Unit-7
Working with databases: SQLite and MySQL

Prof. Manan Thakkar

Assistant Professor, Dept. of Computer Engg.

UVPCE, Ganpat University, Mehsana

Need of Database

- Desktop, web and mobile applications store data and present it to users in a useful way.
- For example, Google stores data about roads and provides directions to get from one location to another by driving through the Maps application. Driving directions are possible because the data is stored in a structured format.
- Collection of interrelated data is called as Database. Databases make storage structured and ordered. Hence, accessing and retrieving data becomes reliable and fast.
- Relational model is a structure widely used to store data in database. It is also called as RDBMS (Relational Database Management Systems)
- Relational databases store data in a series of tables. Every row is accessed using unique primary key. Interconnections between the tables are specified as *foreign keys*. A foreign key is a unique reference from one row in a relational table to another row in a table, which can be the same table but is most commonly a different table.

Python Databases

- The Python standard for database interfaces is DB-API. Most Python database interfaces follow this standard.
- Python Database API supports a wide range of database servers such as:

1) GadFly
2) mSQL
3) MySQL
4) PostgreSQL
5) Microsoft SQL Server 2000
6) Informix
7) Interbase
8) Oracle
9) Sybase

Commonly used databases for Python Applications

- PostgreSQL and MySQL are two of the most common open source databases for storing Python web applications' data.
- SQLite is a database that is stored in a single file on disk. SQLite is built into Python but is only built for access by a single connection at a time. Therefore is highly recommended to not run a production web application with SQLite.
- SQLite is a C library that provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.
- Some applications can use SQLite for internal data storage.
- It’s also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

SQLite3 Database

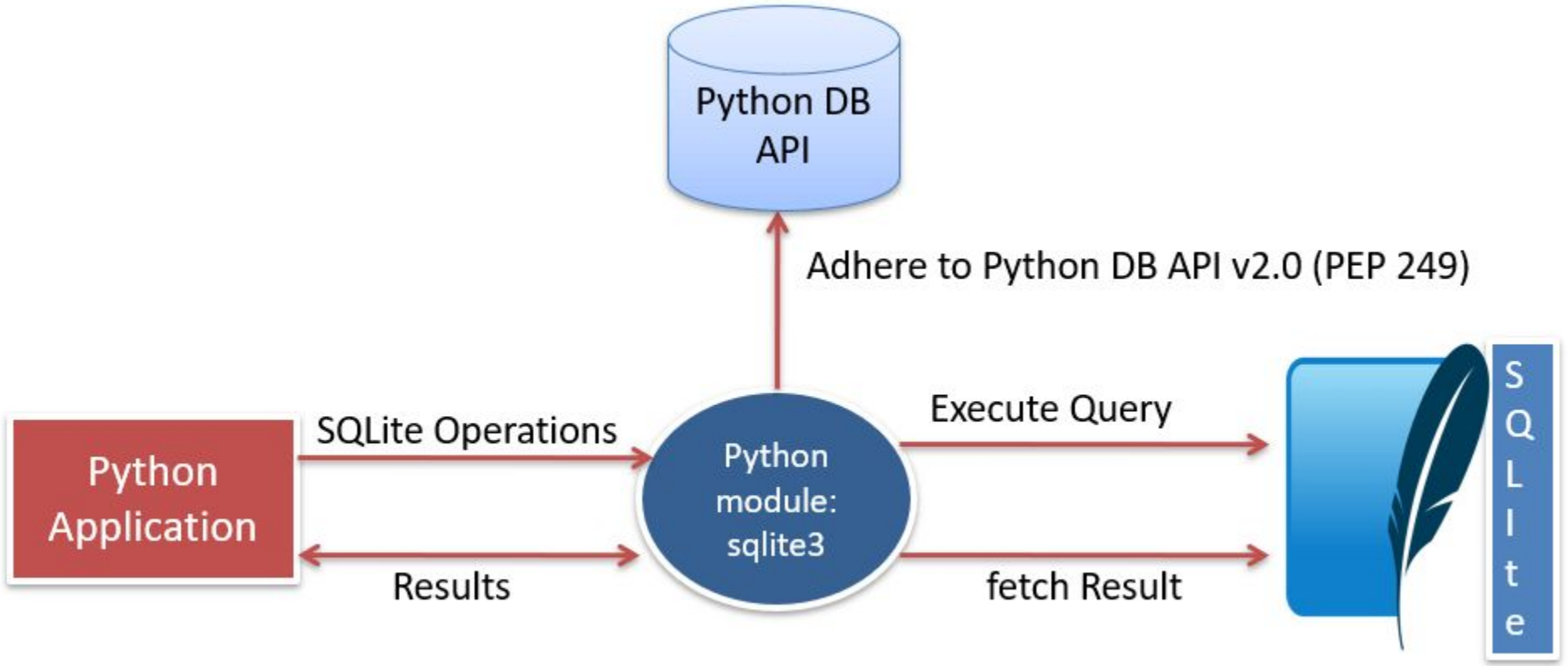
- SQLite3 database can be integrated with Python using sqlite3 module, which was written by Gerhard Haring. It provides an SQL interface compliant with the DB-API 2.0 specification.
- You **do not need to install** this module separately because it is shipped by default along with Python version 2.5.x onwards.
- To use SQLite database, perform following steps:

1) Import sqlite3 module
2) Create a connection object that represents the database.
3) Create a cursor object with help of connection object.
4) Using cursor object call execute / executemany / executescript method to perform SQL statements.
5) Close the connection & cursor.

NOTE: To see contents of SQLite3 database, you will need tool "DB browser for SQLite". Download and install that tool, the link is given below:

https://github.com/sqlitebrowser/sqlitebrowser/releases/download/v3.12.1/DB.Browser.for.SQLite-3.12.1-win64-v2.msi

Working of Python & sqlite3 module



SQLite DataTypes

- NULL: The value is a NULL value.
- INTEGER: To store the numeric value. The integer stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the number.
- REAL: The value is a floating-point value, for example, 3.14 value of Pi
- TEXT: The value is a text string, TEXT value stored using the UTF-8, UTF-16BE or UTF-16LE encoding.
- BLOB: The value is a blob of data, i.e., binary data. It is used to store images and files.

Equivalency of Python and SQLite Datatypes

Python type	**Equivalent SQLite type**
None	NULL
int	INTEGER
float	REAL
str	TEXT
bytes	BLOB

Data Type	Affinity
▪ INT ▪ INTEGER ▪ TINYINT ▪ SMALLINT ▪ MEDIUMINT ▪ BIGINT ▪ UNSIGNED BIG INT ▪ INT2 ▪ INT8	INTEGER
▪ CHARACTER(20) ▪ VARCHAR(255) ▪ VARYING CHARACTER(255) ▪ NCHAR(55) ▪ NATIVE CHARACTER(70) ▪ NVARCHAR(100) ▪ TEXT ▪ CLOB	TEXT
▪ BLOB ▪ no datatype specified	NONE
▪ REAL ▪ DOUBLE ▪ DOUBLE PRECISION ▪ FLOAT	REAL
▪ NUMERIC ▪ DECIMAL(10,5) ▪ BOOLEAN ▪ DATE ▪ DATETIME	NUMERIC

This table shows conversion from common datatypes to basic affinities.

Establishing connection & Create operation

- **Establishing connection with database:**

```
import sqlite3
try:
    con=sqlite3.connect('test.db')
except:
    print('Error occured')
```

- **Creating table in database:**

```
import sqlite3
con=sqlite3.connect('test.db')
cur=con.cursor()
cur.execute('create table company (id int primary key,name text, age int, city char(50),salary real)’)
con.close()
```

Insert operation

- Example:

```
import sqlite3
con=sqlite3.connect('test.db')
cur=con.cursor()
cur.execute('create table company (id int primary key,name text, age int, city char(50),salary real)’)
      # If table already exists, it raises error.
cur.execute('insert into company values(5,"aman", 20,"surat",15200.50)')
cur.execute('insert into company values(2,"tapan", 22,"Ahmedabad",18600.70)')
cur.execute('insert into company values(4,"haresh", 25,"Vadodara",11400.30)')
cur.execute('select * from company')
con.commit()
for row in cur:
    print (row)
con.close()
```

Output:

```
(5, 'aman', 20, 'surat', 15200.5)
(2, 'tapan', 22, 'Ahmedabad', 18600.7)
(4, 'haresh', 25, 'Vadodara', 11400.3)
```

Update operation

- **Continuing previous database and table:**

```
import sqlite3
con=sqlite3.connect('test.db')
cur=con.cursor()
cur.execute('update company set city="rajkot" where id=2')
print(cur.rowcount,'Rows updated')
cur.execute('select * from company')
total_rows=cur.fetchall()
print('Result: ', total_rows)
print('Total rows are: ',len(total_rows))
con.commit()
```

Note: rowcount works with all DML commands except ‘select’ operation. For ‘select’ operation it returns value (-1).

Output:

```
1 Rows updated
Result:  [(5, 'aman', 20, 'surat', 15200.5), (2, 'tapan', 22, 'rajkot', 18600.7), (4, 'haresh', 25, 'Vadodara', 11400.3)]
Total rows are:  3
```

Delete operation

- **Continuing previous database and table:**

```
import sqlite3
con=sqlite3.connect('test.db')
cur=con.cursor()
print('Number of Rows deleted: ',end="")
print(cur.execute('delete from company where id=2').rowcount)
cur.execute('select * from company')
total_rows=cur.fetchall()
print('Remaining Rows: ', total_rows)
print('Number of remaining rows are: ',len(total_rows))
con.commit()
```

Output:

```
Number of Rows deleted: 1
Remaining Rows:  [(5, 'aman', 20, 'surat', 15200.5), (4, 'haresh', 25, 'Vadodara', 11400.3)]
Number of remaining rows are:  2
```

Drop operation

- **Continuing previous example:**

```
import sqlite3
con=sqlite3.connect('test.db')
cur=con.cursor()
cur.execute('drop table company')
```

Placeholder, Timeout, total_changes, execute(), executemany(), fetchone(), fetchmany() & fetchall()

```
import sqlite3
con=sqlite3.connect('test.db',timeout=10)  #defaulit timeout=5 sec
cur=con.cursor()
cur.execute('insert into company values(5,?,20,"surat",?)',('aman',15200.50))  #first-way
val=(4,"haresh", 25,"Vadodara",11400.30)
cur.execute('insert into company values(?,?,?,?,?)',val)     #second-way
examples = [(2, "tapan"), (7, "raman")]
cur.executemany('insert into company values(?,?,20,"Ahmedabad",15000)',examples)
cur.execute('select * from company')
first_row=cur.fetchone()
print('Total columns: ', len(first_row))
print('First row: ',first_row)
print('Next two rows: ',cur.fetchmany(2))
print('Remaining rows: ',cur.fetchall())
print('Total rows affected since connection was opened:',con.total_changes)
con.commit()
```

Cntd…

- **Output of previous program:**

Total columns: 5

First row: (5, 'aman', 20, 'surat', 15200.5)

Next two rows: [(4, 'haresh', 25, 'Vadodara', 11400.3), (2, 'tapan', 20, 'Ahmedabad', 15000.0)]

Remaining rows: [(7, 'raman', 20, 'Ahmedabad', 15000.0)]

Total rows affected since connection was opened: 4

Placeholder, executemany() with update & delete operation

```
import sqlite3
con=sqlite3.connect('test.db')
cur=con.cursor()
options=[('haresh',),('akshat',)]
cur.executemany('update company set city="rajkot" where name=?',options)
my_lst=[(2,),(5,)]
cur.executemany('delete from company where id=?',my_lst)
print('Total rows affected since connection was opened:',con.total_changes)
con.commit()
con.close()
```

Output: Total rows affected since connection was opened: 3

executescript() method

- Example:

```
import sqlite3
con=sqlite3.connect('test.db')
cur=con.cursor()
f=open('direct_sql.sql','r')
query=f.read()
cur.executescript(query)
con.commit()
con.close()
f.close()
```

Output: There is no output, only database is modified. (check database for output)

Note: Content of file ‘direct_sql.sql’ is given in next slide

```
CREATE TABLE DEPOSIT(ACTNO NUMBER(5), CNAME VARCHAR2(18),BNAME VARCHAR2(18),AMOUNT NUMBER(8,2), ADATE DATE);
INSERT INTO DEPOSIT VALUES(100,'ANIL','VRCE',1000.00,'1-MAR-1995');
INSERT INTO DEPOSIT VALUES(101,'SUNIL','AJNI',5000.00,'4-JAN-1996');
INSERT INTO DEPOSIT VALUES(102,'MEHUL','KAROLBAGH',3500.00,'17-NOV-1995');
INSERT INTO DEPOSIT VALUES(104,'MADHURI','CHANDI',1200.00,'17-DEC-1995');
INSERT INTO DEPOSIT VALUES(105,'PRAMOD','M.G.ROAD',3000.00,'27-MAR-1996');
INSERT INTO DEPOSIT VALUES(106,'SANDIP','ANDHERI',2000.00,'31-MAR-1996');
INSERT INTO DEPOSIT VALUES(107,'SHIVANI','VIRAR',1000.00,'5-SEP-1995');
INSERT INTO DEPOSIT VALUES(108,'KRANTI','NEHRUPLACE',5000.00,'2-JULY-1995');
INSERT INTO DEPOSIT VALUES(109,'MINU','POWAI',7000.00,'10-AUG-1995');

CREATE TABLE BRANCH(BNAME VARCHAR2(18),CITY VARCHAR2(18));
INSERT INTO BRANCH VALUES('VRCE','NAGPUR');
INSERT INTO BRANCH VALUES('AJNI','NAGPUR');
INSERT INTO BRANCH VALUES('KAROLBAGH','DELHI');
INSERT INTO BRANCH VALUES('CHANDI','DELHI');
INSERT INTO BRANCH VALUES('DHARAMPETH','NAGPUR');
INSERT INTO BRANCH VALUES('M.G.ROAD','BANGLORE');
INSERT INTO BRANCH VALUES('ANDHERI','BOMBAY');
INSERT INTO BRANCH VALUES('VIRAR','BOMBAY');
INSERT INTO BRANCH VALUES('NEHRUPLACE','DELHI');
INSERT INTO BRANCH VALUES('POWAI','BOMBAY');
```

Methods used in Sqlite3 module

SrNo	Methods	Description
1	sqlite3.connect(database [,timeout ,other optional arguments])	This method opens a connection to the SQLite database file. If database is opened successfully, it returns a connection object. timeout specifies how long the connection should wait for the lock to go away until raising an exception. The default value is 5.0 (seconds). If the given database name does not exist then new database is created. Specify filename with path if you want to create a database other than current directory.
2	connection.commit()	This method commits i.e. permanently stores the current transaction.
3	connection.rollback()	This method rolls back any changes to the database since the last call to commit().
4	connection.close()	This method closes the database connection. Note that it does not automatically call commit(). If you close database connection without calling commit(), your changes will be lost.
5	connection.cursor()	This routine creates a cursor which will be used throughout your database access.

Cntd…

SrNo	Methods	Description
6	cursor.execute(sql [, optional parameters])	This routine executes an SQL statement. The SQL statement may be parameterized (i. e. placeholders instead of SQL literals). The sqlite3 module supports two kinds of placeholders: question marks and named placeholders (named style). For example − cursor.execute("insert into people values (?, ?)", (who, age))
7	cursor.executemany(sql, seq_of_parameters)	This routine executes an SQL command against all parameter sequences or mappings found in the sequence sql.
8	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 a semi colon (;).
9	cursor.fetchone()	This method fetches the next row from query result set. It returns a single sequence, or None when no more data is available.
10	cursor.fetchmany([size = cursor.arraysize])	This routine fetches the next set of rows from query result set, 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.

Cntd…

SrNo	Methods	Description
11	cursor.fetchall()	This routine fetches all (remaining) rows from query result set, returning a list. An empty list is returned when no rows are available.
12	cursor.rowcount	This parameter returns number of rows affected after execution of last insert, update or delete command.
13	connection.total_changes	This parameter returns the total number of database rows that have been modified, inserted, or deleted since the database connection was opened.

Note: TCL commands like commit & rollback works with DML commands only

MySQL database

- MySQL is one of the widely used database. Unlike SQLite, MySQL is an external database, and it creates a database server. To use it thorough python program we need following softwares:

1) Python 3.6 or later: Add 'path' of your 'python.exe' file using 'Environment Variables' otherwise, you won't be able to install MySQL.

2) mysql-connector-python

Python needs MySQL driver to access MySQL database, which is named as: mysql-connector-python. To download & install driver, open **windows command prompt** and write:

```
python -m pip install mysql-connector-python
```

Remember, your system must be connected with internet to install above driver.

3) MySQL database:

We will use web version of MySQL database. Download and install XAMPP distribution from following link to use web MySQL:

https://downloadsapachefriends.global.ssl.fastly.net/7.3.27/xampp-windows-x64-7.3.27-1-VC15-installer.exe?from_af=true

Cntd…

- After installing xampp distribution, open *XAMPP control panel* from start menu.
- Start “Apache” and “MySQL” server by clicking on “Start” button.
- Now, open any web browser and type following link:

  http://127.0.0.1/dashboard/
- Click on “phpMyAdmin” residing on top right corner to open MySQL database. This is homepage of web MySQL database.
- To verify, installation of MySQL and mysql-connector-python, write following lines in python IDLE shell:

```
>>> import mysql.connector
>>>mysql.connector.connect(host='localhost', user='root', password='')
```

- If there is no error, means python and MySQL connection is established.

Steps to connect python application with MySQL database

1) Import mysql.connector module

2) Create the connection object with appropriate parameters.

3) Using connection object, create cursor object.

4) With the help of cursor object, call execute or executemany method to perform SQL statements.

5) Close connection object

Create database

```
import mysql.connector
mycon=mysql.connector.connect(host='localhost',user='root',password='Nilesh@6')
cur=mycon.cursor()
cur.execute('show databases')
for x in cur:
    print (x)
cur.execute('create database test')
cur.execute('show databases')
for x in cur:
    print (x)
```

Output:

```
('information_schema',)
('mysql',)
('performance_schema',)
('sys',)
----------------------------------------
('information_schema',)
('mysql',)
('performance_schema',)
('sys',)
('test',)
```

Create operation

- Consider following code which creates ‘employee’ table in ‘test’ database.

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='
Nilesh@6 ')
cur=con.cursor()
cur.execute('create table employee(id int primary key, name varchar(20) not null,salary
float)')
cur.execute('show tables')
for i in cur:
    print(i)
con.close()
```

Output: ('employee',)

Insert operation

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('insert into employee values(101,"Arbaz",28000.65)')
cur.execute('insert into employee values(102,%s,30000.24)',("Rupesh",))
cur.execute('insert into employee values(%s,%s,%s)',(103,'john',25000.52))
val=[(105,"hiren",27750.38),(107,"arpan",22370.81)]
cur.executemany('insert into employee values(%s,%s,%s)',val)
con.commit()
print(cur.rowcount,"rows inserted")
con.close()
```

Output:

2 rows inserted

Important Point

- The mysql.connector module uses ‘%s’ as the placeholder to escape values in any SQL statement. For example,

cursor.execute(‘insert into employee values (%s,%s), (102, ‘arpit’)’)

- You can see that, 102 is an integer, but ‘%s’ is used for it, because here ‘%s’ is a placeholder and **NOT a** format specifier.
- In python ‘%s’ is a format specifier. For example,

print(‘Hello, my name is %s and I am %d year old’ %(‘amar’,22))

- You can see here, for integer value ‘%d’ is used.
- That means, whenever you are writing SQL statement for MySQL, consider ‘%s’ as place holder and whenever you are writing python code ‘%s’ works as format specifier.

Select operation

- Example-1:

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('select * from employee')
firstrow=cur.fetchone()
print('First row is:',firstrow)
result=cur.fetchall()
for i in result:
    print (i)
cur.execute('select id,salary from employee')
con.commit()
con.close()
```

Output:

```
First row is: (101, 'Arbaz', 28000.7)
(102, 'Rupesh', 30000.2)
(103, 'john', 25000.5)
(105, 'hiren', 27750.4)
(107, 'arpan', 22370.8)
```

Cntd…

- Example-2:

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('select id,name from employee where name like "a%"')
firstrow=cur.fetchone()
print('First row of result:',firstrow)
result=cur.fetchall()
for row in result:
    print ('%d %s'%(row[0],row[1]))
con.commit()
con.close()
```

Output:

```
First row of result: (101, 'Arbaz')
107 arpan
```

Update operation

- Example:

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('update employee set name="tarun" where id=103')
sql='update employee set name=%s where id=%s'
val=("naman",105)
cur.execute(sql,val)
print(cur.rowcount,"row affected")
con.commit()
con.close()
```

Output:

1 row affected

Delete operation

- Example:

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('delete from employee where id in (101,105)')
print(cur.rowcount,"row deleted")
sql='delete from employee where name=%s'
val=('tarun',)
cur.execute(sql,val)
con.commit()
con.close()
```

Output: 2 row deleted (It shows statistics about first delete operation)

Drop table and Join operation

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('drop table employee')
cur.execute('create table emp (id int primary key, name text)')
cur.execute('create table person (name varchar (20) primary key, city varchar(20))')
val=[(101,'ashish'),(103,'laxman'),(104,'arjun')]
cur.executemany('insert into emp values(%s,%s)',val)
temp=[('supan','Ahmedabad'),('ashish','Mumbai'),('vishwa','Delhi')]
cur.executemany('insert into person values(%s,%s)',temp)
cur.execute('select e.id,e.name,p.city from emp e join person p on e.name=p.name')
m=cur.fetchall()
for row in m:
    print(row)
con.commit()
con.close()
```

Output:

(101, 'ashish', 'Mumbai')

Limit operation

- You can limit the number of records returned from the query, by using "LIMIT" statement. For Example,

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('create table emp1 ( id int, name text)')
temp=[(101,'ashish'),(103,'laxman'),(104,'arjun'),(102,'karan'),(105,'irfan'),(106,'Reema'),(108,'Advait'),(107
,'rushi')]
cur.executemany('insert into emp1 values(%s,%s)',temp)
cur.execute('select * from emp1 limit 5')
m=cur.fetchall()
for row in m:
    print(row)
con.commit()
con.close()
```

Output:

```
(101, 'ashish')
(103, 'laxman')
(104, 'arjun')
(102, 'karan')
(105, 'irfan')
```

Cntd…

- If you want to return five records, starting from the third record, you can use the "OFFSET" keyword. For example, (Considering previous table ‘emp1’)

```
import mysql.connector
con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
cur=con.cursor()
cur.execute('select * from emp1 limit 5 offset 2')
m=cur.fetchall()
for row in m:
    print(row)
con.commit()
con.close()
```

Output:

```
(104, 'arjun')
(102, 'karan')
(105, 'irfan')
(106, 'Reema')
(108, 'Advait')
```

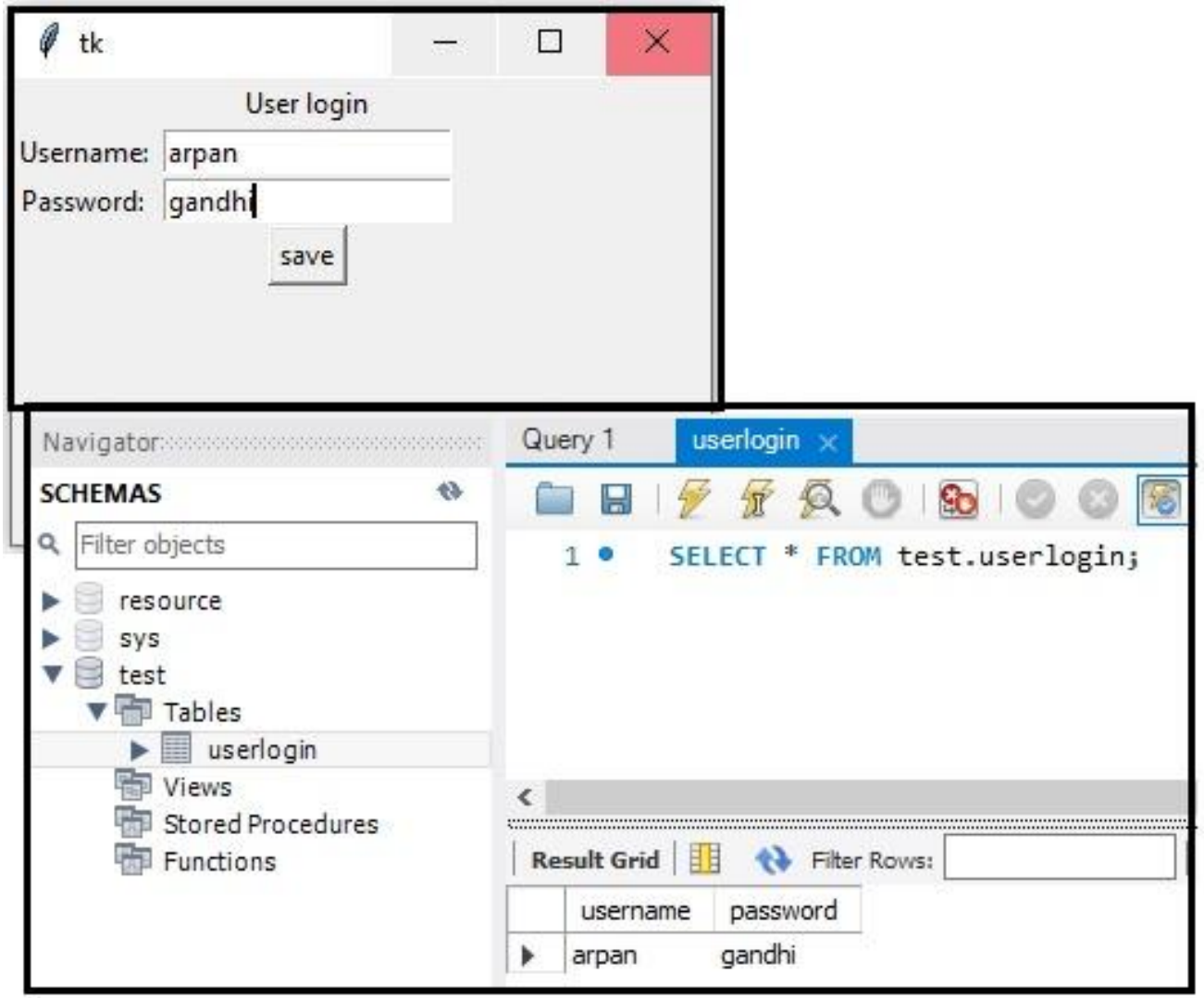
Tkinter and MySQL Database Connectivity Example-1

- Example:

```
import mysql.connector
from tkinter import *
top=Tk()
top.geometry("300x200")
lb=Label(text="User login")
lb.grid(row=0,column=1)
lb2=Label(text="Username: ")
lb2.grid(row=2,column=0)
en1= Entry(top)
en1.grid(row=2,column=1)
lb3=Label(text="Password: ")
lb3.grid(row=3,column=0)
en2= Entry(top)
en2.grid(row=3,column=1)
```

(code continue on next slide)

Output:



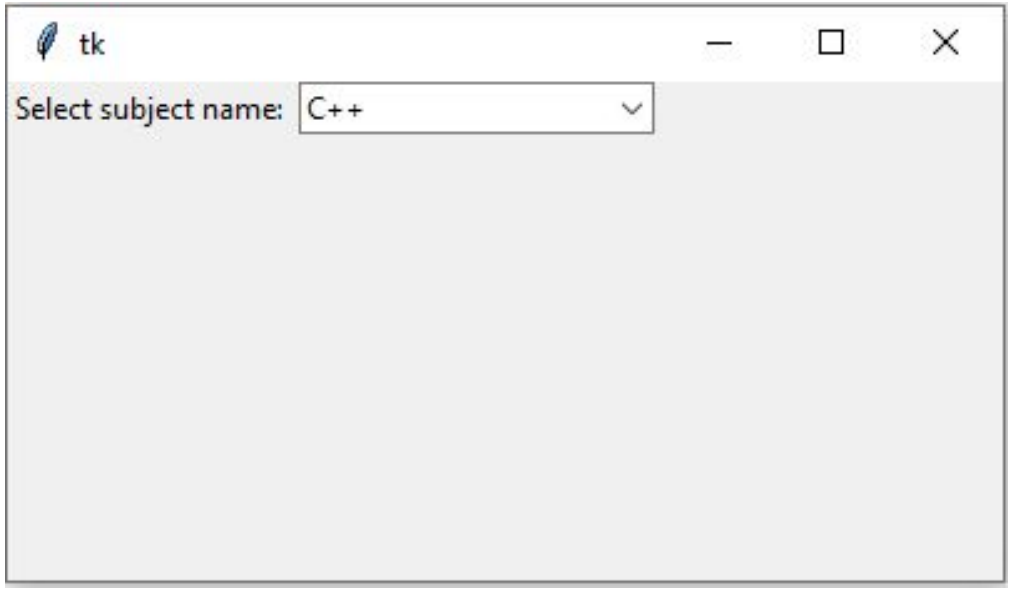
```
def save():
    m=en1.get();n=en2.get()
    con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
    cur=con.cursor()
    cur.execute('create table if not exists userlogin (username text, password char(20))')
    cur.execute('select * from userlogin')
    temp=cur.fetchall()
    first=[x[0] for x in temp]
    if m in first:
        cur.execute('update userlogin set password=%s where username=%s',(n,m))
    else:
        cur.execute('insert into userlogin values (%s,%s)',(en1.get(),en2.get()))
    con.commit()
    con.close()
b=Button(text="save",command=save)
b.grid(row=4,column=1)
mainloop()
```

Combobox widget of ttk (Themed Tkinter)

- Combobox is a drop down list, which can hold multiple values and show one item at a time. Combobox is useful for select one option in many choice.
- Combobox widget is a class of ttk module of tkinter library, so you need to import this module. Let's take an example,

```
from tkinter import *
from tkinter import ttk
top= Tk()
top.geometry("400x200")
lbl= Label(text="Select subject name: ")
lbl.grid(row=0,column=0)
language=["Python","Java","C++","C"] #Tuple can also be used
cb=ttk.Combobox(top,values=language)
cb.current(2)   # Default selected item is at index 1
cb.grid(row=0,column=1)
mainloop()
```

Output:

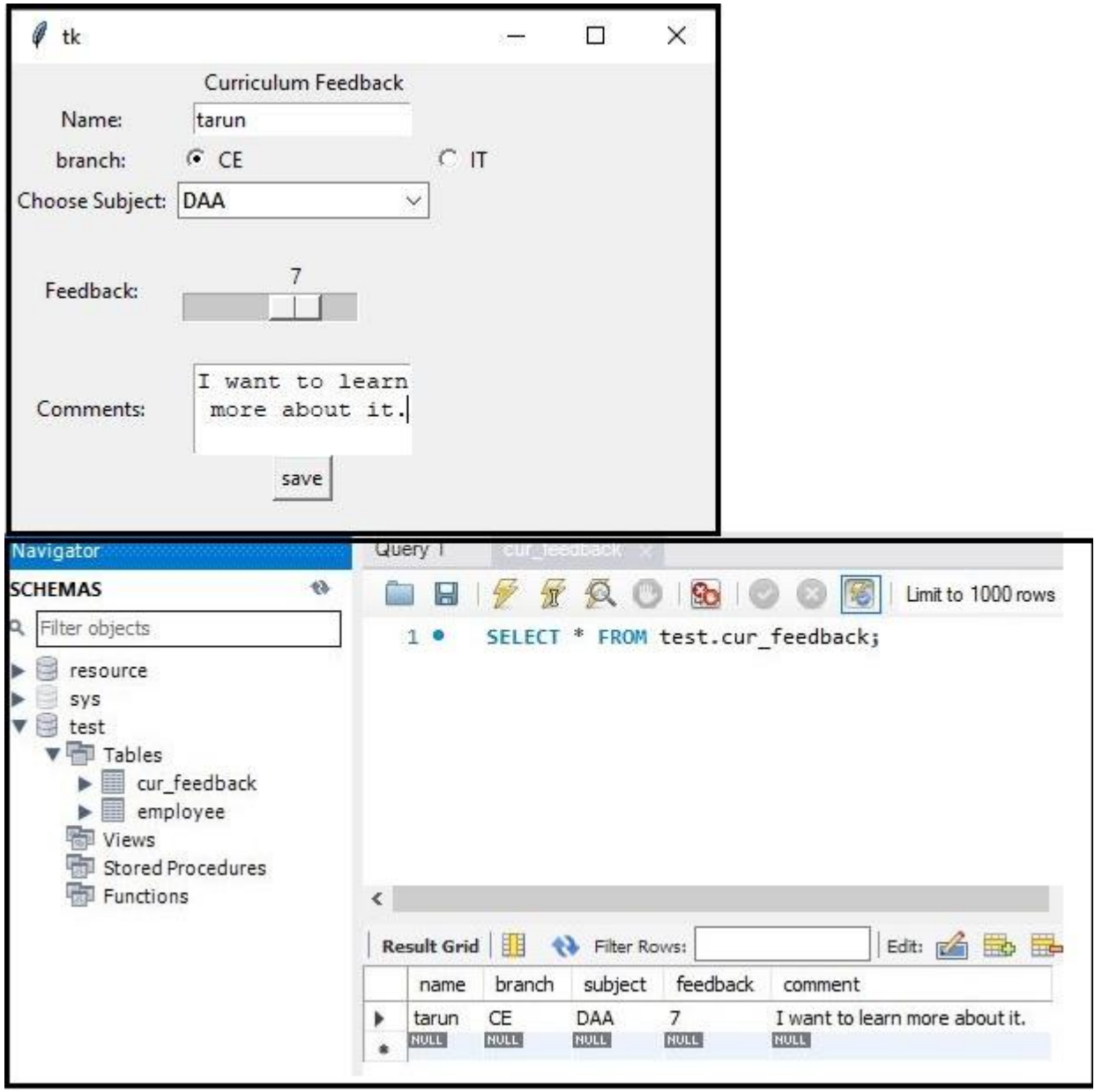


Tkinter and MySQL Database Connectivity Example-2

```
import mysql.connector
from tkinter import *
from tkinter import ttk
top=Tk()
top.geometry("400x300")
lb=Label(text="Curriculum Feedback")
lb.grid(row=0,column=1)
lb1=Label(text="Name: ")
lb1.grid(row=2,column=0)
en1= Entry(top)
en1.grid(row=2,column=1)
lb2=Label(text="branch: ")
lb2.grid(row=3,column=0)
branch=StringVar(top,"CE")
rb1=Radiobutton(top,text="CE",variable=branch,value="CE")
rb1.grid(row=3,column=1,sticky=W)
rb2=Radiobutton(top,text="IT",variable=branch,value="IT")
rb2.grid(row=3,column=2,sticky=W)
lb3=Label(text="Choose Subject: ")
lb3.grid(row=5,column=0)
```

(code continue on next slide)

Output:



```
subjects=["Python","ADT", "OS","Maths","DAA"]
cb=ttk.Combobox(top,values=subjects)
cb.current(0)
cb.grid(row=5,column=1)
lb4=Label(text="Feedback: ")
lb4.grid(row=6,column=0)
my_scale=IntVar()
sc= Scale(top,from_=0,to=10,variable=my_scale,orient=HORIZONTAL)
sc.grid(row=6,column=1,columnspan=2,sticky=W,pady=20)
lb5=Label(text="Comments: ")
lb5.grid(row=7,column=0)
t1=Text(top,width=15,height=3)
t1.grid(row=7,column=1)
def save():
    m=en1.get(); n=branch.get(); o= cb.get(); p=my_scale.get(); q=t1.get("1.0",END)
    con=mysql.connector.connect(host='localhost',database='test',user='root',password='Nilesh@6')
    cur=con.cursor()
    cur.execute('create table if not exists cur_feedback (name char(10) primary key, branch char(2), subject varchar(10),feedback real,comment text)')
    cur.execute('insert into cur_feedback values (%s,%s,%s,%s,%s)',(m,n,o,p,q))
    con.commit()
    con.close()
b=Button(text="save",command=save)
b.grid(row=8,column=1)
mainloop()
```

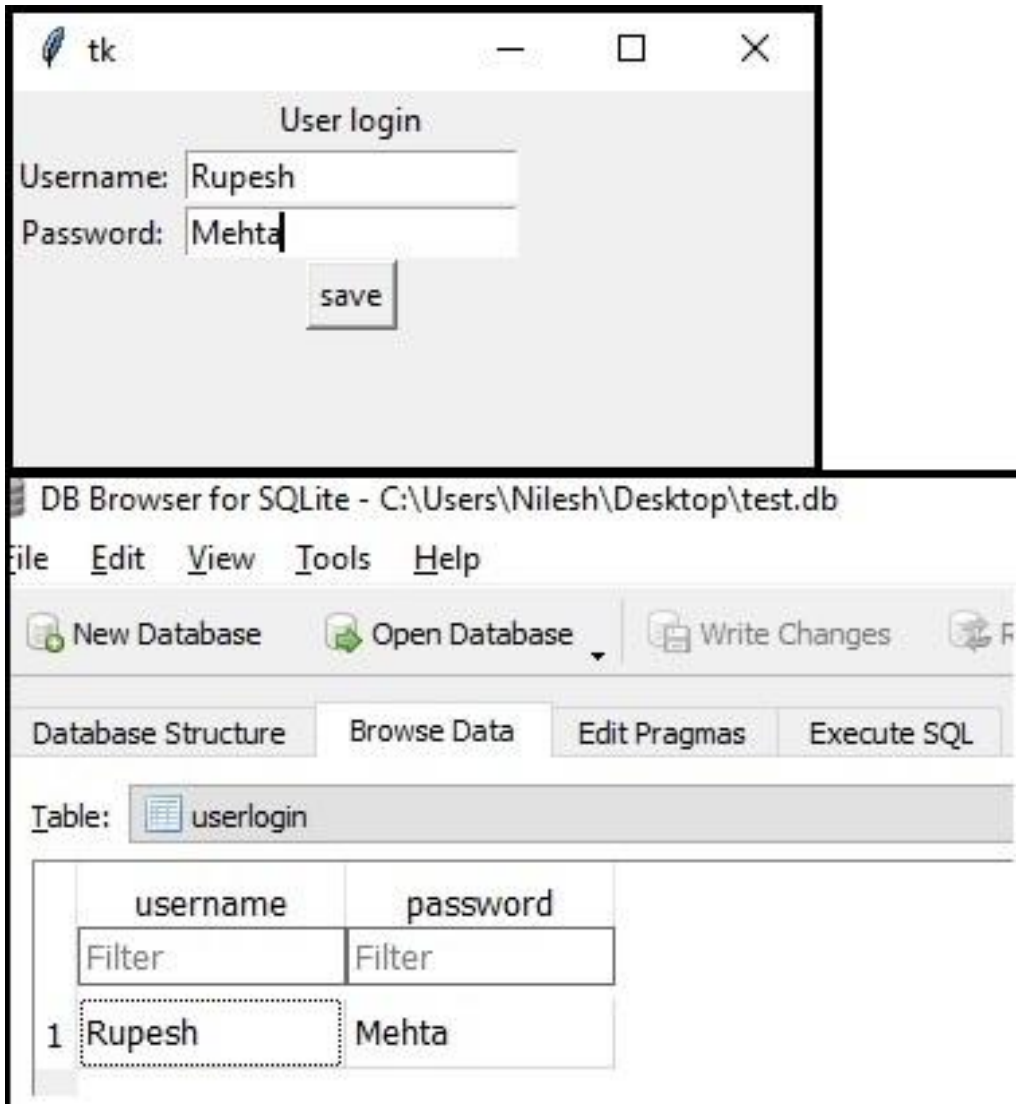
Tkinter & Sqlite3 Database connectivity

- Example:

```
import sqlite3
from tkinter import *
top=Tk()
top.geometry("300x200")
lb=Label(text="User login")
lb.grid(row=0,column=1)
lb2=Label(text="Username: ")
lb2.grid(row=2,column=0)
en1= Entry(top)
en1.grid(row=2,column=1)
lb3=Label(text="Password: ")
lb3.grid(row=3,column=0)
en2= Entry(top)
en2.grid(row=3,column=1)
```

(continue on next slide)

Output:



```
def save():
    m=en1.get();n=en2.get()
    con=sqlite3.connect('test.db')
    cur=con.cursor()
    cur.execute('create table if not exists userlogin (username text, password char(20))')
    cur.execute('select * from userlogin')
    temp=cur.fetchall()
    first=[x[0] for x in temp]
    if m in first:
        cur.execute('update userlogin set password=? where username=?',(n,m))
    else:
        cur.execute('insert into userlogin values (?,?)',(en1.get(),en2.get()))
    con.commit()
    con.close()
b=Button(text="save",command=save)
b.grid(row=4,column=1)
mainloop()
```

Desktop Version of MySQL Database

- MySQL is one of the widely used database. Unlike SQLite, MySQL is an external database, and it creates a database server. To use it thorough python program we need following softwares:

1) Python 3.6 or later: Add 'path' of your 'python.exe' file using 'Environment Variables' otherwise, you won't be able to install MySQL.

2) MySQL database:

Link to download MySQL database: (just click, download will start (400MB))

https://cdn.mysql.com/archives/mysql-installer/mysql-installer-community-8.0.19.0.msi

To get help in installation, see this video: https://youtu.be/kEnD_KN7P-k

3) mysql-connector-python

Python needs a MySQL driver to access MySQL database, which is named as: mysql-connector-python. To download & install driver, open **windows command prompt** and write:

python -m pip install mysql-connector-python

Cntd..

- To verify, installation of MySQL and mysql-connector-python, write following lines in python IDLE shell:

>>> import mysql.connector

>>>mysql.connector.connect(host='localhost',database='mysql',user='root',password='your password')

- If there is no error, means python and MySQL connection is established.
- We will use "MySQL Workbench" to see database contents. This tool already gets installed when you installed MySQL means it is part of your MySQL installer package. So, **NO need** to install it separately. You just need to open it from windows start button.
- After opening "MySQL workbench", create a new connection. When asked, enter password which you had given at the time of installation for 'root' user.

Note: To install & connect MySQL with anaconda and its all IDEs (spyder, jupyter notebook, VScode etc.) see this video: https://www.youtube.com/watch?v=z0jVsEuuLLc