Sr.No	Property & Description
1	**description** This is a read only property which returns the list containing the description of columns in a result-set.
2	**astrowid** This is a read only property, if there are any auto-incremented columns in the table, this returns the value generated for that column in the last INSERT or, UPDATE operation.
3	**rowcount** This returns the number of rows returned/updated in case of SELECT and UPDATE operations.
4	**closed** This property specifies whether a cursor is closed or not, if so it returns true, else false.
5	**connection** This returns a reference to the connection object using which this cursor was created.
6	**name** This property returns the name of the cursor.
7	**scrollable** This property specifies whether a particular cursor is scrollable.

Python SQLite - Introduction

Installation

SQLite3 can be integrated with Python using sqlite3 module, which was written by Gerhard Haring. It provides an SQL interface compliant with the DB-API 2.0 specification described by PEP 249. You do not need to install this module separately because it is shipped by default along with Python version 2.5.x onwards.

To use sqlite3 module, you must first create a connection object that represents the database and then optionally you can create a cursor object, which will help you in executing all the SQL statements.

Python sqlite3 module APIs

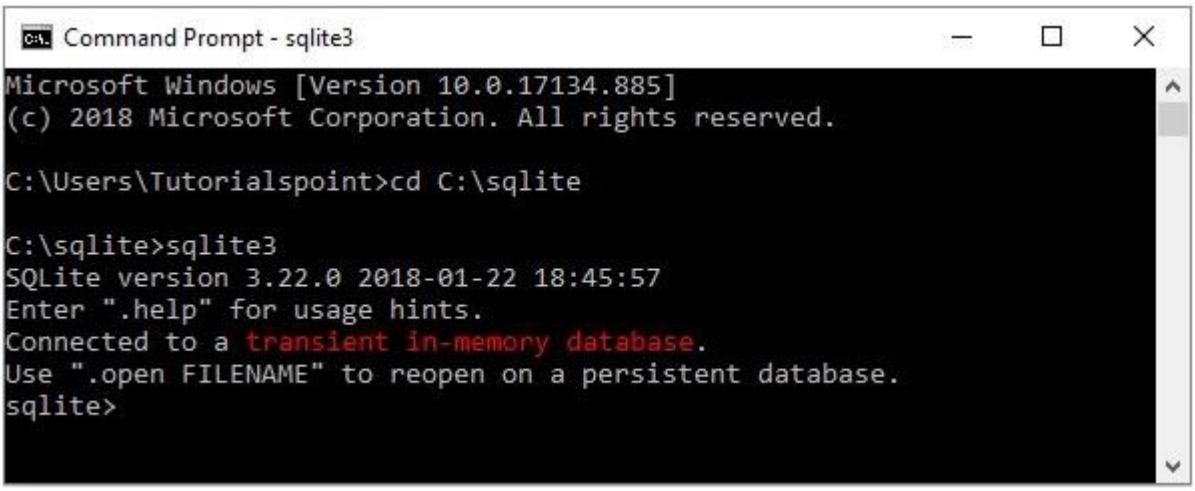
Following are important sqlite3 module routines, which can suffice your requirement to work with SQLite database from your Python program. If you are looking for a more sophisticated application, then you can look into Python sqlite3 module's official documentation.

Sr.No.	API & Description
1	**sqlite3.connect(database [,timeout ,other optional arguments])** This API opens a connection to the SQLite database file. You can use ":memory:" to open a database connection to a database that resides in RAM instead of on disk. If database is opened successfully, it returns a connection object.
2	**connection.cursor([cursorClass])** This routine creates a **cursor** which will be used throughout your database programming with Python. This method accepts a single optional parameter cursorClass. If supplied, this must be a custom cursor class that extends sqlite3.Cursor.
3	**cursor.execute(sql [, optional parameters])** This routine executes an SQL statement. The SQL statement may be parameterized (i. e. placeholders instead of SQL literals). The sqlite3 module supports two kinds of placeholders: question marks and named placeholders (named style). **For example** − cursor.execute("insert into people values (?, ?)", (who, age))
4	**connection.execute(sql [, optional parameters])** This routine is a shortcut of the above execute method provided by the cursor object and it creates an intermediate cursor object by calling the cursor method, then calls the cursor's execute method with the parameters given.
5	**cursor.executemany(sql, seq_of_parameters)** This routine executes an SQL command against all parameter sequences or mappings found in the sequence sql.
6	**connection.executemany(sql[, parameters])** This routine is a shortcut that creates an intermediate cursor object by calling the cursor method, then calls the cursor.s executemany method with the parameters given.
7	**cursor.executescript(sql_script)** This routine executes multiple SQL statements at once provided in the form of script. It issues a COMMIT statement first, then executes the SQL script it gets as a parameter. All the SQL statements should be separated by a semi colon (;).
8	**connection.executescript(sql_script)**

	This routine is a shortcut that creates an intermediate cursor object by calling the cursor method, then calls the cursor's executescript method with the parameters given.
9	**connection.total_changes()** This routine returns the total number of database rows that have been modified, inserted, or deleted since the database connection was opened.
10	**connection.commit()** This method commits the current transaction. If you don't call this method, anything you did since the last call to commit() is not visible from other database connections.
11	**connection.rollback()** This method rolls back any changes to the database since the last call to commit().
12	**connection.close()** This method closes the database connection. Note that this does not automatically call commit(). If you just close your database connection without calling commit() first, your changes will be lost!
13	**cursor.fetchone()** This method fetches the next row of a query result set, returning a single sequence, or None when no more data is available.
14	**cursor.fetchmany([size = cursor.arraysize])** This routine fetches the next set of rows of a query result, returning a list. An empty list is returned when no more rows are available. The method tries to fetch as many rows as indicated by the size parameter.
15	**cursor.fetchall()** This routine fetches all (remaining) rows of a query result, returning a list. An empty list is returned when no rows are available.

Python SQLite - Establishing Connection

To establish connection with SQLite Open command prompt, browse through the location of where you have installed SQLite and just execute the command ***sqlite3*** as shown below −



Establishing connection using python

You can communicate with SQLite2 database using the SQLite3 python module. To do so, first of all you need to establish a connection (create a connection object).

To establish a connection with SQLite3 database using python you need to −

- Import the sqlite3 module using the import statement.
- The connect() method accepts the name of the database you need to connect with as a parameter and, returns a Connection object.

Example

```
import sqlite3
conn = sqlite3.connect('example.db')
```

Output

```
print("Connection established ..........")
```

Python SQLite - Create Table

Using the SQLite CREATE TABLE statement you can create a table in a database.

Syntax

Following is the syntax to create a table in SQLite database −

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

```
   column3 datatype,
   .....
   columnN datatype
);
```

Example

Following SQLite query/statement creates a table with name **CRICKETERS** in SQLite database −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

Let us create one more table OdiStats describing the One-day cricket statistics of each player in CRICKETERS table.

```
sqlite> CREATE TABLE ODIStats (
   First_Name VARCHAR(255),
   Matches INT,
   Runs INT,
   AVG FLOAT,
   Centuries INT,
   HalfCenturies INT
);
sqlite
```

You can get the list of tables in a database in SQLite database using the ***.tables*** command. After creating a table, if you can verify the list of tables you can observe the newly created table in it as −

```
sqlite> . tables
CRICKETERS ODIStats
sqlite>
```

Creating a table using python

The Cursor object contains all the methods to execute quires and fetch data etc. The cursor method of the connection class returns a cursor object.

Therefore, to create a table in SQLite database using python −

- Establish connection with a database using the connect() method.
- Create a cursor object by invoking the cursor() method on the above created connection object.
- Now execute the CREATE TABLE statement using the execute() method of the Cursor class.

Example

Following Python program creates a table named Employee in SQLite3 −

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

#Creating table as per requirement
sql ='''CREATE TABLE EMPLOYEE(
   FIRST_NAME CHAR(20) NOT NULL,
   LAST_NAME CHAR(20),
   AGE INT,
   SEX CHAR(1),
   INCOME FLOAT
)'''
cursor.execute(sql)
print("Table created successfully........")

# Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
Table created successfully........
```

Python SQLite - Insert Data

You can add new rows to an existing table of SQLite using the INSERT INTO statement. In this, you need to specify the name of the table, column names, and values (in the same order as column names).

Syntax

Following is the recommended syntax of the INSERT statement −

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Where, column1, column2, column3,.. are the names of the columns of a table and value1, value2, value3,... are the values you need to insert into the table.

Example

Assume we have created a table with name CRICKETERS using the CREATE TABLE statement as shown below −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

Following PostgreSQL statement inserts a row in the above created table.

```
sqlite> insert into CRICKETERS
   (First_Name, Last_Name, Age, Place_Of_Birth, Country) values
   ('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
sqlite>
```

While inserting records using the INSERT INTO statement, if you skip any columns names, this record will be inserted leaving empty spaces at columns which you have skipped.

```
sqlite> insert into CRICKETERS
   (First_Name, Last_Name, Country) values
   ('Jonathan', 'Trott', 'SouthAfrica');
sqlite>
```

You can also insert records into a table without specifying the column names, if the order of values you pass is same as their respective column names in the table.

```
sqlite> insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale', 'Srilanka');
sqlite> insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur', 'India');
sqlite>
```

After inserting the records into a table you can verify its contents using the SELECT statement as shown below −

```
sqlite> select * from cricketers;
Shikhar  | Dhawan     | 33 | Delhi  | India
Jonathan | Trott      |    |        | SouthAfrica
Kumara   | Sangakkara | 41 | Matale | Srilanka
Virat    | Kohli      | 30 | Delhi  | India
Rohit    | Sharma     | 32 | Nagpur | India
sqlite>
```

Inserting data using python

To add records to an existing table in SQLite database −

- Import sqlite3 package.
- Create a connection object using the connect() method by passing the name of the database as a parameter to it.
- The ***cursor()*** method returns a cursor object using which you can communicate with SQLite3. Create a cursor object by invoking the cursor() object on the (above created) Connection object.
- Then, invoke the execute() method on the cursor object, by passing an INSERT statement as a parameter to it.

Example

Following python example inserts records into to a table named EMPLOYEE −

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

# Preparing SQL queries to INSERT a record into the database.
cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Ramya', 'Rama Priya', 27, 'F', 9000)''')

cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Vinay', 'Battacharya', 20, 'M', 6000)''')

cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Sharukh', 'Sheik', 25, 'M', 8300)''')

cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Sarmista', 'Sharma', 26, 'F', 10000)''')
```

```
cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Tripthi', 'Mishra', 24, 'F', 6000)''')

# Commit your changes in the database
conn.commit()
print("Records inserted........")

# Closing the connection
conn.close()
```

Output

```
Records inserted........
```

Python SQLite - Select Data

You can retrieve data from an SQLite table using the SELCT query. This query/statement returns contents of the specified relation (table) in tabular form and it is called as result-set.

Syntax

Following is the syntax of the SELECT statement in SQLite −

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

Example

Assume we have created a table with name CRICKETERS using the following query −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

And if we have inserted 5 records in to it using INSERT statements as −

```
sqlite> insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown', 'SouthAfrica');
sqlite> insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale', 'Srilanka');
sqlite> insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi', 'India');
```

```
sqlite> insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur', 'India');
sqlite>
```

Following SELECT query retrieves the values of the columns FIRST_NAME, LAST_NAME and, COUNTRY from the CRICKETERS table.

```
sqlite> SELECT FIRST_NAME, LAST_NAME, COUNTRY FROM CRICKETERS;
Shikhar  |Dhawan     |India
Jonathan |Trott      |SouthAfrica
Kumara   |Sangakkara |Srilanka
Virat    |Kohli      |India
Rohit    |Sharma     |India
sqlite>
```

As you observe, the SELECT statement of the SQLite database just returns the records of the specified tables. To get a formatted output you need to set the **header**, and **mode** using the respective commands before the SELECT statement as shown below −

```
sqlite> .header on
sqlite> .mode column
sqlite> SELECT FIRST_NAME, LAST_NAME, COUNTRY FROM CRICKETERS;
First_Name Last_Name           Country
---------- ------------------- ----------
Shikhar    Dhawan              India
Jonathan   Trott               SouthAfric
Kumara     Sangakkara          Srilanka
Virat      Kohli               India
Rohit      Sharma              India
sqlite>
```

If you want to retrieve all the columns of each record, you need to replace the names of the columns with "*" as shown below −

```
sqlite> .header on
sqlite> .mode column
sqlite> SELECT * FROM CRICKETERS;
First_Name Last_Name  Age        Place_Of_Birth Country
---------- ---------- ---------- -------------- ----------
Shikhar    Dhawan     33         Delhi          India
Jonathan   Trott      38         CapeTown       SouthAfric
Kumara     Sangakkara 41         Matale         Srilanka
Virat      Kohli      30         Delhi          India
Rohit      Sharma     32         Nagpur         India
sqlite>
```

In *SQLite* by default the width of the columns is 10 values beyond this width are chopped (observe the country column of 2nd row in above table). You can set the width of each column to required value using the ***.width*** command, before retrieving the contents of a table as shown below −

```
sqlite> .width 10, 10, 4, 10, 13
sqlite> SELECT * FROM CRICKETERS;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Shikhar    Dhawan     33   Delhi      India
Jonathan   Trott      38   CapeTown   SouthAfrica
Kumara     Sangakkara 41   Matale     Srilanka
Virat      Kohli      30   Delhi      India
Rohit      Sharma     32   Nagpur     India
sqlite>
```

Retrieving data using python

READ Operation on any database means to fetch some useful information from the database. You can fetch data from MYSQL using the fetch() method provided by the sqlite python module.

The sqlite3.Cursor class provides three methods namely fetchall(), fetchmany() and, fetchone() where,

- The fetchall() method retrieves all the rows in the result set of a query and returns them as list of tuples. (If we execute this after retrieving few rows it returns the remaining ones).
- The fetchone() method fetches the next row in the result of a query and returns it as a tuple.
- The fetchmany() method is similar to the fetchone() but, it retrieves the next set of rows in the result set of a query, instead of a single row.

Note − A result set is an object that is returned when a cursor object is used to query a table.

Example

Following example fetches all the rows of the EMPLOYEE table using the SELECT query and from the obtained result set initially, we are retrieving the first row using the fetchone() method and then fetching the remaining rows using the fetchall() method.

Following Python program shows how to fetch and display records from the COMPANY table created in the above example.

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Retrieving data
cursor.execute('''SELECT * from EMPLOYEE''')

#Fetching 1st row from the table
result = cursor.fetchone();
print(result)
```

```
#Fetching 1st row from the table
result = cursor.fetchall();
print(result)

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
('Ramya', 'Rama priya', 27, 'F', 9000.0)
[('Vinay', 'Battacharya', 20, 'M', 6000.0),
   ('Sharukh', 'Sheik', 25, 'M', 8300.0),
   ('Sarmista', 'Sharma', 26, 'F', 10000.0),
   ('Tripthi', 'Mishra', 24, 'F', 6000.0)
]
```

Python SQLite - Where Clause

If you want to fetch, delete or, update particular rows of a table in SQLite, you need to use the where clause to specify condition to filter the rows of the table for the operation.

For example, if you have a SELECT statement with where clause, only the rows which satisfies the specified condition will be retrieved.

Syntax

Following is the syntax of the WHERE clause in SQLite −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [search_condition]
```

You can specify a search_condition using comparison or logical operators. like >, <, =, LIKE, NOT, etc. The following examples would make this concept clear.

Example

Assume we have created a table with name CRICKETERS using the following query −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
```

```
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

And if we have inserted 5 records in to it using INSERT statements as −

```
sqlite> insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown', 'SouthAfrica');
sqlite> insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale', 'Srilanka');
sqlite> insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur', 'India');
sqlite>
```

Following SELECT statement retrieves the records whose age is greater than 35 −

```
sqlite> SELECT * FROM CRICKETERS WHERE AGE > 35;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Jonathan   Trott      38   CapeTown   SouthAfrica
Kumara     Sangakkara 41   Matale     Srilanka
sqlite>
```

Where clause using python

The Cursor object/class contains all the methods to execute queries and fetch data, etc. The cursor method of the connection class returns a cursor object.

Therefore, to create a table in SQLite database using python −

- Establish connection with a database using the connect() method.
- Create a cursor object by invoking the cursor() method on the above created connection object.
- Now execute the CREATE TABLE statement using the execute() method of the Cursor class.

Example

Following example creates a table named Employee and populates it. Then using the where clause it retrieves the records with age value less than 23.

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists.
```

```
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
sql = '''CREATE TABLE EMPLOYEE(
   FIRST_NAME CHAR(20) NOT NULL,
   LAST_NAME CHAR(20),
   AGE INT,
   SEX CHAR(1),
   INCOME FLOAT
)'''
cursor.execute(sql)

#Populating the table
cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Ramya', 'Rama priya', 27, 'F', 9000)''')

cursor.execute('''INSERT INTO EMPLOYEE
   (FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Vinay', 'Battacharya', 20, 'M', 6000)''')

cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Sharukh', 'Sheik', 25, 'M', 8300)''')

cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Sarmista', 'Sharma', 26, 'F', 10000)''')

cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Tripthi', 'Mishra', 24, 'F', 6000)''')

#Retrieving specific records using the where clause
cursor.execute("SELECT * from EMPLOYEE WHERE AGE <23")
print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
[('Vinay', 'Battacharya', 20, 'M', 6000.0)]
```

Python SQLite - Order By

While fetching data using SELECT query, you will get the records in the same order in which you have inserted them.

You can sort the results in desired order (ascending or descending) using the ***Order By*** clause. By default, this clause sorts results in ascending order, if you need to arrange them in descending order you need to use “DESC” explicitly.

Syntax

Following is the syntax of the ORDER BY clause in SQLite.

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

Example

Assume we have created a table with name CRICKETERS using the following query −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

And if we have inserted 5 records in to it using INSERT statements as −

```
sqlite> insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown', 'SouthAfrica');
sqlite> insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale', 'Srilanka');
sqlite> insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur', 'India');
sqlite>
```

Following SELECT statement retrieves the rows of the CRICKETERS table in the ascending order of their age −

```
sqlite> SELECT * FROM CRICKETERS ORDER BY AGE;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Virat      Kohli      30   Delhi      India
Rohit      Sharma     32   Nagpur     India
Shikhar    Dhawan     33   Delhi      India
Jonathan   Trott      38   CapeTown   SouthAfrica
```

```
Kumara     Sangakkara 41   Matale     Srilanka
sqlite>
```

You can use more than one column to sort the records of a table. Following SELECT statements sorts the records of the CRICKETERS table based on the columns *AGE* and *FIRST_NAME*.

```
sqlite> SELECT * FROM CRICKETERS ORDER BY AGE, FIRST_NAME;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Virat      Kohli      30   Delhi      India
Rohit      Sharma     32   Nagpur     India
Shikhar    Dhawan     33   Delhi      India
Jonathan   Trott      38   CapeTown   SouthAfrica
Kumara     Sangakkara 41   Matale     Srilanka
sqlite>
```

By default, the **ORDER BY** clause sorts the records of a table in ascending order you can arrange the results in descending order using DESC as −

```
sqlite> SELECT * FROM CRICKETERS ORDER BY AGE DESC;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Kumara     Sangakkara 41   Matale     Srilanka
Jonathan   Trott      38   CapeTown   SouthAfrica
Shikhar    Dhawan     33   Delhi      India
Rohit      Sharma     32   Nagpur     India
Virat      Kohli      30   Delhi      India
sqlite>
```

ORDER BY clause using python

To retrieve contents of a table in specific order, invoke the execute() method on the cursor object and, pass the SELECT statement along with ORDER BY clause, as a parameter to it.

Example

In the following example we are creating a table with name and Employee, populating it, and retrieving its records back in the (ascending) order of their age, using the ORDER BY clause.

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(
   database="mydb", user='postgres', password='password', host='127.0.0.1', port= '5432'
)
#Setting auto commit false
conn.autocommit = True
```

```
#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

#Creating a table
sql = '''CREATE TABLE EMPLOYEE(
FIRST_NAME CHAR(20) NOT NULL,
   LAST_NAME CHAR(20),
   AGE INT, SEX CHAR(1),
   INCOME INT,
   CONTACT INT
)'''
cursor.execute(sql)

#Populating the table
#Populating the table
cursor.execute('''INSERT INTO EMPLOYEE
   (FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Ramya', 'Rama priya', 27, 'F', 9000),
   ('Vinay', 'Battacharya', 20, 'M', 6000),
   ('Sharukh', 'Sheik', 25, 'M', 8300),
   ('Sarmista', 'Sharma', 26, 'F', 10000),
   ('Tripthi', 'Mishra', 24, 'F', 6000)''')
conn.commit()

#Retrieving specific records using the ORDER BY clause
cursor.execute("SELECT * from EMPLOYEE ORDER BY AGE")
print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
[('Vinay', 'Battacharya', 20, 'M', 6000, None),
   ('Tripthi', 'Mishra', 24, 'F', 6000, None),
   ('Sharukh', 'Sheik', 25, 'M', 8300, None),
   ('Sarmista', 'Sharma', 26, 'F', 10000, None),
   ('Ramya', 'Rama priya', 27, 'F', 9000, None)]
```

Python SQLite - Update Table

UPDATE Operation on any database implies modifying the values of one or more records of a table, which are already available in the database. You can update the values of existing records in SQLite using the UPDATE statement.

To update specific rows, you need to use the WHERE clause along with it.

Syntax

Following is the syntax of the UPDATE statement in SQLite −

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

Example

Assume we have created a table with name CRICKETERS using the following query −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

And if we have inserted 5 records in to it using INSERT statements as −

```
sqlite> insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown', 'SouthAfrica');
sqlite> insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale', 'Srilanka');
sqlite> insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur', 'India');
sqlite>
```

Following Statement modifies the age of the cricketer, whose first name is **Shikhar** −

```
sqlite> UPDATE CRICKETERS SET AGE = 45 WHERE FIRST_NAME = 'Shikhar' ;
sqlite>
```

If you retrieve the record whose FIRST_NAME is Shikhar you observe that the age value has been changed to 45 −

```
sqlite> SELECT * FROM CRICKETERS WHERE FIRST_NAME = 'Shikhar';
First_Name Last_Name  Age Place_Of_B  Country
---------- ---------- ---- ---------- -------------
Shikhar    Dhawan     45   Delhi      India
sqlite>
```

If you haven't used the WHERE clause values of all the records will be updated. Following UPDATE statement increases the age of all the records in the CRICKETERS table by 1 −

```
sqlite> UPDATE CRICKETERS SET AGE = AGE+1;
sqlite>
```

If you retrieve the contents of the table using SELECT command, you can see the updated values as −

```
sqlite> SELECT * FROM CRICKETERS;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Shikhar    Dhawan     46   Delhi      India
Jonathan   Trott      39   CapeTown   SouthAfrica
Kumara     Sangakkara 42   Matale     Srilanka
Virat      Kohli      31   Delhi      India
Rohit      Sharma     33   Nagpur     India
sqlite>
```

Updating existing records using python

To add records to an existing table in SQLite database −

- Import sqlite3 package.
- Create a connection object using the *connect()* method by passing the name of the database as a parameter to it.
- The ***cursor()*** method returns a cursor object using which you can communicate with SQLite3 . Create a cursor object by invoking the cursor() object on the (above created) Connection object.
- Then, invoke the execute() method on the cursor object, by passing an UPDATE statement as a parameter to it.

Example

Following Python example, creates a table with name EMPLOYEE, inserts 5 records into it and, increases the age of all the male employees by 1 −

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
```

```
#Creating table as per requirement
sql ='''CREATE TABLE EMPLOYEE(
   FIRST_NAME CHAR(20) NOT NULL,
   LAST_NAME CHAR(20),
   AGE INT,
   SEX CHAR(1),
   INCOME FLOAT
)'''
cursor.execute(sql)

#Inserting data
cursor.execute('''INSERT INTO EMPLOYEE
   (FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Ramya', 'Rama priya', 27, 'F', 9000),
   ('Vinay', 'Battacharya', 20, 'M', 6000),
   ('Sharukh', 'Sheik', 25, 'M', 8300),
   ('Sarmista', 'Sharma', 26, 'F', 10000),
   ('Tripthi', 'Mishra', 24, 'F', 6000)''')
conn.commit()

#Fetching all the rows before the update
print("Contents of the Employee table: ")
cursor.execute('''SELECT * from EMPLOYEE''')
print(cursor.fetchall())

#Updating the records
sql = '''UPDATE EMPLOYEE SET AGE=AGE+1 WHERE SEX = 'M' '''
cursor.execute(sql)
print("Table updated...... ")

#Fetching all the rows after the update
print("Contents of the Employee table after the update operation: ")
cursor.execute('''SELECT * from EMPLOYEE''')
print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
Contents of the Employee table:
[('Ramya', 'Rama priya', 27, 'F', 9000.0),
   ('Vinay', 'Battacharya', 20, 'M', 6000.0),
   ('Sharukh', 'Sheik', 25, 'M', 8300.0),
   ('Sarmista', 'Sharma', 26, 'F', 10000.0),
   ('Tripthi', 'Mishra', 24, 'F', 6000.0)]
Table updated......
```

```
Contents of the Employee table after the update operation:
[('Ramya', 'Rama priya', 27, 'F', 9000.0),
   ('Vinay', 'Battacharya', 21, 'M', 6000.0),
   ('Sharukh', 'Sheik', 26, 'M', 8300.0),
   ('Sarmista', 'Sharma', 26, 'F', 10000.0),
   ('Tripthi', 'Mishra', 24, 'F', 6000.0)]
```

Python SQLite - Delete Data

To delete records from a SQLite table, you need to use the DELETE FROM statement. To remove specific records, you need to use WHERE clause along with it.

To update specific rows, you need to use the WHERE clause along with it.

Syntax

Following is the syntax of the DELETE query in SQLite −

```
DELETE FROM table_name [WHERE Clause]
```

Example

Assume we have created a table with name CRICKETERS using the following query −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

And if we have inserted 5 records in to it using INSERT statements as −

```
sqlite> insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown', 'SouthAfrica');
sqlite> insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale', 'Srilanka');
sqlite> insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur', 'India');
sqlite>
```

Following statement deletes the record of the cricketer whose last name is 'Sangakkara'.

```
sqlite> DELETE FROM CRICKETERS WHERE LAST_NAME = 'Sangakkara';
sqlite>
```

If you retrieve the contents of the table using the SELECT statement, you can see only 4 records since we have deleted one.

```
sqlite> SELECT * FROM CRICKETERS;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Shikhar    Dhawan     46   Delhi      India
Jonathan   Trott      39   CapeTown   SouthAfrica
Virat      Kohli      31   Delhi      India
Rohit      Sharma     33   Nagpur     India
sqlite>
```

If you execute the DELETE FROM statement without the WHERE clause, all the records from the specified table will be deleted.

```
sqlite> DELETE FROM CRICKETERS;
sqlite>
```

Since you have deleted all the records, if you try to retrieve the contents of the CRICKETERS table, using SELECT statement you will get an empty result set as shown below −

```
sqlite> SELECT * FROM CRICKETERS;
sqlite>
```

Deleting data using python

To add records to an existing table in SQLite database −

- Import sqlite3 package.
- Create a connection object using the *connect()* method by passing the name of the database as a parameter to it.
- The ***cursor()*** method returns a cursor object using which you can communicate with SQLite3 . Create a cursor object by invoking the cursor() object on the (above created) Connection object.
- Then, invoke the execute() method on the cursor object, by passing an DELETE statement as a parameter to it.

Example

Following python example deletes the records from EMPLOYEE table with age value greater than 25.

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()
```

```
#Retrieving contents of the table
print("Contents of the table: ")
cursor.execute('''SELECT * from EMPLOYEE''')
print(cursor.fetchall())

#Deleting records
cursor.execute('''DELETE FROM EMPLOYEE WHERE AGE > 25''')

#Retrieving data after delete
print("Contents of the table after delete operation ")
cursor.execute("SELECT * from EMPLOYEE")
print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
Contents of the table:
[('Ramya', 'Rama priya', 27, 'F', 9000.0),
   ('Vinay', 'Battacharya', 21, 'M', 6000.0),
   ('Sharukh', 'Sheik', 26, 'M', 8300.0),
   ('Sarmista', 'Sharma', 26, 'F', 10000.0),
   ('Tripthi', 'Mishra', 24, 'F', 6000.0)]
Contents of the table after delete operation
[('Vinay', 'Battacharya', 21, 'M', 6000.0),
   ('Tripthi', 'Mishra', 24, 'F', 6000.0)]
```

Python SQLite - Drop Table

You can remove an entire table using the DROP TABLE statement. You just need to specify the name of the table you need to delete.

Syntax

Following is the syntax of the DROP TABLE statement in PostgreSQL −

```
DROP TABLE table_name;
```

Example

Assume we have created two tables with name CRICKETERS and EMPLOYEES using the following queries −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255), Last_Name VARCHAR(255), Age int,
   Place_Of_Birth VARCHAR(255), Country VARCHAR(255)
);
sqlite> CREATE TABLE EMPLOYEE(
   FIRST_NAME CHAR(20) NOT NULL, LAST_NAME CHAR(20), AGE INT,
   SEX CHAR(1), INCOME FLOAT
);
sqlite>
```

Now if you verify the list of tables using the ***.tables*** command, you can see the above created tables in it (list) as −

```
sqlite> .tables
CRICKETERS EMPLOYEE
sqlite>
```

Following statement deletes the table named Employee from the database −

```
sqlite> DROP table employee;
sqlite>
```

Since you have deleted the Employee table, if you retrieve the list of tables again, you can observe only one table in it.

```
sqlite> .tables
CRICKETERS
sqlite>
```

If you try to delete the Employee table again, since you have already deleted it you will get an error saying “no such table” as shown below −

```
sqlite> DROP table employee;
Error: no such table: employee
sqlite>
```

To resolve this, you can use the IF EXISTS clause along with the DELTE statement. This removes the table if it exists else skips the DLETE operation.

```
sqlite> DROP table IF EXISTS employee;
sqlite>
```

Dropping a table using Python

You can drop a table whenever you need to, using the DROP statement of MYSQL, but you need to be very careful while deleting any existing table because the data lost will not be recovered after deleting a table.

Example

To drop a table from a SQLite3 database using python invoke the ***execute()*** method on the cursor object and pass the drop statement as a parameter to it.

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists
cursor.execute("DROP TABLE emp")
print("Table dropped... ")

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
Table dropped...
```

Python SQLite - Limit

While fetching records if you want to limit them by a particular number, you can do so, using the LIMIT clause of SQLite.

Syntax

Following is the syntax of the LIMIT clause in SQLite −

```
SELECT column1, column2, columnN
FROM table_name
LIMIT [no of rows]
```

Example

Assume we have created a table with name CRICKETERS using the following query −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
   Country VARCHAR(255)
);
sqlite>
```

And if we have inserted 5 records in to it using INSERT statements as −

```
sqlite> insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown', 'SouthAfrica');
sqlite> insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale', 'Srilanka');
sqlite> insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi', 'India');
sqlite> insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur', 'India');
sqlite>
```

Following statement retrieves the first 3 records of the Cricketers table using the LIMIT clause −

```
sqlite> SELECT * FROM CRICKETERS LIMIT 3;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Shikhar    Dhawan     33   Delhi      India
Jonathan   Trott      38   CapeTown   SouthAfrica
Kumara     Sangakkara 41   Matale     Srilanka
sqlite>
```

If you need to limit the records starting from nth record (not 1st), you can do so, using OFFSET along with LIMIT.

```
sqlite> SELECT * FROM CRICKETERS LIMIT 3 OFFSET 2;
First_Name Last_Name  Age  Place_Of_B Country
---------- ---------- ---- ---------- -------------
Kumara     Sangakkara 41   Matale     Srilanka
Virat      Kohli      30   Delhi      India
Rohit      Sharma     32   Nagpur     India
sqlite>
```

LIMIT clause using Python

If you Invoke the execute() method on the cursor object by passing the SELECT query along with the LIMIT clause, you can retrieve required number of records.

Example

Following python example retrieves the first two records of the EMPLOYEE table using the LIMIT clause.

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Retrieving single row
sql = '''SELECT * from EMPLOYEE LIMIT 3'''

#Executing the query
cursor.execute(sql)

#Fetching the data
result = cursor.fetchall();
print(result)

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
[('Ramya', 'Rama priya', 27, 'F', 9000.0),
   ('Vinay', 'Battacharya', 20, 'M', 6000.0),
   ('Sharukh', 'Sheik', 25, 'M', 8300.0)]
```

Python SQLite - Join

When you have divided the data in two tables you can fetch combined records from these two tables using Joins.

Example

Assume we have created a table with name CRICKETERS using the following query −

```
sqlite> CREATE TABLE CRICKETERS (
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255),
   Age int,
   Place_Of_Birth VARCHAR(255),
```

```
   Country VARCHAR(255)
);
sqlite>
```

Let us create one more table OdiStats describing the One-day cricket statistics of each player in CRICKETERS table.

```
sqlite> CREATE TABLE ODIStats (
   First_Name VARCHAR(255),
   Matches INT,
   Runs INT,
   AVG FLOAT,
   Centuries INT,
   HalfCenturies INT
);
sqlite>
```

Following statement retrieves data combining the values in these two tables −

```
sqlite> SELECT
   Cricketers.First_Name, Cricketers.Last_Name, Cricketers.Country,
   OdiStats.matches, OdiStats.runs, OdiStats.centuries, OdiStats.halfcenturies
   from Cricketers INNER JOIN OdiStats ON Cricketers.First_Name = OdiStats.First_Name;
First_Name Last_Name    Country  Matches    Runs          Centuries  HalfCenturies
---------- ----------   -------  ---------- ------------ ---------- ----------
Shikhar    Dhawan       Indi     133        5518          17         27
Jonathan   Trott        Sout     68         2819          4          22
Kumara     Sangakkara   Sril     404        14234         25         93
Virat      Kohli        Indi     239        11520         43         54
Rohit      Sharma       Indi     218        8686          24         42
sqlite>
```

Join clause using python

Following SQLite example, demonstrates the JOIN clause using python −

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Retrieving data
sql = '''SELECT * from EMP INNER JOIN CONTACT ON EMP.CONTACT = CONTACT.ID'''

#Executing the query
cursor.execute(sql)
```

```
#Fetching 1st row from the table
result = cursor.fetchall();
print(result)

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

Output

```
[('Ramya', 'Rama priya', 27, 'F', 9000.0, 101, 101, 'Krishna@mymail.com', 'Hyderabad'),
   ('Vinay', 'Battacharya', 20, 'M', 6000.0, 102, 102,'Raja@mymail.com', 'Vishakhapatnam'),
   ('Sharukh', 'Sheik', 25, 'M', 8300.0, 103, 103, 'Krishna@mymail.com', 'Pune'),
   ('Sarmista', 'Sharma', 26, 'F', 10000.0, 104, 104, 'Raja@mymail.com', 'Mumbai')]
```

Python SQLite - Cursor Object

The sqlite3.Cursor class is an instance using which you can invoke methods that execute SQLite statements, fetch data from the result sets of the queries. You can create **Cursor** object using the cursor() method of the Connection object/class.

Example

```
import sqlite3

#Connecting to sqlite
conn = sqlite3.connect('example.db')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()
```

Methods

Following are the various methods provided by the Cursor class/object.

Sr.No	Method & Description
1	**execute()** This routine executes an SQL statement. The SQL statement may be parameterized (i.e., placeholders instead of SQL literals). The psycopg2 module supports placeholder using %s sign For example:cursor.execute("insert into people values (%s, %s)", (who, age))
2	**executemany()** This routine executes an SQL command against all parameter sequences or mappings found in the sequence sql.
3	**fetchone()** This method fetches the next row of a query result set, returning a single sequence, or None when no more data is available.
4	**fetchmany()** This routine fetches the next set of rows of a query result, returning a list. An empty list is returned when no more rows are available. The method tries to fetch as many rows as indicated by the size parameter.
5	**fetchall()** This routine fetches all (remaining) rows of a query result, returning a list. An empty list is returned when no rows are available.

Properties

Following are the properties of the Cursor class −

Sr.No	Method & Description
1	**arraySize** This is a read/write property you can set the number of rows returned by the fetchmany() method.
2	**description** This is a read only property which returns the list containing the description of columns in a result-set.
3	**lastrowid** This is a read only property, if there are any auto-incremented columns in the table, this returns the value generated for that column in the last INSERT or, UPDATE operation.
4	**rowcount** This returns the number of rows returned/updated in case of SELECT and UPDATE operations.
5	**connection** This read-only attribute provides the SQLite database Connection used by the Cursor object.

Python MongoDB - Introduction

Pymongo is a python distribution which provides tools to work with MongoDB, it is the most preferred way to communicate with MongoDB database from python.

Installation

To install pymongo first of all make sure you have installed python3 (along with PIP) and MongoDB properly. Then execute the following command.

```
C:\WINDOWS\system32>pip install pymongo
Collecting pymongo
Using cached https://files.pythonhosted.org/packages/cb/a6/b0ae3781b0ad75825e00e29dc5489b5351262
Installing collected packages: pymongo
Successfully installed pymongo-3.9.0
```