Ayudantía 2 — SQL práctico con SQLite

Clases 03 y 04 — IIC2413

Benjamin Saavedra

Agenda de hoy

1. Instalación rápida
2. El código de "setup": conexión, cursor, y crear una BD
3. Repaso: SQL de las Clases 3 y 4
4. 10 ejercicios prácticos
5. Cómo esto se conecta con la Tarea 1

Antes de partir

- Esta semana vieron **todo lo que se puede hacer** con álgebra relacional / lógica en SQL: `SELECT-FROM-WHERE`, joins, subconsultas, operadores de conjunto.
- El **lunes** verán las cosas que **no se pueden hacer** en álgebra relacional: **agregación** (`GROUP BY`, `HAVING`, `COUNT`, `AVG`, `SUM`, ...), **`ORDER BY`** y **`LIKE`**.
- Por lo tanto, **hoy no usaremos nada de eso**. Todos los ejercicios de esta ayudantía se resuelven solo con lo que ya vieron.

Parte 0 — Instalación

`sqlite3` viene **incluido en la librería estándar de Python**.

```
import sqlite3   # no necesita pip install
```

No usaremos ninguna librería externa (ni `pandas` ni nada): todo el manejo de resultados se hace **directo con el cursor**.

Opcional (no obligatorio para la tarea), si quieres explorar la BD gráficamente:

- DB Browser for SQLite
- CLI: `sudo apt install sqlite3` / `brew install sqlite` / `choco install sqlite`

Parte 1 — Conexión y Cursor

Dos conceptos clave para hablar con SQLite desde Python:

- **`Connection`** : el enlace al archivo `.db` .
  - `sqlite3.connect(path)` / `conn.close()` / `conn.commit()`
  - Sin `commit()` , los cambios de DDL/DML **no quedan guardados**.
- **`Cursor`** : el objeto que **ejecuta** SQL y **lee** resultados.
  - `cursor.execute(query)`
  - `cursor.fetchone()` / `cursor.fetchall()`
  - `cursor.description` → nombres de las columnas del resultado

```
conn = sqlite3.connect("tienda.db")
cursor = conn.cursor()
cursor.execute("SELECT * FROM juegos")
filas = cursor.fetchall()
conn.close()
```

Parte 2 — Nuestra base de datos: tienda de videojuegos

```
desarrolladores(id_desarrollador, nombre, pais)                                    ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                       ar = 4
```

donde `juegos.id_desarrollador` referencia a `desarrolladores.id_desarrollador` (cada juego pertenece a un desarrollador), y `compras.id_juego` referencia a `juegos.id_juego` (cada compra corresponde a un juego).

A diferencia de la tarea (que carga un `.csv`), aquí los datos están **hardcodeados** en el notebook.

Creando las tablas (DDL)

```
cursor.executescript("""
    DROP TABLE IF EXISTS compras;
    DROP TABLE IF EXISTS juegos;
    DROP TABLE IF EXISTS desarrolladores;

    CREATE TABLE desarrolladores (
        id_desarrollador INTEGER,
        nombre TEXT NOT NULL,
        pais TEXT
    );

    CREATE TABLE juegos (
        id_juego INTEGER,
        titulo TEXT NOT NULL,
        id_desarrollador INTEGER,
        genero TEXT, plataforma TEXT, precio REAL, anio_lanzamiento INTEGER
    );
    -- ... y compras
""")
```

`executescript` corre **varias sentencias DDL** de una vez.

Insertando los datos (DML)

```
desarrolladores = [
    (1, "CD Projekt Red", "Polonia"),
    (2, "FromSoftware", "Japón"),
    # ...
]

cursor.executemany(
    "INSERT INTO desarrolladores VALUES (?, ?, ?);",
    desarrolladores
)
conn.commit()
```

- `executemany` inserta **todas las filas** de una lista con una sola sentencia parametrizada.
- Cada `?` se reemplaza por un valor de la tupla. Ya volveremos sobre esto en detalle durante los ejercicios prácticos.

Repaso

SQL ≈ Álgebra relacional

SELECT - FROM - WHERE ≈ proyección + selección

```
SELECT [DISTINCT] <lista-att>          π<lista-att> ( σ<condicion> ( R ) )
FROM <relacion>                   ≡
WHERE <condicion>
```

```sql
SELECT titulo, precio        -- proyección (π)
FROM juegos
WHERE genero = 'RPG';        -- selección (σ)
```

- `DISTINCT` elimina duplicados (SQL **sí permite** duplicados por defecto, a diferencia del álgebra relacional pura).
- `AS` renombra columnas: `SELECT precio AS precio_usd`.
- Conectores lógicos en `WHERE`: `AND`, `OR`, `NOT`.

Joins: varias formas de escribir lo mismo

```sql
-- Estilo "WHERE"
SELECT j.titulo, d.pais
FROM juegos AS j, desarrolladores AS d
WHERE j.id_desarrollador = d.id_desarrollador;

-- JOIN ... ON (equivalente, más explícito)
SELECT j.titulo, d.pais
FROM juegos AS j JOIN desarrolladores AS d ON j.id_desarrollador = d.id_desarrollador;

-- JOIN ... USING (cuando la columna se llama igual)
SELECT titulo, nombre FROM juegos JOIN desarrolladores USING (id_desarrollador);

-- NATURAL JOIN (junta automáticamente por todas las columnas de igual nombre)
SELECT titulo, nombre FROM juegos NATURAL JOIN desarrolladores;
```

Importante: `FROM t1, t2` **sin** `WHERE` es un producto cruz: junta cada fila con cada fila.

Subconsultas anidadas

Tipo	Dónde	Ejemplo
En el `FROM`	subconsulta como tabla	`FROM (SELECT ...) AS P`
`IN` / `NOT IN`	en el `WHERE`	`WHERE x IN (SELECT ...)`
`EXISTS` / `NOT EXISTS`	en el `WHERE`, correlacionada	`WHERE EXISTS (SELECT ... WHERE j.id_desarrollador = d.id_desarrollador)`
Por valor	en el `WHERE`, comparación directa	`WHERE precio > (SELECT precio FROM ... WHERE ...)`

- Las subconsultas de `EXISTS` /valor suelen estar **correlacionadas**: hacen referencia a una columna de la consulta externa.
- Importante: en una subconsulta "por valor", si la subconsulta devuelve más de una fila, SQLite lanza error.
- `ANY` / `ALL` **no están soportados en SQLite**.

Operadores de conjunto

Combinan dos `SELECT` con las **mismas columnas**:

```
SELECT plataforma FROM juegos WHERE genero = 'RPG'
UNION                          -- unión de conjuntos (sin duplicados)
SELECT plataforma FROM juegos WHERE genero = 'Acción';
```

Operador	Significado
`UNION`	unión, sin duplicados
`UNION ALL`	unión, con duplicados
`INTERSECT`	intersección
`EXCEPT`	diferencia (A menos B)

10 ejercicios prácticos

Ejercicio 1 — Enunciado

"Escribe una función que reciba una plataforma y un precio máximo, y retorne el título de los juegos de esa plataforma que cuesten menos que ese precio. Pruébala con `'PC'` y `20`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 1 — Primer intento (ingenuo)

Lo más natural parece ser armar la query pegando los valores directo en el string, con un f-string:

```
def juegos_por_plataforma_y_precio(db_path, plataforma, precio_maximo):
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    query = f"""
        SELECT titulo FROM juegos
        WHERE plataforma = '{plataforma}' AND precio < {precio_maximo};
    """
    cursor.execute(query)
    resultados = cursor.fetchall()
    conn.close()
    return resultados
```

Funciona para `'PC'` y `20` … ¿pero qué pasa si `plataforma` fuera el texto `"PC' OR '1'='1"`? ¿O si un valor tuviera un apóstrofe?

Ejercicio 1 — Solución (con placeholder)

```
def juegos_por_plataforma_y_precio(db_path, plataforma, precio_maximo):
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    query = """
        SELECT titulo FROM juegos
        WHERE plataforma = ? AND precio < ?;
    """
    cursor.execute(query, (plataforma, precio_maximo))
    resultados = cursor.fetchall()
    conn.close()
    return resultados

juegos_por_plataforma_y_precio("tienda.db", "PC", 20)
```

Explicación: cada `?` se reemplaza, en orden, por un valor de la tupla `(plataforma, precio_maximo)`. `sqlite3` nunca interpreta esos valores como parte del SQL, así que ni la inyección ni los apóstrofes rompen la query, a diferencia del intento anterior.

Placeholders `?`: la idea en general

Como vimos recién en el Ejercicio 1, dejar `?` en el SQL y pasar los valores aparte a `cursor.execute(query, valores)` es la forma correcta de parametrizar una query.

Para qué sirve:

- Evita mezclar código SQL con datos externos (variables, parámetros de función, input de usuario), por lo que previene inyección SQL.
- Evita errores de comillas/tipos/formato al armar el string a mano (texto con apóstrofes, fechas, etc).
- Permite reutilizar la misma query con distintos valores, sin reconstruir el string cada vez.
- Es el patrón que van a usar en varias funciones de la Tarea 1.

Ejercicio 2 — Enunciado

"Muestra, con los alias `Juego` y `Precio`, el título y precio de los juegos de género `'Acción'` lanzados desde el año 2020 en adelante."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 2 — Solución

```sql
SELECT titulo AS Juego, precio AS Precio
FROM juegos
WHERE genero = 'Acción' AND anio_lanzamiento >= 2020;
```

```python
consulta = """
    SELECT titulo AS Juego, precio AS Precio
    FROM juegos
    WHERE genero = 'Acción' AND anio_lanzamiento >= 2020;
"""

conn = sqlite3.connect("tienda.db")
cursor = conn.cursor()
cursor.execute(consulta)
resultados = cursor.fetchall()
conn.close()

resultados
```

Explicación: los alias (`AS`) solo cambian el nombre de la columna en el resultado; la condición del `WHERE` se sigue escribiendo con el nombre original de la columna (`anio_lanzamiento`), no con el alias.

Ejercicio 3 — Enunciado

"Muestra, sin duplicados, los géneros de los juegos que cuestan más de $50."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 3 — Solución

```sql
SELECT DISTINCT genero
FROM juegos
WHERE precio > 50;
```

```python
consulta = """
    SELECT DISTINCT genero
    FROM juegos
    WHERE precio > 50;
"""

conn = sqlite3.connect("tienda.db")
cursor = conn.cursor()
cursor.execute(consulta)
resultados = cursor.fetchall()
conn.close()

resultados
```

Explicación: varios juegos caros comparten género (por ejemplo, hay más de un RPG y más de un juego de Acción sobre $50); `DISTINCT` colapsa esas repeticiones y deja cada género una sola vez.

Ejercicio 4 — Enunciado

"Muestra el título del juego y el país de su desarrollador, para los juegos de género `'RPG'`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 4 — Solución

```sql
SELECT j.titulo, d.pais
FROM juegos AS j, desarrolladores AS d
WHERE j.id_desarrollador = d.id_desarrollador
  AND j.genero = 'RPG';
```

```python
consulta = """
    SELECT j.titulo, d.pais
    FROM juegos AS j, desarrolladores AS d
    WHERE j.id_desarrollador = d.id_desarrollador
      AND j.genero = 'RPG';
"""

conn = sqlite3.connect("tienda.db")
cursor = conn.cursor()
cursor.execute(consulta)
resultados = cursor.fetchall()
conn.close()

resultados
```

Explicación: producto cruz (`FROM j, d`) + condición de igualdad entre las columnas que conectan las tablas (`j.id_desarrollador = d.id_desarrollador`) + el filtro del enunciado. Es el estilo de join más común en código real.

Ejercicio 5 — Enunciado

"Escribe una función que reciba el nombre de un comprador y retorne el título del juego y la fecha de compra de todas sus compras. Pruébala con `'Bruno Silva'`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 5 — Solución

```
def compras_de(db_path, nombre_comprador):
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    query = """
        SELECT j.titulo, c.fecha_compra
        FROM compras AS c
        JOIN juegos AS j ON c.id_juego = j.id_juego
        WHERE c.nombre_comprador = ?;
    """
    cursor.execute(query, (nombre_comprador,))
    resultados = cursor.fetchall()
    conn.close()
    return resultados

compras_de("tienda.db", "Bruno Silva")
```

Explicación: un solo parámetro, un solo `?`. Nota la coma en `(nombre_comprador,)`: sin ella, no sería una tupla (maña de python).

Ejercicio 6 — Enunciado

"Muestra el nombre del comprador, el título del juego y el nombre del desarrollador, para las compras de juegos de `'FromSoftware'`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 6 — Solución

```sql
SELECT c.nombre_comprador, j.titulo, d.nombre
FROM compras AS c
JOIN juegos AS j ON c.id_juego = j.id_juego
JOIN desarrolladores AS d ON j.id_desarrollador = d.id_desarrollador
WHERE d.nombre = 'FromSoftware';
```

```python
consulta = """
    SELECT c.nombre_comprador, j.titulo, d.nombre
    FROM compras AS c
    JOIN juegos AS j ON c.id_juego = j.id_juego
    JOIN desarrolladores AS d ON j.id_desarrollador = d.id_desarrollador
    WHERE d.nombre = 'FromSoftware';
"""

conn = sqlite3.connect("tienda.db")
cursor = conn.cursor()
cursor.execute(consulta)
resultados = cursor.fetchall()
conn.close()

resultados
```

Explicación: encadenamos dos `JOIN` siguiendo el camino `compras → juegos → desarrolladores` y filtramos al final, sobre las tres tablas ya unidas.

Ejercicio 7 — Enunciado

"Escribe una función que reciba el nombre de un desarrollador y retorne, sin duplicados, los nombres de los compradores que compraron algún juego de ese desarrollador. Pruébala con `'Larian Studios'`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 7 — Solución

```
def compradores_de_desarrollador(db_path, nombre_desarrollador):
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    query = """
        SELECT DISTINCT nombre_comprador
        FROM compras
        WHERE id_juego IN (
            SELECT id_juego FROM juegos
            WHERE id_desarrollador = (
                SELECT id_desarrollador FROM desarrolladores WHERE nombre = ?
            )
        );
    """
    cursor.execute(query, (nombre_desarrollador,))
    resultados = cursor.fetchall()
    conn.close()
    return resultados

compradores_de_desarrollador("tienda.db", "Larian Studios")
```

Explicación: aunque hay tres niveles de subconsulta, solo un valor viene de afuera (`nombre_desarrollador`), así que basta un solo `?`, usado en la subconsulta más interna.

Ejercicio 7 — Solución alternativa (con JOIN)

```sql
SELECT DISTINCT c.nombre_comprador
FROM desarrolladores AS d
JOIN juegos AS j
  ON d.id_desarrollador = j.id_desarrollador
JOIN compras AS c
  ON j.id_juego = c.id_juego
WHERE d.nombre = 'Larian Studios';
```

Explicación: en vez de subconsultas anidadas, encadenamos los tres `JOIN` directamente (`desarrolladores → juegos → compras`) y filtramos por el nombre al final. Da exactamente el mismo resultado que la versión con `IN`.

Ejercicio 7 — Esta solución también es válida

```sql
SELECT DISTINCT nombre_comprador
FROM compras
WHERE id_juego IN (
    SELECT j.id_juego
    FROM juegos AS j
    JOIN desarrolladores AS d
      ON j.id_desarrollador = d.id_desarrollador
    WHERE d.nombre = 'Larian Studios'
);
```

Explicación: una forma híbrida: la subconsulta del `IN` usa un `JOIN` (en vez de subconsultas anidadas por valor) para obtener los `id_juego` de Larian Studios. También produce el mismo resultado — en SQL casi siempre hay más de una forma correcta de escribir una consulta.

`IN` vs. `JOIN`: cuidado con la multiplicidad

`WHERE columna IN (subconsulta)` normalmente se puede reescribir como un `JOIN` — pero no siempre tienen la misma semántica respecto de **cuántas veces aparece cada fila**:

- `IN` es una pregunta de **existencia**: cada fila externa aparece **como máximo una vez**, sin importar cuántas coincidencias haya en la subconsulta.
- `JOIN` genera **una fila por cada combinación** que coincide: si el lado "muchos" tiene más de una coincidencia, la fila externa se **repite**.

Por eso agregamos `DISTINCT` en la versión con `JOIN`: sin él, si algún comprador hubiera comprado *más de un* juego de Larian Studios, aparecería repetido (con nuestros datos actuales no pasa, pero con otros datos sí podría).

Ejercicio 8 — Enunciado

"Muestra el nombre de los desarrolladores que tienen al menos un juego de género `'RPG'`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                          ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                              ar = 4
```

Ejercicio 8 — Solución

```sql
SELECT nombre
FROM desarrolladores AS d
WHERE EXISTS (
    SELECT * FROM juegos AS j
    WHERE j.id_desarrollador = d.id_desarrollador AND j.genero = 'RPG'
);
```

```python
consulta = """
    SELECT nombre
    FROM desarrolladores AS d
    WHERE EXISTS (
        SELECT * FROM juegos AS j
        WHERE j.id_desarrollador = d.id_desarrollador AND j.genero = 'RPG'
    );
"""

conn = sqlite3.connect("tienda.db")
cursor = conn.cursor()
cursor.execute(consulta)
resultados = cursor.fetchall()
conn.close()

resultados
```

Explicación: la subconsulta está *correlacionada* — `j.id_desarrollador = d.id_desarrollador` la asocia a cada fila externa de `desarrolladores`. `EXISTS` es verdadero si existe **al menos una** fila que cumpla, sin importar cuántas.

Ejercicio 9 — Enunciado

"Muestra los compradores que compraron algún juego de `'PS5'` pero *ningún* juego de `'PC'`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                    ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                        ar = 4
```

Ejercicio 9 — Solución

```sql
SELECT c.nombre_comprador
FROM compras AS c JOIN juegos AS j ON c.id_juego = j.id_juego
WHERE j.plataforma = 'PS5'
EXCEPT
SELECT c.nombre_comprador
FROM compras AS c JOIN juegos AS j ON c.id_juego = j.id_juego
WHERE j.plataforma = 'PC';
```

```python
consulta = """
    SELECT c.nombre_comprador
    FROM compras AS c JOIN juegos AS j ON c.id_juego = j.id_juego
    WHERE j.plataforma = 'PS5'
    EXCEPT
    SELECT c.nombre_comprador
    FROM compras AS c JOIN juegos AS j ON c.id_juego = j.id_juego
    WHERE j.plataforma = 'PC';
"""

conn = sqlite3.connect("tienda.db")
cursor = conn.cursor()
cursor.execute(consulta)
resultados = cursor.fetchall()
conn.close()

resultados
```

Explicación: calculamos el conjunto de compradores de PS5 y le restamos (`EXCEPT`) el conjunto de compradores de PC, por lo que quedan solo quienes están en el primero y no en el segundo.

Ejercicio 10 — Enunciado

"Escribe una función que reciba dos países y retorne los compradores que compraron un juego de un desarrollador del primer país y también un juego de un desarrollador del segundo país. Pruébala con `'Estados Unidos'` y `'Japón'`."

Esquema (recordatorio):

```
desarrolladores(id_desarrollador, nombre, pais)                                    ar = 3
juegos(id_juego, titulo, id_desarrollador, genero, plataforma, precio, anio_lanzamiento)    ar = 7
compras(id_compra, id_juego, nombre_comprador, fecha_compra)                        ar = 4
```

Ejercicio 10 — Solución

```
def compradores_de_dos_paises(db_path, pais1, pais2):
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    query = """
        SELECT DISTINCT c.nombre_comprador
        FROM compras AS c JOIN juegos AS j ON c.id_juego = j.id_juego
        WHERE j.id_desarrollador IN
            (SELECT id_desarrollador FROM desarrolladores WHERE pais = ?)
        INTERSECT
        SELECT DISTINCT c.nombre_comprador
        FROM compras AS c JOIN juegos AS j ON c.id_juego = j.id_juego
        WHERE j.id_desarrollador IN
            (SELECT id_desarrollador FROM desarrolladores WHERE pais = ?);
    """
    cursor.execute(query, (pais1, pais2))
    resultados = cursor.fetchall()
    conn.close()
    return resultados

compradores_de_dos_paises("tienda.db", "Estados Unidos", "Japón")
```

Explicación: dos `?`, uno por país, en el mismo orden que `(pais1, pais2)`. Toda la lógica de joins, subconsultas e `INTERSECT` que vimos hoy son validas si usamos parámetros, no son técnicas separadas.

Ahora les toca a ustedes

En el notebook van a encontrar **17 ejercicios** organizados en 4 bloques:

A. Proyección / selección · **B.** Joins · **C.** Subconsultas · **D.** Operadores de conjunto

- Algunos ya **resueltos y explicados** (como los que acabamos de ver).
- Otros solo con el enunciado y una celda vacía para que lo intenten, con la solución un poco más abajo.

Dudas?

Para la Tarea 1

- El **flujo de trabajo** es el mismo: conexión → cursor → crear tablas (DDL) → insertar datos (DML) → `cursor.fetchall()` para leer resultados.
- Hoy cubrimos: `SELECT-FROM-WHERE`, joins (varias formas), subconsultas anidadas, operadores de conjunto, y funciones parametrizadas con `?`.
- El **lunes** verán agregación, `ORDER BY` y `LIKE`, que son herramientas que también van a necesitar para la tarea.
- Practiquen combinando estas herramientas: la mayoría de los ejercicios "difíciles" son varias herramientas simples, combinadas (como el Ejercicio 10).

Referencias

- Documentación de `sqlite3` en Python
- Documentación oficial de SQLite