SQLITE PYTHON TUTORIAL

Installation

The 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 its being 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 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.

S.N.	API & Description
1	**sqlite3.connect***database*[*, timeout, otheroptionalarguments*] This API opens a connection to the SQLite database file database. 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 *fiveseconds*. If given database name does not exist then this call will create the database. You can specify filename with required path as well if you want to create database anywhere else except in 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*[*, optionalparameters*] This routine executes an SQL statement. The SQL statement may be parameterized *i. e. placeholdersinsteadofSQLliterals*. The sqlite3 module supports two kinds of placeholders: question marks and named placeholders *namedstyle*. For example:cursor.execute " *insertintopeoplevalues*(?, ?", *who, age*)
4	**connection.execute***sql*[*, optionalparameters*] This routine is a shortcut of the above execute method provided by 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$_o$f$_p$arameters*

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 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.

Connecting To Database

Following Python code shows how to connect to an existing database. If database does not exist, then it will be created and finally a database object will be returned.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print "Opened database successfully";
```

Here you can also supply database name as the special name **:memory:** to create a database in RAM. Now, let's run above program to create our database **test.db** in the current directory. You can change your path as per your requirement. Keep above code in sqlite.py file and execute it as shown below. If database is successfully created then it will give following message:

```
$chmod +x sqlite.py
$./sqlite.py
Open database successfully
```

Create a Table

Following Python program will be used to create a table in previously created database:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute('''CREATE TABLE COMPANY
       (ID INT PRIMARY KEY     NOT NULL,
       NAME           TEXT    NOT NULL,
       AGE            INT     NOT NULL,
       ADDRESS        CHAR(50),
       SALARY         REAL);''')
print "Table created successfully";

conn.close()
```

When above program is executed, it will create COMPANY table in your **test.db** and it will display following messages:

```
Opened database successfully
Table created successfully
```

INSERT Operation

Following Python program shows how we can create records in our COMPANY table created in above example:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");
```

```
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print "Records created successfully";
conn.close()
```

When above program is executed, it will create given records in COMPANY table and will display the following two lines:

```
Opened database successfully
Records created successfully
```

SELECT Operation

Following Python program shows how we can fetch and display records from our COMPANY table created in above example:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

When above program is executed, it will produce the following result:

```
Opened database successfully
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  20000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

UPDATE Operation

Following Python code shows how we can use UPDATE statement to update any record and then fetch and display updated records from our COMPANY table:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID=1")
conn.commit
print "Total number of rows updated :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

When above program is executed, it will produce the following result:

```
Opened database successfully
Total number of rows updated : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

DELETE Operation

Following Python code shows how we can use DELETE statement to delete any record and then fetch and display remaining records from our COMPANY table:

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("DELETE from COMPANY where ID=2;")
conn.commit
print "Total number of rows deleted :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

When above program is executed, it will produce the following result:

```
Opened database successfully
Total number of rows deleted : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  20000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

Processing math: 100%