escarabajo de oro
```

Consultas

- En el siguiente ejemplo se recupera una única columna de la tabla y luego se usa fetchall:

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT titulo FROM Libros')
titulos=cur.fetchall()
print titulos
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
[(u'El quijote',), (u'El escarabajo de oro',)]
```

Consultas

- En este último caso se podría formatear para que en vez de devolver una tupla con un solo valor devolviera una lista con los valores recuperados usando comprensión de listas

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT * FROM Libros')
titulos=[rec[0] for rec in cur.fetchall()]
print titulos
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
[u'El quijote', u'El escarabajo de oro']
```

Consultas

- Si las tablas que se usan son muy grandes es mejor no pedir todas las filas mediante fetchall, usando otros métodos alternativos como fetchone que devuelve la siguiente tupla del conjunto resultado o None si no existen más

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT * FROM Libros')
while True:
    fila=cur.fetchone()
    if not fila: break
    print fila
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
(u'El quijote', 20)
(u'El escarabajo de oro', 15)
```

Consultas

- Otra alternativa a fetchall es el método fetchmany que devuelve el número de tuplas indicado por el entero pasado como parámetro o bien el número indicado por el atributo Cursor.arraysize si no se pasa ningún parámetro(por defecto vale 1).

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT * FROM Libros')
while True:
    filas=cur.fetchmany()
    if not filas: break
    for fila in filas:
        print fila
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
(u'El quijote', 20)
(u'El escarabajo de oro', 15)
...
```

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.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT * FROM Libros')
for resultado in cur:
    print resultado
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
(u'El quijote', 20)
(u'El escarabajo de oro', 15)
```

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 "El quijote" y se va a rellenar con el valor 22.

Actualizaciones

- Si se hace la consulta para ver lo que hay en la base de datos:

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El quijote', 20 ) )
cur.execute('INSERT INTO Libros (titulo, ejemplares) VALUES ( ?, ? )',
( 'El escarabajo de oro', 15 ) )
conn.commit()
print 'Libros:'
cur.execute('SELECT titulo, ejemplares FROM Libros')
for (titulo,ejemplar) in cur.fetchall():
    print "Titulo:", titulo
    print "Ejemplar:", ejemplar
cur.execute('DELETE FROM Libros')
cur.close()
conn.commit()
```

```
Libros:
Titulo: El quijote
Ejemplar: 20
Titulo: El escarabajo de oro
Ejemplar: 15
```

Actualizaciones

- Ahora se realiza la actualización:

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('UPDATE Libros set ejemplares=? WHERE titulo=?',[22,"El quijote"])
conn.commit()
cur.execute('SELECT * FROM Libros')
for (titulo,ejemplar) in cur.fetchall():
    print "Titulo:", titulo
    print "Ejemplar:", ejemplar
cur.close()
conn.commit()
```

```
Titulo: El quijote
Ejemplar: 22
Titulo: El escarabajo de oro
Ejemplar: 15
```

Actualizaciones

- En el siguiente ejemplo se va eliminar la fila correspondiente al libro con título “El quijote”.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('DELETE from Libros WHERE titulo=?',["El quijote"])
conn.commit()
cur.execute('SELECT * FROM Libros')
for (titulo,ejemplar) in cur.fetchall():
    print "Titulo:", titulo
    print "Ejemplar:", ejemplar
cur.close()
conn.commit()
```

```
...
Titulo: El escarabajo de oro
Ejemplar: 15
```

Operaciones sobre columnas

- Se pueden realizar operaciones de columna cuando se ejecuta una sentencia SELECT. En el siguiente ejemplo se van a sumar los ejemplares de todos los libros almacenados en la base de datos.

```
import sqlite3
conn = sqlite3.connect('biblioteca.sqlite3')
cur = conn.cursor()
cur.execute('SELECT sum(ejemplares) FROM Libros')
print cur.fetchall()
cur.close()
conn.commit()
```

```
[(35,)]
```

Ejemplo-SQLite

- Se quiere implementar un programa que guarde información de los amigos que una persona tiene en Twitter y de las relaciones que tienen a su vez estos amigos entre sí. Esta información se almacenará en una base de datos.

Ejemplo SQLite

- En primer lugar se fija la estructura de la base de datos que se va a utilizar:
  - Se creará una tabla llamada Personas que almacenará la información de las cuentas de Twitter.
  - Se creará una tabla llamada Seguimientos que almacenará las relaciones que existen entre las personas.

Ejemplo SQLite

- La tabla Personas dispondrá de 3 columnas:
  - Un id que actuará como clave primaría de la tabla.
  - El nombre de usuario de la cuenta de Twitter.
  - Un valor entero que puede valer 0 o 1 que indica si la información de dicha cuenta ha sido recuperada o no.

```
cur.execute('''CREATE TABLE IF NOT EXISTS Personas
    (id INTEGER PRIMARY KEY, nombre TEXT UNIQUE, recuperado INTEGER)''')
```

Ejemplo SQLite

- La tabla Seguimiento dispondrá de 2 columnas que contendrán id´s de usuarios registrados en la tabla Personas y que representarán una relación de seguimiento entre ambos usuarios con un sentido definido.

```
cur.execute('''CREATE TABLE IF NOT EXISTS Seguimientos
    (desde_id INTEGER, hacia_id INTEGER, UNIQUE(desde_id, hacia_id))''')
```

Ejemplo SQLite

- Observar que la combinación de los dos números de cada fila de la tabla Seguimientos se especifica como única para evitar que se cometan errores como añadir la misma relación entre las mismas personas más de una vez.

Ejemplo SQLite

- Estas restricciones obligan a que las inserciones que se hagan en las tablas añadan la clausula IGNORE en la sentencia INSERT para indicar que si este INSERT en concreto causara una violación de la regla “el nombre debe ser ´unico”, el sistema de la base de datos está autorizado a ignorar el INSERT.

```
cur.execute('''INSERT OR IGNORE INTO Personas (nombre, recuperado)
    VALUES ( ?, 0)''', ( amigo, ) )
```

```
cur.execute('''INSERT OR IGNORE INTO Seguimientos
    (desde_id, hacia_id) VALUES (?, ?)''', (id, amigo_id) )
```

Ejemplo SQLite

- La estructura que se va a seguir para implementar el programa será:
  - Crear tablas con claves primarias y restricciones.
  - Cuando se disponga de una cuenta, y se necesita el valor del id de esa persona, dependiendo de si esa persona ya está o no en la tabla Personas:
    - Buscar lapersona en la tabla Personas y recuperar el valor de id para esa persona,
    - Añadir la persona a la tabla Personas y obtener el valor del id para la fila recién añadida.
  - Insertar la fila que indica la relación de "seguimiento".

Ejemplo SQLite

- En el siguiente trozo de código se crean las tablas.

```
import urllib
import twurl
import json
import sqlite3
TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'
conn = sqlite3.connect('amigos.sqlite3')
cur = conn.cursor()
#Se crean las tablas que almacenarán la información
cur.execute('''CREATE TABLE IF NOT EXISTS Personas (id INTEGER PRIMARY KEY, nombre TEXT UNIQUE, recuperado INTEGER)''')
cur.execute('''CREATE TABLE IF NOT EXISTS Seguimientos(desde_id INTEGER, hacia_id INTEGER, UNIQUE(desde_id, hacia_id))''')
```

Ejemplo SQLite

- En el siguiente trozo de código se pide al usuario una cuenta de Twitter, de forma que si la cuenta ya existe se debe averiguar el valor de su id, y si la cuenta no existe aún en la tabla Personas, se debe insertar el registro y obtener el valor del id de la fila recién insertada(para ello se us cur.lastrowid que proporciona el valorque la base de datos ha asignado a la columna id en la fila recién creada)

Ejemplo SQLite

```
#Se recupera una cuenta para procesar sus relaciones de seguimiento
while True:
    cuenta = raw_input('Introduzca una cuenta de Twitter, o salir: ')
    if ( cuenta == 'salir' ) : break
    if ( len(cuenta) < 1 ) :
        cur.execute('''SELECT id, nombre FROM Personas WHERE recuperado = 0 LIMIT 1''')
        try:
            (id, cuenta) = cur.fetchone()
        except:
            print 'No se han encontrado cuentas de Twitter sin recuperar'
            continue
    else:
        cur.execute('SELECT id FROM Personas WHERE nombre = ? LIMIT 1',(cuenta, ) )
        try:
            id = cur.fetchone()[0]
        except:
            cur.execute('''INSERT OR IGNORE INTO Personas (nombre, recuperado) VALUES ( ?, 0)''', ( cuenta, ) )
            conn.commit()
            if cur.rowcount != 1 :
                print 'Error insertando cuenta:',cuenta
                continue
            id = cur.lastrowid
```

Ejemplo SQLite

- A continuación se recupera la información de amistades del usuario y se actualiza la tabla Personas para indicar que esa cuenta ya ha sido procesada(para lo cual se fija el campo recuperado a 1).

```
url = twurl.augment(TWITTER_URL,{'screen_name': cuenta, 'count': '20'} )
print 'Recuperando cuenta', cuenta
conexion = urllib.urlopen(url)
datos = conexion.read()
js = json.loads(datos)
cur.execute('UPDATE Personas SET recuperado=1 WHERE nombre = ?', (cuenta, ) )
```

Ejemplo SQLite

- El siguiente trozo de código realiza la búsqueda de las amistades del usuario en el documento json recuperado desde Twitter.

Ejemplo SQLite

- Una vez extraido el nombre de usuario del documento JSON, se intenta buscar el id de esa cuenta en la tabla Personas:
  - En la sección try se intenta recuperar el id mediante un fetchone que se almacena en amigo_id.
  - Si el SELECT falla, el fetchone()[0] falla, y el control se transfiere a la sección except.

Ejemplo SQLite

- Si se entra en el código except significa que la fila no se ha encontrado en la tabla Personas, por lo que hay que insertarla usando un INSERT OR IGNORE y luego un commit() para forzar a que la base de datos se actualice. Si la inserción ha tenido éxito se usa cur.lastrowid para averiguar el valor asignado a la columna id en la fila creada.
- Conocidos los dos id´s se insertan ambos en la tabla de Seguimientos.

Ejemplo SQLite

```
for u in js['users'] :
    amigo = u['screen_name']
    print amigo
    cur.execute('SELECT id FROM Personas WHERE nombre = ? LIMIT 1',(amigo, ) )
    try:
        amigo_id = cur.fetchone()[0]
    except:
        cur.execute('''INSERT OR IGNORE INTO Personas (nombre, recuperado) VALUES ( ?, 0)''', ( amigo, ) )
        conn.commit()
        if cur.rowcount != 1 :
            print 'Error al insertar cuenta:',amigo
            continue
        amigo_id = cur.lastrowid
    cur.execute('''INSERT OR IGNORE INTO Seguimientos (desde_id, hacia_id) VALUES (?, ?)''', (id, amigo_id) )
    conn.commit()
```

Ejemplo SQLite

- Una vez se haya ejecutado varias veces el programa, se habrán poblado las 2 tablas, y entonces se podrán consultar para saber las relaciones de amistad. Para ello se va a crear otro programa que mostrará:
  - La información de las tablas Personas y Seguimientos
  - La información compactada de una persona y a las personas que sigue

Ejemplo SQLite

- Para mostrar la información de las tablas Personas y Seguimientos se consultan las tablas con SELECT:

```
import sqlite3
conn = sqlite3.connect('amigos.sqlite3')
cur = conn.cursor()

cur.execute('SELECT * FROM Personas')
contador = 0
print 'Personas:'
for fila in cur :
    print fila
    contador = contador + 1
print contador, 'filas.'
cur.execute('SELECT * FROM Seguimientos')
contador = 0
print 'Seguimientos:'
for fila in cur :
    print fila
    contador = contador + 1
print contador, 'filas.'
```

Ejemplo SQLite

- Para recuperar a las personas que sigue una persona en concreto se realiza un SELECT con una claúsula JOIN.

```
ids=cur.execute('SELECT id FROM Personas')
for id in ids:
    print 'Conexiones para id=',id,':'
    cur.execute('''SELECT * FROM Seguimientos JOIN Personas
ON Seguimientos.hacia_id = Personas.id WHERE Seguimientos.desde_id =? ''',(id,))
    for fila in cur :
        print fila
cur.close()
```

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()
```