MÓDULO 4. ACCESO A LA INFORMACIÓN DE BASES DE DATOS

Loreto Pelegrín Castillo

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Región de Murcia
Sef Servicio Regional DE EMPLEO y FORMACIÓN
AVANZA

Índice:

1

Introducción

1. Introducción

- En esta unidad abordaremos como guardar información en bases de datos. Estas son muy importantes para crear aplicaciones mas completas, puesto que permiten guardar conjuntos de datos relacionados entre sí de manera mas estructurada y fácil de manejar.

2

Bases de datos

2. Bases de datos

- Una **base de datos** es un conjunto de datos pertenecientes a un mismo ámbito y que se encuentran guardados en el ordenador o en un entorno cloud.

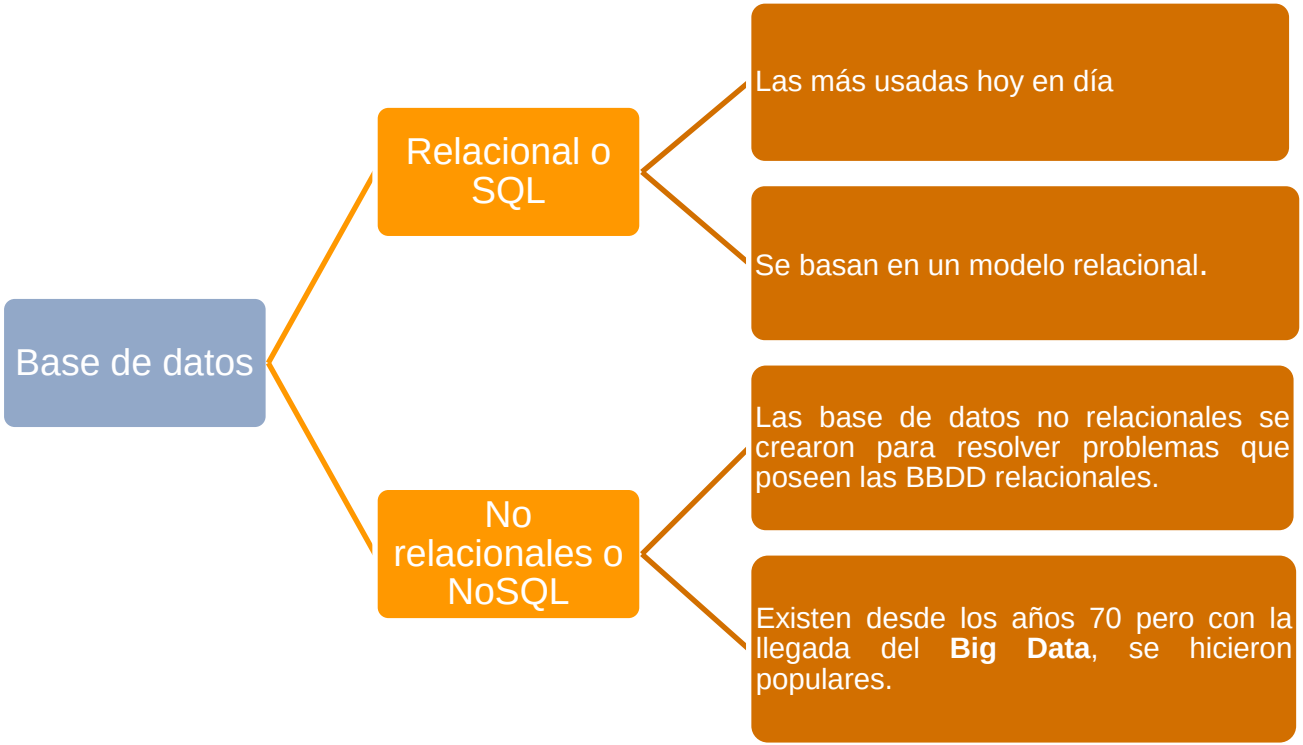
- Estos datos suelen estar almacenados siguiendo una estructura o esquema, que veremos más adelante, que facilita su posterior acceso, consulta o manipulación.

2. Bases de datos

- El **sistema gestor de bases de datos (SGBD)** es un software especifico que nos permite interactuar con las bases de datos para crear y manipular estas de forma rápida y eficiente.

- Algunos ejemplos: MySQL, PostgreSQL, Orable, MongoDB, …

2. Bases de datos

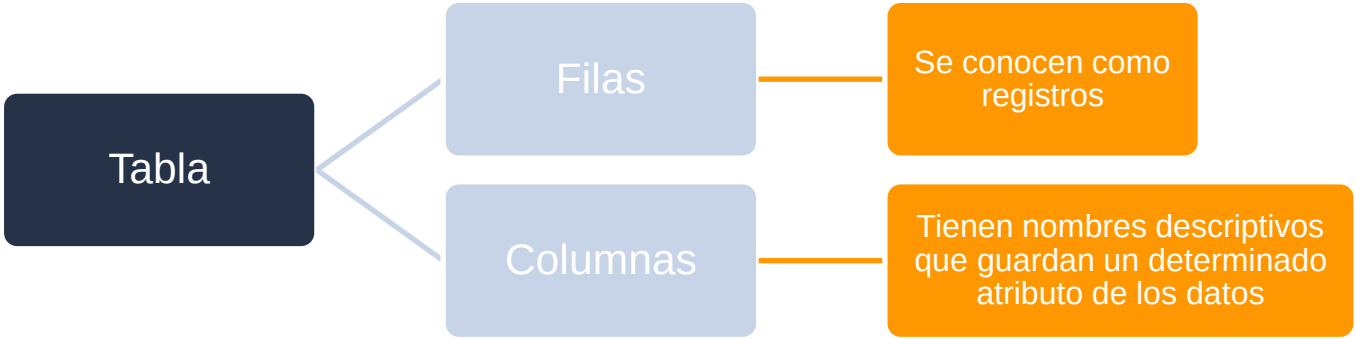


2.1. Bases de datos relacionales o SQL

- Las bases de datos relacionales se basan en un sistema relacional, es decir los datos se organizan en **tablas relacionadas** entre sí, por lo que podemos acceder a los datos en relación a otros datos de la propia base de datos que pertenece a otra tabla.

2.1.1. Tabla

- Una **tabla** en base de datos es un tipo de modelado de datos donde se guarda una información recogida por un sistema. Es decir, son objetos o estructuras que contienen los datos organizados en filas y columnas. Cada fila representa un registro único, y cada columna un campo dentro del registro.



2.1.1. Tabla

- Elementos clave de una tabla:
  - **Nombre de la tabla:** Identifica de forma única a la tabla dentro de la base de datos.
    - Por ejemplo, "Clientes", "Productos", "Pedidos".
  - **Columnas (o campos):** Son las cabeceras de cada columna y definen los tipos de datos que se almacenarán en esa columna.
    - Por ejemplo, "Nombre" (texto), "Edad" (número), "Fecha de nacimiento" (fecha).

2.1.1. Tabla

- **Filas (o registros):** Cada fila representa un registro individual y contiene un valor para cada columna.
  - Por ejemplo, en una tabla de "Clientes", una fila podría representar a un cliente específico con su nombre, edad y fecha de nacimiento.
- **Clave primaria:** Es una columna (o conjunto de columnas) que identifica de forma **única** cada fila en una tabla. No puede contener valores nulos ni duplicados.
  - Por ejemplo, un "ID de cliente" podría ser la clave primaria de la tabla "Clientes".

2.1.1. Tabla

- **Tipos de datos:** Definen el tipo de información que puede almacenarse en una columna. Los tipos de datos comunes incluyen:
  - **Texto:** Para almacenar nombres, direcciones, etc.
  - **Numérico:** Para almacenar números enteros o decimales.
  - **Fecha:** Para almacenar fechas y horas.
  - **Booleano:** Para almacenar valores verdaderos o falsos

2.1.1. Tabla

- Ejemplo:

Id (Clave Primaria)	Título	Autor	Año de Publicación
1	Don Quijote	Cervantes	1605
2	1984	Orwell	1949
3	El Principito	Saint-Exupéry	1943

2.2. Bases de datos no relacionales

- Las bases de datos no relacionales son sistemas de gestión de datos que ofrecen una alternativa flexible a las tradicionales bases de datos relacionales (como MySQL o PostgreSQL).

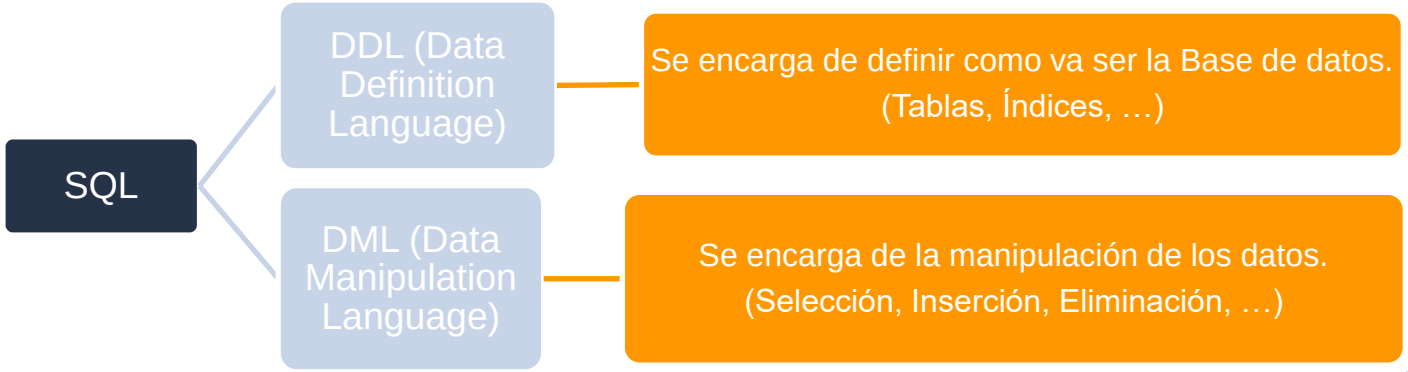
- En lugar de utilizar el rígido esquema de tablas, filas y columnas, las bases de datos no relacionales permiten almacenar datos de manera más flexible y adaptable.

3

Bases de datos relacionales con Python

3.1. Introducción a SQL

- **SQL** es un lenguaje estandarizado, aunque cada base de datos posee sus propias extensiones y funcionalidades añadidas, los usos básicos de este son los mismos en las diferentes bases de datos.
- https://docs.python.org/es/3/library/sqlite3.html



3.1.1. Principales comandos DDL

CREATE DATABASE

- Se una para crear una nueva base de datos

CREATE TABLE

- Crear una nueva tabla en la base de datos

ALTER TABLE

- Modificar las tablas de la base de datos, ya sea agregar, modificar o eliminar columnas.

DROP TABLE

- Borrar tablas de la base de datos.

3.1.2. Principales comandos DML

SELECT nombre_columna FROM

- Permite consultar datos de una o varias columnas.
- El comando FROM indica el nombre de la tabla para realizar la consulta

UPDATE

- Actualizar los datos existentes en una tabla.

DELETE

- Eliminar datos de una tabla.

INSERT INTO nombre_tabla VALUES

- Añadir datos a una tabla ya existente.
- El término VALUES indica los valores a introducir

3.1.3. Clausulas

WHERE

- Sirve para especificar una condición que deben cumplir los datos que recuperar de una o varias tablas

DISTINCT

- Devuelve solo alores distintos al realizar una consulta.

ORDER BY

- Ordenar el conjunto de los resultados en forma ascendente o descendente.

NOT NULL

- Sirve para especificar que la columna a la que se le asignemos la clausula siempre debe de tener un valor. La columna no puede estar vacía.

AUTO INCREMENT

- Permite generar un número único de manera automática al insertar nuevos registros en una tabla.

3.2. SQLite

- La base de datos relacional que vamos a manejar en Python es **SQLite.**

- Esta es una base datos fácil de usar, ligera y de código abierto, que se puede utilizar en diferentes lenguajes. Podemos usarla directamente desde Python sin tener que realizar instalaciones adicionales.

- Esta diseñada para ser insertada fácilmente dentro de diferentes aplicaciones y permite almacenar una base de datos en un solo archivo que se guardará en la memoria del ordenador.

3.2. SQLite

- Debemos tener en cuenta que, para trabajar con bases de datos, al igual que con ficheros, necesitamos abrir y cerrar la conexión con la base de datos correspondiente.

- Un concepto que vamos a necesitar para trabajar con las diferentes bases de datos es el **cursor**; este es el análogo al usado en el manejo de los ficheros y nos indicará en qué parte de la base de datos estamos operando.

3.2. SQLite

- Una cosa importante que debemos de tener en cuenta es la correspondencia entre los tipos de datos en SQLite y su correspondencia en Python.

SQLite	Python
NULL	None
INTEGER	int
REAL	float
TEXT	str
BLOB	bytes

3.3. Creación Base de Datos

```python
#Importamos la base de datos, no hay que instalarla
import sqlite3

#Creamos la conexion
"""Nombramos la base de datos que se creará de forma
automatica la primera vez que se ejecute el programa"""
conexion = sqlite3.connect("coches.db")

#Creamos el cursor
#El cursor es el que se encarga de ejecutar las ordenes SQL
#En este ejemplo no hay codigo SQL
cursor = conexion.cursor()

#Cerramos la conexion
conexion.close()
```

- Al ejecutar el código, se nos creará la base de datos: *coches.db*

3.4. Sentencias SQL

- CREATE , INSERT Y SELECT

```
#Importamos la base de datos, no hay que instalarla
import sqlite3

#Creamos la conexion
"""Nombramos la base de datos que se creará de forma
automatica la primera vez que se ejecute el programa"""
conexion = sqlite3.connect("coches.db")

#Creamos el cursor
#El cursor es el que se encarga de ejecutar las ordenes SQL
cursor = conexion.cursor()

#Creamos las ordendes de la base de datos
#Dentro de execute introducimos el codido SQL por ejecutar
"""La clausula IF NOT EXISTS se puede usar a la hora de crear tablar, esto permite
que la tabla solo se cree si no existe previamente.
Como la tabla se crea de forma automatica la primera vez que se ejecuta el programa,
nos evitará problemas de duplicidad"""
cursor.execute("CREATE TABLE IF NOT EXISTS Modelos (name text, modelo text, color text, plazas integer)")

cursor.execute("INSERT INTO Modelos VALUES ('Mercedes','Cabrio','Negro',2)")
```

```
#Ejecutamos las ordenes para guardar cambios
conexion.commit()

#Vemos los datos almacenados
cursor.execute("SELECT * FROM Modelos")

#Mostramos los resultados del query por pantalla
print(cursor.fetchall())

#Cerramos la conexion
conexion.close()
```

- El resto de las sentencias SQL se encuentran en el **Anexo I.**

3.4. Sentencias SQL

- CREATE , INSERT Y SELECT con funciones

```python
import sqlite3
def crear_conexion():
    conexion = sqlite3.connect("coches1.db")
    cursor = conexion.cursor()
    return cursor,conexion

def crear_tabla(cursor):
    cursor.execute('''
            CREATE TABLE IF NOT EXISTS Modelos (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT,
                modelo TEXT,
                color TEXT,
                plazas INTEGER
                )
            ''')

def anadir_valores_modelo(cursor,conexion,id, nombre, modelo, color, plazas):
    cursor.execute("INSERT INTO Modelos VALUES (?,?, ?, ?, ?)",
                   (id, nombre, modelo, color, plazas))
    #Guardar
    conexion.commit()
```

```python
def mostrar_modelos(cursor):
    cursor.execute("SELECT * FROM Modelos")
    #Mostrar
    print(cursor.fetchall())

def cerrar_conexion(conexion):
    conexion.close()

cursor,conexion = crear_conexion()
crear_tabla(cursor)
anadir_valores_modelo(cursor,conexion,None,"Mercedes", "GLA", "Rojo", 7)
anadir_valores_modelo(cursor,conexion,None, "Dacia", "Duster", "Blanco", 5)

mostrar_modelos(cursor)
cerrar_conexion(conexion)
```

- **El signo de interrogación (?)**, lo usaremos cuando dentro de una consulta necesitamos parámetros en vez de datos ya definidos.

3.4. Sentencias SQL

- DELETE – BORRAR
  - En este caso vamos a utilizar la columna name como referencia, pero se podría usar cualquier otra.

```
def borrar_datos(cursor,conexion):
    cursor.execute("DELETE FROM Modelos Where name = 'Mercedes'")
    conexion.commit()
```

3.4. Sentencias SQL

UPDATE - ACTUALIZAR

- Usamos la sentencia UPDATE indicando con SET la columna y el nuevo dato que introducir, mediante WHERE buscamos el registro donde realizar el cambio.
- En este caso hemos usado la columna name y el modelo correspondiente.

```
def actulizar_datos(cursor,conexion):
    cursor.execute("UPDATE Modelos SET color = 'Amarillo' WHERE name = 'Honda'")
    conexion.commit()
```

3.4. Sentencias SQL

- INNER JOIN - COMBINAR
  - La cláusula JOIN se utiliza para combinar filas de dos o más tablas basándose en una relación común.
  - Para este ejemplo hemos creado otra tabla llamada Clientes. En este ejemplo el elemento común es el Id del coche.

```
def combinar(cursor,conexion):
    cursor.execute('''
        SELECT
            c.nombre AS NombreCliente,
            c.DNI,
            m.name AS NombreModelo,
            m.modelo,
            m.color,
            m.plazas
        FROM
            Cliente c
        INNER JOIN
            Modelos m ON c.idCoche = m.Id
    ''')
    filas = cursor.fetchall()
    for fila in filas:
        print(fila)
```

Ejercicio Clase

- **Ejercicio 1.** Macarena tiene un pequeño restaurante y quiere llevar un registro de su carta.
- Diseña una base de datos:
  - T_Carta: Contiene información sobre los platos, como su nombre, descripción, precio y categoría (entrada, plato principal, postre).
  - T_Pedido: Registra los pedidos realizados, incluyendo la fecha, los platos pedidos (con cantidades).

Ejercicio Clase

- Crear un menú con las siguientes opciones:
  - Añadir nuevos platos
  - Consultar la carta, se puede hacer la consulta por categoría.
  - Eliminar platos del menú.
  - Consultar pedidos, todos o asociados a un cliente en particular.

¡Gracias!