«Talento Tech»

Iniciación a la Programación con Python

CLASE 14

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 14 | Fundamentos SQL

Temario:

- Instalación y uso del módulo SQLite
- Consultas SQL básicas desde Python con SQLite
- Consultas SQL básicas: SELECT, INSERT, UPDATE, DELETE

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Módulo SQLite.

Para comenzar con los fundamentos de SQL en esta clase, retomaremos el concepto de módulo en Python que vimos en la clase anterior y veremos cómo utilizarlo de forma tal que nos ayude a gestionar bases de datos de manera eficiente. Hoy trabajaremos con SQLite.

¿Qué es SQLite y para qué sirve?

SQLite es un módulo en Python que nos permite crear y trabajar con bases de datos directamente desde nuestros programas. Su gran ventaja consiste en que es una base de datos ligera y embebida: todo lo que necesitamos se guarda en un solo archivo. Esto hace que sea ideal para proyectos pequeños o medianos, como el sistema de inventario que estamos desarrollando.

A diferencia de otras bases de datos que requieren una instalación y configuración más complejas, SQLite ya viene integrado en Python, por lo que no necesitamos instalar nada. Esto significa que desde el primer momento podemos empezar a crear y manejar bases de datos sin complicaciones.

¿Qué es SQL?

SQL, o Structured Query Language (Lenguaje de Consulta Estructurada), es un lenguaje que se utiliza para gestionar y manipular eficientemente bases de datos. Nos permite realizar operaciones como crear tablas, insertar, actualizar y borrar datos, además de hacer consultas para encontrar información específica. SQL actúa como un puente entre quien interactúa con el software y la base de datos, ya que con comandos sencillos podemos organizar, filtrar y recuperar datos de acuerdo a nuestras necesidades. En el caso de SQLite, SQL es el lenguaje que usamos para interactuar con el archivo de base de datos desde nuestro programa en Python.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

¿Cómo accedemos a las funciones y objetos de SQLite?

Para acceder a las funciones de SQLite en Python, sólo tenemos que importar el módulo sqlite3 usando la misma sintaxis que empleamos para otros módulos. Una vez que importamos sqlite3, podemos usar todas sus funciones y objetos.

```
import sqlite3
```

Al importar sqlite3, obtenemos una serie de herramientas para ejecutar consultas y almacenar o recuperar datos. Aquí algunos de los objetos y funciones más importantes que nos proporciona:

Conexión a la base de datos: Usamos la función sqlite3.connect() para conectarnos o crear una base de datos nueva si no existe. Al hacerlo, creamos un "puente" entre nuestro programa y el archivo.

Ejecutar comandos SQL: A través del objeto cursor, que obtenemos con el método .cursor() de la conexión, podemos ejecutar instrucciones SQL. Este cursor es como un "control remoto" para interactuar con la base de datos, permitiéndonos realizar tareas como crear tablas, insertar datos y hacer consultas.

Cerrar la conexión: Una vez que terminamos de trabajar, es una buena práctica cerrar la conexión usando el método .close(). Esto asegura que todos los cambios se guarden correctamente y que el archivo de base de datos esté disponible para otros programas o personas usuarias.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Ejemplo básico de conexión a una base de datos con SQLite

Supongamos que queremos crear una conexión para almacenar nuestro inventario de productos. El primer paso luego de hacerlo es obtener el cursor.

```
import sqlite3

# Conectar a la base de datos (o crearla si no existe)
conexion = sqlite3.connect("inventario.db")

# Crear un cursor para interactuar con la base de datos
cursor = conexion.cursor()

# Aquí podemos agregar las instrucciones SQL para crear tablas,
insertar datos, etc.

# Cerrar la conexión cuando terminamos
conexion.close()
```

En este código, la línea sqlite3.connect("inventario.db") establece una conexión con el archivo de base de datos llamado "inventario.db". Si este archivo no existe, se crea automáticamente. A través de conexion.cursor(), obtenemos un cursor que nos permite ejecutar comandos SQL para interactuar con la base de datos, como crear tablas y gestionar los datos que vamos a guardar. Finalmente, al llamar a conexion.close(), cerramos la conexión para que la información quede guardada correctamente.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Consultas SQL básicas desde Python con SQLite.

Las consultas SQL son instrucciones que enviamos para obtener, modificar o eliminar información. En Python, podemos realizar consultas SQL a través del módulo sqlite3, lo cual nos permite interactuar con los datos de forma estructurada y precisa. 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. Para entender cómo se usan, vamos a ver primero el comando SELECT, que nos permite consultar datos almacenados.

SELECT.

El comando SELECT es fundamental en SQL, ya que nos permite recuperar datos específicos de una tabla, solicita la información que indiques. La consulta básica con SELECT nos permite obtener todos los registros de una tabla o sólo algunos en particular, según los filtros o condiciones que le consultemos.

Ejemplo básico de SELECT

Imaginemos que tenemos una tabla llamada Personas con los campos nombre, edad y ciudad, tal como se ve en la figura siguiente:

Campo	Tipo de Dato	Descripción
nombre	TEXT	Almacena el nombre de la persona
edad	INTEGER	Almacena la edad de la persona en años
ciudad	TEXT	Almacena el nombre de la ciudad de residencia

Y que el contenido de la tabla es:

Nombre	Edad	Ciudad
Ana	23	Buenos Aires
Luis	35	Rosario
Clara	29	Mendoza
Pedro	40	Córdoba
Marta	31	La Plata
José	22	San Juan
Lucía	26	Buenos Aires
Diego	30	Santa Fe
Sofía	28	Mendoza
Joaquín	34	Rosario

Para ver todos los datos en esa tabla, escribimos una consulta simple de SELECT para seleccionar y mostrar todas las filas:

SELECT * FROM Personas;

En este caso, * significa "todos los campos". Así, si ejecutamos esta consulta, la base de datos nos devolverá cada registro de la tabla con todos los campos de cada persona. En Python, usando SQLite, podemos implementar esta consulta de la siguiente forma:

```
import sqlite3

# Conectarse a la base de datos y crear un cursor
conexion = sqlite3.connect("base_datos.db")
cursor = conexion.cursor()
```

```
# Ejecutar la consulta SELECT y obtener todos los registros
cursor.execute("SELECT * FROM Personas")
resultados = cursor.fetchall()

# Mostrar los resultados
for registro in resultados:
   print("Nombre:", registro[0], "Edad:", registro[1], "Ciudad:",
registro[2])

# Cerrar la conexión
conexion.close()
```

En este ejemplo, cursor.execute("SELECT * FROM Personas") envía la consulta SQL. El método fetchall() recupera todos los registros que cumplen con la consulta y luego los muestra en pantalla. Cada registro contiene todos los campos de una persona y los accedemos en el mismo orden en que están en la tabla. La salida por pantalla es esta:

```
Nombre: Ana Edad: 23 Ciudad: Buenos Aires
Nombre: Luis Edad: 35 Ciudad: Rosario
Nombre: Clara Edad: 29 Ciudad: Mendoza
Nombre: Pedro Edad: 40 Ciudad: Córdoba
Nombre: Marta Edad: 31 Ciudad: La Plata
Nombre: José Edad: 22 Ciudad: San Juan
Nombre: Lucía Edad: 26 Ciudad: Buenos Aires
Nombre: Diego Edad: 30 Ciudad: Santa Fe
Nombre: Sofía Edad: 28 Ciudad: Mendoza
Nombre: Joaquín Edad: 34 Ciudad: Rosario
```

SELECT con WHERE

Podemos hacer que nuestra consulta sea más específica usando la cláusula WHERE, que permite definir una condición o filtro. Supongamos que queremos ver sólo los registros de personas que viven en "Buenos Aires". En este caso, la consulta sería:

SELECT * FROM Personas WHERE ciudad = 'Buenos Aires';

Esto le indica a la base de datos que solo queremos ver los registros donde el campo ciudad es igual a "Buenos Aires". En Python, podríamos escribirlo así:

```
cursor.execute("SELECT * FROM Personas WHERE ciudad = 'Buenos Aires'")
resultados = cursor.fetchall()
```

Si aplicamos esto a un sistema de inventario como el del Trabajo Final Integrador, podríamos querer ver sólo los productos que pertenecen a la categoría "Lácteos". Para eso, imaginemos una tabla llamada Productos que tiene un campo categoria. La consulta para ver sólo los productos de la categoría "Lácteos" sería:

SELECT * FROM Productos WHERE categoria = 'Lácteos';

Este ejemplo ayuda a que los datos específicos de los productos sean fáciles de ubicar.

SELECT con columnas específicas

A veces no queremos ver todos los campos, sino solo algunos. En lugar de *, podemos especificar los nombres de los campos que queremos recuperar. Por ejemplo, si sólo queremos ver el nombre y la edad de cada persona, usamos:

SELECT nombre, edad FROM Personas;

Esta consulta nos devuelve únicamente los campos nombre y edad, ignorando el campo ciudad. En Python, el código se vería así:

```
cursor.execute("SELECT nombre, edad FROM Personas")
resultados = cursor.fetchall()
```

Para el Trabajo Final Integrador, si sólo nos interesa ver el nombre y el precio de cada producto en el inventario, la consulta sería:

SELECT nombre, precio FROM Productos;

Esto permite centrarnos en los campos específicos que necesitamos, optimizando la búsqueda.

SELECT con ORDER BY

La cláusula ORDER BY nos ayuda a ordenar los resultados de una consulta. Si queremos ver a las personas en orden alfabético según su nombre, agregamos ORDER BY nombre al final de la consulta:

SELECT * FROM Personas ORDER BY nombre;

Esto le indica a la base de datos que muestre los registros en orden ascendente (A a Z) por el campo nombre. Para ordenar en orden descendente, agregamos DESC al final:

SELECT * FROM Personas ORDER BY nombre DESC;

Si consideramos la tabla del ejemplo anterior y corremos el siguiente código Python

```
import sqlite3

# Conectarse a la base de datos y crear un cursor
conexion = sqlite3.connect("base_datos.db")
cursor = conexion.cursor()

# Ejecutar la consulta SELECT y obtener todos los registros
cursor.execute("SELECT * FROM Personas ORDER BY nombre DESC;")
resultados = cursor.fetchall()

# Mostrar los resultados
for registro in resultados:
   print("Nombre:", registro[0], "Edad:", registro[1], "Ciudad:",
registro[2])

# Cerrar la conexión
conexion.close()
```

Obtendremos la siguiente salida en pantalla:

```
Nombre: Sofía Edad: 28 Ciudad: Mendoza
Nombre: Pedro Edad: 40 Ciudad: Córdoba
Nombre: Marta Edad: 31 Ciudad: La Plata
Nombre: Luis Edad: 35 Ciudad: Rosario
Nombre: Lucía Edad: 26 Ciudad: Buenos Aires
```

```
Nombre: José Edad: 22 Ciudad: San Juan
Nombre: Joaquín Edad: 34 Ciudad: Rosario
Nombre: Diego Edad: 30 Ciudad: Santa Fe
Nombre: Clara Edad: 29 Ciudad: Mendoza
Nombre: Ana Edad: 23 Ciudad: Buenos Aires
```

En el contexto del inventario, podemos ordenar los productos de menor a mayor precio con:

SELECT * FROM Productos ORDER BY precio;

De esta manera, al combinar SELECT con otras cláusulas como WHERE y ORDER BY, tenemos un control completo sobre cómo acceder y organizar los datos, ajustando los resultados a nuestras necesidades. Este es el primer paso para realizar consultas SQL en Python con SQLite y será muy útil al momento de gestionar la información en el sistema de inventario.

INSERT.

El comando INSERT en SQL nos permite agregar nuevos datos a una tabla dentro de la base de datos. En el caso de nuestra tabla Personas, podemos usar INSERT para agregar información de una persona nueva a los registros ya existentes. Cuando hacemos un INSERT, especificamos tanto los campos en los que queremos añadir datos como los valores que queremos almacenar en esos campos.

Para agregar un registro en la tabla Personas, la consulta SQL básica tiene la forma:

Buenos Aires aprende
Agencia de Habilidades para el Futuro

INSERT INTO Personas (nombre, edad, ciudad) VALUES ('Carlos', 27, 'Tucumán');

Esta instrucción le indica a la base de datos que queremos insertar un nuevo registro en la tabla Personas y que el valor de nombre será “Carlos”, el de edad será 27, y el de ciudad será “Tucumán”. Cada campo especificado en VALUES debe coincidir con el orden y el tipo de datos de los campos de la tabla.

En Python, podemos utilizar esta instrucción INSERT para agregar un registro desde nuestro programa, aprovechando el módulo sqlite3. Veamos un ejemplo donde añadimos una persona nueva a la tabla Personas.

```
import sqlite3

# Conectarse a la base de datos y crear un cursor
conexion = sqlite3.connect("base_datos.db")
cursor = conexion.cursor()

# Insertar un nuevo registro en la tabla Personas
cursor.execute("INSERT INTO Personas (nombre, edad, ciudad) VALUES
('Carlos', 27, 'Tucumán')")

# Guardar los cambios
conexion.commit()

# Cerrar la conexión
conexion.close()
```

En este código, cursor.execute(...) ejecuta la instrucción SQL de INSERT, y luego conexion.commit() guarda el nuevo registro en la base de datos de manera permanente. Finalmente, cerramos la conexión con conexion.close().

Imaginemos ahora que estamos usando este mismo comando en el contexto de un inventario. Supongamos que estamos insertando un producto nuevo en una tabla Productos con los campos nombre, cantidad, precio y categoria. La instrucción INSERT sería similar:

INSERT INTO Productos (nombre, cantidad, precio, categoria) VALUES ('Leche', 50, 1.5, 'Lácteos');

```
# Insertar un producto nuevo en la tabla Productos
cursor.execute("INSERT INTO Productos (nombre, cantidad, precio,
categoria) VALUES ('Leche', 50, 1.5, 'Lácteos')")
```

El comando INSERT es indispensable para agregar información a nuestras tablas de manera organizada y nos permite actualizar y enriquecer la base de datos con datos nuevos a medida que los necesitemos. Cada registro que añadimos sigue la estructura definida por los campos de la tabla, lo que facilita el acceso y la manipulación de la información en el futuro.

UPDATE.

El comando UPDATE en SQL nos permite modificar datos que ya están guardados en una tabla de la base de datos. Con UPDATE, podemos actualizar uno o varios campos de un registro específico sin necesidad de agregar un nuevo registro. Esto es útil cuando queremos corregir o ajustar la información que ya existe.

Para usar UPDATE, especificamos el nombre de la tabla, los campos que queremos modificar y los nuevos valores que deseamos asignar. Además, es común usar la cláusula WHERE para indicar qué registros deben actualizarse, evitando que otros se modifiquen por error.

Supongamos que queremos actualizar la edad de una persona llamada "Ana" en la tabla Personas y cambiarla a 24. La consulta SQL para hacer esto sería:

Buenos Aires aprende
Agencia de Habilidades para el Futuro

UPDATE Personas SET edad = 24 WHERE nombre = 'Ana';

Esta consulta le dice a la base de datos que busque el registro donde el nombre es "Ana" y cambie el valor del campo edad a 24. La cláusula WHERE es crucial aquí porque indica qué registro modificar. Si omitimos WHERE, se cambiará la edad en todos los registros de la tabla Personas.

En Python, podemos usar el módulo sqlite3 para ejecutar la instrucción UPDATE y actualizar un registro en la tabla Personas. Veamos cómo sería el código para realizar esta modificación.

```
import sqlite3

# Conectarse a la base de datos y crear un cursor
conexion = sqlite3.connect("base_datos.db")
cursor = conexion.cursor()

# Actualizar la edad de Ana en la tabla Personas
cursor.execute("UPDATE Personas SET edad = 24 WHERE nombre = 'Ana'")

# Guardar los cambios
conexion.commit()

# Cerrar la conexión
conexion.close()
```

Aquí, cursor.execute(...) ejecuta la consulta SQL para actualizar el registro. conexion.commit() guarda los cambios de forma permanente y luego cerramos la conexión.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

En nuestro Trabajo Final Integrador trabajaremos con una tabla Productos: imaginemos que queremos actualizar la cantidad de un producto en el inventario. Supongamos además que queremos cambiar la cantidad del producto "Leche" a 60 unidades. La consulta SQL sería:

UPDATE Productos SET cantidad = 60 WHERE nombre = 'Leche';

Y en Python, el código para realizar esta actualización sería:

```
# Actualizar la cantidad de un producto en la tabla Productos
cursor.execute("UPDATE Productos SET cantidad = 60 WHERE nombre =
'Leche'")
```

El comando UPDATE nos permite realizar cambios puntuales en la información sin necesidad de eliminar ni volver a agregar registros, optimizando la gestión de la base de datos. La combinación con WHERE hace que esta instrucción sea muy versátil y segura, al poder aplicar las modificaciones solo a los registros específicos que queremos actualizar.

DELETE.

El comando DELETE en SQL nos permite eliminar registros de una tabla. Con DELETE, podemos borrar uno o varios registros específicos, según los criterios que definamos. Esto es útil cuando queremos quitar información que ya no interesa o que fue ingresada por error.

Para usar DELETE, indicamos el nombre de la tabla y, generalmente, agregamos una cláusula WHERE para especificar qué registros deben eliminarse. La cláusula WHERE es importante aquí porque, si no la incluimos, todos los registros de la tabla serían eliminados.

Imaginemos que queremos eliminar el registro de una persona llamada "José" en la tabla Personas. La consulta SQL para hacer esto es:

DELETE FROM Personas WHERE nombre = 'José';

Esta instrucción le indica a la base de datos que busque el registro donde el nombre es "José" y lo elimine de la tabla. La cláusula WHERE asegura que solo eliminemos el registro correspondiente a "José".

Podemos usar el módulo sqlite3 en Python para ejecutar la instrucción DELETE y eliminar un registro en la tabla Personas. A continuación, vemos el código que eliminaría el registro de "José" de la tabla.

```
import sqlite3

# Conectarse a la base de datos y crear un cursor
conexion = sqlite3.connect("base_datos.db")
cursor = conexion.cursor()

# Eliminar el registro de José en la tabla Personas
cursor.execute("DELETE FROM Personas WHERE nombre = 'José'")

# Guardar los cambios
conexion.commit()

# Cerrar la conexión
conexion.close()
```

En este código, cursor.execute(...) ejecuta la instrucción DELETE, mientras que conexion.commit() guarda los cambios para que el registro sea eliminado de forma permanente. Por último, cerramos la conexión.

En un sistema de inventario como el del Trabajo Final Integrador, podríamos querer eliminar un producto que ya no está en stock o que ha sido descontinuado. Supongamos que queremos borrar el producto "Jugo" de la tabla Productos. La instrucción SQL sería:

DELETE FROM Productos WHERE nombre = 'Jugo';

Y en Python, el código para eliminar este producto se vería así:

```
# Eliminar el producto Jugo de la tabla Productos
cursor.execute("DELETE FROM Productos WHERE nombre = 'Jugo'")
```

El comando DELETE nos da un control preciso sobre la eliminación de datos. Al combinarlo con WHERE, nos aseguramos de que sólo los registros específicos que queremos eliminar sean afectados, manteniendo la integridad de la información que sigue siendo útil. Esto es importante para la gestión de un inventario o en cualquier sistema donde los datos cambian frecuentemente.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Ejercicios prácticos:

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. Usa 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")
]
```

Eliminación de registros por condición

Desarrolla un programa en Python que elimine todos los registros en la tabla Personas donde la edad sea menor a 25 años. Usa un bucle para consultar y eliminar cada registro que cumpla con esta condición. Al final del programa, muestra todos los registros restantes para confirmar que se han eliminado correctamente los registros que cumplen la condición.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad