Python Programming

Módulo 4

SQLite3 en Python

El módulo sqlite3 en Python

La librería estándar de Python incluye el módulo `sqlite3` que permite la comunicación con bases de datos SQLite, así que por el momento no tendremos que instalar nada con `pip`.

Para crear una conexión desde un script de Python, utilizamos la función `connect()` indicando el nombre del archivo (si no existe, es creado automáticamente).

```
conn = sqlite3.connect("database.sqlite")
```

Para poder ejecutar una consulta, primero es necesario crear un *cursor*.

```
cursor = conn.cursor()
```

Hecho esto, vía el método `execute()` procedemos a correr un código SQL.

```
# Ejecutar una consulta.
cursor.execute("CREATE TABLE personas
(nombre TEXT, edad NUMERIC)")
```

Cuando una consulta realiza una modificación (cláusulas como `CREATE`, `INSERT`, `UPDATE`, `DELETE`, etc.) en la información o estructura de alguna tabla de la base de datos, es necesario "guardar los cambios" a través de una operación llamada *commit* y que en Python se realiza a través de la función homónima.

```
# Guardar los cambios.
conn.commit()
```

Si la conexión se cierra y no se ha llamado a la función `commit()`, toda consulta que haya alterado la base de datos habrá sido desestimada.

Una vez terminada de usar la conexión, debe cerrarse:

```
conn.close()
```

Para agregar algo de información, vamos a crear una tupla personas que contenga otras tuplas con la estructura (`nombre, edad`), y luego para cada una de ellas ejecutaremos una consulta `INSERT` para añadirla a la tabla.

```
personas = (
    ("Pablo", 30),
    ("Jorge", 41),
    ("Pedro", 27)
)
for nombre, edad in personas:
    cursor.execute("INSERT INTO personas
VALUES (?, ?)", (nombre, edad))
conn.commit()
```

En las consultas nunca se debe utilizar ninguno de los métodos que provee Python para incluir variables dentro de una cadena. En su lugar, se coloca un signo de interrogación por cada valor que se quiera reemplazar, y luego se pasan los objetos dentro de una tupla como segundo argumento. Se trata de una medida de seguridad para evitar inyección de código SQL (véase el apartado sobre cómo prevenir inyecciones SQL más abajo).

Hecho esto, vamos a ejecutar una simple consulta que retorne todas las filas de dicha tabla:

```
cursor.execute("SELECT * FROM personas")
personas = cursor.fetchall()
print(personas)
```

Esto imprime en pantalla:

```
[('Pablo', 30), ('Jorge', 41), ('Pedro',
27)]
```

La función `fetchall()` retorna la información devuelta por la consulta en una lista que contiene tuplas, y cada una de éstas corresponde a una fila de la tabla. La lista puede estar vacía si la tabla está vacía o si los criterios de la consulta no coinciden con ninguna fila de la tabla.

Si se quiere retornar únicamente el primer resultado obtenido, puede utilizarse `fetchone()`.

```
print(cursor.fetchone())
```

El valor de retorno de esta función es una tupla, no una lista, ya que solo obtiene una fila de la tabla especificada. En este caso, imprime:

```
('Pablo', 30)
```

Del mismo modo, el valor de retorno puede ser `None` si la tabla está vacía o si la consulta no devolvió ningún resultado.

Cuando la ejecución de una consulta falla, la excepción lanzada es: `sqlite3.OperationalError`.

Por ejemplo, la siguiente consulta tiene un error de sintaxis.

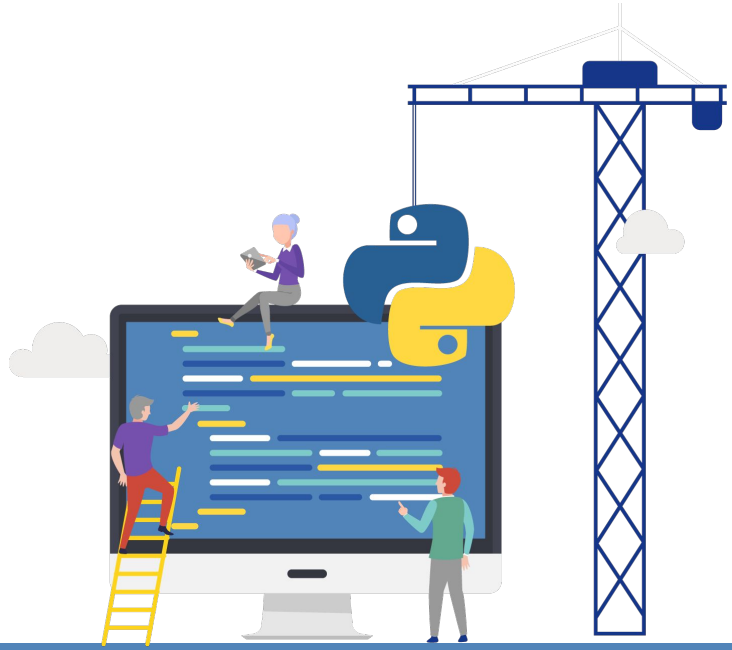
```
cursor.execute("SELECT FROM personas")
```

La excepción es la siguiente:

```
Traceback (most recent call last):
      [...]
      cursor.execute("SELECT FROM
personas")
sqlite3.OperationalError: near "FROM":
syntax error
```

Puede ser capturada usando `try/except` como cualquier otra:

```
try:
   cursor.execute("SELECT FROM personas")
except sqlite3.OperationalError:
   print("La consulta no se ejecutó correctamente.")
```



¡Sigamos trabajando!