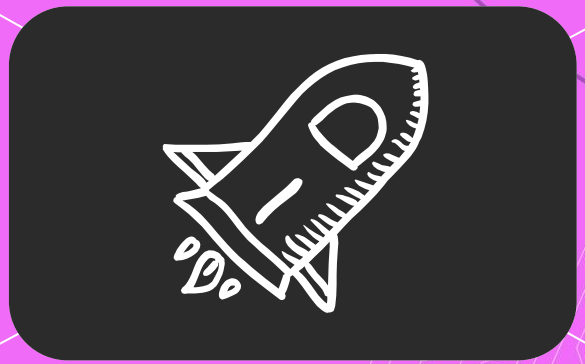
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

Semana 8
¡Bienvenidos!



Temas de hoy

1. Integrando SQL y Python
2. Ejecutando consultas Desde Python.
3. Ejercicios prácticos usando Bases de datos y Python.

1

Integrando SQL y Python

La integración de Python y SQL permite combinar las fortalezas de ambos lenguajes para realizar tareas de manipulación de datos de forma rápida y eficiente.

Python se utiliza para realizar tareas de preprocesamiento de datos, análisis estadístico y visualización, mientras que SQL se utiliza para acceder a datos almacenados en la base de datos.

🗄️👉

Integrando SQL y Python

Introducción

En el mundo de la programación, trabajar con bases de datos es una tarea fundamental. Las bases de datos SQL son una herramienta muy útil para almacenar grandes cantidades de información y para poder acceder a ella de manera estructurada.
En Python, trabajar con bases de datos SQL es esencial en el ámbito informático, ya que permite interactuar con bases de datos relacionales y realizar operaciones CRUD (Create, Read, Update, Delete).

Trabajar con bases de datos en Python se puede hacer a través de varias librerías, siendo las más conocidas SQLite, MySQL y PostgreSQL.

Ventajas de la integración de Python y SQL

La integración de Python y SQL ofrece las siguientes ventajas:

Eficiencia: Python y SQL son lenguajes muy eficientes en términos de rendimiento. La integración permite aprovechar esta eficiencia para realizar tareas de manipulación de datos de forma rápida y eficiente.
Flexibilidad: Python es un lenguaje muy flexible, al integrarse con SQL se puede aprovechar esta flexibilidad para realizar tareas de manipulación de datos de forma flexible.
Reutilización: La integración de ambos lenguajes permite reutilizar código Python para realizar tareas de manipulación de datos.

Integrando SQL y Python

Cómo integrar Python y SQL

Para integrar Python y SQL, es necesario instalar las bibliotecas adecuadas. **Para Python, existen varias bibliotecas que permiten acceder a bases de datos SQL**. Las bibliotecas más populares son:

sqlite3: Esta biblioteca permite acceder a bases de datos SQLite, que son bases de datos relacionales de código abierto.

mysql-connector-python: Esta biblioteca permite acceder a bases de datos MySQL, proporciona una API consistente para varias versiones de MySQL.

pyodbc: Esta biblioteca permite acceder a bases de datos SQL Server, que son bases de datos relacionales de Microsoft.

psycopg2: Para trabajar con bases de datos PostgreSQL, se puede utilizar la psycopg2biblioteca, un adaptador popular para Python y PostgreSQL.

Una vez instaladas las librerías adecuadas, se puede comenzar a integrar Python y SQL

Integrando SQL y Python

Gestor de paquetes pip

Pip es un gestor de paquetes muy popular en el universo Python, que nos permite realizar la gestión de paquetes y librerías de Python de forma muy sencilla en nuestro sistema Linux.
Con Pip, podemos instalar, actualizar, eliminar y buscar paquetes de Python, y nos proporciona una forma conveniente de gestionar dependencias y versiones entre diversas librerías de Python que tengamos en nuestro sistema.
Los paquetes Python instalados mediante pip proceden del siguiente repositorio:

https://pypi.org/

Para instalar el gestor de paquetes PIP hay que proceder del siguiente modo en función del sistema operativo.
GNU-Linux : Se ejecuta el siguiente comando en la terminal

```
sudo apt install python3-pip
```

Windows:
Abrimos el navegador web y accedemos a bootstrap.pypa.io/get-pip.py para descargar el fichero get-pip.py.
Hacemos clic derecho en el enlace y seleccionamos “Guardar como” y lo guardamos en cualquier carpeta de la pc, o en la carpeta de “Descargas”. Abrimos la terminal y navegamos hasta el archivo get-pip.py
Ejecutamos el siguiente comando:

```
python3 get-pip.py
```

Luego de instalar, verificamos que está correctamente instalado ejecutando en la terminal:

```
pip --version
```

Integrando SQL y Python

Conexión a Base de datos usando SQLite3

SQLite es un motor de base de datos liviano, autónomo y sin servidor que se usa ampliamente en aplicaciones donde no se requiere un servidor de base de datos completo.
El módulo integrado de Python **SQLite3** proporciona una interfaz fácil de usar para trabajar con bases de datos SQLite.
Para poder utilizar SQLite3 integrado con Python, primero tenemos que instalar el paquete usando pip. Ejecutamos en la terminal:

```
pip install pysqlite3
```

Una vez instalado el paquete tenemos que importarlo para poder conectarnos a la base de datos.
Creamos un archivo *.py* e importamos el paquete, para conectarnos utilizamos el comando connect. Nuestro código debería quedar de la siguiente forma:

```
import sqlite3

#Conectarnos a la base de datos
con = sqlite3.connect("data.db")
```

Integrando SQL y Python

PostgreSQL

Conexión a Base de datos usando psycopg2

Psycopg es el adaptador de base de datos PostgreSQL más popular para Python.
Fue diseñado para aplicaciones con múltiples subprocesos que crean y destruyen muchos cursores y realizan una gran cantidad de "INSERT" o "UPDATE" simultáneos.
El módulo integrado de Python **psycopg2** proporciona una interfaz fácil de usar para trabajar con bases de datos PostgreSQL.
Para poder utilizar PostgreSQL integrado con Python, primero tenemos que instalar el paquete usando pip. Ejecutamos en la terminal:

```
pip install psycopg2
```

Y, una vez instalado el paquete, nuevamente tenemos que importarlo para poder conectarnos a la base de datos.
Creamos un archivo .py e importamos el paquete, para conectarnos utilizamos el comando connect(), que crea una nueva sesión de base de datos y devuelve una nueva instancia de conexión. Nuestro código debería quedar de la siguiente forma:

```
import psycopg2

#Conectarnos a la base de datos
conn = psycopg2.connect(database = "nombredb",
                        user = "root",
                        host= 'localhost',
                        password = "rootpass",
                        port = 3306)
```

2

Ejecución de consultas desde Python

Ya aprendimos cómo integrar SQL con distintos motores de bases de datos en nuestros programas hechos en Python, ahora veamos cómo ejecutar consultas SQL dependiendo el motor de bases de datos que estemos utilizando 👉

Ejecución de consultas desde Python

SQLite

Ejecución de consultas en SQLite3

- **Creación de una tabla**:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print("¡Conexión exitosa de base de datos!");

conn.execute('''CREATE TABLE PERSONA
         (ID INT PRIMARY KEY NOT NULL,
          NOMBRE TEXT NOT NULL,
          EDAD INT NOT NULL,
          DIRECCION CHAR(50));''')
print("¡Tabla creada exitosamente!");

conn.close()
```

- **Insertar datos en una tabla de SQLite con Python**:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print("¡Conexión exitosa de base de datos!");

conn.execute("INSERT INTO PERSONA
            (ID,NOMBRE,EDAD,DIRECCION) \
            VALUES (1,'Pablo',32,'Av. Chaco 123')");

conn.execute("INSERT INTO PERSONA
            (ID,NOMBRE,EDAD,DIRECCION) \
            VALUES (2, 'Ana', 25, 'Av. Nueva 123')");

conn.commit()
print("¡Registros guardados exitosamente!");
conn.close()
```

Ejecución de consultas desde Python

SQLite

- Consultar datos de una tabla en SQLite con Python:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print("¡Conexión exitosa de base de datos!");

cursor = conn.execute("SELECT id, nombre,
                      direccion from PERSONA")
for row in cursor:
   print("ID = ", row[0])
   print("NOMBRE = ", row[1])
   print("DIRECCION = ", row[2], "\n")

print("Operación realizada exitosamente.");
conn.close()
```

- Actualizar datos de una tabla en SQLite con Python

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print("¡Conexión exitosa de base de datos!");

conn.execute("UPDATE PERSONA set DIRECCION = "Calle 3"
              where ID = 5")
conn.commit()
print("Filas actualizadas :", conn.total_changes)

cursor = conn.execute("SELECT id, nombre, direccion
                       from PERSONA")
for row in cursor:
   print("ID = ", row[0])
   print("NOMBRE = ", row[1])
   print("DIRECCION = ", row[2], "\n")
print("Operación realizada exitosamente.");
conn.close()
```

Ejecución de consultas desde Python

SQLite

- **Eliminar datos de una tabla en SQLite con Python**:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print("¡Conexión exitosa de base de datos!");

conn.execute("DELETE from PERSONA where ID = 2;")
conn.commit()
print("Filas actualizadas :", conn.total_changes)

cursor = conn.execute("SELECT id, nombre,
                      direccion from PERSONA")
for row in cursor:
   print("ID = ", row[0])
   print("NOMBRE = ", row[1])
   print("DIRECCION = ", row[2])

print("Operación realizada exitosamente.");
conn.close()
```

Se puede consultar más info sobre el uso de SQLite3 en la documentación de python:

https://docs.python.org/es/3.8/library/sqlite3.html

Ejecución de consultas desde Python

PostgreSQL

Ejecución de consultas con psycopg2

- **Creación de una tabla con psycopg2 y Python**:

```
#!/usr/bin/python

import psycopg2

cur = conn.cursor()
cur.execute("""CREATE TABLE PERSONA(ID INT
            PRIMARY KEY NOT NULL,
            NOMBRE TEXT NOT NULL,
            EDAD INT NOT NULL,
            DIRECCION CHAR(50));""")
conn.commit()

cur.close()
conn.close()
```

- **Insertar datos en una tabla con psycopg2 y Python**:

```
#!/usr/bin/python

import psycopg2

cur = conn.cursor()

conn.execute("INSERT INTO PERSONA
            (ID,NOMBRE,EDAD,DIRECCION) \
            VALUES (1,'Pedro',32,'Av. Chaco 123')");

conn.execute("INSERT INTO PERSONA
            (ID,NOMBRE,EDAD,DIRECCION) \
            VALUES (2, 'Mariana', 25, 'Av. Nueva 123')");


conn.commit()
cur.close()
conn.close()
```

Ejecución de consultas desde Python

PostgreSQL

- Consultar datos de una tabla con psycopg2 y Python:

```
#!/usr/bin/python

import psycopg2

cur = conn.cursor()

cur.execute('SELECT * FROM PERSONA ORDER BY NOMBRE;')

rows = cur.fetchall()

conn.commit()
conn.close()

for row in rows:
    print(row)
```

- Actualizar datos de una tabla con psycopg2 y Python:

```
#!/usr/bin/python

import psycopg2

cur = conn.cursor()

cur.execute("UPDATE PERSONA SET DIRECCION = "Calle 123"
WHERE NOMBRE = 'Pedro';")

conn.commit()

conn.close()
```

Ejecución de consultas desde Python

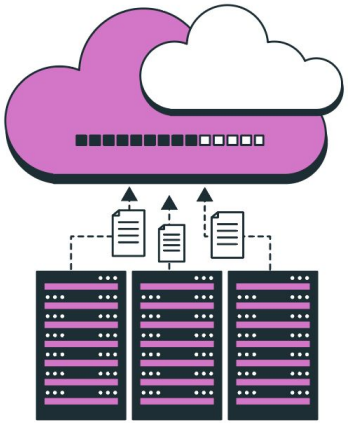
PostgreSQL

- **Eliminar datos de una tabla con psycopg2 y Python**:

```
#!/usr/bin/python

import psycopg2

cur = conn.cursor()

cur.execute("""DELETE from PERSONA WHERE
           NOMBRE = 'Pedro'""");

conn.commit()

cur.close()
```

Se puede consultar más info sobre el uso de la librería psycopg2 en la documentación que consta en el link:

https://www.psycopg.org/docs/



3

Ejercicios Prácticos

Veamos ahora cómo aplicar lo que vimos de bases de datos integrado con Python como propuesta de solución para situaciones que pueden aplicarse en la vida cotidiana.
En los ejemplos que veremos a continuación, se incluirán temas de bases de datos y programación que vimos anteriormente 👉

Agenda de Contactos

Se desea generar un pequeño sistema para gestionar los contactos en una Agenda.
Se debe almacenar para cada contacto el nombre, apellido, el teléfono y el email. Además, deberá mostrar un menú con las siguientes opciones:

- Añadir contacto
- Borrar contacto
- Listar contactos
- Buscar contacto
- Editar contacto

Tablas de Base de Datos

```
CREATE TABLE agenda(
id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
nombre VARCHAR(20) NOT NULL,
apellido VARCHAR(20) NOT NULL,
telefono VARCHAR(14) NOT NULL,
email VARCHAR(20) NOT NULL)
```

- Ver resolución usando sqlite3
- Ver resolución usando sqlite3 y Tkinter
  Archivo BaseDatos

Menú Restaurante

Se desea generar un pequeño sistema para gestionar los platos del menú de un restaurante.
Para la base de datos crearemos dos tablas, teniendo en cuenta el esquema que figura a la derecha.
Para el código, necesitaremos crear los métodos para conexión de bases de datos, una función llamada agregar_plato() que muestre al usuario las categorías disponibles y le permita escoger una (escribiendo un número). Luego le pedirá introducir el nombre del plato y lo añadirá a la base de datos, teniendo en cuenta que la categoría del plato concuerde con el id de la categoría y que el nombre del plato no pueda repetirse (no es necesario comprobar si la categoría realmente existe, en ese caso simplemente no se insertará el plato).
Crearemos una función llamada mostrar_menu() que muestre el menú con todos los platos de forma ordenada: los primeros, los segundos y los postres. Optativamente se puede adornar la forma en que muestra el menú por pantalla.

Tablas de Base de Datos

```sql
CREATE TABLE categoria(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre VARCHAR(100) UNIQUE NOT NULL)
```

```sql
CREATE TABLE plato(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre VARCHAR(100) UNIQUE NOT NULL,
    categoria_id INTEGER NOT NULL,
    FOREIGN KEY(categoria_id) REFERENCES categoria(id))
```

- Ver resolución usando sqlite3
- Ver resolución usando psycopg
- Ver resolución usando sqlite3 y Tkinter

Lo que vimos hoy 😉

Hasta acá pudimos ver como implementar propuestas de solución usando Python y SQL, además pudimos ver la aplicación de todo lo que avanzamos hasta aquí en ejercicios con Tkinter y las distintas librerías de SQL.

Pudimos conocer un poco más sobre cómo funcionan los paquetes y librerías aplicado a las bases de datos, además de ver cómo escribir código en SQL dependiendo de la librería y motor de bases de datos que seleccionamos para nuestra propuesta de solución.

Vimos de manera práctica, de qué manera podemos implementar nuestros programas utilizando Python en la programación y distintos motores de bases de datos.

Con esto se proporcionó una visión integral sobre cómo implementar y gestionar bases de datos utilizando nuestros programas en Python según las necesidades que se requieran.

¡Nos vemos En la próxima clase!