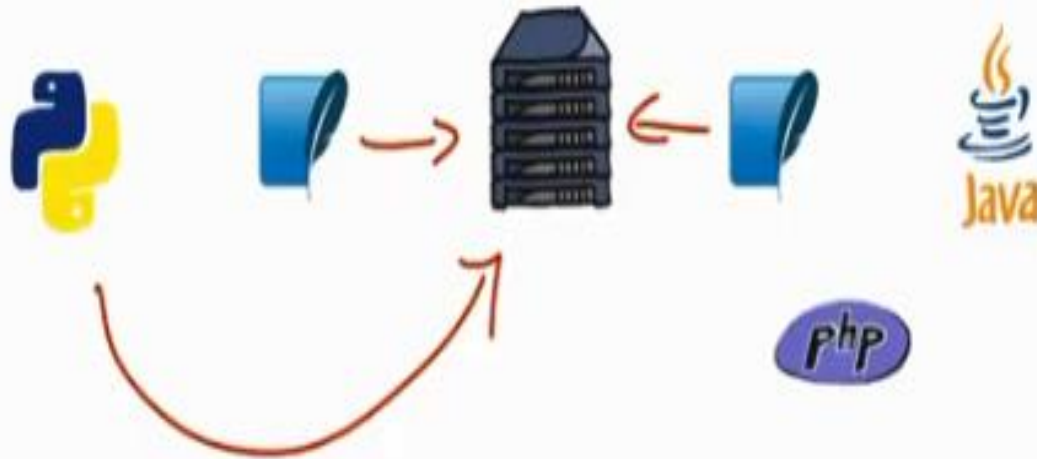


SQLite GUI

Nguyen Viet Hung
MSc. (I.T)



```
22.02 Connecting to SQLite.py  X
1  import sqlite3
2
3  try:
4      conn = sqlite3.connect('test.db')
5      print "Opened database successfully";
6  except Exception as e:
7      print("Error during connection: ",str(e))
8
9  conn.close()
```



Explanation

SSQL stands for “Structured Query Language” and is the main language that the large database packages use. SSQlite is free software that can be used as an SQL database. You can download the latest version of the software from www.sqlite.org.



Explanation

From the download page you need to select one of the “Precompiled Binaries” options for either Mac OS or Windows that includes the “command-line-shell”.

Precompiled Binaries for Mac OS X (x86)

[sqlite-tools-osx-x86-3190300.zip](#) (1.14 MB) A bundle of command-line tools for managing SQLite database files, including the [command-line-shell](#) program, the [sqldiff](#) program, and the [sqlite3_analyzer](#) program.
(sha1: 1012be9d387f2d0adb7b27e5967600d6566c798c)

Precompiled Binaries for Windows

[sqlite-dll-win32-x86-3190300.zip](#) (434.80 KB) 32-bit DLL (x86) for SQLite version 3.19.3.
(sha1: 92d2f94c0f528e92993f346a7b3375f92419b7e)

[sqlite-dll-win64-x64-3190300.zip](#) (722.63 KB) 64-bit DLL (x64) for SQLite version 3.19.3.
(sha1: 80034d5738990e07bac72fbfd0d0b8cd39d2782a)

[sqlite-tools-win32-x86-3190300.zip](#) (1.56 MB) A bundle of command-line tools for managing SQLite database files, including the [command-line-shell](#) program, the [sqldiff.exe](#) program, and the [sqlite3_analyzer.exe](#) program.
(sha1: 21a42e8103a5a89a7305af2f58174cebab34d1c6)

Explanation

To use SQL you need to load the “DB Browser for SQLite” which you can download from **<https://sqlitebrowser.org>**.




<https://sqlitebrowser.org/dl/>

Explanation

Understanding a Relational Database

We will use the example of a small manufacturing company that stores the details of their employees in an **SQL** database.

Below is an example of the Employees table, which contains the details of all the employees in the company. The contents of a table can be viewed by clicking on the Browse Data tab.

Table:  Employees ▼

	ID	Name	Dept	Salary
	Filter	Filter	Filter	Filter
1	1	Bob	Sales	25000
2	2	Sue	IT	28500
3	3	Tim	Sales	25000
4	4	Anne	Admin	18500
5	5	Paul	IT	28500
6	6	Simon	Sales	22000
7	7	Karen	Manufacturing	18500
8	8	Mark	Manufacturing	19000
9	9	George	Manufacturing	18500
10	10	Keith	Manufacturing	15000

Explanation

Here we can see the Departments table holding the details about each department. We have simplified it and only include one piece of data for each department (in this case the manager's name) but it will still save having to input the manager's name on every record, which we would have to do if the data were all being saved in one large table.

By splitting the data into two tables like this, if we need to update the manager it only needs to be updated in one place rather than updating it several times, which we would need to do if it was all stored in a single table.

This is known as a **one-to-many** relationship as one department can have many employees in it.



A **primary key** is the field (usually the first one) in each table that stores the unique identifier for that record. Therefore, in the Employees table the primary key will be the ID column and in the Department table the primary key will be Dept.

When **creating a table**, you need to identify the following for each field: the name of the field (field names cannot contain spaces and must follow the same rules as variable names);

If it is a primary key;

The data type for that field.

The data types you can use are as follows:

integer: the value is an integer value;

real: the value is a floating-point value;

text: the value is a text string;

blob: the value is stored exactly as it was input.

You can also specify if the field cannot be left blank by adding **NOT NULL** to the end of the field when you create it.

Example Code

```
import sqlite3
```

This must be the first line of the program to allow Python to use the SQLite3 library.

```
with sqlite3.connect("company.db") as db:
```

```
    cursor=db.cursor()
```

Connects to the company database. If no such database exists, it will create one. The file will be stored in the same folder as the program.

Example Code

```
cursor.execute("""CREATE TABLE IF NOT EXISTS  
employees(  
id integer PRIMARY KEY, name text NOT NULL,  
dept text NOT NULL, salary integer);""")
```

Creates a table called employees which has four fields (id, name, dept and salary). It specifies the data type for each field, defines which field is the primary key and which fields cannot be left blank. The triple speech marks allow the code to be split over several lines to make it easier to read rather than having it all displayed in one line.

Example Code

```
cursor.execute(“““INSERT INTO  
employees(id,name,dept,salary)  
VALUES(“1”,“Bob”,“Sales”,“25000”)”””)  
db.commit()
```

Inserts data into the employees table. The **db.commit()** line saves the changes.

Example Code

```
newID = input("Enter ID number: ")  
newName = input("Enter name: ")  
newDept = input("Enter department: ")  
newSalary = input("Enter salary: ")  
cursor.execute("""INSERT INTO  
employees(id,name,dept,salary)  
VALUES(?,?,?,?)""",(newID,newName,newDept,newSalary))  
db.commit()
```

Allows a user to enter new data which is then inserted into the table.

Example Code

```
cursor.execute("SELECT * FROM employees")  
print(cursor.fetchall())
```

Displays all the data from the employees table.

```
db.close()
```

This must be the last line in the program to close the database.

Example Code

```
cursor.execute("SELECT * FROM employees")
```

```
for x in cursor.fetchall():
```

```
    print(x)
```

Displays all the data from the employees table and displays each record on a separate line.

```
cursor.execute("SELECT * FROM employees ORDER BY  
name")
```

```
for x in cursor.fetchall():
```

```
    print(x)
```

Selects all the data from the employees table, sorted by name, and displays each record on a separate line.

Example Code

```
cursor.execute("SELECT * FROM employees WHERE  
salary>20000")
```

Selects all the data from the employees table where the salary is over 20,000.

```
cursor.execute("SELECT * FROM employees WHERE  
dept='Sales'")
```

Selects all the data from the employees table where the department is "Sales".

Example Code

```
cursor.execute(“““SELECT employees.id,employees.name,  
employees.dept  
FROM employees WHERE employees.dept=“IT”  
AND employees.salary >20000””””)
```

Selects the ID and name fields from the employees table and the manager field from the department table if the salary is over 20,000.

```
cursor.execute(“SELECT id,name,salary FROM employees”)
```

Selects the ID, name and salary fields from the employees table.

Example Code

```
whichDept = input("Enter a department: ")  
cursor.execute("SELECT * FROM employees WHERE  
dept=?", [whichDept])  
for x in cursor.fetchall():  
    print(x)
```

Allows the user to type in a department and displays the records of all the employees in that department.

Example Code

```
cursor.execute("""SELECT  
employees.id,employees.name,dept.manager  
FROM employees,dept WHERE employees.dept=dept.dept"""))
```

Selects the ID and name fields from the employees table and the manager field from the department table, using the dept fields to link the data. If you do not specify how the tables are linked, Python will assume every employee works in every department and you will not get the results you are expecting.

Example Code

```
cursor.execute(“UPDATE employees SET name = ‘Tony’  
WHERE id=1”)
```

```
db.commit()
```

Updates the data in the table (overwriting the original data) to change the name to “Tony” for employee ID 1.

```
cursor.execute(“DELETE from employees WHERE id=1”)
```

Challenges

Ex1: Create an SQL database called PhoneBook that contains a table called Names with the following data:

ID	First Name	Surname	Phone Number
1	Simon	Howels	01223 349752
2	Karen	Phillips	01954 295773
3	Darren	Smith	01583 749012
4	Anne	Jones	01323 567322
5	Mark	Smith	01223 855534

Challenges

Ex2: Using the PhoneBook database from program Ex1, write a program that will display the following menu.

```
Main Menu
```

- ```
1) View phone book
2) Add to phone book
3) Search for surname
4) Delete person from phone book
5) Quit
```

```
Enter your selection:
```

If the user selects 1, they should be able to view the entire phonebook. If they select 2, it should allow them to add a new person to the phonebook. If they select 3, it should ask them for a surname and then display only the records of people with the same surname. If they select 4, it should ask for an ID and then delete that record from the table. If they select 5, it should end the program. Finally, it should display a suitable message if they enter an incorrect selection from the menu. They should return to the menu after each action, until they select 5.

# Challenges

Ex3: Create a new SQL database called BookInfo that will store a list of authors and the books they wrote. It will have two tables. The first one should be called Authors and contain the following data:

| Name            | Place of Birth |
|-----------------|----------------|
| Agatha Christie | Torquay        |
| Cecelia Ahern   | Dublin         |
| J.K. Rowling    | Bristol        |
| Oscar Wilde     | Dublin         |