SQLITE-BASIC COMMANDS

SQLite is a software library that implements a self-contained, server-less, zero configuration, transactional SQL database engine. The Python Standard Library includes a module called sqlite3 intended for working with this database. So you don't have to install it separately. Let's see some basic commands used in SQLite.

Importing Python's SQLite Module

To use the SQLite3 module we need to add an import statement to our python script. Any program that use sqlite should have the following import statement

import sqlite3

Connecting SQLite to the Database

We use the function sqlite3.connect to connect to the database.

Create or open a file (database) called mydb with a SQLite3 DB

conn = sqlite3.connect(‘mydb')

Creating (CREATE) Tables

In order to make any operation with the database we need to get a cursor object (something like a pointer to the database) and pass the SQL statements to the cursor object to execute them. Finally it is necessary to commit the changes.

```
import sqlite3
db = sqlite3.connect('mydb')
cursor = db.cursor() # Getting the cursor to the database mydb
cursor.execute('''CREATE TABLE users(id INTEGER PRIMARY KEY, name TEXT, email
TEXT)”')
db.commit()
```

Once the table is created we can insert data into the table using INSERT INTO command.

```
import sqlite3
cursor = db.cursor()
name1=‘abc’
email1='abc@xyz.com'

name2=‘def’
email2='def@xyz.com'
```

```
cursor.execute('''INSERT INTO users(name,email)
                  VALUES(?,?)''', (name1,email1))

db.commit()
```

OR If you need to insert several users use executemany and a list with the tuples:

```
users = [(name1,email1),
         (name2,email2)]

cursor.executemany(''' INSERT INTO users(name,email) VALUES(?,?)''', users)
db.commit()
```

OR using a dictionary

```
cursor.execute('''INSERT INTO users(name, email)
                      VALUES(:name,:email)''', {'name':name1, 'email':email1})
```

Retrieving Data (SELECT) with SQLite

To retrieve data, execute the query against the cursor object and then use fetchone() to retrieve a single row or fetchall() to retrieve all the rows.

```
import sqlite3
cursor = db.cursor()
cursor.execute('''SELECT name, email FROM users''')
user1 = cursor.fetchone() #retrieve the first row
print("The name of first user is",user1[0])
```

Output :

```
The name of first user is abc
```

```
import sqlite3
cursor = db.cursor()
cursor.execute('''SELECT name, email FROM users''')
all_rows = cursor.fetchall()
print ("The contact list\n")
for row in all_rows:
        print('{0} \t {1} \t {2}'.format(row[0], row[1], row[2]))
db.commit()
```

Output:

```
The contact list
abc	abc@xyz.com
def	def@xyz.com
```

Update the Database

The following code updates the email id of user with id=2 with UPDATE command and displays the updated database using SELECT command

```
import sqlite3
cu= db.cursor()
newemail = ‘ddd@xyz.com’
userid = 2 #second user
cu.execute('''UPDATE users SET email = ? WHERE id = ? ''',(newemail,userid))
cu.execute('''SELECT name, email, FROM users''')
all_rows = cu.fetchall()
print ("The updated list\n")
for row in all_rows:
	print('{0} \t {1}\t {2}'.format(row[0], row[1], row[2]))
db.commit()
```

Output:

```
The updated list

abc	abc@xyz.com
def	ddd@xyz.com
```

For additional information please refer http://www.sqlitetutorial.net

Do It Now Exercise:

Create a simple TODO list application using PySimpleGUI. Use SQLite as backend.