# SQL

## Introduction to SQL and Databases

Accura Tequipment

### Introduction to Databases

- A database is an organized collection of data.

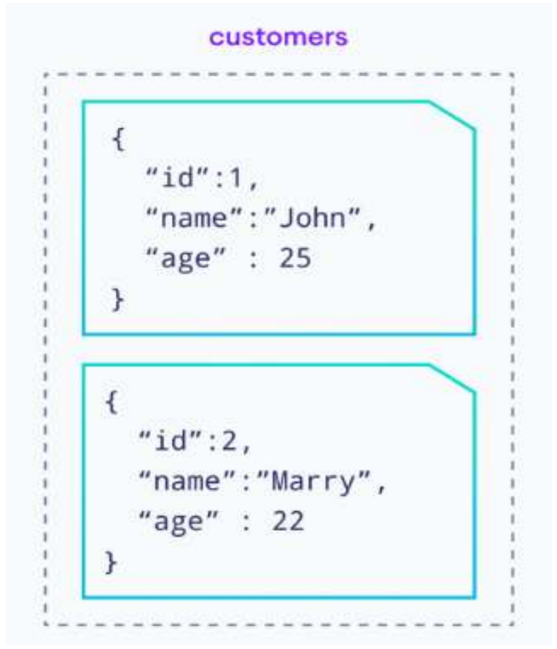
### Types of Databases

- In general, there are two common types of databases:
    1. Non-Relational
    2. Relational

## Non-Relational Database

In a non-relational database, data is stored in **key-value pairs**.

For example:



```
{
  "id":1,
  "name":"John",
  "age" : 25
}
```

```
{
  "id":2,
  "name":"Marry",
  "age" : 22
}
```

- Here, customers' data are stored in key-value pairs.
- Commonly used non-relational database management systems (Non-RDBMS) are MongoDB, Amazon DynamoDB, Redis, etc

## Relational Database

In a relational database, data is stored in **tabular format**.

For example,

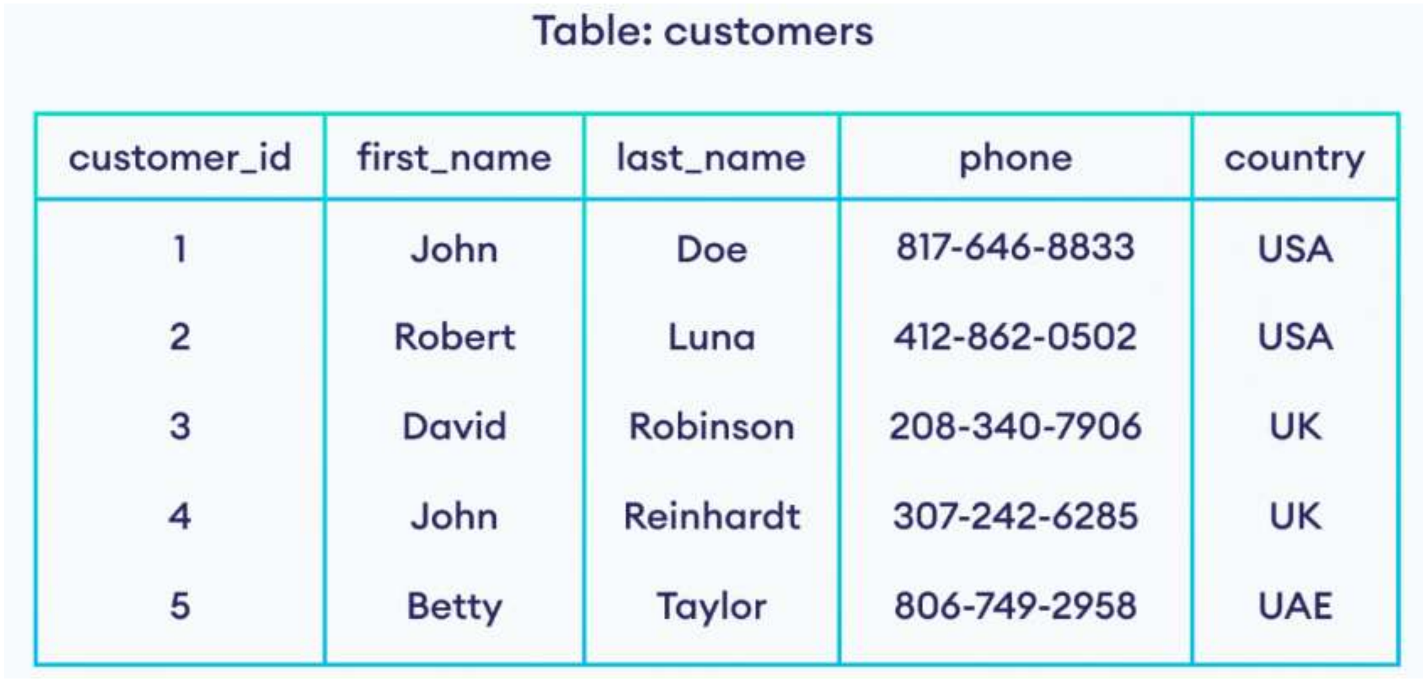


Table: customers

| customer_id | first_name | last_name | phone | country |
|---|---|---|---|---|
| 1 | John | Doe | 817-646-8833 | USA |
| 2 | Robert | Luna | 412-862-0502 | USA |
| 3 | David | Robinson | 208-340-7906 | UK |
| 4 | John | Reinhardt | 307-242-6285 | UK |
| 5 | Betty | Taylor | 806-749-2958 | UAE |

- Here, customers is a table inside the database.
- The first row is the attributes of the table. Each row after that contains the data of a customer.
- Commonly used relational database management systems (RDBMS) are MySQL, PostgreSQL, MSSQL, Oracle etc.

- In a relational database, two or more tables may be related to each other. Hence the term "**Relational**".

For example,

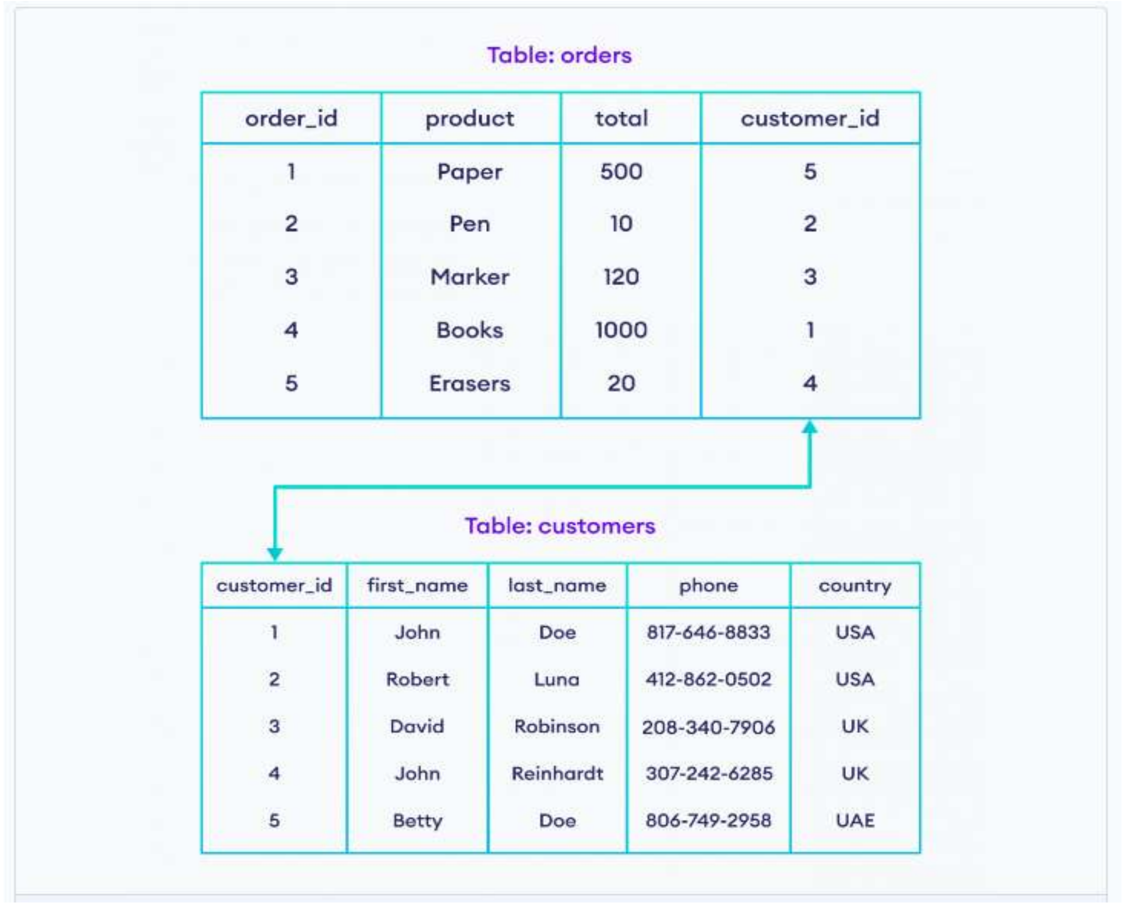


Table: orders

| order_id | product | total | customer_id |
|---|---|---|---|
| 1 | Paper | 500 | 5 |
| 2 | Pen | 10 | 2 |
| 3 | Marker | 120 | 3 |
| 4 | Books | 1000 | 1 |
| 5 | Erasers | 20 | 4 |

Table: customers

| customer_id | first_name | last_name | phone | country |
|---|---|---|---|---|
| 1 | John | Doe | 817-646-8833 | USA |
| 2 | Robert | Luna | 412-862-0502 | USA |
| 3 | David | Robinson | 208-340-7906 | UK |
| 4 | John | Reinhardt | 307-242-6285 | UK |
| 5 | Betty | Doe | 806-749-2958 | UAE |

Here, orders and customers are related through customer_id.

# SQL

- SQL (Structured Query Language) is a powerful and standard query language for relational database systems. This language is used by all the relational database systems.

- SQL allows users to create databases, add data, modify, maintain data and fetch selected data. It is governed by standards maintained by ISO (International Standards Organization).

- We use SQL to perform CRUD (Create, Read, Update, Delete) operations on databases along with other various operations like:
  - create databases
  - create tables in a database
  - read data from a table
  - insert data in a table
  - update data in a table
  - delete data from a table
  - delete database tables
  - delete databases
  - and many more database operations

## Benefits of SQL:

1. SQL is non procedural, that means you do not need to write lengthy programs to get data, there are commands, expressions and operators to use to get data from tables.

2. SQL is portable, databases using SQL can be moved from one device to another without any problems.

3. Easy to learn as SQL is in form of ENGLISH statements.

Accura Tequipment

# SQLite3

## SQLite

- SQLite is embedded relational database management system.

- It is self-contained, serverless, zero configuration and transactional SQL database engine.

- SQLite is free to use for any purpose commercial or private. In other words, “SQLite is an open source, zero-configuration, self-contained, stand alone, transaction relational database engine designed to be embedded into an application”.

- SQLite is different from other SQL databases because unlike most other SQL databases, SQLite does not have a separate server process. It reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file.

## Features of SQLite

- **SQLite is totally free:** SQLite is open-source. So, no license is required to work with it.
- **SQLite is serverless:** SQLite doesn't require a different server process or system to operate.
- **SQLite is very flexible:** It facilitates you to work on multiple databases on the same session on the same time.
- **Configuration Not Required:** SQLite doesn't require configuration. No setup or administration required.
- **SQLite is a cross-platform DBMS:** You don't need a large range of different platforms like Windows, Mac OS, Linux, and Unix. It can also be used on a lot of embedded operating systems like Symbian, and Windows CE.

- **Storing data is easy:** SQLite provides an efficient way to store data.

- **Variable length of columns:** The length of the columns is variable and is not fixed. It facilitates you to allocate only the space a field needs. For example, if you have a varchar(200) column, and you put a 10 characters' length value on it, then SQLite will allocate only 20 characters' space for that value not the whole 200 space.

- **Provide large number of API's:** SQLite provides API for a large range of programming languages. **For example:** .Net languages (Visual Basic, C#), PHP, Java, Objective C, Python and a lot of other programming language.

- **SQLite** is written in **ANSI-C** and provides simple and easy-to-use API.

- **SQLite** is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

# SQLite Advantages

- SQLite is a very popular database which has been successfully used with on disk file format for desktop applications like version control systems, financial analysis tools, media cataloging and editing suites, CAD packages, record keeping programs etc.

- There are a lot of advantages to use SQLite as an application file format:

1) Lightweight

    - SQLite is a very light weighted database so, it is easy to use it as an embedded software with devices like televisions, Mobile phones, cameras, home electronic devices, etc

2) Better Performance

    - Reading and writing operations are very fast for SQLite database. It is almost 35% faster than File system.
    - It only loads the data which is needed, rather than reading the entire file and hold it in memory.
    - If you edit small parts, it only overwrite the parts of the file which was changed.

## 3) No Installation Needed

- SQLite is very easy to learn. You don’t need to install and configure it. Just download SQLite libraries in your computer and it is ready for creating the database.

## 4) Reliable

- It updates your content continuously so, little or no work is lost in a case of power failure or crash.
- SQLite is less bugs prone rather than custom written file I/O codes.
- SQLite queries are smaller than equivalent procedural codes so, chances of bugs are minimal.

## 5) Portable

- SQLite is portable across all 32-bit and 64-bit operating systems and big- and little-endian architectures.
- Multiple processes can be attached with same application file and can read and write without interfering each other.
- It can be used with all programming languages without any compatibility issue.

6) Accessible

- SQLite database is accessible through a wide variety of third-party tools.
- SQLite database's content is more likely to be recoverable if it has been lost. Data lives longer than code.

7) Reduce Cost and Complexity

- It reduces application cost because content can be accessed and updated using concise SQL queries instead of lengthy and error-prone procedural queries.
- SQLite can be easily extended in in future releases just by adding new tables and/or columns. It also preserve the backwards compatibility.

## SQLite Disadvantages

- SQLite is used to handle low to medium traffic HTTP requests.
- Database size is restricted to 2GB in most cases.

# Differences between SQL and SQLite

| SQL | SQLite |
|---|---|
| SQL is Structured Query Language used to query Relational Database System. It is written in C language. | SQLite is an Relational Database Management System which is written in ANSI-C. |
| SQL is standard which specifies how relational schema is created, data is inserted or updated in relations, transactions are started and stopped, etc. | SQLite is file-based. It is different from other SQL databases because unlike most other SQL databases, SQLite does not have separate server process. |
| Main components of SQL are Data Definition Language(DDL), Data Manipulation Language(DML), Data Control Language(DCL). | SQLite supports many features of SQL and has high performance but does not support stored procedures. |
| SQL is Structured Query Language which is used with databases like MySQL, Oracle, Microsoft SQL Server, IBM DB2, etc. | SQLite is portable database resource. It could get an extension in whatever programming language used to access that database. |
| A conventional SQL database needs to be running as service like Oracle DB to connect to and provide lot of functionalities. | SQLite database system does not provide such functionalities. |
| SQL is query language which is used by other SQL databases. It is not database itself. | SQLite is relational database management system itself which uses SQL. |

# SQLite Commands

SQLite commands are similar to SQL commands. There are three types of SQLite commands:

- **DDL:** Data Definition Language
- **DML:** Data Manipulation Language
- **DQL:** Data Query Language

## Data Definition Language

There are three commands in this group:

- **CREATE:** This command is used to create a table, a view of a table or other object in the database.
- **ALTER:** It is used to modify an existing database object like a table.
- **DROP:** The DROP command is used to delete an entire table, a view of a table or other object in the database.

Accura Tequipment

## Data Manipulation language

There are three commands in data manipulation language group:

- **INSERT:** This command is used to create a record.
- **UPDATE:** It is used to modify the records.
- **DELETE:** It is used to delete records.

## Data Query Language

- **SELECT:** This command is used to retrieve certain records from one or more table.

## SQLite dot Command

The SQLite dot commands are not terminated by a semicolon (;)

## Creating a Table in SQLite3

Step 1: Start SQLite3

import sqlite3

Step 2: Create a New Database

sqlite3 DatabaseName.db

Step 3: Create a New Table

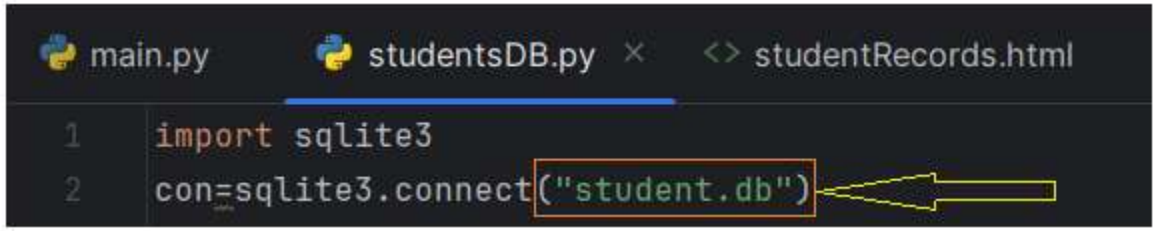
To create a table in SQLite3, you use the __CREATE TABLE` statement.

# SQLite Create Database

- In SQLite, the sqlite3 command is used to create a new database.

  **Syntax:**

  sqlite3 DatabaseName.db
- Your database name should be unique within the RDBMS.
- The sqlite3 command is used to create database. But, if the database does not exist then a new database file with the given name will be created automatically.
- Example



```
import sqlite3
con=sqlite3.connect("student.db")
```

# CREATE Table

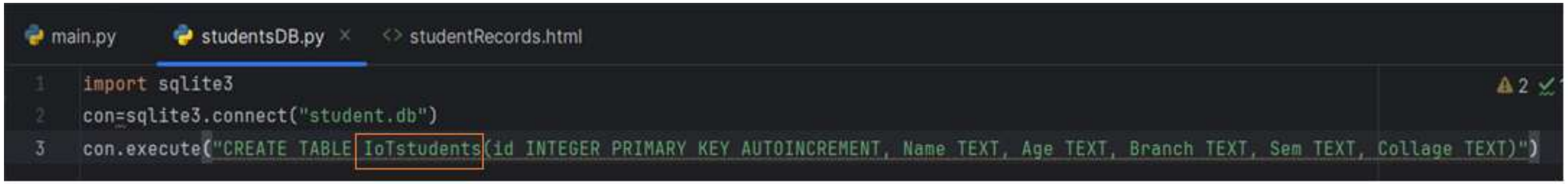
- To create a table in SQLite3, you use the __CREATE TABLE` statement. The general syntax is:

```
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY(one or more columns)
);
```

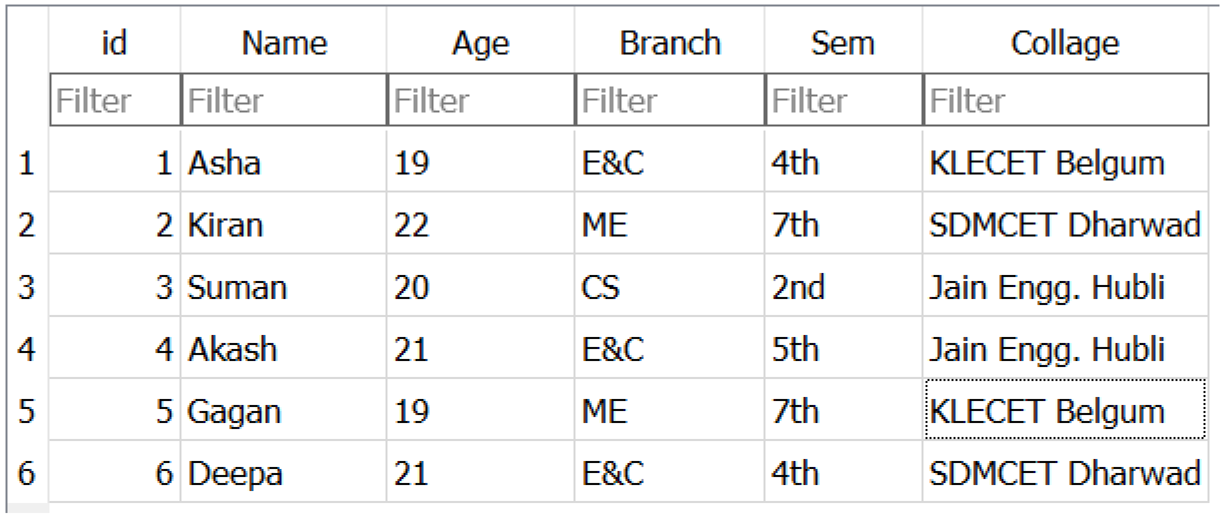
**For example:**

```
import sqlite3
con=sqlite3.connect("student.db")
con.execute("CREATE TABLE IoTstudents(id INTEGER PRIMARY KEY AUTOINCREMENT,
Name TEXT, Age TEXT, Branch TEXT, Sem TEXT, Collage TEXT)")
```

- Table with the name **IoTstudents**



```python
import sqlite3
con=sqlite3.connect("student.db")
con.execute("CREATE TABLE IoTstudents(id INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT, Age TEXT, Branch TEXT, Sem TEXT, Collage TEXT)")
```



| | id | Name | Age | Branch | Sem | Collage |
|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | Asha | 19 | E&C | 4th | KLECET Belgum |
| 2 | 2 | Kiran | 22 | ME | 7th | SDMCET Dharwad |
| 3 | 3 | Suman | 20 | CS | 2nd | Jain Engg. Hubli |
| 4 | 4 | Akash | 21 | E&C | 5th | Jain Engg. Hubli |
| 5 | 5 | Gagan | 19 | ME | 7th | KLECET Belgum |
| 6 | 6 | Deepa | 21 | E&C | 4th | SDMCET Dharwad |

## SQLite Insert

- To insert data into a table, you use the INSERT statement. SQLite provides various forms of the INSERT statements that allow you to insert a single row, multiple rows, and default values into a table.

- In addition, you can insert a row into a table using data provided by a SELECT statement.

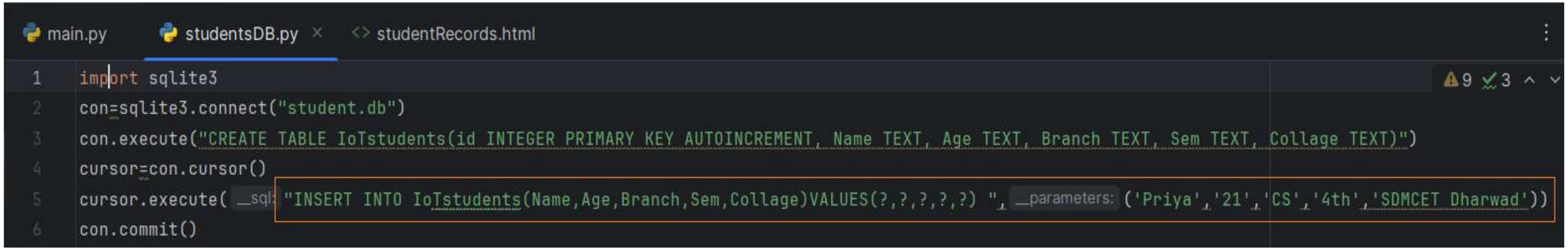
- **SQLite INSERT – inserting a single row into a table**

  To insert a single row into a table, you use the following form of the INSERT statement:

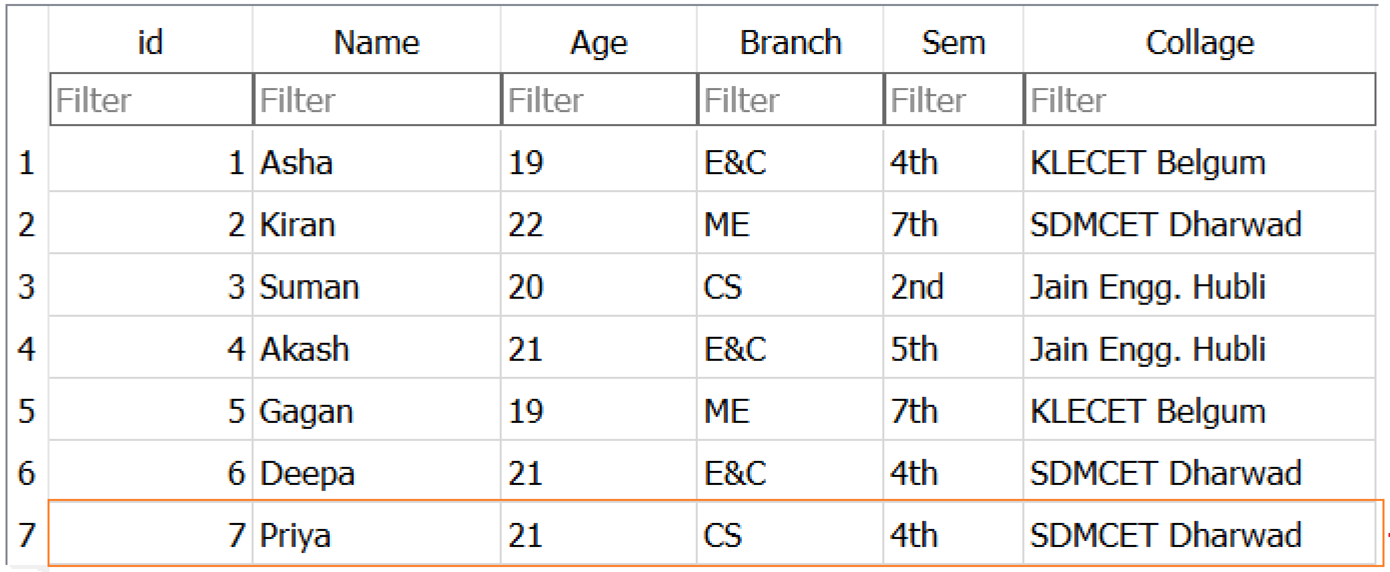
  ```
  INSERT INTO table (column1,column2 ,..) VALUES( value1,value2 ,...);
  ```

  - First, specify the name of the table to which you want to insert data after the INSERT INTO keywords.
  - Second, add a comma-separated list of columns after the table name. The column list is optional. However, it is a good practice to include the column list after the table name.
  - Third, add a comma-separated list of values after the VALUES keyword. If you omit the column list, you have to specify values for all columns in the value list. The number of values in the value list must be the same as the number of columns in the column list.

## Example for INSERT statement:



```
import sqlite3
con=sqlite3.connect("student.db")
con.execute("CREATE TABLE IoTstudents(id INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT, Age TEXT, Branch TEXT, Sem TEXT, Collage TEXT)")
cursor=con.cursor()
cursor.execute( __sql: "INSERT INTO IoTstudents(Name,Age,Branch,Sem,Collage)VALUES(?,?,?,?,?) ", __parameters: ('Priya','21','CS','4th','SDMCET Dharwad'))
con.commit()
```



| | id | Name | Age | Branch | Sem | Collage |
|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | Asha | 19 | E&C | 4th | KLECET Belgum |
| 2 | 2 | Kiran | 22 | ME | 7th | SDMCET Dharwad |
| 3 | 3 | Suman | 20 | CS | 2nd | Jain Engg. Hubli |
| 4 | 4 | Akash | 21 | E&C | 5th | Jain Engg. Hubli |
| 5 | 5 | Gagan | 19 | ME | 7th | KLECET Belgum |
| 6 | 6 | Deepa | 21 | E&C | 4th | SDMCET Dharwad |
| 7 | 7 | Priya | 21 | CS | 4th | SDMCET Dharwad |

# SQLite Update

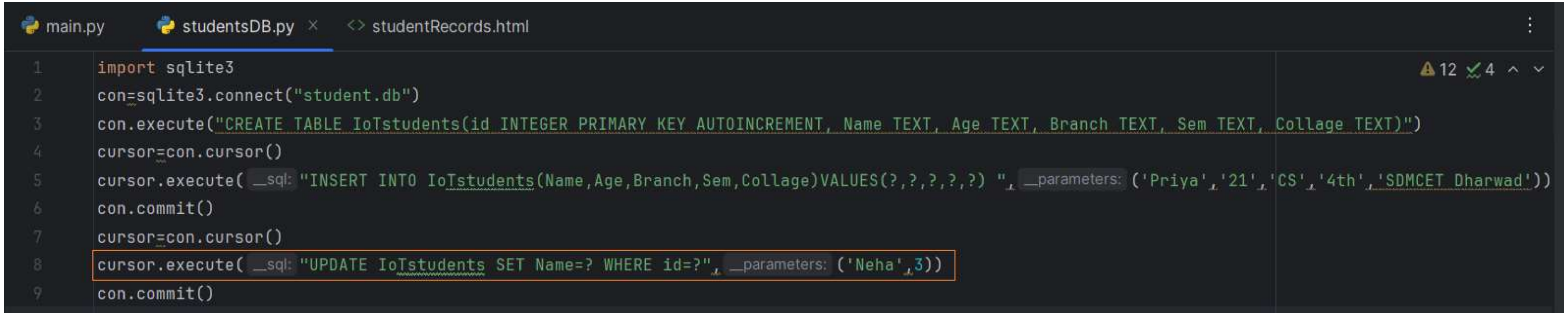
- To update existing data in a table, you use SQLite UPDATE statement.
- The syntax of the UPDATE statement is:

```
UPDATE table
SET column_1 = new_value_1,
    column_2 = new_value_2
WHERE
    search_condition
ORDER column_or_expression
LIMIT row_count OFFSET offset;
```

In this syntax:

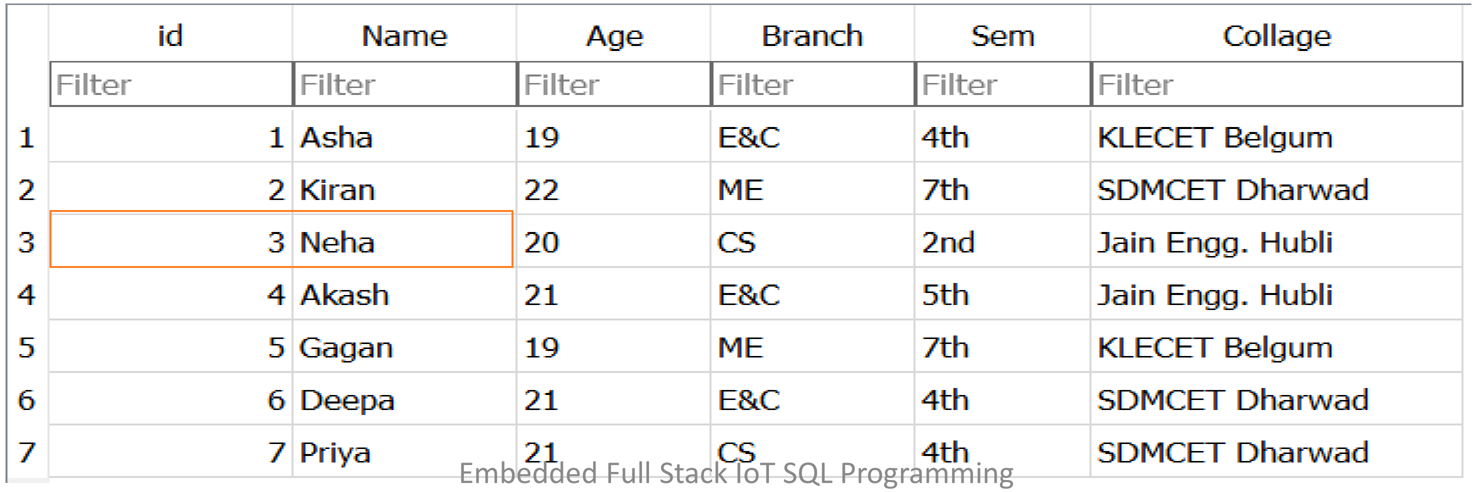
- First, specify the table where you want to update after the UPDATE clause.
- Second, set new value for each column of the table in the SET clause.
- Third, specify rows to update using a condition in the WHERE clause. The WHERE clause is optional. If you skip it, the UPDATE statement will update data in all rows of the table.
- Finally, use the ORDER BY and LIMIT clauses in the UPDATE statement to specify the number of rows to update.
- Notice that if use a negative value in the LIMIT clause, SQLite assumes that there are no limit and updates all rows that meet the condition in the preceding WHERE clause.
- The ORDER BY clause should always goes with the LIMIT clause to specify exactly which rows to be updated. Otherwise, you will never know which row will be actually updated; because without the ORDER BY clause, the order of rows in the table is unspecified.

## Example for UPDATE statement


main.py   studentsDB.py ×   <> studentRecords.html

```
import sqlite3
con=sqlite3.connect("student.db")
con.execute("CREATE TABLE IoTstudents(id INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT, Age TEXT, Branch TEXT, Sem TEXT, Collage TEXT)")
cursor=con.cursor()
cursor.execute( _sql: "INSERT INTO IoTstudents(Name,Age,Branch,Sem,Collage)VALUES(?,?,?,?,?) ", _parameters: ('Priya','21','CS','4th','SDMCET Dharwad'))
con.commit()
cursor=con.cursor()
cursor.execute( _sql: "UPDATE IoTstudents SET Name=? WHERE id=?", _parameters: ('Neha',3))
con.commit()
```

The value is updated WHERE id=3 in the table IoTstudents



| | id | Name | Age | Branch | Sem | Collage |
|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | Asha | 19 | E&C | 4th | KLECET Belgum |
| 2 | 2 | Kiran | 22 | ME | 7th | SDMCET Dharwad |
| 3 | 3 | Neha | 20 | CS | 2nd | Jain Engg. Hubli |
| 4 | 4 | Akash | 21 | E&C | 5th | Jain Engg. Hubli |
| 5 | 5 | Gagan | 19 | ME | 7th | KLECET Belgum |
| 6 | 6 | Deepa | 21 | E&C | 4th | SDMCET Dharwad |
| 7 | 7 | Priya | 21 | CS | 4th | SDMCET Dharwad |

## SQLite Select

- SQLite SELECT statement is used to fetch the data from a SQLite database table which returns data in the form of a result table. These result tables are also called result sets.

- The syntax of SQLite SELECT statement.

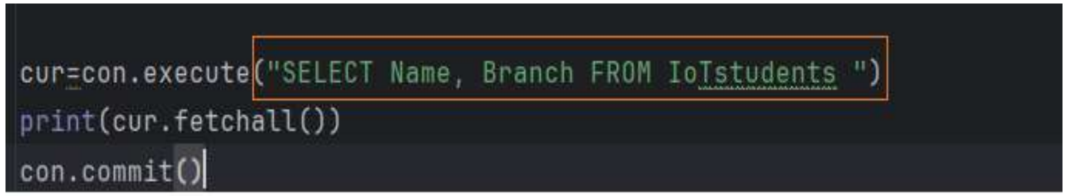
```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2 ... are the fields of a table, whose values you want to fetch.

- If you want to fetch all the fields available in the field, then you can use the following syntax

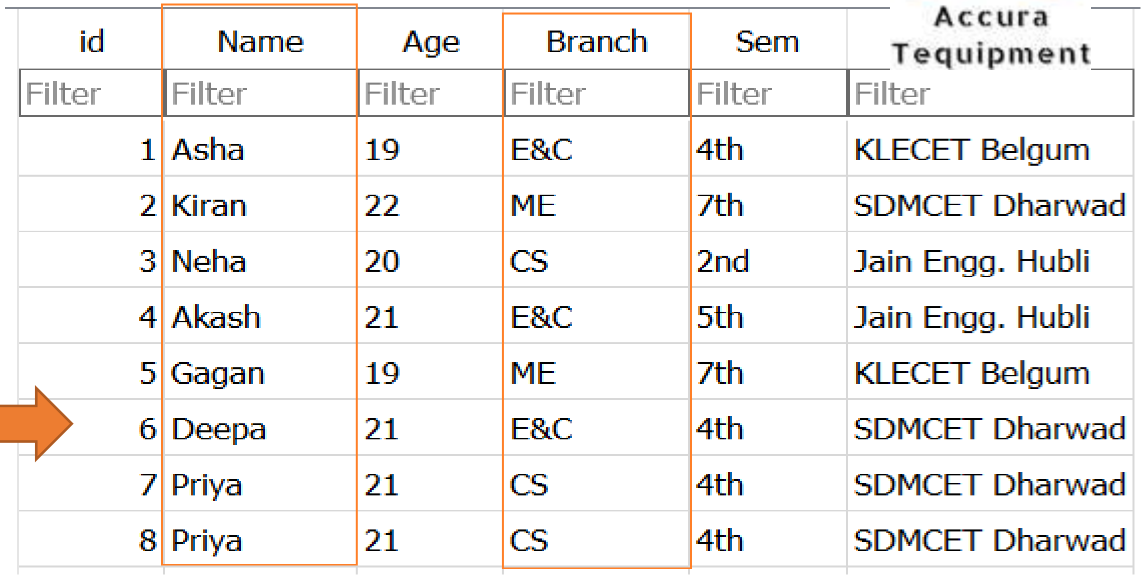
```
SELECT * FROM table_name;
```

## Example for SELECT statement:



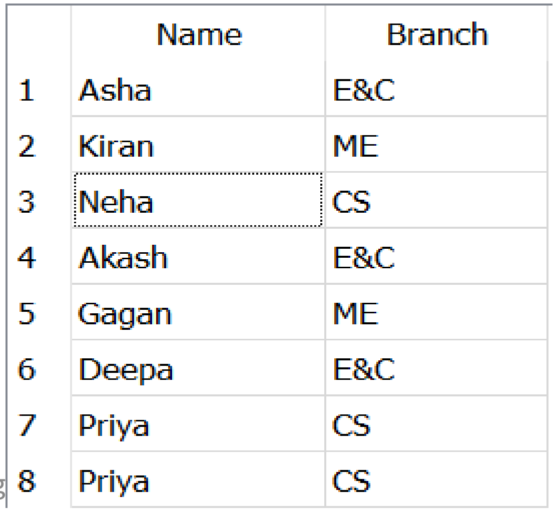
```
cur=con.execute("SELECT Name, Branch FROM IoTstudents ")
print(cur.fetchall())
con.commit()
```

The Name and Branch is selected FROM IoTstudents



| id | Name | Age | Branch | Sem | |
|---|---|---|---|---|---|
| Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | Asha | 19 | E&C | 4th | KLECET Belgum |
| 2 | Kiran | 22 | ME | 7th | SDMCET Dharwad |
| 3 | Neha | 20 | CS | 2nd | Jain Engg. Hubli |
| 4 | Akash | 21 | E&C | 5th | Jain Engg. Hubli |
| 5 | Gagan | 19 | ME | 7th | KLECET Belgum |
| 6 | Deepa | 21 | E&C | 4th | SDMCET Dharwad |
| 7 | Priya | 21 | CS | 4th | SDMCET Dharwad |
| 8 | Priya | 21 | CS | 4th | SDMCET Dharwad |

```
SELECT Name, Branch FROM IoTstudents
```



| | Name | Branch |
|---|---|---|
| 1 | Asha | E&C |
| 2 | Kiran | ME |
| 3 | Neha | CS |
| 4 | Akash | E&C |
| 5 | Gagan | ME |
| 6 | Deepa | E&C |
| 7 | Priya | CS |
| 8 | Priya | CS |

# SQLite Delete

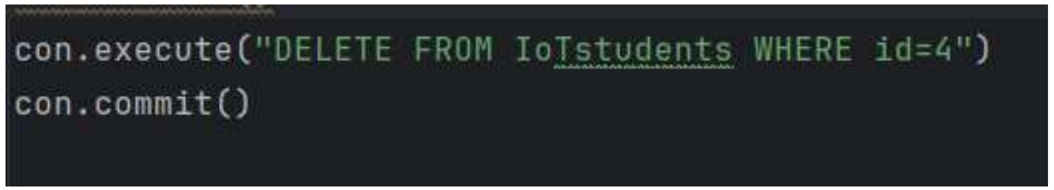
- The SQLite DELETE statement is used to remove rows from a table
- The SQLite DELETE statement allows you to delete one row, multiple rows, and all rows in a table.
- The syntax of the SQLite DELETE statement is as follows:

```
DELETE FROM table WHERE search_condition;
```

In this syntax:

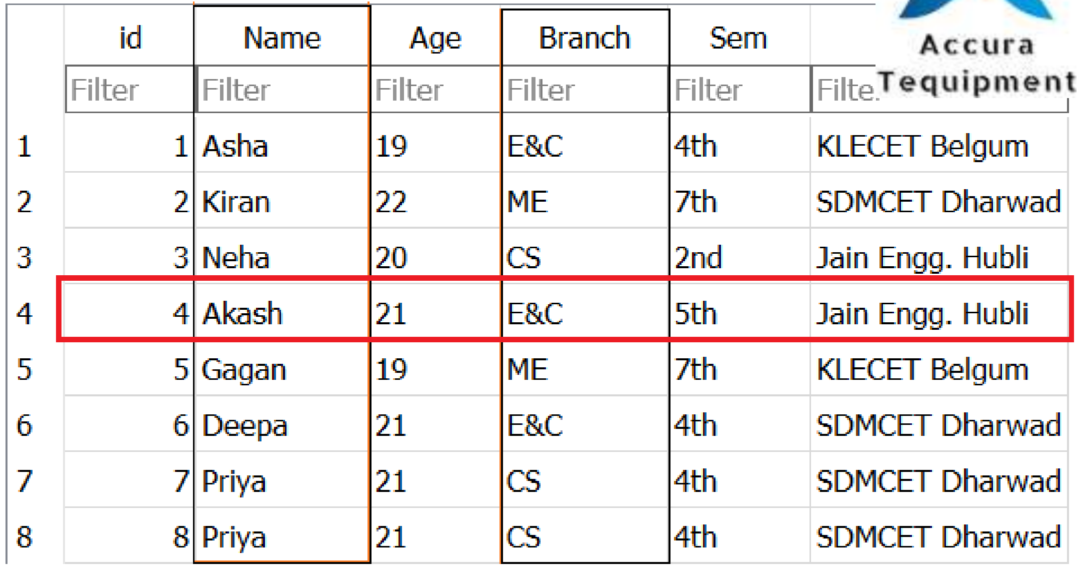
- First, specify the name of the table which you want to remove rows after the DELETE FROM keywords.
- Second, add a search condition in the WHERE clause to identify the rows to remove. The WHERE clause is an optional part of the DELETE statement. If you omit the WHERE clause, the DELETE statement will delete all rows in the table.

## Example for DELETE statement



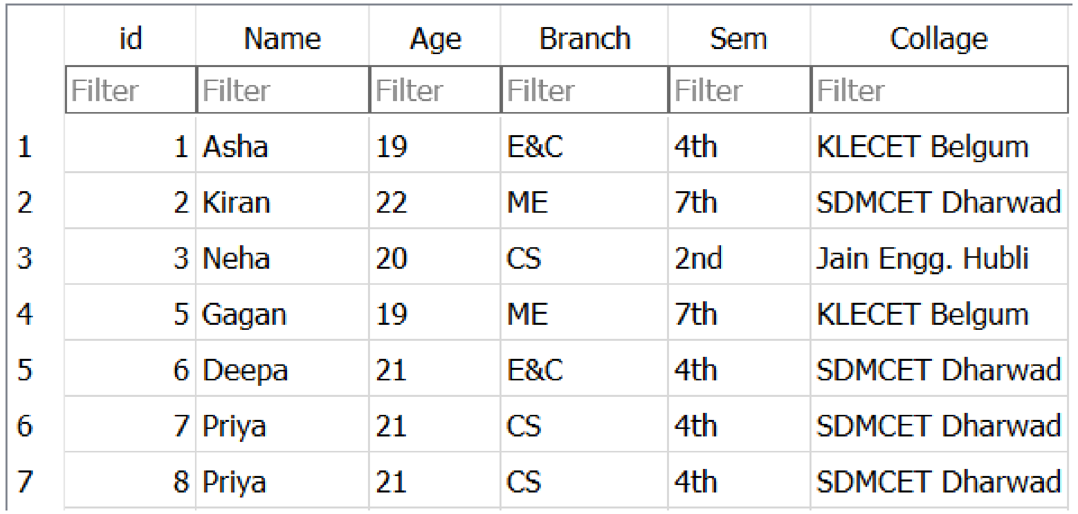
```
con.execute("DELETE FROM IoTstudents WHERE id=4")
con.commit()
```

The id=4 is deleted FROM IoTstudents



| | id | Name | Age | Branch | Sem | |
|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filte |
| 1 | 1 | Asha | 19 | E&C | 4th | KLECET Belgum |
| 2 | 2 | Kiran | 22 | ME | 7th | SDMCET Dharwad |
| 3 | 3 | Neha | 20 | CS | 2nd | Jain Engg. Hubli |
| 4 | 4 | Akash | 21 | E&C | 5th | Jain Engg. Hubli |
| 5 | 5 | Gagan | 19 | ME | 7th | KLECET Belgum |
| 6 | 6 | Deepa | 21 | E&C | 4th | SDMCET Dharwad |
| 7 | 7 | Priya | 21 | CS | 4th | SDMCET Dharwad |
| 8 | 8 | Priya | 21 | CS | 4th | SDMCET Dharwad |

**DELETE FROM IoTstudents WHERE id=4;**



| | id | Name | Age | Branch | Sem | Collage |
|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | Asha | 19 | E&C | 4th | KLECET Belgum |
| 2 | 2 | Kiran | 22 | ME | 7th | SDMCET Dharwad |
| 3 | 3 | Neha | 20 | CS | 2nd | Jain Engg. Hubli |
| 4 | 5 | Gagan | 19 | ME | 7th | KLECET Belgum |
| 5 | 6 | Deepa | 21 | E&C | 4th | SDMCET Dharwad |
| 6 | 7 | Priya | 21 | CS | 4th | SDMCET Dharwad |
| 7 | 8 | Priya | 21 | CS | 4th | SDMCET Dharwad |

# SQLite Drop Table

- SQLite **DROP TABLE** statement is used to remove a table definition and all associated data, indexes, triggers, constraints, and permission specifications for that table.

- To remove a table in a database, use SQLite DROP TABLE statement.

```
DROP TABLE [IF EXISTS] [schema_name.]table_name;
```

- SQLite allows you to drop only one table at a time. To remove multiple tables, you need to issue multiple DROP TABLE statements.

- If you remove a non-existing table, SQLite issues an error. If you use IF EXISTS option, then SQLite removes the table only if the table exists, otherwise, it just ignores the statement and does nothing.

- If you want to remove a table in a specific database, you use the [schema_name.] explicitly.

- In case the table has dependent objects such as triggers and indexes, the DROP TABLE statement also removes all the dependent objects.

# SQLite Data Types

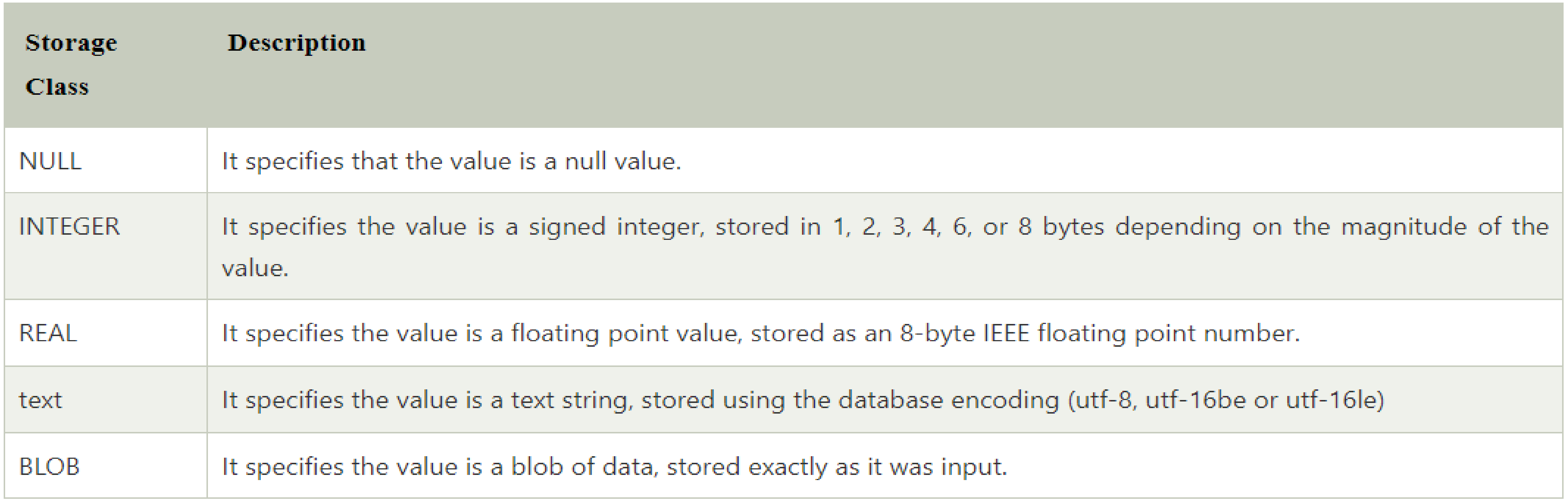
- SQLite data types are used to specify type of data of any object.
- A  data type is an attribute that tells the type of data that a column can store in a database. It defines the kind of values` that a column can accept. Different data types can include integers, decimal numbers, strings, dates, etc.
- Each column, variable and expression has related data type in SQLite. These data types are used while creating table.
- SQLite uses a more general dynamic type system. In SQLite, the datatype of a value is associated with the value itself, not with its container.

Types of SQLite data types

1. SQLite Storage Classes
2. SQLite Afinity Types
3. Date and Time Data Type
4. Boolean Data Type

## SQLite Storage Classes

- The stored values in a SQLite database has one of the following storage classes:

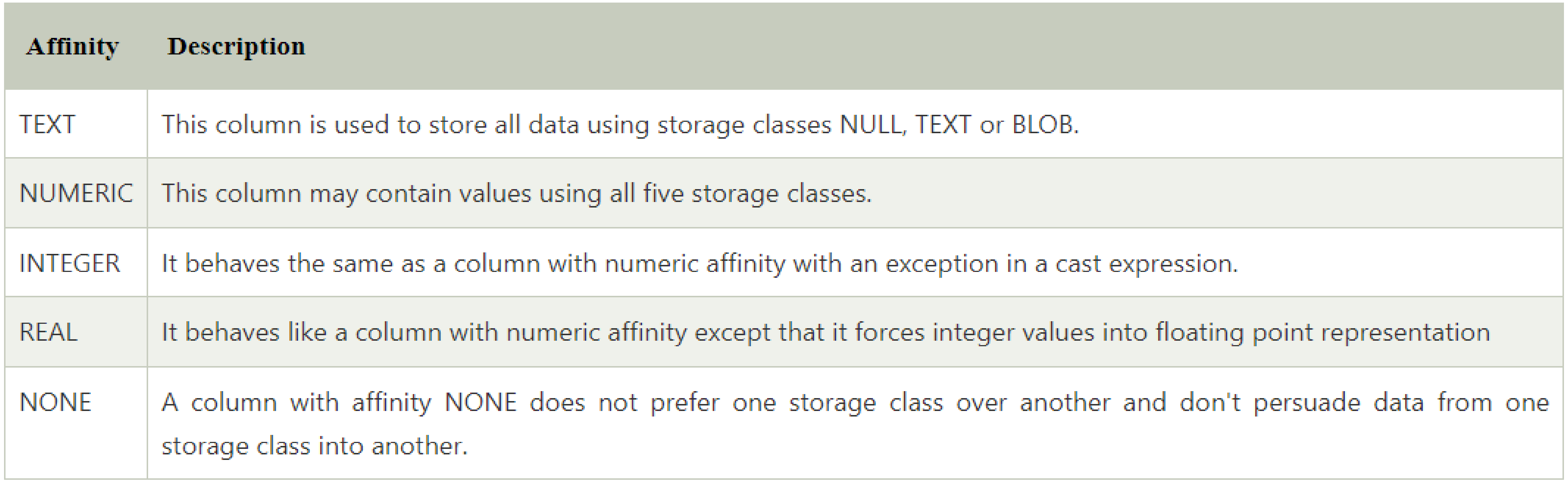


| Storage Class | Description |
|---|---|
| NULL | It specifies that the value is a null value. |
| INTEGER | It specifies the value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value. |
| REAL | It specifies the value is a floating point value, stored as an 8-byte IEEE floating point number. |
| text | It specifies the value is a text string, stored using the database encoding (utf-8, utf-16be or utf-16le) |
| BLOB | It specifies the value is a blob of data, stored exactly as it was input. |

- SQLite storage class is slightly more general than a data type. For example: The INTEGER storage class includes 6 different integer data types of different lengths.

## SQLite Afinity Types

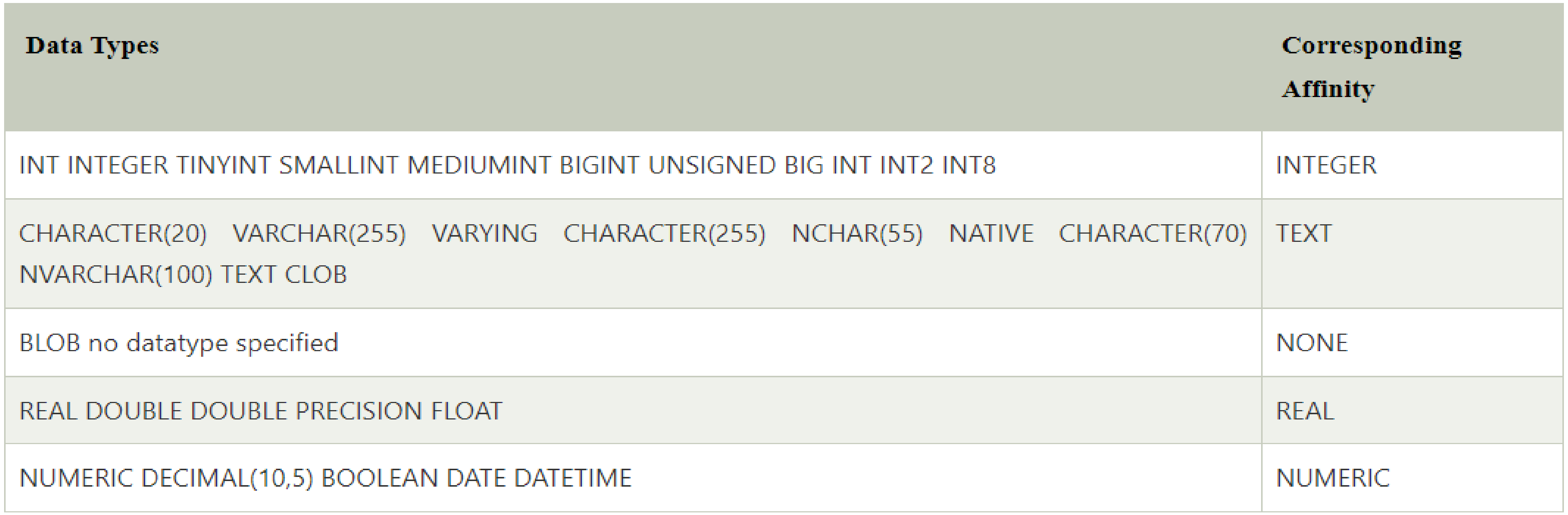
- SQLite supports type affinity for columns. Any column can still store any type of data but the preferred storage class for a column is called its affinity.
- There are following type affinity used to assign in SQLite3 database.



| Affinity | Description |
|---|---|
| TEXT | This column is used to store all data using storage classes NULL, TEXT or BLOB. |
| NUMERIC | This column may contain values using all five storage classes. |
| INTEGER | It behaves the same as a column with numeric affinity with an exception in a cast expression. |
| REAL | It behaves like a column with numeric affinity except that it forces integer values into floating point representation |
| NONE | A column with affinity NONE does not prefer one storage class over another and don't persuade data from one storage class into another. |

## SQLite Affinity and Type Names

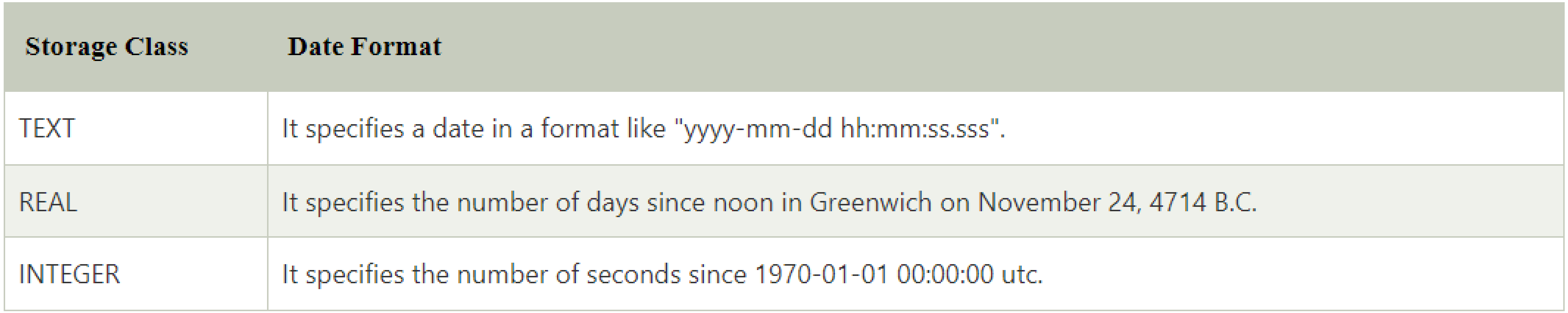
- Following is a list of various data types names which can be used while creating SQLite tables.



| Data Types | Corresponding Affinity |
|---|---|
| INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8 | INTEGER |
| CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB | TEXT |
| BLOB no datatype specified | NONE |
| REAL DOUBLE DOUBLE PRECISION FLOAT | REAL |
| NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME | NUMERIC |

## Date and Time Data Type

- In SQLite, there is no separate class to store dates and times. But you can store date and times as TEXT, REAL or INTEGER values.



| Storage Class | Date Format |
|---|---|
| TEXT | It specifies a date in a format like "yyyy-mm-dd hh:mm:ss.sss". |
| REAL | It specifies the number of days since noon in Greenwich on November 24, 4714 B.C. |
| INTEGER | It specifies the number of seconds since 1970-01-01 00:00:00 utc. |

## Boolean Data Type

- In SQLite, there is not a separate Boolean storage class. Instead, Boolean values are stored as integers 0 (false) and 1 (true).

# SQLite Operators

- SQLite operators are reserved words or characters used in SQLite statements when we use WHERE clause to perform operations like comparisons and arithmetic operations.

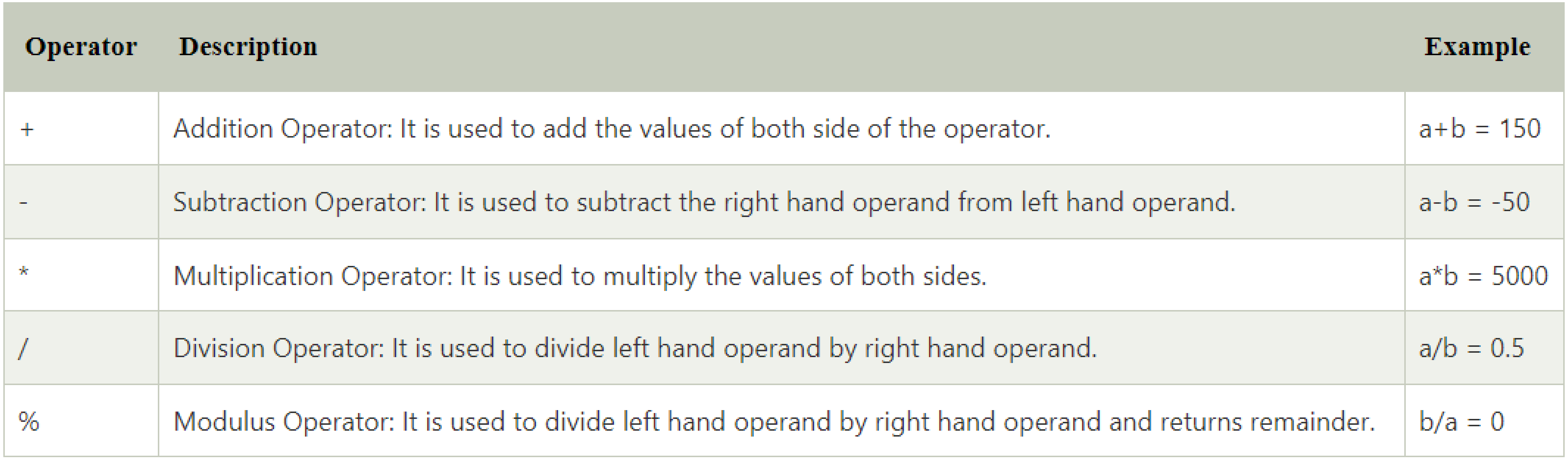
- Operators can be used to specify conditions and as conjunction for multiple conditions in SQLite statements.

There are mainly 4 type of operators in SQLite:

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Bitwise operators

## SQLite Arithmetic Operators

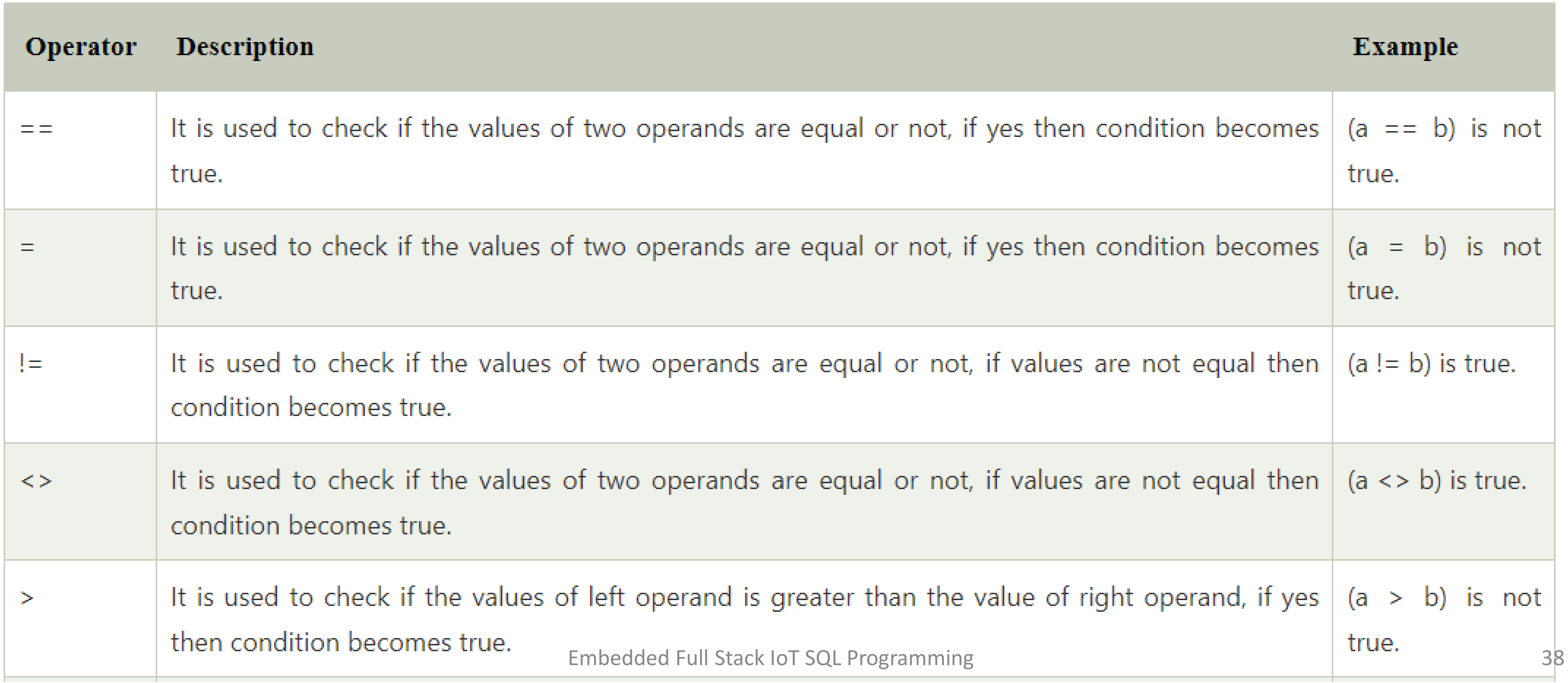
- The table specifies the different arithmetic operators in SQLite. In this table, we have two variables "a" and "b" holding values 50 and 100 respectively.



| Operator | Description | Example |
|---|---|---|
| + | Addition Operator: It is used to add the values of both side of the operator. | a+b = 150 |
| - | Subtraction Operator: It is used to subtract the right hand operand from left hand operand. | a-b = -50 |
| * | Multiplication Operator: It is used to multiply the values of both sides. | a*b = 5000 |
| / | Division Operator: It is used to divide left hand operand by right hand operand. | a/b = 0.5 |
| % | Modulus Operator: It is used to divide left hand operand by right hand operand and returns remainder. | b/a = 0 |

# SQLite Comparison Operator

- The table specifies the different comparison operators in SQLite. In this table, we have two variables "a" and "b" holding values 50 and 100 respectively.

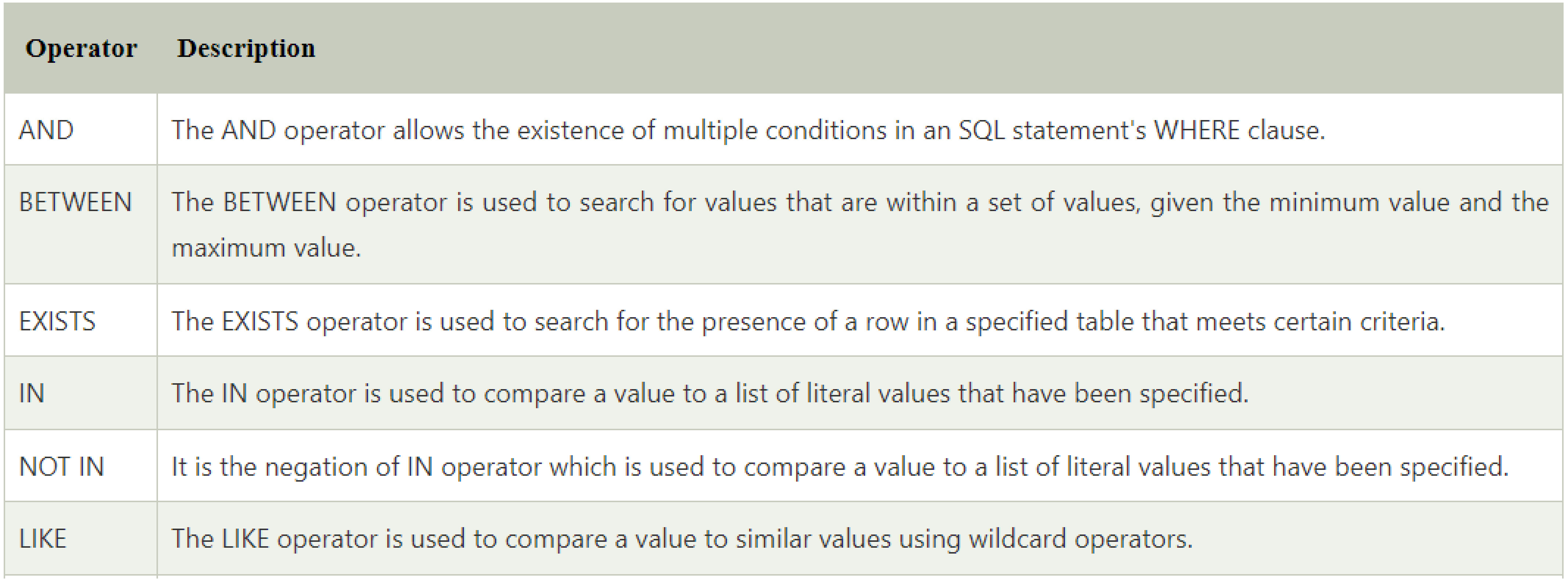


| Operator | Description | Example |
|---|---|---|
| == | It is used to check if the values of two operands are equal or not, if yes then condition becomes true. | (a == b) is not true. |
| = | It is used to check if the values of two operands are equal or not, if yes then condition becomes true. | (a = b) is not true. |
| != | It is used to check if the values of two operands are equal or not, if values are not equal then condition becomes true. | (a != b) is true. |
| <> | It is used to check if the values of two operands are equal or not, if values are not equal then condition becomes true. | (a <> b) is true. |
| > | It is used to check if the values of left operand is greater than the value of right operand, if yes then condition becomes true. | (a > b) is not true. |

| | | |
|---|---|---|
| < | It is used to check if the values of left operand is less than the value of right operand, if yes then condition becomes true. | (a < b) is true. |
| >= | It is used to check if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (a >= b) is not true. |
| <= | It is used to check if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (a <= b) is true. |
| !< | It is used to check if the value of left operand is not less than the value of right operand, if yes then condition becomes true. | (a !< b) is false. |
| !> | It is used to check if the value of left operand is not greater than the value of right operand, if yes then condition becomes true. | (a !> b) is true. |

## SQLite Logical Operator

Following is a list of logical operators in SQLite:

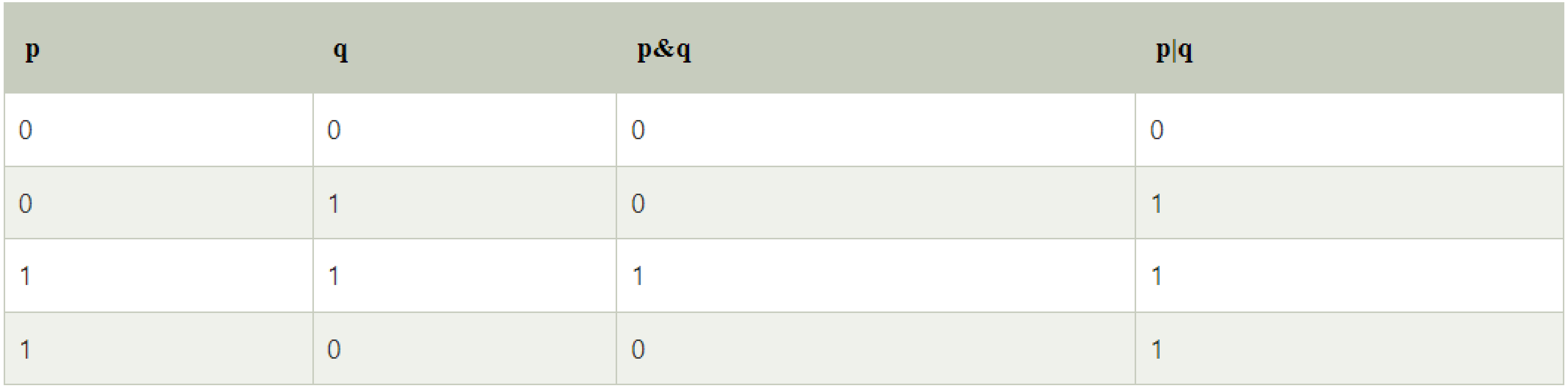


| Operator | Description |
|---|---|
| AND | The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause. |
| BETWEEN | The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value. |
| EXISTS | The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria. |
| IN | The IN operator is used to compare a value to a list of literal values that have been specified. |
| NOT IN | It is the negation of IN operator which is used to compare a value to a list of literal values that have been specified. |
| LIKE | The LIKE operator is used to compare a value to similar values using wildcard operators. |

|  |  |
|---|---|
| GLOB | The GLOB operator is used to compare a value to similar values using wildcard operators. Also, glob is case sensitive, unlike like. |
| NOT | The NOT operator reverses the meaning of the logical operator with which it is used. For example: EXISTS, NOT BETWEEN, NOT IN, etc. These are known as negate operator. |
| OR | The OR operator is used to combine multiple conditions in an SQL statement's where clause. |
| IS NULL | The NULL operator is used to compare a value with a null value. |
| IS | The IS operator work like = |
| IS NOT | The IS NOT operator work like != |
| \|\| | This operator is used to add two different strings and make new one. |
| UNIQUE | The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates). |

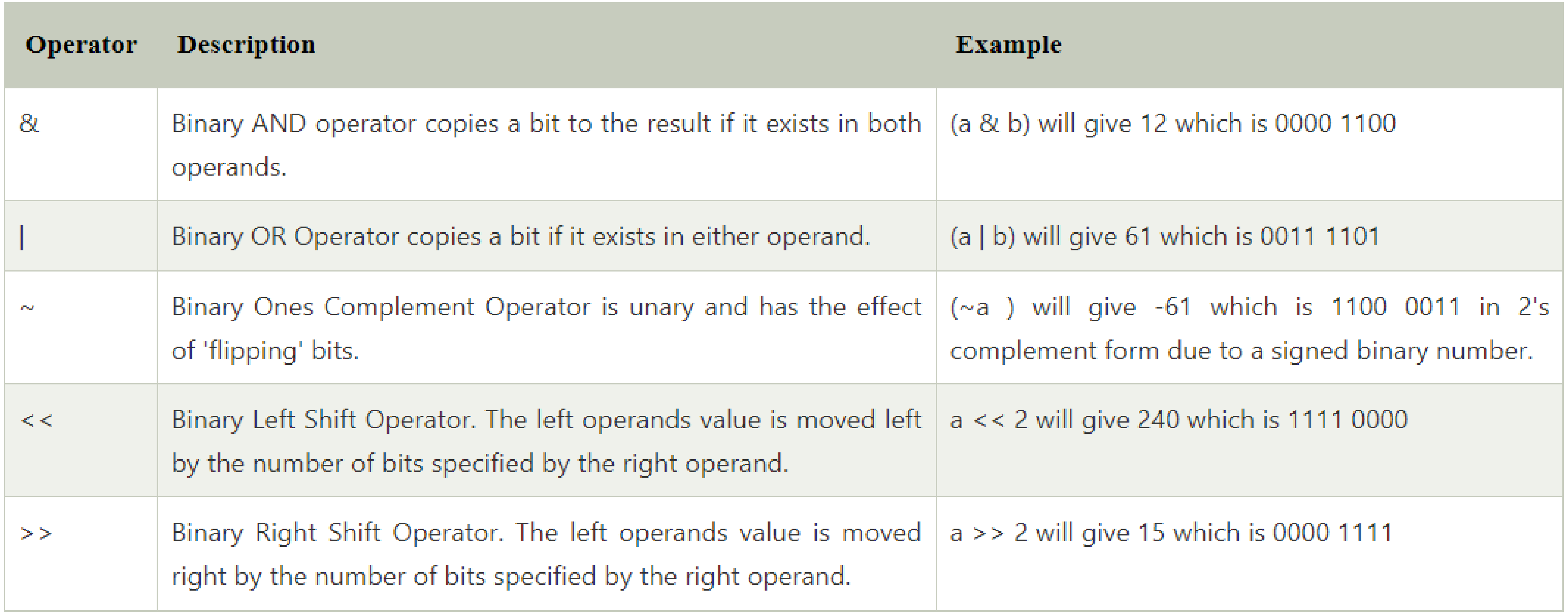
# SQLite Bitwise Operators

- SQLite Bitwise operators work on bits and perform bit by bit operation.
- See the truth table for Binary AND (&) and Binary OR (|):



| p | q | p&q | p\|q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |

- Assume two variables "a" and "b", having values 60 and 13 respectively. So Binary values of a and b are:  a= 0011 1100, b= 0000 1101



| Operator | Description | Example |
|---|---|---|
| & | Binary AND operator copies a bit to the result if it exists in both operands. | (a & b) will give 12 which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (a \| b) will give 61 which is 0011 1101 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~a ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | a << 2 will give 240 which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 will give 15 which is 0000 1111 |

## SQLite Syntax

Syntax is a unique set of rules and guidelines.

**Case sensitivity:**

- SQLite is not case sensitive. But, there are some commands which are case sensitive.
- For example: GLOB and glob have different meaning in SQLite statements.

**Comments:**

- Comments are used to add more readability in your SQLite code.
- Comments cannot be nested.
- Comments begin with two consecutive "-" characters.
- Sometimes it also appears with "/*" and extend up to and including the next "*/" character pair.

## SQLite Statements

- All the SQLite statement is started with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, etc. All the statement will end with a semicolon (;).

## SQLite ANALYZE Statement

Syntax:

```
ANALYZE;
        or
ANALYZE  database_name;
        or
ANALYZE  database_name.table_name;
```

## SQLite AND/OR Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

## SQLite ALTER TABLE Statement

Syntax:

```
ALTER TABLE table_name ADD COLUMN column_def...;
```

## SQLite ALTER TABLE Statement (Rename)

Syntax:

```
ALTER TABLE table_name RENAME TO new_table_name;
```

## SQLite ATTACH DATABASE Statement

Syntax:

```
ATTACH DATABASE 'DatabaseName' As 'Alias-Name';
```

## SQLite BEGIN TRANSACTION Statement

Syntax:

```
BEGIN;
   or
BEGIN EXCLUSIVE TRANSACTION;
```

## SQLite BETWEEN Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name BETWEEN val-1 AND val-2;
SQLite COMMIT Statement:
COMMIT;
```

## SQLite CREATE INDEX Statement

Syntax:

```
CREATE INDEX index_name
ON table_name ( column_name COLLATE NOCASE );
```

## SQLite CREATE UNIQUE INDEX Statement

Syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...columnN);
```

## SQLite CREATE TABLE Statement

Syntax:

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY( one or more columns ));
```

## SQLite CREATE TRIGGER Statement

Syntax:

```
CREATE TRIGGER database_name.trigger_name
BEFORE INSERT ON table_name FOR EACH ROW
BEGIN
  stmt1;
  stmt2;
  ....
END;
```

## SQLite CREATE VIEW Statement

Syntax:

```
CREATE VIEW database_name.view_name  AS
SELECT statement....;
```

## SQLite CREATE VIRTUAL TABLE Statement

Syntax:

```
CREATE VIRTUAL TABLE database_name.table_name USING weblog( access.log );
                  or
CREATE VIRTUAL TABLE database_name.table_name USING fts3( );
```

## SQLite COMMIT TRANSACTION Statement

Syntax:

```
COMMIT;
```

## SQLite COUNT Clause

Syntax:

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

## SQLite DELETE Statement

Syntax:

```
DELETE FROM table_name
WHERE  {CONDITION};
```

## SQLite DETACH DATABASE Statement

Syntax:

```
DETACH DATABASE 'Alias-Name';
```

## SQLite DISTINCT Clause

Syntax:

```
SELECT DISTINCT column1, column2....columnN
FROM   table_name;
```

## SQLite DROP INDEX Statement

Syntax:

```
DROP INDEX database_name.index_name;
```

## SQLite DROP TABLE Statement

Syntax:

```
DROP TABLE database_name.table_name;
```

## SQLite DROP VIEW Statement

Syntax:

```
DROP INDEX database_name.view_name;
```

## SQLite DROP TRIGGER Statement

Syntax:

```
DROP INDEX database_name.trigger_name;
```

## SQLite EXISTS Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name EXISTS (SELECT * FROM   table_name );
```

## SQLite EXPLAIN Statement

Syntax:

```
EXPLAIN INSERT statement...;
        or
EXPLAIN QUERY PLAN SELECT statement...;
```

## SQLite GLOB Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name GLOB { PATTERN };
```

## SQLite GROUP BY Clause

Syntax:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name;
```

## SQLite HAVING Clause

Syntax:

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithematic function condition);
```

## SQLite INSERT INTO Statement

Syntax:

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

## SQLite IN Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name IN (val-1, val-2,...val-N);
```

## SQLite Like Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name LIKE { PATTERN };
```

## SQLite NOT IN Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name NOT IN (val-1, val-2,...val-N);
```

## SQLite ORDER BY Clause

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION
ORDER BY column_name {ASC|DESC};
```

## SQLite PRAGMA Statement

Syntax:

```
PRAGMA pragma_name;
```

## SQLite RELEASE SAVEPOINT Statement

Syntax:

```
RELEASE savepoint_name;
```

## SQLite REINDEX Statement

Syntax:

```
REINDEX collation_name;
REINDEX database_name.index_name;
REINDEX database_name.table_name;
```

## SQLite ROLLBACK Statement

Syntax:

```
ROLLBACK;
        or
ROLLBACK TO SAVEPOINT savepoint_name;
```

## SQLite SAVEPOINT Statement

Syntax:

```
SAVEPOINT savepoint_name;
```

## SQLite SELECT Statement

Syntax:

```
SELECT column1, column2....columnN
FROM   table_name;
```

## SQLite UPDATE Statement

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2....columnN=valueN
[ WHERE  CONDITION ];
```

## SQLite VACUUM Statement

Syntax:

```
VACUUM;
SQLite WHERE Clause:
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION;
```