Installation

SQLite3 can be integrated with Python using the 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. When a database is accessed by multiple connections, and one of the processes modifies the database, the SQLite database is locked until that transaction is committed. The timeout parameter specifies how long the connection should wait for the lock to go away until raising an exception. The default for the timeout parameter is 5.0 (five seconds). If the given database name does not exist then this call will create the database. You can specify filename with the required path as well if you want to create a database anywhere else except in the current directory.
2	**connection.cursor([cursorClass])** This routine creates a **cursor** which will be used throughout of 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.