

SQLite con Python



Que es SQL ?

- SQL (Structured Query Language) es un lenguaje **estándar** e interactivo de acceso a bases de datos relacionales que permite especificar operaciones
 - Altas : Create database, create table, insert record
 - Cambios: Update
 - Eliminaciones: delete, drop
 - Consultas: select from

- 2 Características generales de SQL
 - 2.1 Tipos de datos
 - 2.2 Optimización
- 3 Lenguaje de definición de datos (DDL)
 - 3.1 CREATE | CREAR
 - 3.2 ALTER | MODIFICAR
 - 3.3 DROP | ELIMINAR
 - 3.4 TRUNCATE | TRUNCAR
- 4 Lenguaje de manipulación de datos DML(Data Manipulation Language)
 - 4.1 Definición
 - 4.2 SELECT | SELECCIONAR
 - 4.2.1 Forma básica
 - 4.2.2 Cláusula WHERE
 - 4.2.3 Cláusula ORDER BY
 - 4.3 SUBCONSULTAS
 - 4.4 INSERT | INSERTAR
 - 4.4.1 Forma básica
 - 4.4.2 Ejemplo
 - 4.4.3 Formas avanzadas
 - 4.4.3.1 Copia de filas de otras tablas
 - 4.5 UPDATE
 - 4.5.1 Ejemplo
 - 4.6 DELETE
 - 4.6.1 Forma básica
 - 4.6.2 Ejemplo



5 Recuperación de clave

6 Disparadores

7 Sistemas de gestión de base de datos

Formas de trabajar con una BBDD

1. **Comandos SQL**, estándares, con variaciones en algunas marcas.
2. **Gestores de BBDD** : Aplicaciones visuales, adaptadas a una o más marcas.
3. **API db** : Programas de manejo para usar las bases de datos desde un programa.



SYBASE



SQLITE3

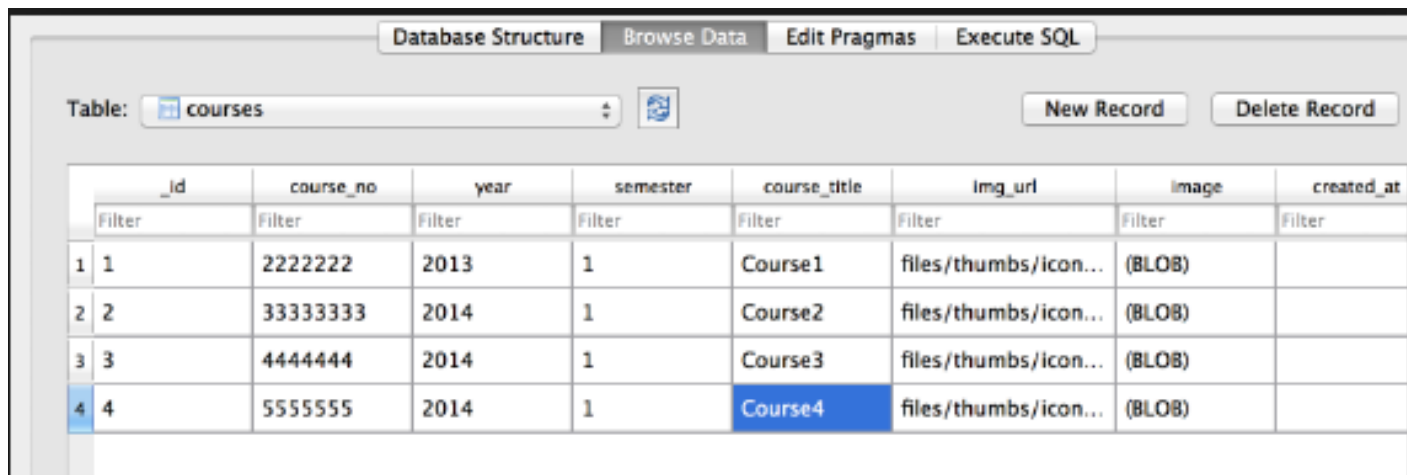
- SQLite es una base de datos escrita en C, **muy ligera, basada en archivos**.
- almacena la información en dispositivos de **forma sencilla**, eficaz, potente, rápida y en equipos con pocas capacidades de hardware, como puede ser una PDA o un teléfono móvil.
- SQLite **implementa el estándar SQL92** y también agrega extensiones. Esto permite que SQLite soporte desde las consultas más básicas hasta las más complejas del lenguaje SQL,
- Se puede usar tanto en dispositivos móviles como en sistemas de escritorio, sin necesidad de realizar procesos complejos de importación y exportación de datos, ya que existe compatibilidad al 100% entre las diversas plataformas disponibles, haciendo que la **portabilidad entre dispositivos** y plataformas sea transparente

Características

Estas son algunas de las características principales de SQLite:

- La base de datos completa se encuentra en un solo archivo.
- Puede funcionar enteramente en memoria, lo que la hace muy rápida.
- Tiene un footprint menor a 230KB.
- Es totalmente autocontenida (sin dependencias externas).
- Cuenta con librerías de acceso para muchos lenguajes de programación.
- Soporta texto en formato UTF-8 y UTF-16, así como datos numéricos de 64 bits.
- Soporta funciones SQL definidas por el usuario (UDF).
- El código fuente es de dominio público y se encuentra muy bien documentado.

<https://www.tutorialspoint.com/sqlite>



The screenshot shows a database browser window with tabs for 'Database Structure', 'Browse Data', 'Edit Pragma', and 'Execute SQL'. The 'Browse Data' tab is active, showing a table named 'courses'. The table has 8 columns: _id, course_no, year, semester, course_title, img_url, image, and created_at. The first four columns have filter icons. The table contains 4 records. The first record has _id 1, course_no 2222222, year 2013, semester 1, and course_title 'Course1'. The second record has _id 2, course_no 33333333, year 2014, semester 1, and course_title 'Course2'. The third record has _id 3, course_no 4444444, year 2014, semester 1, and course_title 'Course3'. The fourth record has _id 4, course_no 5555555, year 2014, semester 1, and course_title 'Course4'. The 'img_url' and 'image' columns show truncated values and '(BLOB)' respectively. The 'created_at' column is empty for all records.

	_id	course_no	year	semester	course_title	img_url	image	created_at
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2222222	2013	1	Course1	files/thumbs/icon...	(BLOB)	
2	2	33333333	2014	1	Course2	files/thumbs/icon...	(BLOB)	
3	3	4444444	2014	1	Course3	files/thumbs/icon...	(BLOB)	
4	4	5555555	2014	1	Course4	files/thumbs/icon...	(BLOB)	

Cómo se crea una base de datos SQL

MY SQL :

CREATE DATABASE

[IF NOT EXISTS] *database_name*

[CHARACTER SET *charset_name*]

[COLLATE *collation_name*]

SQLITE :

Consola> **sqlite3** *ejemplo.db*

O abriendo la base de datos desde un programa, si no existe la crea.

API -> Sqlite Python

- Existe el módulo sqlite 3 para python que incluye clases y métodos para gestionar la base de datos está incluido en Python.

- conectar
- obtener cursor
- ejecutar

```
import sqlite3

conexion = sqlite3.connect('usuarios_autoincremental.db')
cursor = conexion.cursor()

# Recuperamos un registro de la tabla de usuarios
cursor.execute("SELECT * FROM usuarios WHERE id=1")

usuario = cursor.fetchone()
print(usuario)

conexion.close()
```

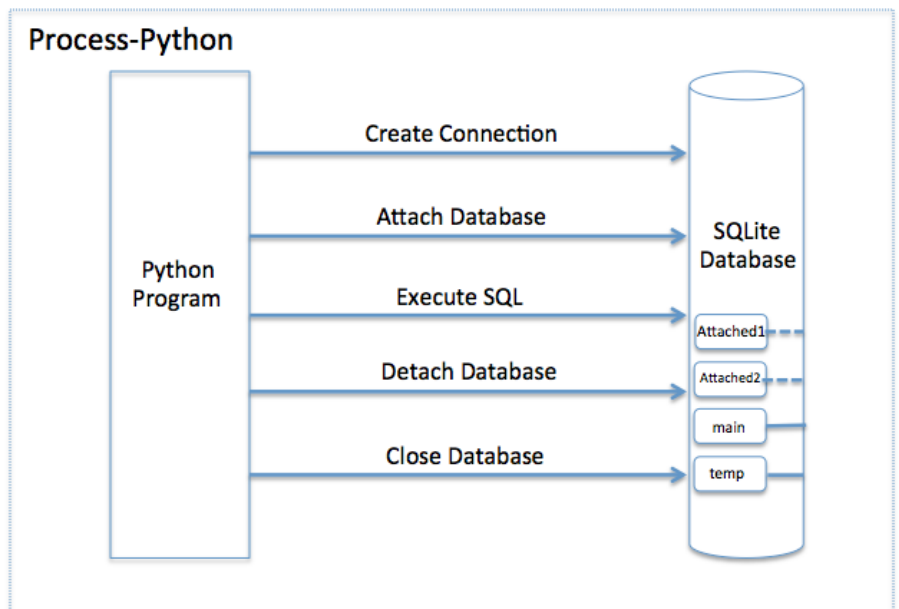

Primeros pasos

Conexión a la base de datos, creación y desconexión:

```
# Importamos el módulo
import sqlite3

# Nos conectamos a la base de datos ejemplo.db (la crea si no existe)
conexion = sqlite3.connect('ejemplo.db')

# Cerramos la conexión, si no la cerramos quedará abierta
# y no podremos gestionar el fichero
conexion.close()
```



Cómo se crea una tabla en SQL

CREATE TABLE **usuarios** (

id INTEGER PRIMARY KEY AUTOINCREMENT

dni VARCHAR(20) NOT NULL ,

nombre VARCHAR(100),

apellidos VARCHAR(100),

email VARCHAR(100),

tipo_usu INTEGER,

)

Los tipos más usuales

INTEGER
CHARACTER(20)
VARCHAR(255)
DECIMAL(10,5)
BOOLEAN
DATE
DATETIME
CLOB
BLOB

Crear una tabla

Antes de ejecutar una consulta (query) en código SQL, tenemos que crear un cursor:

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')

# Creamos el cursor
cursor = conexion.cursor()

# Ahora crearemos una tabla de usuarios con nombres, edades y emails
cursor.execute("CREATE TABLE IF NOT EXISTS usuarios " \
               "(nombre VARCHAR(100), edad INTEGER, email VARCHAR(100))")

# Guardamos los cambios haciendo un commit
conexion.commit()

conexion.close()
```

Inserción con INSERT

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')
cursor = conexion.cursor()

# Insertamos un registro en la tabla de usuarios
cursor.execute("INSERT INTO usuarios VALUES " \
               "('Hector', 27, 'hector@ejemplo.com')")

# Guardamos los cambios haciendo un commit
conexion.commit()

conexion.close()
```

Lectura con SELECT

Recuperando el primer registro con `.fetchone()`:

Código

Resultado

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')
cursor = conexion.cursor()

# Recuperamos los registros de la tabla de usuarios
cursor.execute("SELECT * FROM usuarios")

# Mostrar el cursos a ver que hay ?
print(cursor)

# Recorremos el primer registro con el método fetchone, devuelve una tupla
usuario = cursor.fetchone()
print(usuario)

conexion.close()
```

Inserción múltiple

Insertando varios registros con `.executemany()`:

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')
cursor = conexion.cursor()

# Creamos una lista con varios usuarios
usuarios = [('Mario', 51, 'mario@ejemplo.com'),
            ('Mercedes', 38, 'mercedes@ejemplo.com'),
            ('Juan', 19, 'juan@ejemplo.com')]

# Ahora utilizamos el método executemany() para insertar varios
cursor.executemany("INSERT INTO usuarios VALUES (?, ?, ?)", usuarios)

# Guardamos los cambios haciendo un commit
conexion.commit()

conexion.close()
```

Lectura múltiple

Recuperando varios registros con `.fetchall()`:

Código

Resultado

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')
cursor = conexion.cursor()

# Recuperamos los registros de la tabla de usuarios
cursor.execute("SELECT * FROM usuarios")

# Recorremos todos los registros con fetchall
# y los volcamos en una lista de usuarios
usuarios = cursor.fetchall()

# Ahora podemos recorrer todos los usuarios
for usuario in usuarios:
    print(usuario)

conexion.close()
```

Arquitectura 3 Capas



Si ponemos la base de datos en un módulo independiente nos acercamos a esta arquitectura, Pero aún tenemos juntas la capa de presentación y la de negocio.