Tag: SQL

Python Database Programming: SQLite (tutorial)

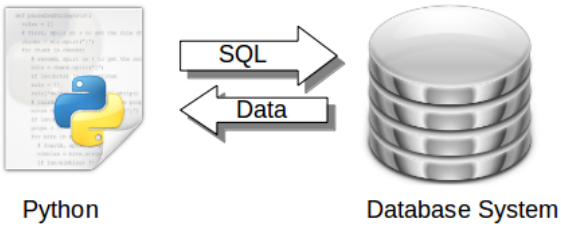
In this tutorial you will learn how to use the SQLite database management system with Python. You will learn how to use SQLite, SQL queries, RDBMS and more of this cool stuff!

Related course:
SQLite Made Easy

Pyton Database

Data is everywhere and software applications use that. Data is either in memory, files or databases.

Python has



Python Database.
Data is retrieved from a database system using the SQL language.

SQL is a spe designed for managing data held in a databases. The language has been around since 1986 and is worth learning. The is an old funny video about SQL

SQLite

SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.

SQLite, a relational database management system.

It is a self-contained, serverless, zero-configuration, transactional SQL database engine. The SQLite project is sponsored by Bloomberg and Mozilla.

Install SQLite:

Use this command to install SQLite:

```
$ sudo apt-get install sqlite
```

Verify if it is correctly installed. Copy this program and save it

```
    con = l
    cur = con.cursor()
    cur.execute('SELECT SQLITE_VERSION()')
    data = cur.fetchone()
    print "SQLite version: %s" % data
except lite.Error, e:
    print "Error %s:" % e.args[0]
    sys.exit(1)
finally:
    if con:
        con.close()
```

Execute with:

```
$ python test1.py
```

It should output:

```
SQLite version: 3.8.2
```

What did the script above do?

The script connected to a new database called test.db with this line:

```
con = lite.connect('test.db')
```

It then queries the database management system with the command

```
SELECT SQLITE_VERSION()
```

which in turn returned its version number. That line is known as an SQL query.

SQL Create and Insert

```
con = lite.connect('user.db')

with con:

    cur = con.cursor()
    cur.execute("CREATE TABLE Users(Id INT, Name TEXT)")
    cur.execute("INSERT INTO Users VALUES(1,'Michelle')")
    cur.execute("INSERT INTO Users VALUES(2,'Sonya')")
    cur.execute("INSERT INTO Users VALUES(3,'Greg')")
```

SQLite is a database management system that uses tables. These tables can have relations with other tables: it's called relational database management system or RDBMS. The table defines the structure of the data and can hold the data. A database can hold many different tables. The table gets created using the command:

```
cur.execute("CREATE TABLE Users(Id INT, Name TEXT)")
```

We add records into the table with these commands:

```
cur.execute("INSERT INTO Users VALUES(2,'Sonya')")
cur.execute("INSERT INTO Users VALUES(3,'Greg')")
```

The first value is the ID. The second value is the name. Once we run the script the data gets inserted into the database table Users:

	Id	Name
1	1	Michelle
2	2	Sonya
3	3	Greg

SQL Table

From consol e type these commands:

```
sqlite3 user.db
.tables
SELECT * FROM Users;
```

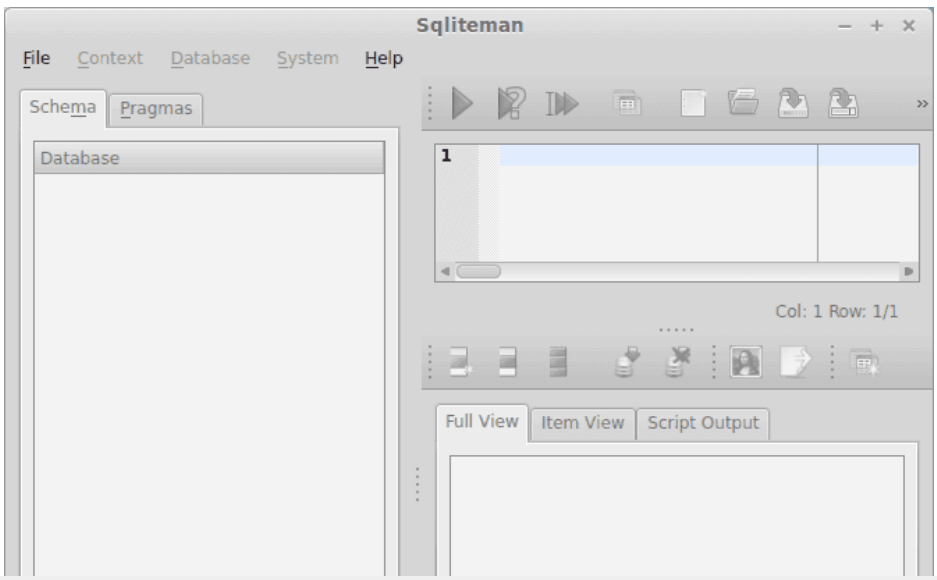
This will output the data in the table Users.

```
sqlite&gt; SELECT * FROM Users;
1|Michelle
2|Sonya
3|Greg
```

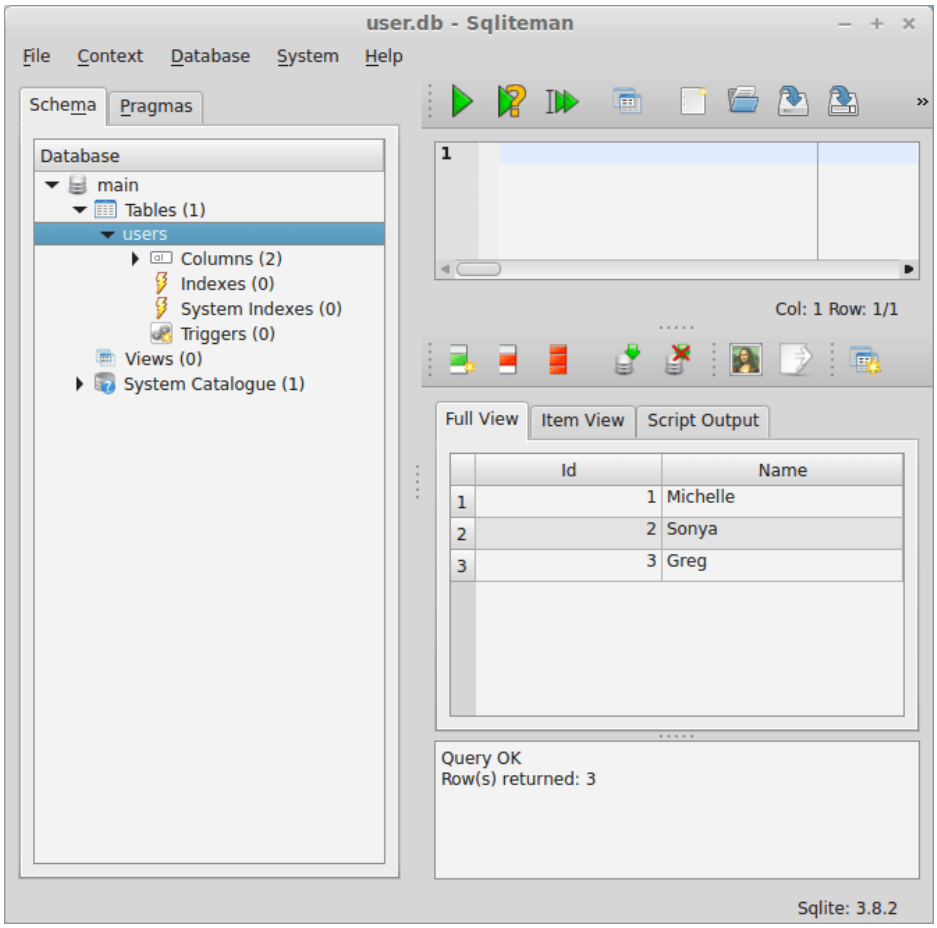
From GUI: If you want to use a GUI instead, there is a lot of choice. Personally I picked sqllite-man but there are many others. We install using:

```
sudo apt-get install sqliteman
```

We start the application sqliteman. A gui pops up.



Press File > Open > user.db. It appears like not much has changed, do not worry, this is just the user interface. On the left is a small tree view, press Tables > users. The full table including all records will be showing now.



sqliteman

This GUI can be used to modify the records (data) in the table and to add new tables.

```
SELECT coun
SELECT name FROM Users;
SELECT * FROM Users WHERE id = 2;
DELETE FROM Users WHERE id = 6;
```

We can use those queries in a Python program:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sqlite3 as lite
import sys


con = lite.connect('user.db')

with con:

    cur = con.cursor()
    cur.execute("SELECT * FROM Users")

    rows = cur.fetchall()

    for row in rows:
        print row
```

This will output all data in the Users table from the database:

```
$ python get.py
(1, u'Michelle')
(2, u'Sonya')
(3, u'Greg')
```

Creating a user information database

We can structure our data across multiple tables. This keeps our data structured, fast and organized. If we would have a single table to store everything, we would quickly have a big chaotic mess. What we will do is create multiple tables and use them in a combination. We create two tables:

Jobs:

	Id	Uid	Profession
1	1	1	Scientist
2	2	2	Marketeer
3	3	3	Developer

SQL Table

To create these tables, you can do that by hand in the GUI or use the script below:

```
# -*- coding: utf-8 -*-

import sqlite3 as lite
import sys

con = lite.connect('system.db')

with con:

    cur = con.cursor()
    cur.execute("CREATE TABLE Users(Id INT, Name TEXT)")
    cur.execute("INSERT INTO Users VALUES(1,'Michelle')")
    cur.execute("INSERT INTO Users VALUES(2,'Howard')")
    cur.execute("INSERT INTO Users VALUES(3,'Greg')")

    cur.execute("CREATE TABLE Jobs(Id INT, Uid INT, Profes
    cur.execute("INSERT INTO Jobs VALUES(1,1,'Scientist')"
    cur.execute("INSERT INTO Jobs VALUES(2,2,'Marketeer')"
    cur.execute("INSERT INTO Jobs VALUES(3,3,'Developer')"
```

The jobs table has an extra parameter, Uid. We use that to connect the two tables in an SQL query:

```
SELECT users.name, jobs.profession FROM jobs INNER JOIN us
```

```
with con:

    cur = con.cursor()
    cur.execute("SELECT users.name, jobs.profession FROM j

    rows = cur.fetchall()

    for row in rows:
        print row
```

It should output:

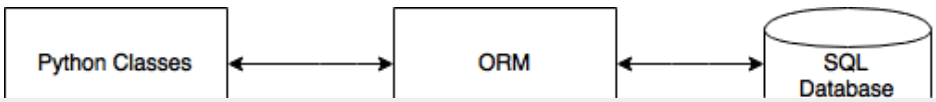
```
$ python get2.py
(u'Michelle', u'Scientist')
(u'Howard', u'Marketeer')
(u'Greg', u'Developer')
```

You may like: Databases and data analysis

ORM with SqlAlchemy

An object relational mapper maps a relational database system to objects. If you are unfamiliar with object orientated programming, read this tutorial first. The ORM is independent of which relational database system is used. From within Python, you can talk to objects and the ORM will map it to the database. In this article you will learn to use the SqlAlchemy ORM.

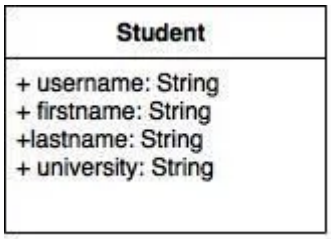
What an ORM does is shown in an illustration below:



- Complete[illegible] hero in Python
- The Complete SQL Bootcamp

Creating a class to feed the ORM

We create the file tabledef.py. In this file we will define a class Student. An abstract visualization of the class below:



Class definition

Observe we do not define any methods, only variables of the class. This is because we will map this class to the database and thus won't need any methods.

This is the contents of tabledef.py:

```
from sqlalchemy import *
from sqlalchemy import create_engine, ForeignKey
from sqlalchemy import Column, Date, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, backref

engine = create_engine('sqlite:///student.db', echo=True)
Base = declarative_base()

########################################################
class Student(Base):
    """"""
    __tablename__ = "student"

    id = Column(Integer, primary_key=True)
```

```
# create tables
Base.metadata.create_all(engine)
```

Execute with:

```
python tabledef.py
```

The ORM created the database file tabledef.py. It will output the SQL query to the screen, in our case it showed:

```
CREATE TABLE student (
        id INTEGER NOT NULL,
        username VARCHAR,
        firstname VARCHAR,
        lastname VARCHAR,
        university VARCHAR,
        PRIMARY KEY (id)
)
```

Thus, while we defined a class, the ORM created the database table for us. This table is still empty.

Inserting data into the database

The database table is still empty. We can insert data into the database using Python objects. Because we use the SqlAlchemy ORM we do not have to write a single SQL query. We now simply create Python objects that we feed to the ORM. Save the code below as dummy.py

```
import datetime
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from tabledef import *

engine = create_engine('sqlite:///student.db', echo=True)
```

```
session.add
```



```
# commit the record the database
session.commit()
```

Execute with:

```
python dummy.py
```

The ORM will map the Python objects to a relational database. This means you do not have any direct interaction from your application, you simply interact with objects. If you open the database with SQLiteman or an SQLite database application you'll find the table has been created:

	id	username	firstname	lastname	university
1	1	james	James	Boogie	MIT
2	2	lara	Lara	Miami	UU
3	3	eric	Eric	York	Stanford

Data in database table.

Query the data

We can query all items of the table using the code below. Note that Python will see every record as a unique object as defined by the Students class. Save the code as demo.py

```
import datetime
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from tabledef import *

engine = create_engine('sqlite:///student.db', echo=True)

# create a Session
```

```
Eric York
```


To select a single object use the filter() method. A demonstration below:

```
import datetime
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from tabledef import *

engine = create_engine('sqlite:///student.db', echo=True)

# create a Session
Session = sessionmaker(bind=engine)
session = Session()

# Select objects
for student in session.query(Student).filter(Student.first
    print student.firstname, student.lastname
```

Output:

```
Eric York
```

Finally, if you do not want the ORM the output any of the SQL queries change the create_engine statement to:

```
engine = create_engine('sqlite:///student.db', echo=False)
```

- Robotics
- Matplotlib
- Network
- Machine Learning