MÓDULO 4: Parte 3

SQLITE

BD + Python

1000 PROGRAMADORES

>SQLite Python

PySQLite

PySQLite proporciona una interfaz compatible con Python DBI API 2.0 estandarizada para la base de datos SQLite. Si su aplicación necesita admitir no solo la base de datos SQLite sino también otras bases de datos como MySQL, PostgreSQL y Oracle, PySQLite es una buena opción.
PySQLite es parte de la biblioteca estándar de Python desde la versión 2.5.
En este curso usaremos SQlite

>Conexión a la base de datos

Al realizar la conexión, si la base de datos no existe, entonces la crea. Siempre debe cerrarse la conexión al finalizar el uso, sino luego no se podrá seguir manipulandola.

```
import sqlite3    # Importamos el modulo
# Conexion a la DB
conexion = sqlite3.connect('ejemplo.db')
conexion.close()     # Cerramos la conexion
```

>Crear una tabla

Antes de ejecutar una consulta (query) en código SQL, tenemos que crear un cursor:

cursor: es un objeto que ayuda a ejecutar la consulta y obtener los registros de la base de datos. El cursor juega un papel muy importante en la ejecución de la consulta.

>Crear una tabla

```
import sqlite3
conexion = sqlite3.connect('ejemplo.db')

# Creamos el cursor
cursor= conexion.cursor()

# Creamos una tabla de usuarios con nombres, edades y emails
cursor.execute("CREATE TABLE IF NOT EXISTS usuario(nombre
VARCHAR(100),edad INTEGER,email VARCHAR(100))")
conexion.close()
```

>Inserción o carga de datos en una tab

Cargar un registro de datos

```
import sqlite3
conexion = sqlite3.connect('ejemplo.db')
cursor= conexion.cursor()
#Insertamos un registro en la tabla usuario
cursor.execute("INSERT INTO usuario VALUES
('Hector',27,'hector@ejemplo.com')")
# Guardamos los cambios haciendo un commit
conexion.commit()
conexion.close()
```

módulo 6.

* 2022 / 3° Cohorte.

>Cargar varios registros

Insertando varios registros con .executemany():

```
import sqlite3
conexion = sqlite3.connect('ejemplo.db')
cursor= conexion.cursor()
usuarios= [('Mario',51,'mario@ejemplo.com'),
           ('Mercedes',38,'mercedes@ejemplo.com'),
           ('Juan',19,'Juan@ejemplo.com')]
# Ahora utilizamos el metodo executemany() para insertar varios
cursor.executemany("INSERT INTO usuario VALUES(?,?,?)" ,usuarios)
# Guardamos los cambios haciendo commit
conexion.commit()
conexion.close()
```

>Lectura de datos en una tabla

Lectura de una tupla Recuperando el primer registro con .fetchone():

```
import sqlite3
conexion = sqlite3.connect('ejemplo.db')
cursor= conexion.cursor()

# Recuperamos los registros de la tabla usuario
cursor.execute("select * from usuario")

# Recorremos el 1er registro con fetchone que devuelve una tupla
usuario= cursor.fetchone()
conexion.close()
```

>Lectura múltiple

Recuperando varios registros con .fetchall():

```
import sqlite3
conexion = sqlite3.connect('ejemplo.db')
cursor= conexion.cursor()
cursor.execute("select * from usuario")

# Se recorren los registros con fetchall y se vuelvan a una lista
usuarios= cursor.fetchall()
for usuario in usuarios:
    print("Un usuario: ",usuario)
conexion.close()
```

>Actualizar/Modificar un dato en una tabla

Indicamos cuál campo queremos modificar, con qué valor lo haremos y a cuáles filas afectará el cambio:

```
import sqlite3
conexion = sqlite3.connect('ejemplo.db')
cursor= conexion.cursor()
# Actualizamos un registro de la tabla usuario
cursor.execute("UPDATE usuario SET edad=80 WHERE nombre=
'Juan'")
conexion.close()
```

>Eliminar un registro de una tabla

Indicamos cuál registro/fila queremos eliminar de acuerdo a alguna condición de alguno de los campos:

```
import sqlite3
conexion= sqlite3.connect('ejemplo.db')
cursor= conexion.cursor()
# Actualizamos un registro de la tabla usuario
cursor.execute("UPDATE usuario SET edad=80 WHERE nombre=
'Juan'")
conexion.close()
```

>Mostrar resultado ordenado

Con **SELECT** obtenemos todos los registros de la tabla y con **ORDER BY** los ordenamos por edad de menor a mayor

```
cursor.execute("SELECT * FROM usuario ORDER BY
edad")
personas= cursor.fetchall()
for persona in personas:
    print(persona)
```

módulo 6.

* 2022 / 3° Cohorte.

>Mostrar resultado ordenado

DESC nos permite ordenar en orden descendente

```
cursor.execute("SELECT * FROM usuario ORDER BY
edad DESC")
personas= cursor.fetchall()
for persona in personas:
    print(persona)
```

módulo 6.

1000 PROGRAMADORES

* 2022 / 3° Cohorte.

>COUNT

COUNT() nos permite contar cuantas veces se repite una búsqueda. Por ejemplo, en el código siguiente, cuanta la cantidad nombres guardados

```
def cuenta_nombre():
    cursor.execute(f"SELECT COUNT (nombre) FROM usuario")
    salida= cursor.fetchone()    # Devuelve una tupla
    return salida[0]             # Retorno sólo la cantidad
print(f"Actualmente existen {cuenta_nombre()} usuarios")
```

>WHERE

WHERE nos permite filtrar los resultados según el parámetro que necesitemos usar

```
def cuenta_nombre(nombre):
    cursor.execute(f"SELECT COUNT (nombre) FROM usuario WHERE
nombre= '{nombre}'")
    return cursor.fetchone()[0]

nombre= "Juan"
print(f"EL nombre {nombre} se repite {cuenta_nombre(nombre)}
veces")
```

>LIKE y %

LIKE se puede utilizar con los caracteres comodín, como por ejemplo el **%** que reemplaza a cualquier combinación de caracteres.

```
def empieza_con(inicio):
    cursor.execute(f"SELECT nombre, edad,email FROM usuario
WHERE nombre LIKE '{inicio}%' ")
    return cursor.fetchall()
inicial= "M"
print(f"Los nombres que inicial con  {inicial} son:")
personas= empieza_con(inicial)
for persona in personas:
    print(persona)
```

web: **http://milprogramadores.unsa.com.ar**

telegram: **https://t.me/milprogramadoressaltenios**

centro de ayuda: **http://ayudamilprogramadores.com/**