Databases with SQlite using Python (Tutorial)

Chapter · April 2020

CITATIONS
0

READS
16,410

1 author:

Muhammad Ilyas
Université Paris-Est Créteil Val de Marne - Université Paris 12
15 PUBLICATIONS **46** CITATIONS

SEE PROFILE

Databases with SQlite using Python (Tutorial)

Introduction

An organized file for storing data is known as database. A database can be easily accessed and developed through proper modeling, complex design and techniques. Normally a database can be manipulated electronically from a computer. To communicate with the database, we need a DataBase Management System (DBMS), that creates a bridge between the end user and data. DBMS offers a facility to administer the data inside the database.

In this section of chapter, we will provide some useful links to install SQlite and the required documentation for connecting python with existing databases or newly created databases. In these links, a brief introduction is provided for SQlite. In the second section, steps are described to create connection between database and python. While in third section, an example for database creation is provided and in forth section, an exercise is given for practice.

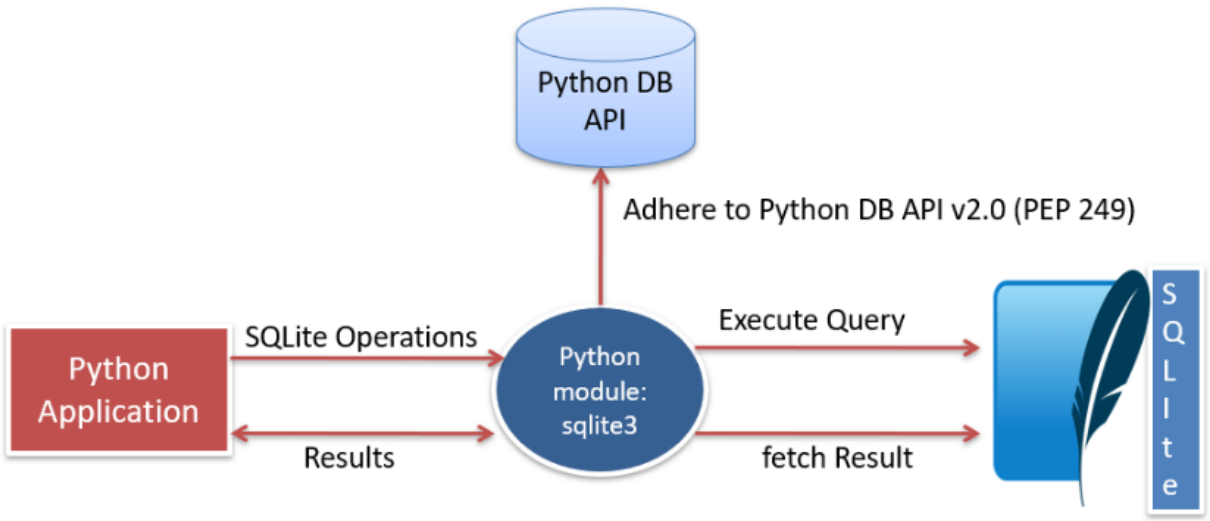
1. Links

A brief introduction for SQlite and related information is briefly explained here:
https://www.sqlite.org/index.html

SQlite can be downloaded according to the operating system here:
https://www.sqlite.org/download.html

The required library is :

https://pypi.org/project/db-sqlite3/



SQlite module

Connection between python and SQlite

Open command prompt and write: *pip install db-sqlite3*

Write the following code in python IDE: *import sqlite3*

Connecting to Database

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

```
import sqlite3

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

print ("Opened database successfully")
```

We can provide the name of the database here in this section *:memory:* to create a database in RAM which can renewed each time. Run the above program to create a database *test_abc.db* in the directory, while you can choose any kind of format such csv, text, etc. The path can be modified accordingly. Save code in test.py file as required and execute the code. After successful execution the following message will be displayed.

```
Opened database successfully
[Finished in 0.2s]
```

2. Create a Table

The following Python program can create a table in the previously created database. We need to create the cursor to handle the database.

```
import sqlite3

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


c = conn.cursor()

c.execute("""CREATE TABLE students (
            FIRST text,
            LAST text,
            AGE INT,
            EMAIL text,
            ID VARCHAR
            )""")

print ("Table created successfully")

conn.close()
```

INSERT Operation

The Python program below will insert the information to students table created in the example above

```
import sqlite3

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

c = conn.cursor()

c.execute("INSERT INTO students (FIRST,LAST,AGE,EMAIL,ID) \
      VALUES ('Jean', 'Paul', 32, 'JP@HOTMAIL.COM', 1 )");

c.execute("INSERT INTO students (FIRST,LAST,AGE,EMAIL,ID) \
      VALUES ('Sandra', 'Allen', 25, 'SA@HOTMAIL.COM', 2)");

c.execute("INSERT INTO students (FIRST,LAST,AGE,EMAIL,ID) \
      VALUES ('Salma', 'Teddy', 23, 'ST@HOTMAIL.COM', 3 )");

c.execute("INSERT INTO students (FIRST,LAST,AGE,EMAIL,ID) \
      VALUES ('Zack', 'Mark', 25, 'ZM@HOTMAIL.COM', 4 )");

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

```
Opened database successfully
Records created successfully
[Finished in 0.4s]
```

SELECT Operation

The following Python program represent how to fetch and display records from students table.

```
import sqlite3

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

c = conn.cursor()
cursor=conn.execute("SELECT FIRST, LAST, AGE, EMAIL, ID from students")
for row in cursor:
   print ("FIRST = ", row[0])
   print ("LAST = ", row[1])
   print ("AGE = ", row[2])
   print ("EMAIL = ", row[3])
   print ("ID = ", row[4]), "\n"

print ("Operation done successfully");

conn.commit()
conn.close()
```

```
Opened database successfully
FIRST =  Jean
LAST =  Paul
AGE =  32
EMAIL =  JP@HOTMAIL.COM
ID =  1
FIRST =  Sandra
LAST =  Allen
AGE =  25
EMAIL =  SA@HOTMAIL.COM
ID =  2
```

UPDATE Operation

The following code will update the existing information in the students table.

```
import sqlite3

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

c = conn.cursor()

c.execute("UPDATE students set AGE = 22 where ID = 1")

conn.commit()
conn.close()
```

```
Opened database successfully
FIRST =  Jean
LAST =  Paul
AGE =  22
```

DELETE Operation

The use of DELETE statement to delete any record from the students table.

```
import sqlite3

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

c = conn.cursor()

c.execute("DELETE from students where ID = 1")
```

```
Opened database successfully
FIRST =  Sandra
LAST =  Allen
AGE =  25
EMAIL =  SA@HOTMAIL.COM
ID =  2
FIRST =  Salma
LAST =  Teddy
AGE =  23
EMAIL =  ST@HOTMAIL.COM
ID =  3
```

3. Example to practice

Example of creating a database in SQlite using Python

```
#install library
import sqlite3
#importing tha class of the db
from test import Students
#create any kind of document with an extension to .csv .db . xlx  .text etc
#conn = sqlite3.connect('students.csv')

#this can take the db from the memory each time creates a fresh new database
conn = sqlite3.connect(':memory:')

#creating cursor to connect to the db and handling the db
c = conn.cursor()
#""" is just to write a string withour multiple breaks -long
c.execute("""CREATE TABLE test (
        first text,
        last text,
        ID int
        )""")

#A function that insert the ID
def insert_emp(emp):
   with conn:
      c.execute("INSERT INTO test VALUES (:first, :last, :ID)", {'first': emp.first, 'last': emp.last, 'ID': emp.ID})

#A function that search in data
def get_emps_by_name(lastname):
   c.execute("SELECT * FROM test WHERE last=:last", {'last': lastname})
#dislay all the data related with the proposed condition
    return c.fetchall()
#Display only one value with the proposed condition
   #return c.fetchone()
#Display first five value with the proposed condition or according to requirements
   #return c.fetchmany(5)

#A function that updates the ID
def update_ID(emp, ID):
   with conn:
      c.execute("""UPDATE test SET ID = :ID
            WHERE first = :first AND last = :last""",
```

```
            {'first': emp.first, 'last': emp.last, 'ID': ID})

#A function that remove employee
def remove_emp(emp):
    with conn:
        c.execute("DELETE from test WHERE first = :first AND last = :last",
            {'first': emp.first, 'last': emp.last})

#Information of the students
emp_1 = Students('abdel' , 'tst', 123)
emp_2 = Students('peter' , 'zee', 125')
emp_3 = Students('kahina', 'abc', 128)
emp_4 = Students('assia' , 'abz', 126')

#Inserting all the data at once
insert_emp(emp_1)
insert_emp(emp_2)
insert_emp(emp_3)
insert_emp(emp_4)

emps = get_emps_by_name('zee')
print(emps)

#Updating existing data
update_ID(emp_2, 123)
#Remove an entry
remove_emp(emp_1)

emps = get_emps_by_name('zee')
print(emps)
#Closing the created connection to the proposed table
conn.close()
#Connection.commit() is a necessary statement after any modification to the table
by adding, updating or deleting.
```

Function to add data automatically to the database

```
class Students:
    """A sample Students class"""

    def __init__(self, first, last, ID):
        self.first = first
        self.last = last
        self.ID = ID
```

```
@property
def email(self):
    return '{}.{}@email.com'.format(self.first, self.last)

@property
def fullname(self):
    return '{} {}'.format(self.first, self.last)

def __repr__(self):
    return "Students('{}', '{}', {})".format(self.first, self.last, self.ID)
```

4. Exercise

-Create a database for doctors having information about id, first name, last name, joining date, salary, and address.

-Use a function to manipulate the database such as select statement, insert statement, updating etc.

References:

[1] Krogh, J. W., Krogh, & Gennick. (2018). *MySQL Connector/Python Revealed*. Apress.

[2] Beazley, D. M. (2009). *Python essential reference*. Addison-Wesley Professional.