



Iniciación con Python

Clase 14 - “Fundamentos SQL”

¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase

Clase 13.

► Bases de Datos

1. Concepto y utilidad de los módulos en Python
2. Introducción a Bases de datos.
3. Idea de tabla, campo, índice, clave, etc.

Clase 14.

► Bases de Datos

1. Instalación y uso del módulo SQLite
2. Consultas SQL básicas desde Python con SQLite
3. Consultas SQL básicas: SELECT, INSERT, UPDATE, DELETE

Clase 15.

► SQLite

1. Definimos y creamos la base de datos
2. Agregamos a las funciones creadas antes de las consultas de SQL
3. Revisión del enunciado del TFI

Pero antes...



¡Resolvamos los “**Ejercicios prácticos**”
de la clase anterior!



Fundamentos SQL



¿Qué es SQLite y para qué sirve?



- SQLite es un módulo de Python para gestionar bases de datos de forma ligera y práctica.
- Es una base de datos embebida en un solo archivo.
- Viene integrada en Python.
- Permite usar desde Python comandos SQL para acceder a los datos.
- Proporciona un mecanismo para la persistencia de datos.



Ventajas y desventajas de SQLite

SQLite es ideal para sistemas como el inventario que estamos desarrollando, debido a su simplicidad y accesibilidad. No requiere instalación adicional ni configuración compleja. Es ideal para pequeños proyectos.

Pero no debemos olvidar que:

- Tiene limitaciones en la concurrencia y permite un solo proceso de escritura a la vez.
- No tiene un sistema de usuarios con roles o permisos avanzados.
- Su rendimiento disminuye a medida que el tamaño de la base de datos aumenta.
- Realizar backups en SQLite puede ser más complicado si la base de datos está en uso, ya que el acceso concurrente durante el proceso de respaldo puede corromper el archivo.



Consultas SQL



Las consultas SQL son instrucciones que enviamos a la base de datos para obtener, modificar o eliminar información. Desde Python podemos hacerlo mediante el módulo `sqlite3`.

Cada consulta SQL se compone de un comando o instrucción, como `SELECT`, `INSERT`, `UPDATE` o `DELETE`, que especifica una acción a realizar y puede incluir cláusulas adicionales para definir detalles de esa acción, como qué datos queremos ver o cuáles queremos cambiar.



Importar el módulo SQLite

Para acceder a las funciones de SQLite en Python, solo tenemos que importar el módulo `sqlite3` usando la misma sintaxis que empleamos para otros módulos.

Ejemplo:

```
import sqlite3                # Importar el módulo
```

Una vez que importamos `sqlite3`, podemos usar todas sus funciones y objetos para conectarnos a una base de datos, ejecutar consultas y almacenar o recuperar datos.



Conexión a la base de datos

Luego de importar el módulo, usamos la función `sqlite3.connect()` para conectarnos (o crear una base de datos nueva si no existe). Al conectar con la base de datos, creamos un "puente" entre nuestro programa y el archivo de base de datos.

Ejemplo:


```
import sqlite3    # Importar el módulo
conexion = sqlite3.connect("inventario.db") # Conectar a la base de datos (o crearla)
cursor = conexion.cursor()      # Crear un cursor para interactuar con la base de datos
```

El cursor nos permite ejecutar comandos SQL para interactuar con la base de datos, como crear tablas y gestionar los datos que vamos a guardar.

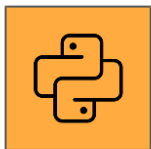


SELECT: Recuperar datos

La consulta SELECT nos permite obtener todos los registros de una tabla o sólo algunos en particular, según los filtros o condiciones que le indiquemos:




```
import sqlite3                                # Importar el módulo
conexion = sqlite3.connect("base_datos.db")    # Conectarse a la base de datos
cursor = conexion.cursor()                    # Crear un cursor
cursor.execute("SELECT * FROM Personas")        # Ejecutar la consulta SELECT
resultados = cursor.fetchall()                # Obtener todos los registros
for registro in resultados:                    # Mostrar los resultados
    print("Nombre:", registro[0], "Edad:", registro[1], "Ciudad:", registro[2])
conexion.close()                              # Cerrar la conexión
```

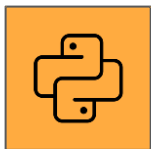


SELECT con WHERE

Usando la cláusula WHERE, podemos definir una condición o filtro. Supongamos que queremos ver sólo los registros de personas que viven en "Buenos Aires". En este caso, la consulta se ve así:




```
import sqlite3                                # Importar el módulo
conexion = sqlite3.connect("base_datos.db")    # Conectarse a la base de datos
cursor = conexion.cursor()                    # Crear un cursor
# Ejecutar la consulta SELECT:
cursor.execute("SELECT * FROM Personas WHERE ciudad = 'Buenos Aires'")
resultados = cursor.fetchall()                # Obtener todos los registros
for registro in resultados:                  # Mostrar los resultados
    print("Nombre:", registro[0], "Edad:", registro[1], "Ciudad:", registro[2])
conexion.close()                             # Cerrar la conexión
```



SELECT con campos específicos

Si en lugar de * ponemos el nombre de los campos que queremos recuperar, SELECT devuelve sólo estos datos. Por ejemplo, si solo queremos ver el nombre y la edad de cada persona usamos:




```
import sqlite3                                # Importar el módulo
conexion = sqlite3.connect("base_datos.db")    # Conectarse a la base de datos
cursor = conexion.cursor()                    # Crear un cursor
# Ejecutar la consulta SELECT, devolviendo solo las columnas nombre y edad:
cursor.execute("SELECT nombre, edad FROM Personas")
resultados = cursor.fetchall()                # Obtener todos los registros
for registro in resultados:                   # Mostrar los resultados
    print("Nombre:", registro[0], "Edad:", registro[1], "Ciudad:", registro[2])
conexion.close()                             # Cerrar la conexión
```



SELECT con ORDER BY

La cláusula ORDER BY ordena los resultados de una consulta alfabéticamente en orden creciente. Si usamos “DESC”, el orden se invierte. Podés probar el siguiente código:




```
import sqlite3                                # Importar el módulo
conexion = sqlite3.connect("base_datos.db")   # Conectarse a la base de datos
cursor = conexion.cursor()                    # Crear un cursor
# Ejecutar la consulta SELECT, ordenada por nombre de "Z" a "A":
cursor.execute("SELECT * FROM Personas ORDER BY nombre DESC;")
resultados = cursor.fetchall()                # Obtener todos los registros
for registro in resultados:                   # Mostrar los resultados
    print("Nombre:", registro[0], "Edad:", registro[1], "Ciudad:", registro[2])
conexion.close()                             # Cerrar la conexión
```



Agregando datos con INSERT

El comando INSERT permite agregar nuevos datos a una tabla. Es necesario especificar tanto los campos en los que queremos añadir datos como los valores que queremos almacenar en esos campos:



```
import sqlite3                                # Importar el módulo
conexion = sqlite3.connect("base_datos.db")    # Conectarse a la base de datos
cursor = conexion.cursor()                    # Crear un cursor
# Insertar un nuevo registro en la tabla Personas
cursor.execute("INSERT INTO Personas (nombre, edad, ciudad) VALUES ('Carlos',
27, 'Tucumán')")
conexion.commit()                             # Guardar los cambios
conexion.close()                              # Cerrar la conexión
```




La importancia de commit()

En SQLite (y en la mayoría de los sistemas de gestión de bases de datos), cuando llevás adelante operaciones como `cursor.execute` para insertar, actualizar o eliminar datos, dichas modificaciones no se guardan inmediatamente.. En su lugar, los cambios se mantienen en una **transacción**.


Una transacción es un conjunto de operaciones que se ejecutan de forma "*temporal*" hasta que decidís confirmarlas o descartarlas. SQLite mantiene estas operaciones en memoria (buffer) o en un archivo temporal y no las aplica al archivo de la base de datos hasta que se haga un **commit**.

Si no se hace el `commit()` y la conexión se cierra, SQLite deshace automáticamente los cambios pendientes (un rollback implícito), por lo que los datos **no se guardan**.



Actualizando datos con UPDATE

El comando UPDATE permite modificar datos que ya están guardados en una tabla. Podemos actualizar uno o varios campos de un registro específico sin necesidad de agregar un nuevo registro.




```
import sqlite3                                # Importar el módulo
conexion = sqlite3.connect("base_datos.db")    # Conectarse a la base de datos
cursor = conexion.cursor()                    # Crear un cursor
# Actualizar la edad de Ana en la tabla Personas
cursor.execute("UPDATE Personas SET edad = 24 WHERE nombre = 'Ana'")

conexion.commit()                             # Guardar los cambios
conexion.close()                             # Cerrar la conexión
```



Eliminando datos con DELETE

Finalmente, el comando DELETE en SQL nos permite eliminar registros de una tabla. Es importante recordar usar la cláusula WHERE para indicar la condición que deben cumplir los registros a eliminar. Sin ella, borramos toda la tabla.



```
import sqlite3                                # Importar el módulo
conexion = sqlite3.connect("base_datos.db")   # Conectarse a la base de datos
cursor = conexion.cursor()                    # Crear un cursor
# Eliminar los registros con nombre "José" en la tabla Personas:
cursor.execute("DELETE FROM Personas WHERE nombre = 'José'")

conexion.commit()                             # Guardar los cambios
conexion.close()                              # Cerrar la conexión
```

¡Vamos a la práctica!



Ejercicios prácticos



Optativos | No entregables

Agregar múltiples registros

Crea un programa en Python que inserte varios registros en la tabla Personas usando una lista de tuplas predefinida. Cada tupla debe contener un nombre, una edad y una ciudad.

Usá un bucle para recorrer la lista e insertar cada persona en la base de datos. La lista debe tener al menos cinco personas nuevas y al finalizar el programa deben mostrarse todos los registros en la tabla Personas.

La lista de tuplas tiene una estructura como la siguiente:

```
nuevas_personas = [  
    ("Esteban", 32, "Mar del Plata"),  
    ("Valeria", 27, "Bahía Blanca"),  
    ("Fernando", 41, "Rosario"),  
    ("Carolina", 29, "La Plata"),  
    ("Juan", 35, "Córdoba")  
]
```

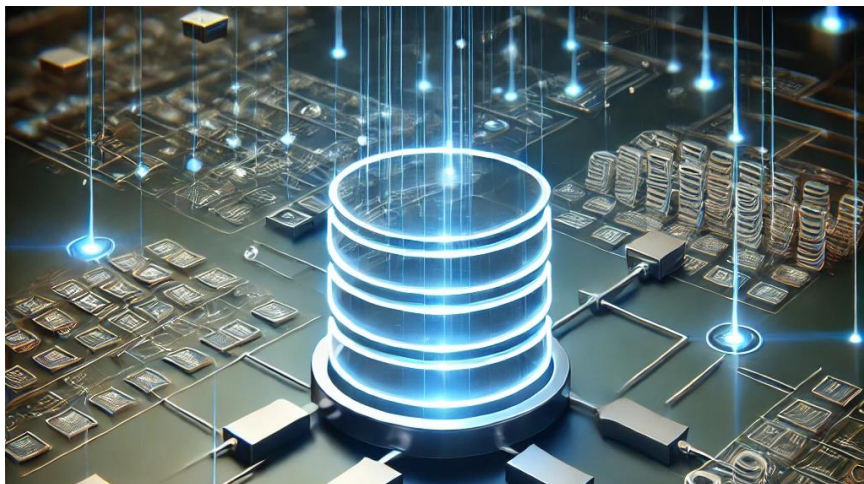


Ejercicios prácticos



Optativos | No entregables

Eliminación de registros según una condición



Desarrollá un programa en Python que elimine todos los registros en la tabla Personas donde la edad sea menor a 25 años

Al final del programa, mostrará todos los registros restantes para confirmar que se han eliminado correctamente aquellos que cumplen con la condición establecida.