

PROGRAMACIÓN AVANZADA CON PYTHON

(CEFIRE CTEM)



Acceso a bases de datos con Python y SQLite

Esta obra está sujeta a la licencia Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional de Creative Commons. Para ver una copia de esta licencia, visitad

<http://creativecommons.org/licenses/by-nc-sa/4.0/>.



Autora: María Paz Segura Valero (segura_marval@gva.es)

CONTENIDO

1. Introducción.....	2
2. Conexión y desconexión de una BD.....	2
3. Uso de cursores.....	4
3.1. Creación y borrado de tablas.....	4
3.2. Inserción, actualización y borrado de registros.....	5
3.3. Consulta de datos.....	7
4. Conclusión.....	8
5. Fuentes de información.....	8

1. Introducción

Como ya dijimos anteriormente, no necesitamos hacer ninguna instalación extra para que Python pueda trabajar con bases de datos SQLite ya que existe una librería incorporada en el lenguaje llamada **sqlite3**. Así que, simplemente, deberemos importar dicha librería para poder hacer uso de las funciones y elementos que incluye para crear bases de datos y manejar su información.

La sintaxis para importar la librería es la siguiente:

```
import sqlite3
```

2. Conexión y desconexión de una BD

Lo primero que tenemos que hacer para poder trabajar con una base de datos es conectarnos a una de ellas. Puede que nos conectemos a una base de datos existente, a una base de datos nueva que aún no tiene creada su estructura o, incluso, a una base de datos temporal para hacer pruebas.

Las dos primeras residirán en el disco duro de nuestra máquina, es decir, se creará un fichero con extensión **.db** que contendrá toda la información de la base de datos. Pero la última será una base de datos que sólo residirá en la RAM. En cuanto acabe el programa, desaparecerá.

```
nombre_conexion = sqlite3.connect(nombre_fichero_db)
```

Donde:

- La variable **nombre_conexion** es la variable del programa Python que representa la base de datos. Podemos ponerle cualquier nombre.
- En **nombre_fichero_db** deberemos indicar el nombre del fichero de la base de datos que queremos utilizar para almacenarla (entre comillas) o el nombre especial **:memory:** para trabajar con una base de datos temporal.

Cuando queramos cerrar la conexión de la base de datos, utilizaremos esta instrucción:

```
nombre_conexion.close()
```

Donde **nombre_conexion** es la variable del programa Python que representa la base de datos.

Ejemplo 1: Nos conectamos a la base de datos "ejemplos.db" cuyo fichero está ubicado en la misma carpeta que el programa Python.

```
import sqlite3;
mi_conexion = sqlite3.connect("ejemplos.db")
#Instrucciones de utilización de la base de datos
mi_conexion.close()
```

Ejemplo 2: Nos conectamos a la base de datos "ejemplos.db" cuyo fichero está ubicado en una carpeta diferente a la del programa Python.

```
import sqlite3;
mi_conexion = sqlite3.connect("C:/proyectos/ejemplos.db")
#Instrucciones de utilización de la base de datos
mi_conexion.close()
```

Ejemplo 3: Vamos a trabajar con una base de datos temporal, que no se almacenará en un soporte físico de la máquina.

```
import sqlite3;
mi_conexion = sqlite3.connect(":memory:")
#Instrucciones de utilización de la base de datos
mi_conexion.close()
```

En un programa Python puedo trabajar con varias bases de datos a la vez. Por ejemplo, para hacer una migración de datos de una a otra.

Para ello, deberíamos abrir una conexión por cada base de datos que vamos a necesitar y ejecutar las funciones de las conexiones correspondientes.

Ejemplo: Abrimos varias conexiones con distintas bases de datos.

```
import sqlite3;
con_sec = sqlite3.connect("secundaria.db")
con_pri = sqlite3.connect("primaria.db")
con_inf = sqlite3.connect("infantil.db")
#Instrucciones de utilización de las bases de datos
con_sec.close()
con_pri.close()
con_inf.close()
```

3. Uso de cursores

Un **cursor** es un objeto de la base de datos que nos permite ejecutar instrucciones SQL en una base de datos y recuperar registros de la misma.

Para poder utilizar las funciones de un cursor, primero debemos crearlo:

```
nombre_cursor = nombre_conexion.cursor()
```

Donde:

- **nombre_cursor** es el nombre de la variable del programa de Python que contendrá el cursor de la base de datos
- **nombre_conexion** es el nombre de la conexión a la base de datos que habremos definido previamente.

Ejemplo: Definición de un cursor de una base de datos.

```
import sqlite3;
mi_conexion = sqlite3.connect(":memory:")
mi_cursor = mi_conexion.cursor()
#Instrucciones de utilización del cursor
mi_conexion.close()
```

A continuación, vamos a ver las operaciones que podemos hacer en una base de datos a través de un cursor.

3.1. Creación y borrado de tablas

Podemos ejecutar sentencias SQL a través de un cursor de una base de datos teniendo en cuenta que siempre que hagamos cambios en la estructura de una base de datos, deberemos ejecutar la función **commit()** para hacer permanentes dichos cambios.

La sintaxis es la siguiente:

```
mi_cursor.execute(sentencia_SQL_sin_punto_y_coma)  
mi_conexion.commit()
```

Donde:

- **mi_cursor** es la variable cursor del programa.
- **sentencia_SQL_sin_puntoycoma** es una instrucción SQL sin el punto y coma del final. Podemos escribir directamente la instrucción SQL o pasar una variable de tipo texto que la contenga.

Para escribir en Python un texto que ocupa varias líneas deberás utilizar la triple comilla.

- **mi_conexion** es la variable conexión del programa.

Ejemplo: Creamos la tabla LENGUA_MATERNA.

```
import sqlite3;
mi_conexion = sqlite3.connect(":memory:")
mi_cursor = mi_conexion.cursor()

sentencia_sql = '''CREATE TABLE IF NOT EXISTS LENGUA_MATERNA
    (abreviatura TEXT, nombre TEXT,
    descripcion TEXT,
    PRIMARY KEY(abreviatura) ) ''' #sin punto y coma final

mi_cursor.execute(sentencia_sql)
mi_conexion.commit()

mi_conexion.close()
```

Ejemplo 2: Borramos la tabla AFILIACIONES.

```
import sqlite3;
mi_conexion = sqlite3.connect("ejemplos.db")
mi_cursor = mi_conexion.cursor()

sentencia_sql = "DROP TABLE IF EXISTS AFILIACIONES"

mi_cursor.execute(sentencia_sql)
mi_conexion.commit()

mi_conexion.close()
```

3.2. Inserción, actualización y borrado de registros

Al igual que hemos ejecutados sentencias de creación y borrado de tablas, podemos ejecutar sentencias para insertar, borrar y actualizar registros de la base de datos. Aquí también deberemos utilizar la función **commit()** para hacer permanentes los cambios en la base de datos.

Ejemplo: Ejecutar instrucciones de manipulación de registros.

```
import sqlite3;
mi_conexion = sqlite3.connect(":memory:")
mi_cursor = mi_conexion.cursor()

ins = '''INSERT INTO AFILIACIONES
        VALUES(555555555, "1", "15/01/2020")'''
mod = '''UPDATE AFILIACIONES
        SET dni = 333333333 WHERE dni = 555555555'''
bor = "DELETE FROM AFILIACIONES WHERE dni = 333333333"

#Insertamos un registro
mi_cursor.execute(ins)
#Actualizamos registros
mi_cursor.execute(mod)
#Borramos registros
mi_cursor.execute(bor)

mi_conexion.commit()
mi_conexion.close()
```

También podemos utilizar parámetros a la hora de construir las sentencias SQL. Por ejemplo, podríamos insertar registros utilizando los valores almacenados en variables del programa. Para ello, utilizaremos el carácter **?** por cada parámetro de la sentencia.

Ejemplo: Insertamos un registro con dos parámetros que pasamos a la función en forma de tupla.

```
ins = '''INSERT INTO AFILIACIONES
        VALUES(?, ?, "15/01/2020")'''
dni = 555555555
mi_cursor.execute(ins, (dni, "1"))
```

El segundo parámetro de la función **execute()** es una tupla con tantos valores como parámetros (?) se esperen en la sentencia.

Si solo vamos a pasar un valor como parámetro escribiremos **(valor,)** en lugar de **(valor)**.

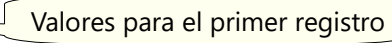
Ejemplo: Pasamos un solo parámetro a la instrucción SQL.

```
ins = '''INSERT INTO AFILIACIONES
        VALUES(?, "1", "15/01/2020")'''
dni = 555555555
mi_cursor.execute(ins, (dni, ))
```

Utilizando una técnica parecida se puede insertar un grupo de registros en una tabla con una sola instrucción. Para ello se utiliza la función **executemany()** a la que se le pasa una lista de tuplas con los valores de los campos de cada registro.

Ejemplo: Insertamos tres registros en la tabla AFILIACIONES basándonos en los valores de una lista de tuplas que pasamos a la función.

```
lista = [(333333333, '1', '15/01/2020'),  
        (333333333, '2', '02/01/2021'),  
        (444444444, '3', '09/04/2021')]
```



```
ins = "INSERT INTO AFILIACIONES VALUES(?, ?, ?)"  
mi_cursor.executemany(ins, lista)
```

3.3. Consulta de datos

Y, por último, vamos a ver cómo ejecutar consultas de datos en una base de datos y poder utilizar los valores recuperados en nuestros programas. En este caso no será necesario ejecutar la función **commit()** porque no vamos a hacer cambios en la base de datos.

Seguiremos utilizando la función **execute()** para ejecutar la sentencia SQL pero, además, usaremos las funciones **fetchone()** y **fetchall()** de los cursores para recuperar los resultados de la consulta.

La función **fetchone()** devuelve una tupla con los valores de las columnas correspondientes. Si no recupera ningún registro entonces devuelve el valor **None**.

Ejemplo 1: Uso de la función **fetchone()** para recuperar un registro de la consulta.

```
#Ejecutamos la consulta SQL  
mi_cursor.execute("SELECT * FROM personas")  
  
#Recuperamos el primer registro de la consulta.  
registro = mi_cursor.fetchone()  
if registro == None:  
    print("La consulta no ha devuelto ningún registro")  
else:  
    print(registro)
```

Podemos utilizar la función **fetchone()** dentro de un bucle para recuperar todos los registros de la consulta.

Ejemplo 2: Uso de la función **fetchone()** con un bucle.

```
#Ejecutamos la consulta SQL
mi_cursor.execute("SELECT * FROM personas")

#Recuperamos el primer registro con el método fetchone.
registro = mi_cursor.fetchone()
while registro != None:
    print(registro)
    registro = cursor.fetchone() #Recuperamos el siguiente registro
```

La función **fetchall()** devuelve una lista de tuplas con los valores de todos los registros de la consulta. Si la consulta no devuelve ningún registro entonces la función devuelve una lista vacía.

Ejemplo 3: Uso de la función **fetchall()** para recuperar todos los registros de la consulta.

```
#Ejecutamos la consulta SQL
mi_cursor.execute("SELECT * FROM personas")

#Recuperamos todos los registros con el método fetchall.
registros = mi_cursor.fetchall()

for reg in registros:
    print(registro)
```

4. Conclusión

En este documento hemos visto como conectarnos a una base de datos SQLite desde un programa de Python y ejecutar distintas instrucciones SQL tanto para modificar la estructura de una base de datos como para manipular o consultar los registros que contiene.

5. Fuentes de información

- Página oficial del lenguaje Python: <https://www.python.org/>
- Bases de datos SQLite de Hektor Profe: <https://docs.hektorprofe.net/python/bases-de-datos-sqlite/>
- Tutorial de SQLite: <https://www.sqlitetutorial.net>

- Data Management With Python, SQLite, and SQLAlchemy:
<https://realpython.com/python-sqlite-sqlalchemy>