

Lenguaje SQL

▼ Funciones Básicas de SQLITE

1. **SELECT** - Extraer datos de una tabla.

- Sintaxis:

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

- Ejemplo:

```
SELECT nombre, edad  
FROM estudiantes  
WHERE edad > 20;
```

2. **INSERT INTO** - Insertar nuevos registros en una tabla.

- Sintaxis:

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

- Ejemplo:

```
INSERT INTO estudiantes (nombre, edad)  
VALUES ('Juan', 25);
```

3. **UPDATE** - Modificar registros existentes en una tabla.

- Sintaxis:

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

- Ejemplo:

```
UPDATE estudiantes  
SET edad = 26  
WHERE nombre = 'Juan';
```

4. **DELETE** - Eliminar registros de una tabla.

- Sintaxis:

```
DELETE FROM table_name WHERE condition;
```

- Ejemplo:

```
DELETE FROM estudiantes  
WHERE nombre = 'Juan';
```

5. **AVG()** - Calcular el promedio de una columna.

- Sintaxis:

```
SELECT AVG(column_name)  
FROM table_name  
WHERE condition;
```

- Ejemplo:

```
SELECT AVG(edad)  
FROM estudiantes;
```

6. **COUNT()** - Contar el número de registros que cumplen una condición.

- Sintaxis:

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

- Ejemplo:

```
SELECT COUNT(*)
FROM estudiantes
WHERE edad > 20;
```

7. **MAX()** y **MIN()** - Encontrar el valor máximo y mínimo de una columna.

- Sintaxis:

```
SELECT MAX(column_name), MIN(column_name)
FROM table_name
WHERE condition;
```

- Ejemplo:

```
SELECT MAX(edad), MIN(edad)
FROM estudiantes;
```

Estas son solo algunas de las funciones básicas de SQLite.

▼ COMANDO SELECT

La sintaxis esquemática para el comando **SELECT** en SQL es la siguiente:

```
SELECT [DISTINCT | ALL] { * | [column_expression [AS new_name]] [, ...] }
FROM table_name [, ...]
[WHERE condition]
[GROUP BY column_name [, ...] ]
[HAVING condition]
[ORDER BY column_name [ASC | DESC] [, ...] ]
[LIMIT number]
[OFFSET number];
```

Desglose de la sintaxis.

- `SELECT` : La palabra clave para iniciar una consulta.
- `DISTINCT | ALL` : Son opciones que permiten seleccionar registros únicos o todos los registros. Si no especificas ninguno de los dos, se asume `ALL`.
- `*` : Selecciona todas las columnas de la tabla.
- `column_expression` : Puedes seleccionar columnas específicas en lugar de todas las columnas.
- `AS new_name` : Permite renombrar la columna de salida.
- `FROM table_name` : Especifica la tabla de donde se seleccionarán los registros.
- `WHERE condition` : Permite filtrar los registros basados en una condición específica.
- `GROUP BY column_name` : Agrupa registros por una o más columnas.
- `HAVING condition` : Se utiliza para filtrar valores después de un `GROUP BY`.
- `ORDER BY column_name [ASC | DESC]` : Ordena los registros de acuerdo a `column_name` en orden ascendente (`ASC`) o descendente (`DESC`).
- `LIMIT number` : Limita el número de registros a mostrar.
- `OFFSET number` : Especifica el número de registros a saltar antes de empezar a devolver registros.

▼ Ejemplo

Seleccionar todos los registros de una tabla:

```
SELECT * FROM student;
```

Seleccionar columnas específicas de una tabla:

```
SELECT name, sex FROM student;
```

Seleccionar y renombrar una columna:

```
SELECT name AS "Nombre" FROM student;
```

Seleccionar registros con una condición específica:

```
SELECT * FROM student WHERE sex = "female";
```

Seleccionar registros con múltiples condiciones:

```
SELECT name FROM student WHERE sex = "male" AND mark > 40;
```

Ordenar registros de acuerdo a una columna:

```
SELECT * FROM student ORDER BY mark;
```

Ordenar registros en orden descendente:

```
SELECT * FROM student ORDER BY mark DESC;
```

Limitar la cantidad de registros a mostrar:

```
SELECT * FROM nombre_tabla LIMIT 5;
```

Seleccionar registros únicos (eliminar duplicados):

```
SELECT DISTINCT class from student;
```

Contar registros:

```
SELECT COUNT(*) FROM student;
```

```
SELECT count(sex) FROM student WHERE
```

▼ INSERT y UPDATE

1. **INSERT:** Para añadir un nuevo estudiante a la tabla.

- Sintaxis:

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

- Ejemplo: Añadir un nuevo estudiante llamado "David Lope", que está en la clase "Three", obtuvo una marca de 70, y es masculino.

```
INSERT INTO student (ID, name, class, mark, sex)  
VALUES (36, 'David Lope', 'Three', 70, 'male');
```

2. **UPDATE:** Modificar datos de un estudiante existente basado en un criterio específico.

- Sintaxis:

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

- Ejemplo 1: Cambiar la marca de "John Deo" a 80.

```
UPDATE student  
SET mark = 80  
WHERE name = 'John Deo';
```

- Ejemplo 2: Cambiar el sexo de "John Mike" a masculino.

```
UPDATE student  
SET sex = 'male'  
WHERE name = 'John Mike';
```

3. Insertar varios registros a la vez:

Añadir múltiples estudiantes en una sola instrucción:

- Sintaxis:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES
(value1a, value2a, value3a, ...),
(value1b, value2b, value3b, ...),
...;
```

- Ejemplo:

```
INSERT INTO student (ID, name, class, mark, sex)
VALUES
(37, 'Anna Bell', 'Five', 85, 'female'),
(38, 'Chris Top', 'Six', 75, 'male'),
(39, 'Eve Lyn', 'Four', 90, 'female');
```

4. Insertar datos desde otra tabla: (SALTARSE)

Insertar datos en una tabla a partir de otra tabla (por ejemplo, copiar ciertos registros de una tabla a otra).

- Sintaxis:

```
INSERT INTO table_name1 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table_name2
WHERE condition;
```

5. Insertar un registro con valores predeterminados:

Si solo se conocen algunos de los valores y se quiere insertar valores predeterminados para las demás columnas, puedes hacerlo así:

- Sintaxis:

```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

- Ejemplo:
Insertar el nombre y la clase de un estudiante, y quieres que los demás campos sean NULL o tomen un valor predeterminado:

```
INSERT INTO student (name, class)
VALUES ('Lucas Mint', 'Seven');
```

6. Actualizar múltiples columnas a la vez:

Modificar más de una columna en una sola instrucción, puedes hacerlo de la siguiente manera:

- Sintaxis:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

- Ejemplo:
Supongamos que quieres actualizar la clase y la marca para el estudiante con ID 1:

```
UPDATE student
SET class = 'Five', mark = 80
WHERE ID = 1;
```

7. Actualizar valores basándote en otra tabla:

Actualizar valores en una tabla utilizando información de otra tabla.

- Sintaxis: `bonus_marks` integer

```
UPDATE table_name1
SET column1 = (SELECT column1 FROM table_name2 WHERE condition2),
   column2 = (SELECT column2 FROM table_name2 WHERE condition2)
WHERE condition1;
```


- Ejemplo:

Imagina que tienes otra tabla llamada `extra_marks` con columnas `ID` y `bonus_marks`. Si deseas añadir estas `bonus_marks` a la columna `mark` de la tabla `student` para cada estudiante que coincida con un ID, podrías hacerlo así:

```
--PASO 1
CREATE TABLE extra_marks (
    ID INTEGER PRIMARY KEY,
    bonus_marks INTEGER
);

--PASO 2
INSERT INTO extra_marks (id, bonus_marks)
VALUES (5,10), (10,10), (13,10);

--Paso3

UPDATE student
SET mark = mark + (SELECT bonus_marks FROM extra_marks WHERE student.ID = extra_marks.ID)
WHERE ID IN (SELECT ID FROM extra_marks);
```

8. Actualizar registros usando funciones:

Utilizar funciones para actualizar registros. Por ejemplo, para incrementar todos los valores de una columna numérica.

- Sintaxis:

```
UPDATE table_name
SET column = function(column)
WHERE condition;
```

- Ejemplo:

Si quieres incrementar en 10 las marcas de todos los estudiantes que están en la clase 'Six':

```
UPDATE student
SET mark = mark + 10
WHERE class = 'Six';
```

9. Actualizar registros condicionalmente basado en su contenido actual:

- Sintaxis:

```
UPDATE table_name
SET column = CASE
    WHEN condition1 THEN value1
    WHEN condition2 THEN value2
    ELSE default_value
END
WHERE main_condition;
```

- Ejemplo:

Supongamos que deseas otorgar bonificaciones a los estudiantes: +5 para marcas por debajo de 60 y +3 para marcas entre 60 y 75:

```
UPDATE student
SET mark = CASE
    WHEN mark < 60 THEN mark + 5
    WHEN mark BETWEEN 60 AND 75 THEN mark + 3
    ELSE mark
END;
```

Estos ejemplos deberían darte una idea más profunda de las diversas maneras en las que puedes utilizar el comando `UPDATE` en SQLite.

▼ Crear una nueva tabla

Crear una nueva tabla llamada `new_students` e insertar datos provenientes de la tabla `student`.

1. Creación de la tabla `new_students`:

Primero, deberíamos crear una estructura idéntica (o adecuada según el requerimiento) a la tabla `student`.

```
CREATE TABLE new_students (
    ID INTEGER PRIMARY KEY,
    name TEXT,
```

```
class TEXT,  
mark INTEGER,  
sex TEXT  
);
```

2. Insertar datos de `student` en `new_students` :

Supongamos que deseas insertar solamente los estudiantes de la clase 'Seven' de la tabla `student` a la tabla `new_students` .

```
INSERT INTO new_students (ID, name, class, mark, sex)  
SELECT ID, name, class, mark, sex  
FROM student  
WHERE class = 'Seven';
```

Si quisieras insertar todos los datos de `student` en `new_students` , simplemente omite la cláusula `WHERE` .

3. Verificar los datos insertados:

Una vez que hayas realizado la inserción, puedes verificar los datos en `new_students` para confirmar que los datos se hayan insertado correctamente.

```
SELECT * FROM new_students;
```

Esto insertará los datos de los estudiantes de la clase 'Seven' de la tabla `student` en la tabla `new_students` . Puedes adaptar este ejemplo a tus propias necesidades.

▼ Actividad

Cuenta cual es el numero de Mujeres y Hombres

```
SELECT sex,COUNT(sex) from student GROUP BY sex;
```

¿Cual es el promedio de la calificación.?

```
SELECT AVG(mark) from student;
```

Obtén el mínimo y máximo de la calificación.

```
SELECT min(mark) as Minimo , max(mark) as Maximo from student;
```

Inserta el registro de 'Aaron Flores', 'Two', 85, 'male'

```
--Insertamos el registro
INSERT INTO student(ID, name, "class", mark, sex)
VALUES (100, 'Aaron Flores', 'Two', 85, "male");
-- Comprobamos que el registro exista
SELECT id, name FROM student WHERE id=100;
```

Elimina el registro de "Aaron Flores"

```
--EN este caso eliminaremos el registro recién creado.
DELETE FROM student WHERE ID = 100;
```

Realizar el punto 7. (Actualizar tabla utilizando valores de otra tabla.)

Imagina que tienes otra tabla llamada `extra_marks` con columnas `ID` y `bonus_marks`. Si deseas añadir estas `bonus_marks` a la columna `mark` de la tabla `student` para cada estudiante que coincida con un ID, podrías hacerlo así:

```
--PASO 1, crear la tabla
CREATE TABLE extra_marks (
    ID INTEGER PRIMARY KEY,
    bonus_marks INTEGER
);

--PASO 2, insertar registros a la tabla.
INSERT INTO extra_marks (id, bonus_marks)
VALUES (5,10), (10,10), (13,10);

--Paso3, actualizar la tabla con los registros creados en la tabla nueva.

UPDATE student
SET mark = mark + (SELECT bonus_marks FROM extra_marks WHERE student.ID = extra_ma
rks.ID)
WHERE ID IN (SELECT ID FROM extra_marks);
```