What is SQLite?

SQLite is an **Open Source database**. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime (approx. 250 KByte) which makes it a light weight database to embed into other runtimes.

SQLite supports the data types TEXT (like Str in Python), INTEGER (like int in python) and REAL (like float in Python).

All other types must be converted into one of these fields before getting saved in the database.

SQLite is different from most other SQL database engines in that its primary design goal is to be simple:

1. Simple to administer
2. Simple to operate
3. Simple to embed in a larger program
4. Simple to maintain and customize

Features of SQLite

1. **Zero configuration** – SQLite does not need to be installed as there is no setup procedure to use it.
2. **Serverless** – SQLite is not implemented as a separate server process.
3. **Stable Cross-Platform Database File** – The SQLite file format is cross-platform. A database file written on one machine can be copied to and used on a different machine with a different architecture.

4. **Single Database File** – A SQLite database is a single ordinary disk file that can be located anywhere in the directory hierarchy.
5. **Compact** – When optimized for size, the whole SQLite library with everything enabled is less than 400KB in size

Advantages of using SQLite

1. There is no file parsing and no need to generate code to read/write/update the file.
2. Content can be accessed and updated using powerful SQL queries, greatly reducing the complexity of the application code.
3. The content can be viewed using third-party tools like Toad.
4. The application file is portable across all operating systems, 32-bit and 64-bit and big- and little-endian architectures.
5. Content is updated continuously and atomically so that there is no work lost in the event of a power failure or crash.

Well Known Users of SQLite

Adobe Mozilla Google McAfee Skype PHP Microsoft

SQL Commands

Create, Read, Update and Delete (CRUD) operations in SQLite DB

1. **Command to create a table**

   Syntax

   Create table <table-name> (col-name data-type, col-name data-type, ….n);

SQLite supports the data types TEXT (like String in Python), INTEGER (like long in Python) and REAL (like double in Python).

Example:

Create table Empoyee (Id INTEGER, Name TEXT, Salary REAL);

2. Command to insert data into a table

Syntax

Insert into <table-name> values (val1,val2,val3,…..n);

Example

Insert into Employee values (1212, 'Adwaith', 1500000.00);

Note: - TEXT must be enclosed in single quotations.

3. Command to Update data in a table

Syntax

Update <table-name> set col1 = value, col2 = value, ….n where condition;

Example

Update Employee set Salary=1800000.00 where id=1212;

4. Command to Delete data from a table

Syntax

Delete from <table-name> where condition;

Example

Delete from Employee where id=1212;

5. Command to Read set of data from a table

Syntax

Select * from <table-name> ;

Example

Select * from Employee;

Syntax

Select * from <table-name> where condition;

Example

Select * from Employee where id = 1212;

Syntax

Select <col1>, <col2> from <table-name> where condition;

Example

Select Name, Salary from Employee where Id = 1212;

Using Database in Python

To use SQLite, you have to **import** the module **sqlite3**.

Example:

```
import sqlite3
```

There are four stages of database communication with python:

1. Create a connection object.

Example:

```
connection = sqlite3.connect('naveen.db')
```

2. Create a cursor object to read/write.

Example:

```
curs = connection.cursor()
```

3. Interact with the database.

Example:

```
curs.execute('create table student(name, age, cno, address,
username, password)')
connection. commit()
```

4. Close the connection.

Example: connection.close()

Complete Example

```
import sqlite3 as sql
connection = sql.connect("naveen.db")
curs = connection.cursor()
curs.execute("create table
student(name,age,cno,address,username,password)")
        <sqlite3.Cursor object at 0x0000000003033B90>
curs.execute("insert into student
values('Ravi',23,9052492329,'Hyderabad','ravi','kumar')")
        <sqlite3.Cursor object at 0x0000000003033B90>
 connection.commit()
 connection.close()
```

Cursor class functions

execute ( )

Will only execute a single SQL statement.
If you try to execute more than one statement with it, it will raise a Warning.

executescript( )

If you want to execute multiple SQL statements with one call.

executemany(*sql, seq_of_parameters*)

Executes an SQL command against all parameter sequences or mappings found in the sequence *seq_of_parameters*

Fetching results

1) fetchall( )

To fetch result we use the fetchall( ) function of the cursor object.

Syntax: cur.fetchall( )

On success it returns tuple of rows where each row is a tuple.

2) fetchone( )

This method returns one record as a tuple, If there are no more records then it returns None

3) fetchmany(number_of_records)

This method accepts number of records to fetch and returns tuple where each records itself is a tuple. If there are not more records then it returns an empty tuple.