PYTHON

File Processing

SQLite3 - interacting with SQLite databases

FILE PROCESSING

SQLite3 - interacting with SQLite databases

FILE PROCESSING

- Database Management System (DBMS) o Sistema Gestor de Bases de Datos (SGBD) es responsable de:
  - Crear la estructura de la base de datos;
  - Insertar, actualizar, eliminar y buscar datos;
  - Garantizar la seguridad de los datos;
  - Gestión de transacciones;
  - Garantizar el acceso simultáneo a los datos para múltiples usuarios;
  - Permitir el intercambio de datos con otros sistemas de bases de datos.

FILE PROCESSING

- Algunos SGBD:
  - Gratuitos:
    - MySQL
    - PostgreSQL
    - SQLite
  - Bajo pago:
    - Oracle Database
    - Microsoft SQL Server
    - IBM DB2

FILE PROCESSING

- SQLite:
  - Biblioteca de C.
  - Se almacena en un único archivo.
  - No necesita de un proceso en ejecución independiente.
  - No requiere configuración.
  - Admite transacciones.
  - Se utiliza en dispositivos móviles (iOS y Android).
  - Es multiplataforma.
  - Python incorpora en la biblioteca estándar (desde la versión 2.5 de Python) un módulo que proporciona una interfaz para comunicarse con SQLite, concretamente con la especificación DB-API 2.0 descrita en PEP 249.
  - El propósito de la especificación DB-API 2.0 es definir un estándar común para la creación de módulos que funcionen con bases de datos en Python.

FILE PROCESSING

- SQL:
  - Desarrollado por IBM en 1970.
  - Existen varios estándares. SQLite utiliza una versión simplificada del estándar SQL-92.
  - No admite procedimientos almacenados.
  - No dispone de gestión de usuarios.

FILE PROCESSING

- Uso:
  - Importación del módulo sqlite3:
    - `import sqlite3`
  - Creación de la base de datos:
    - Método **connect** → Devuelve un objeto **Connection**.
    - Si no existe el fichero con la base de datos lo crea.
      - `conn = sqlite3.connect('hello.db')`
    - Se puede crear la base de datos en **memoria**:
      - `conn = sqlite3.connect(':memory:')`

FILE PROCESSING

- Uso:
  - Tipos de datos:
    - NULL.
    - INTEGER.
    - REAL. 8-byte IEEE floating point number.
    - TEXT. Text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
    - BLOB. The value is a blob of data, stored exactly as it was input.
  - Creación de tablas
    - `CREATE TABLE table_name (column1 datatype, column2 datatype, column3 datatype, … columnN datatype);`

https://www.sqlite.org/datatype3.html

FILE PROCESSING

- Uso. Restricciones
  - NOT NULL → De nulabilidad
  - PRIMARY KEY → De unicidad (implica NOT NULL)

```
CREATE TABLE tasks (
id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
priority INTEGER NOT NULL
);
```

FILE PROCESSING

- Cursor.
  - El método **cursor** de **Connection** proporciona un cursor, que es el mecanismo para poder acceder a los datos de la base de datos.
  - Funciones principales de un cursor:
    - Ejecutar consultas SQL → execute(), executemany(), executescript()
    - Obtener resultados → fetchone(), fetchall(), fetchmany(n)
    - Administrar transacciones → commit(), rollback()
    - Cerrar el cursor → close()

FILE PROCESSING

- Cursor.
  - Métodos más útiles.

Método	Descripción
execute(sql, params)	Ejecuta una consulta SQL con parámetros.
executemany(sql, seq_of_params)	Ejecuta la misma consulta con múltiples valores.
executescript(sql_script)	Ejecuta múltiples consultas separadas por ;.
fetchone()	Obtiene una fila del resultado.
fetchall()	Obtiene todas las filas del resultado.
fetchmany(n)	Obtiene n filas del resultado.

FILE PROCESSING

- Creación de tablas.

```
import sqlite3
conn = sqlite3.connect('todo.db')
c = conn.cursor()
c.execute('''CREATE TABLE tasks (
id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
priority INTEGER NOT NULL
);''')
```

- Si la tabla existe: `sqlite3.OperationError`: `table tasks already exists.`
- SOLUCIÓN → `CREATE TABLE IF NOT EXISTS`

FILE PROCESSING

- Insertando registros.

```
INSERT INTO table_name VALUES (value1, value2, value3, ..., valueN);
```

```
cursor.execute('INSERT INTO tasks (name, priority) VALUES (?,?)', ('My first task', 1))
```

Variables (?,?) y valores (tupla o lista)

FILE PROCESSING

- Insertando registros.
  - Confirmación de cambios. Método **commit** de Connection.
  - Cierre de la conexión. Método **close** de Connection.

FILE PROCESSING

- Insert simple. Método **execute**

```
import sqlite3

conn = sqlite3.connect('todo.db')
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS tasks (
id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
priority INTEGER NOT NULL
);''')
c.execute('INSERT INTO tasks (name, priority) VALUES (?,?)', ('My first task', 1))
conn.commit()
conn.close()
```

FILE PROCESSING

- Insert múltiple. Método **executemany**

```
import sqlite3

conn = sqlite3.connect('todo.db')
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS tasks (
id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
priority INTEGER NOT NULL
);''')
tasks = [
    ('My first task', 1),
    ('My second task', 5),
    ('My third task', 10),
]
c.executemany('INSERT INTO tasks (name, priority) VALUES (?,?)', tasks)
conn.commit()
conn.close()
```

FILE PROCESSING

- Refactorizando el código:

```
import sqlite3

class Todo:
    def __init__(self):
        self.conn = sqlite3.connect('todo.db')
        self.c = self.conn.cursor()
        self.create_task_table()

    def create_task_table(self):
        pass

    def add_task(self):
        pass

app = Todo()
app.add_task()
```

FILE PROCESSING

- Sentencia **Select**
  - `SELECT column FROM table_name;`
  - `SELECT column1, column2, column3, …, columnN FROM table_name;`
  - `SELECT * FROM table_name;`
  - **Se realiza sobre el cursor.**
  - **El resultado es un iterador.**
  - **El acceso a las columnas se hace a través de un índice.**

FILE PROCESSING

- Sentencia **Select**

```
import sqlite3

conn = sqlite3.connect('todo.db')
c = conn.cursor()
for row in c.execute('SELECT * FROM tasks'):
    print(row)
conn.close()
```

FILE PROCESSING

- Sentencia **Select.** Método **fetchall.**
  - Si se quieren cargar todos los registros en memoria → **fechall**.
  - Devuelve una lista de tuplas con el resultado de la consulta.
  - Si no hay resultados → lista vacía.
  - Menos eficiente. Puede provocar la saturación de la memoria.

FILE PROCESSING

- Sentencia **Select.** Método **fetchall.**

```
import sqlite3

conn = sqlite3.connect('todo.db')
c = conn.cursor()
c.execute('SELECT * FROM tasks')
rows = c.fetchall()
for row in rows:
    print(row)
conn.close()
```

FILE PROCESSING

- Sentencia **Select.** Método **fetchone.**
  - Proporciona el siguiente registro del cursor.
  - Si no hay datos → None

```
import sqlite3

conn = sqlite3.connect('todo.db')
c = conn.cursor()
c.execute('SELECT * FROM tasks')
row = c.fetchone()
print(row)
row = c.fetchone()
print(row)
conn.close()
```

FILE PROCESSING

- **Sentencia Update.**
- UPDATE nombre_tablaSET columna1 = valor1, columna2 = valor2, columna3 = valor3, …, columnaN = valorN WHERE condición;
- Sin cláusula WHERE se actualizan todos los registros de la tabla.

```
import sqlite3

conn = sqlite3.connect('todo.db')
c = conn.cursor()
c.execute('UPDATE tasks SET priority = ? WHERE id = ?', (20, 1))
c.commit()
c.close()
```

FILE PROCESSING

- Sentencia **Delete.**
- `DELETE FROM table_name WHERE condition;`
- Sin cláusula WHERE se ELIMINAN todos los registros de la tabla.

```
import sqlite3

conn = sqlite3.connect('todo.db')
c = conn.cursor()
c.execute('DELETE FROM tasks WHERE id = ?', (1,))
c.commit()
c.close()
```