Unit V

# CHAPTER 15

# DATA MANIPULATION THROUGH SQL

## Learning Objectives

After the completion of this chapter, the student will be able to write Python script to

- Create a table and to add new rows in the database.
- Update and Delete record in a table
- Query the table
- Write the Query in a CSV file

QIM6PU

## 15.1 Introduction

A database is an organized collection of data. The term "database" can both refer to the data themselves or to the database management system. The Database management system is a software application for the interaction between users and the databases. Users don't have to be human users. They can be other programs and applications as well. We will learn how Python program can interact as a user of an SQL database.

## 15.2 SQLite

SQLite is a simple relational database system, which saves its data in regular data files or even in the internal memory of the computer. It is designed to be embedded in applications, instead of using a separate database server program such as MySQLor Oracle. SQLite is fast, rigorously tested, and flexible, making it easier to work. Python has a native library for SQLite. To use SQLite,

Step 1 import sqlite3

Step 2 create a connection using connect () method and pass the name of the database File

Step 3 Set the cursor object cursor = connection. cursor ()

- Connecting to a database in step2 means passing the name of the database to be accessed. If the database already exists the connection will open the same. Otherwise, Python will open a new database file with the specified name.

- Cursor in step 3: is a control structure used to traverse and fetch the records of the database.
- Cursor has a major role in working with Python. All the commands will be executed using cursor object only.

To create a table in the database, create an object and write the SQL command in it.

**Example:-** sql_comm = "SQL statement"

For executing the command use the cursor method and pass the required sql command as a parameter. Many number of commands can be stored in the sql_comm and can be executed one after other. Any changes made in the values of the record should be saved by the commend "Commit" before closing the "Table connection".

## 15.3 Creating a Database using SQLite

The following example 15.3 explains how a connection to be made to a database through Python sqlite3

```
# Python code to demonstrate table creation and  insertions with SQL
# importing module
import sqlite3
# connecting to the database
connection = sqlite3.connect ("Academy.db")
# cursor
cursor = connection.cursor()
```

In the above example a database with the name "Academy" would be created. It's similar to the sql command "CREATE DATABASE Academy;" to SQL server."sqlite3.connect ('Academy.db')" is again used in some program, "connect" command just opens the already created database.

### 15.3.1 Creating a Table

After having created an empty database, you will most probably add one or more tables to this database. The SQL syntax for creating a table "Student" in the database "Academy" looks like as follows :

```
CREATE TABLE Student (

RollnoINTEGER, SnameVARCHAR(20), GradeCHAR(1), gender CHAR(1),

Average float(5.2), birth_date DATE, PRIMARY KEY (Rollno) );
```

This is the way, somebody might do it on a SQL command shell. Of course, we want to do this directly from Python. To be capable to send a command to "SQL", or SQLite, we need a cursor object. Usually, **a cursor in SQL and databases is a control structure to traverse over the records in a database.** So it's used for the fetching of the results.

Note

Cursor is used for performing all SQL commands.

The cursor object is created by calling the cursor() method of connection. The cursor is used to traverse the records from the result set. You **can define a SQL command with a triple quoted string in Python.** The reason behind the triple quotes is sometime the values in the table might contain single or double quotes.

### Example 15.3.1

```
sql_command = """
CREATE TABLE Student (
Rollno INTEGER PRIMARY KEY ,
Sname VARCHAR(20),
Grade CHAR(1),
gender CHAR(1),
Average DECIMAL(5,2),
birth_date DATE);"""
```

In the above example the Emp_no field as "INTEGER PRIMARY KEY" A column which is labeled like this will be automatically auto-incremented in SQLite3. To put it in other words: **If a column of a table is declared to be an INTEGER PRIMARY KEY, then whenever a NULL will be used as an input for this column, the NULL will be automatically converted into an integer which will one larger than the highest value so far used in that column.** If the table is empty, the value **1** will be used.

## 15.3.2 Adding Records

To populate (add record) the table "INSERT" command is passed to SQLite. “execute” method executes the SQL command to perform some action. The following example 15.3.2 is a complete working example. To run the program you will either have to remove the file Academy. db or uncomment the "DROP TABLE" line in the SQL command:

**Example 15.3.2 -1**

```
import sqlite3
connection = sqlite3.connect ("Academy.db")
cursor = connection.cursor()
# if the table already exits then delete it using cursor.execute ("""DROP TABLE Student;""")
# where  Student is name of the table.
sql_command = """
CREATE TABLE Student (
Rollno INTEGER PRIMARY KEY , Sname VARCHAR(20), Grade CHAR(1),
gender CHAR(1), Average DECIMAL (5, 2), birth_date DATE);"""
cursor.execute(sql_command)
sql_command = """INSERT INTO Student (Rollno, Sname, Grade, gender, Average,
    birth_date) VALUES (NULL, "Akshay", "B", "M","87.8", "2001-12-12");"""
cursor.execute(sql_command)
sql_command = """INSERT INTO Student (Rollno, Sname, Grade, gender, Average,
    birth_date) VALUES (NULL, "Aravind", "A", "M","92.50","2000-08-17");"""
cursor.execute(sql_command)
# never forget this, if you want the changes to be saved:
connection.commit()
connection.close()
print("STUDENT TABLE CREATED")
```

**OUTPUT**

```
    STUDENT TABLE CREATED
```

Of course, in most cases, you will not literally insert data into a SQL table. You will rather have a lot of data inside of some Python data type e.g. a dictionary or a list, which has to be used as the input of the insert statement.

The following working example, assumes that you have an already existing database Academy.db and a table Student. We have a list with data of persons which will be used in the INSERT statement:

**Example 15.3.2-2**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
student_data = [("BASKAR", "C", "M","75.2","1998-05-17"),
    ("SAJINI", "A", "F","95.6","2002-11-01"),
    ("VARUN", "B", "M","80.6","2001-03-14"),
    ("PRIYA", "A", "F","98.6","2002-01-01"),
    ("TARUN", "D", "M","62.3","1999-02-01") ]
for p in student_data:
    format_str = """INSERT INTO Student (Rollno, Sname, Grade, gender,Average,
    birth_date) VALUES (NULL,"{name}", "{gr}", "{gender}","{avg}","{birthdate}");"""
    sql_command = format_str.format(name=p[0], gr=p[1], gender=p[2],avg=p[3],
    birthdate = p[4])
cursor.execute(sql_command)
connection.commit()
connection.close()
print("RECORDS ADDED TO STUDENT TABLE ")
```

**OUTPUT**

```
RECORDS ADDED TO STUDENT TABLE
```

In the above program {gr} is a place holder (variable) to get the value. format_str.format() is a function used to format the value to the required datatype.

## 15.4 SQL Query Using Python

The time has come now to finally query our "Student" table. Fetching the data from record is as simple as inserting them. The execute method uses the SQL command to get all the data from the table.

### 15.4.1 SELECT Query

**"Select"** is the most commonly used statement in SQL. The SELECT Statement in SQL is used to retrieve or fetch data from a table in a database. The syntax for using this statement is **"Select * from table_name"** and all the table data can be fetched in an object in the form of list of lists.

If you run this program, saved as "sql_Academy_query.py", you would get the following result, depending on the actual data:

It should be noted that the database file that will be created will be in the same folder as that of the python file. If we wish to change the path of the file, change the path while opening the file.

**Example 15.4.1-1**

```
import sqlite3
#save the file as “sql_Academy_query.py”
# cursor object
crsr = connection.cursor()
# execute the command to fetch all the data from the table Student
crsr.execute("SELECT * FROM Student")
# store all the fetched data in the ans variable
ans= crsr.fetchall()
# loop to print all the data
for i in ans:
        print(i)
```

### 15.4.1.1 Displaying all records using fetchall()

The fetchall() method is used to fetch all rows from the database table

**Example 15.4.1.1-1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM student")
print("fetchall:")
result = cursor.fetchall()
for r in result:
     print(r)
```

**OUTPUT**

```
     fetchall:
     (1, 'Akshay', 'B', 'M', 87.8, '2001-12-12')
     (2, 'Aravind', 'A', 'M', 92.5, '2000-08-17')
     (3, 'BASKAR', 'C', 'M', 75.2, '1998-05-17')
     (4, 'SAJINI', 'A', 'F', 95.6, '2002-11-01')
     (5, 'VARUN', 'B', 'M', 80.6, '2001-03-14')
     (6, 'PRIYA', 'A', 'F', 98.6, '2002-01-01')
     (7, 'TARUN', 'D', 'M', 62.3, '1999-02-01')
```

Note

cursor.fetchall() -fetchall () method is to fetch all rows from the database table

cursor.fetchone() - The fetchone () method returns the next row of a query result set or None in case there is no row left.

cursor.fetchmany() method that returns the next number of rows (n) of the result set

### 15.4.1.2 Displaying A record using fetchone()

The fetchone() method returns the next row of a query result set or None in case there is no row left.

**Example 15.4.1.2-1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM student")
print("\nfetch one:")
res = cursor.fetchone()
print(res)
```

**OUTPUT**

```
    fetch one:
    (1, 'Akshay', 'B', 'M', 87.8, '2001-12-12')
```

### 15.4.1.3 Displaying all records using fetchone()

Using while loop and fetchone() method we can display all the records from a table.

**Example 15.4.1.3 -1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM student")
print("fetching all records one by one:")
result = cursor.fetchone()
while result is not None:
    print(result)
    result = cursor.fetchone()
```

**OUTPUT**

```
fetching all records one by one:
(1, 'Akshay', 'B', 'M', 87.8, '2001-12-12')
(2, 'Aravind', 'A', 'M', 92.5, '2000-08-17')
(3, 'BASKAR', 'C', 'M', 75.2, '1998-05-17')
(4, 'SAJINI', 'A', 'F', 95.6, '2002-11-01')
(5, 'VARUN', 'B', 'M', 80.6, '2001-03-14')
(6, 'PRIYA', 'A', 'F', 98.6, '2002-01-01')
(7, 'TARUN', 'D', 'M', 62.3, '1999-02-01')
```

### 15.4.1.4 Displayingusing fetchmany()

Displaying specified number of records is done by using fetchmany(). This method returns the next number of rows (n) of the result set.

**Example 15.4.1.4-1: Program to display the content of tuples using fetchmany()**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM student")
print("fetching first 3 records:")
result = cursor.fetchmany(3)
print(result)
```

**OUTPUT**

```
fetching first 3 records:
[(1, 'Akshay', 'B', 'M', 87.8, '2001-12-12'), (2, 'Aravind', 'A', 'M', 92.5, '2000-08-17'), (3,
'BASKAR', 'C', 'M', 75.2, '1998-05-17')]
```

**Example 15.4.1.4-2: Program to display the content of tuples in newline without using loops**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM student")
print("fetching first 3 records:")
result = cursor.fetchmany(3)
print(*result,sep="\n") # * is used for unpacking a tuple.
```

**OUTPUT**

```
fetching first 3 records:
(1, 'Akshay', 'B', 'M', 87.8, '2001-12-12')
(2, 'Aravind', 'A', 'M', 92.5, '2000-08-17')
(3, 'BASKAR', 'C', 'M', 75.2, '1998-05-17')
```

Note

symbol is used to print the list of all elements in a single line with space. To print all elements in new lines or separated by space use sep= "\n" or sep= "," respectively.

### 15.4.2 CLAUSES IN SQL

SQL provides various clauses that can be used in the SELECT statements. This clauses can be called through python script. Almost all clauses will work with SQLite. The following frequently used clauses are discussed here

- DISTINCT
- WHERE
- GROUP BY
- ORDER BY.
- HAVING

#### 15.4.2.1 SQL DISTINCT CLAUSE

The distinct clause is helpful when there is need of avoiding the duplicate values present in any specific columns/table. When we use distinct keyword only the unique values are fetched. In this example we are going to display the different grades scored by students from "student table".

**Example 15.4.2.1-1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT DISTINCT (Grade) FROM student")
result = cursor.fetchall()
print(result)
```

**OUTPUT**

```
        [('B',), ('A',), ('C',), ('D',)]
```

Without the keyword "distinct" in the above example displays 7 records instead of 4, since in the original table there are actually 7 records and some are with the duplicate values.

#### 15.4.2.2 SQL WHERE CLAUSE

The WHERE clause is used to extract only those records that fulfill a specified condition. In this example we are going to display the different grades scored by male students from "student table"

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT DISTINCT (Grade) FROM student where gender='M'")
result = cursor.fetchall()
print(*result,sep="\n")
```

**OUTPUT**

```
('B',)
('A',)
('C',)
('D',)
```

### 15.4.2.3 SQL Group By Clause

The SELECT statement can be used along with GROUP BY clause. The GROUP BY clause groups records into summary rows. It returns one records for each group. It is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns. The following example count the number of male and female from the student table and display the result.

**Example 15.4.2.3 -1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT gender,count(gender) FROM student Group BY gender")
result = cursor.fetchall()
print(*result,sep="\n")
```

**OUTPUT**

```
('F', 2)
('M', 5)
```

### 15.4.2.4 SQL ORDER BY Clause

The ORDER BY Clause can be used along with the SELECT statement to sort the data of specific fields in an ordered way. It is used to sort the result-set in ascending or descending order. In this example name and Rollno of the students are displayed in alphabetical order of names

### Example 15.4.2.4 -1

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT Rollno,sname  FROM student Order BY sname")
result = cursor.fetchall()
print(*result,sep="\n")
```

**OUTPUT**

```
        (1, 'Akshay')
        (2, 'Aravind')
        (3, 'BASKAR')
        (6, 'PRIYA')
        (4, 'SAJINI')
        (7, 'TARUN')
        (5, 'VARUN')
```

### 15.4.2.5  SQL HAVING Clause

Having clause is used to filter data based on the group functions. This is similar to WHERE condition but can be used only with group functions. Group functions cannot be used in WHERE Clause but can be used in HAVING clause.

### Example 15.4.2.5 -1

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT GENDER,COUNT(GENDER) FROM Student GROUP BY
GENDER HAVING COUNT(GENDER)>3")
result = cursor.fetchall()
co = [i[0] for i in cursor.description]
print(co)
print(result)
```

**OUTPUT**

```
        ['gender', 'COUNT(GENDER)']
        [('M', 5)]
```

## 15.5  The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators. The AND and OR operators are used to filter records based on more than one condition. In this example you are going to display the details of students who have scored other than ‘A’ or ‘B’ from the “student table”

**Example for WHERE WITH NOT Operator**

**Example 15.5 -1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT * FROM student where grade<>'A' and Grade<>'B'")
result = cursor.fetchall()
print(*result,sep="\n")
```

**OUTPUT**

```
(3, 'BASKAR', 'C', 'M', 75.2, '1998-05-17')
(7, 'TARUN', 'D', 'M', 62.3, '1999-02-01')
```

**Example for WHERE WITH AND Operator**

In this example we are going to display the name, Rollno and Average of students who have scored an average between 80 to 90% (both limits are inclusive)

**Example 15.5 -2**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT   Rollno,Same,Average   FROM   student   WHERE
(Average>=80 AND Average<=90)")
result = cursor.fetchall()
print(*result,sep="\n")
```

**OUTPUT**

```
(1, 'Akshay', 87.8)
(5, 'VARUN', 80.6)
```

**Example for WHERE WITH OR Operator**

In this example we are going to display the name and Rollno of students who have not scored an average between 60 to 70%

**Example 15.5 -3**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT Rollno,sname  FROM student WHERE (Average<60
OR Average>70)")
result = cursor.fetchall()
print(*result,sep="\n")
```

**OUTPUT**

```
(1, 'Akshay')
(2, 'Aravind')
(3, 'BASKAR')
(4, 'SAJINI')
(5, 'VARUN')
(6, 'PRIYA')
```

## 15.6 Querying A Date Column

In this example we are going to display the name and grade of students who have born in the year 2001

**Example 15.6 -1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT Rollno,sname FROM student
WHERE(Birth_date>='2001-01-01' AND Birth_date<='2001-12-01')")
result = cursor.fetchall()
print(*result,sep="\n")
```

**OUTPUT**

```
(5, 'VARUN')
```

## 15.7 Aggregate Functions

These functions are used to do operations from the values of the column and a single value is returned.

- COUNT()
- AVG()
- SUM()
- MAX()
- MIN()

### 15.7.1 COUNT() function

The SQL COUNT() function returns the number of rows in a table satisfying the criteria specified in the WHERE clause. COUNT() returns 0 if there were no matching rows.

#### Example 15.7.1-1

**Example 1 : In this example we are going to count the number of records(rows)**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT COUNT(*)  FROM student ")
result = cursor.fetchall()
print(result)
```

**Output:**

```
[(7,)]
```

#### EXAMPLE 15.7.1-2

**Example 2 : In this example we are going to count the number of records by specifying a column**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT COUNT(AVERAGE)  FROM student ")
result = cursor.fetchall()
print(result
```

**Output:**

```
[(7,)]
```

Note

NULL values are not counted. In case if we had null in one of the records in student table for example in Average field then the output would be 6

### 15.7.2AVG():

The following SQL statement in the python program finds the average mark of all students.

**Example 15.7.2-1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT AVG(AVERAGE)  FROM student ")
result = cursor.fetchall()
print(result)
```

**OUTPUT**

```
        [(84.65714285714286,)]
```

Note

NULL values are ignored.

### 15.7.3 SUM():

The following SQL statement in the python program finds the sum of all average in the Average field of "Student table".

**Example 15.7.1-3**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("SELECT SUM(AVERAGE)  FROM student ")
result = cursor.fetchall()
print(result)
```

**OUTPUT**

```
[(592.6,)]
```

Note

NULL values are ignored.

### 15.7.4 MAX() AND MIN() FUNCTIONS

The MAX() function returns the largest value of the selected column.

The MIN() function returns the smallest value of the selected column.

The following example show the highest and least average student's name.

**Example 15.7.4-1**

```
import sqlite3
connection = sqlite3.connect("Organization.db")
cursor = connection.cursor()
print("Displaying the name of the Highest Average")
cursor.execute("SELECT sname,max(AVERAGE)  FROM student ")
result = cursor.fetchall()
print(result)
print("Displaying the name of the Least Average")
cursor.execute("SELECT sname,min(AVERAGE)  FROM student ")
result = cursor.fetchall()
print(result)
```

**OUTPUT**

```
Displaying the name of the Highest Average
[('PRIYA', 98.6)]
Displaying the name of the Least Average
[('TARUN', 62.3)]
```

## 15.8 Updating A Record

You can even update a record (tuple) in a table through python script. The following example change the name "Priya" to "Priyanka" in a record in "student table"

**Example 15.8 -1**

```
# code for update operation
import sqlite3
# database name to be passed as parameter
conn = sqlite3.connect("Academy.db")
# update the student record
conn.execute("UPDATE Student SET sname ='Priyanka' where Rollno='6'")
conn.commit()
print ("Total number of rows updated :", conn.total_changes)
cursor = conn.execute("SELECT * FROM Student")
for row in cursor:
        print (row)
conn.close()
```

**OUTPUT**

```
Total number of rows updated : 1
(1, 'Akshay', 'B', 'M', 87.8, '2001-12-12')
(2, 'Aravind', 'A', 'M', 92.5, '2000-08-17')
(3, 'BASKAR', 'C', 'M', 75.2, '1998-05-17')
(4, 'SAJINI', 'A', 'F', 95.6, '2002-11-01')
(5, 'VARUN', 'B', 'M', 80.6, '2001-03-14')
(6, 'Priyanka', 'A', 'F', 98.6, '2002-01-01')
(7, 'TARUN', 'D', 'M', 62.3, '1999-02-01')
```

Note

Remember throughout this chapter student table what we have created is taken as example to explain the SQL queries .Example 15.3.2 -2 contain the student table with records

## 15.9 Deletion Operation

Similar to Sql command to delete a record, Python also allows to delete a record. The following example delete the content of Rollno 2 from "student table"

**Example 15.9-1**

```
# code for delete operation
import sqlite3
  # database name to be passed as parameter
conn = sqlite3.connect("Academy.db")
# delete student record from database
conn.execute("DELETE from Student where Rollno='2'")
conn.commit()
print("Total number of rows deleted :", conn.total_changes)
cursor =conn.execute("SELECT * FROM Student")
for row in cursor:
        print(row)
conn.close()
```

**OUTPUT**

```
Total number of rows deleted : 1
(1, 'Akshay', 'B', 'M', 87.8, '2001-12-12')
(3, 'BASKAR', 'C', 'M', 75.2, '1998-05-17')
(4, 'SAJINI', 'A', 'F', 95.6, '2002-11-01')
(5, 'VARUN', 'B', 'M', 80.6, '2001-03-14')
(6, 'Priyanka', 'A', 'F', 98.6, '2002-01-01')
(7, 'TARUN', 'D', 'M', 62.3, '1999-02-01')
```

## 15.10 Data input by User

In this example we are going to accept data using Python input() command during runtime and then going to write in the Table called "Person"

**Example 15.10 -1**

```
# code for executing query using input data
import sqlite3
# creates a database in RAM
con =sqlite3.connect("Academy.db")
cur =con.cursor()
cur.execute("DROP Table person")
cur.execute("create table person (name, age, id)")
print("Enter 5 students names:")
who =[input() for i in range(5)]
print("Enter their ages respectively:")
age =[int(input()) for i in range(5)]
print("Enter their ids respectively:")
p_id =[int(input())for i in range(5)]
n =len(who)
for i in range(n):
# This is the q-mark style:
        cur.execute("insert into person values (?, ?, ?)", (who[i], age[i], p_id[i]))
cur.execute("select * from person")
# Fetches all entries from table
print("Displaying All the Records From Person Table")
print (*cur.fetchall(), sep='\n' )
```

**OUTPUT**

```
Enter 5 students names:
        RAM
        KEERTHANA
        KRISHNA
        HARISH
        GIRISH
Enter their ages respectively:
        28
        12
        21
        18
        16
Enter their ids respectively:
        1
        2
        3
        4
        5
Displaying All the Records From Person Table
        ('RAM', 28, 1)
        ('KEERTHANA', 12, 2)
        ('KRISHNA', 21, 3)
        ('HARISH', 18, 4)
        ('GIRISH', 16, 5)
```

You can even add records to the already existing table like "Student" Using the above coding with appropriate modification in the Field Name. To do so you should comment the create table statement

Note

Execute (sql[, parameters]) :- Executes a single SQL statement. The SQL statement may be parametrized (i. e. Use placeholders instead of SQL literals). The sqlite3 module supports two kinds of placeholders: question marks? ("qmark style") and named placeholders :name ("named style").

## 15.11 Using Multiple Table for Querying

Python allows to query more than one table by joining them. In the following example a new table called "Appointments" which contain the details of students Rollno, Duty, Age is created. The tables "student" and "Appointments" are joined and displayed the result with the column headings.

**Example 15.11-1**

```
import sqlite3
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
cursor.execute("""DROP TABLE Appointment;""")
sql_command = """
CREATE TABLE Appointment(rollnointprimarkey,Dutyvarchar(10),age int)"""
cursor.execute(sql_command)
sql_command = """INSERT INTO Appointment (Rollno,Duty ,age )
        VALUES ("1", "Prefect", "17");"""
cursor.execute(sql_command)
sql_command = """INSERT INTO Appointment (Rollno, Duty, age)
        VALUES ("2", "Secretary", "16");"""
cursor.execute(sql_command)
#  never forget this, if you want the changes to be saved:
connection.commit()
cursor.execute ("SELECT student.rollno,student.sname,
        Appointment.Duty, Appointment.Age FROM student,Appointment
        where student.rollno=Appointment.rollno")
#print (cursor.description) to display the field names of the table
co = [i[0] for i in cursor.description]
print(co)
#  Field informations can be read from cursor.description.
result = cursor.fetchall()
for r in result:
        print(r)
```

**OUTPUT**

```
        ['Rollno', 'Sname', 'Duty', 'age']
        (1, 'Akshay', 'Prefect', 17)
        (2, 'Aravind', 'Secretary', 16)
```

Note

cursor. description contain the details of each column headings .It will be stored as a tuple and the first one that is 0(zero) index refers to the column name. From index 1 onwards the values of the column(Records) are refered. Using this command you can display the table's Field names.

## 15.12 Integrating Query With Csv File

You can even store the query result in a CSV file. This will be useful to display the query output in a tabular format. In the following example **(EXAMPLE 15.12 -1)** Using Python script the student table is sorted "gender" wise in descending order and then arranged the records alphabetically. The output of this Query will be written in a CSV file called "SQL.CSV", again the content is read from the CSV file and displayed the result.

### Example 15.12 -1

```
import sqlite3
import io                                  #  to access replace()
import csv
                                        #  CREATING CSV FILE
d=open('c:/pyprg/sql.csv','w')
c=csv.writer(d)
connection = sqlite3.connect("Academy.db")
cursor = connection.cursor()
                                        #  a=Connection.cursor()
cursor.execute("SELECT * FROM student ORDER BY GENDER DESC,SNAME")
                                  #  WRITING THE COLUMN HEADING
co = [i[0] for i in cursor.description]
c.writerow(co)
data=cursor.fetchall()
for item in data:
        c.writerow(item)
d.close()
#  Reading the CSV File
#  replace() is used to eliminate the newline character at the end of each row
with open('c:/pyprg/sql.csv', "r", newline=None) as fd:
        #  r = csv.reader(fd)
        for line in fd:
                line = line.replace("\n", "")
                print(line)
cursor.close()
connection.close()
```

**OUTPUT**

```
        Rollno,Sname,Grade,gender,Average,birth_date
        1, Akshay, B, M, 87.8, 2001-12-12
        2, Aravind, A, M, 92.5, 2000-08-17
        3, BASKAR, C, M, 75.2, 1998-05-17
        7, TARUN, D, M, 62.3, 1999-02-01
        5, VARUN, B, M, 80.6, 2001-03-14
        6, PRIYA, A, F, 98.6, 2002-01-01
        4, SAJINI, A, F, 95.6, 2002-11-01
```

**Example 15.12 -2 Opening the file ("sqlexcel.csv") through MS-Excel and view the result (Program is same similar to EXAMPLE 15.12 -1 script)**

```
import sqlite3
import io                                    #to access replace()
import csv
# database name to be passed as parameter
conn = sqlite3.connect("Academy.db")
print("Content of the table before sorting and writing in CSV file")
cursor = conn.execute("SELECT * FROM Student")
for row in cursor:
        print (row)
#  CREATING CSV FILE
d=open('c:\pyprg\sqlexcel.csv','w')
c=csv.writer(d)
cursor = conn.cursor()
cursor.execute("SELECT * FROM student ORDER BY GENDER DESC,SNAME")
#WRITING THE COLUMN HEADING
co = [i[0] for i in cursor.description]
c.writerow(co)
data=cursor.fetchall()
for item in data:
        c.writerow(item)
d.close()
print("sqlexcel.csv File is created open by visiting c:\pyprg\sqlexcel.csv")
conn.close()
```

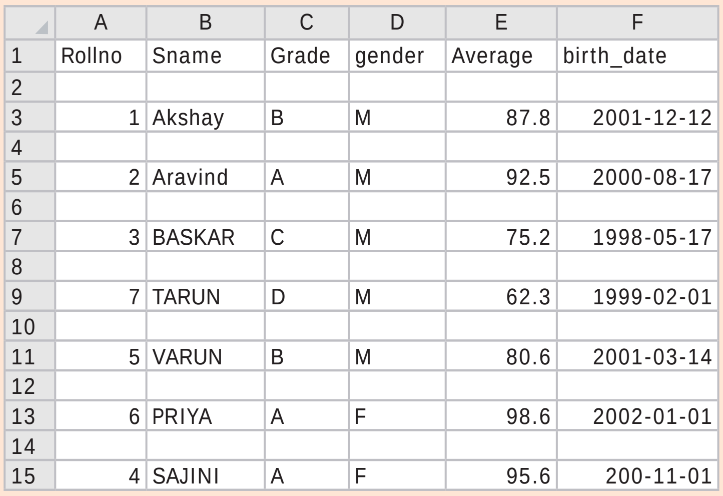
Note

By default while writing in a csv file each record ends with \n (newline) to eliminate this newline replace () is used during the reading of csv file.

**OUTPUT**

```
Content of the table before sorting and writing in CSV file
(1, 'Akshay', 'B', 'M', 87.8, '2001-12-12')
(2, 'Aravind', 'A', 'M', 92.5, '2000-08-17')
(3, 'BASKAR', 'C', 'M', 75.2, '1998-05-17')
(4, 'SAJINI', 'A', 'F', 95.6, '2002-11-01')
(5, 'VARUN', 'B', 'M', 80.6, '2001-03-14')
(6, 'Priyanka', 'A', 'F', 98.6, '2002-01-01')
(7, 'TARUN', 'D', 'M', 62.3, '1999-02-01')
sqlexcel.csv File is created open by visiting c:\pyprg\sqlexcel.csv
```

**OUTPUT THROUGH EXCEL**



|  | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Rollno | Sname | Grade | gender | Average | birth_date |
| 2 |  |  |  |  |  |  |
| 3 | 1 | Akshay | B | M | 87.8 | 2001-12-12 |
| 4 |  |  |  |  |  |  |
| 5 | 2 | Aravind | A | M | 92.5 | 2000-08-17 |
| 6 |  |  |  |  |  |  |
| 7 | 3 | BASKAR | C | M | 75.2 | 1998-05-17 |
| 8 |  |  |  |  |  |  |
| 9 | 7 | TARUN | D | M | 62.3 | 1999-02-01 |
| 10 |  |  |  |  |  |  |
| 11 | 5 | VARUN | B | M | 80.6 | 2001-03-14 |
| 12 |  |  |  |  |  |  |
| 13 | 6 | PRIYA | A | F | 98.6 | 2002-01-01 |
| 14 |  |  |  |  |  |  |
| 15 | 4 | SAJINI | A | F | 95.6 | 200-11-01 |

## 15.3 Table List

To show (display) the list of tables created in a database the following program **(Example 15.3 – 1)** can be used.

**Example 15.3 – 1**

```
import sqlite3
con = sqlite3.connect('Academy.db')
cursor = con.cursor()
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
print(cursor.fetchall())
```

**OUTPUT**

```
[('Student',), ('Appointment',), ('Person',)]
```

The above program (Example 15.3-1) display the names of all tables created in 'Academy.db' database. **The master table holds the key information about your database tables and it is called sqlite_master**

So far, you have been using the Structured Query Language in Python scripts. This chapter has covered many of the basic SQL commands. Almost all sql commands can be executed by Python SQLite module. You can even try the other commands discussed in the SQL chapter.

**Points to remember:**

- A database is an organized collection of data.
- Users of database can be human users, other programs or applications
- SQLite is a simple relational database system, which saves its data in regular data files.
- Cursor is a control structure used to traverse and fetch the records of the database. All the SQL commands will be executed using cursor object only.
- As data in a table might contain single or double quotes, SQL commands in Python are denoted as triple quoted string.
- "Select" is the most commonly used statement in SQL
- The SELECT Statement in SQL is used to retrieve or fetch data from a table in a database
- The GROUP BY clause groups records into summary rows
- The ORDER BY Clause can be used along with the SELECT statement to sort the data of specific fields in an ordered way
- Having clause is used to filter data based on the group functions.
- Where clause cannot be used along with 'Group by'
- The WHERE clause can be combined with AND, OR, and NOT operators
- The 'AND' and 'OR' operators are used to filter records based on more than one condition
- Aggregate functions are used to do operations from the values of the column and a single value is returned.
- COUNT() function returns the number of rows in a table.
- AVG() function retrieves the average of a selected column of rows in a table.
- SUM() function retrieves the sum of a selected column of rows in a table.
- MAX() function returns the largest value of the selected column.
- MIN() function returns the smallest value of the selected column
- sqlite_master is the master table which holds the key information about your database tables.
- The path of a file can be either represented as '/' or using '\' in Python. For example the path can be specified either as 'c:/pyprg/sql.csv', or c:\pyprg\sql.csv'.

## Hands on Experience

1. Create an interactive program to accept the details from user and store it in a csv file using Python for the following table.

   Database name;- DB1

   Table name : Customer

| Cust_Id | Cust_Name | Address | Phone_no | City |
|---|---|---|---|---|
| C008 | Sandeep | 14/1 Pritam Pura | 41206819 | Delhi |
| C010 | Anurag Basu | 15A, Park Road | 61281921 | Kolkata |
| C012 | Hrithik | 7/2 Vasant Nagar | 26121949 | Delhi |

2. Consider the following table GAMES. Write a python program to display the records for question (i) to (iv) and give outputs for SQL queries (v) to (viii)

   Table: GAMES

| Gcode | Name | GameName | Number | PrizeMoney | ScheduleDate |
|---|---|---|---|---|---|
| 101 | Padmaja | Carom Board | 2 | 5000 | 01-23-2014 |
| 102 | Vidhya | Badminton | 2 | 12000 | 12-12-2013 |
| 103 | Guru | Table Tennis | 4 | 8000 | 02-14-2014 |
| 105 | Keerthana | Carom Board | 2 | 9000 | 01-01-2014 |
| 108 | Krishna | Table Tennis | 4 | 25000 | 03-19-2014 |

(i) To display the name of all Games with their Gcodes in descending order of their schedule date.

(ii) To display details of those games which are having Prize Money more than 7000.

(iii) To display the name and gamename of the Players in the ascending order of Gamename.

(iv) To display sum of PrizeMoney for each of the Numberof participation groupings (as shown in column Number 4)

(v) Display all the records based on GameName

## Evaluation

### Part - I

RDEZQU

1. Which of the following is an organized collection of data?

   (A) Database (B) DBMS (C) Information (D) Records

2. SQLite falls under which database system?

   (A) Flat file database system (B) Relational Database system

   (C) Hierarchical database system (D) Object oriented Database system

3. Which of the following is a control structure used to traverse and fetch the records of the database?

   (A) Pointer (B) Key

   (C) Cursor (D) Insertion point

4. Any changes made in the values of the record should be saved by the command

   (A) Save (B) Save As (C) Commit (D) Oblige

5. Which of the following executes the SQL command to perform some action?

   (A) execute() (B) key() (C) cursor() (D) run()

6. Which of the following function retrieves the average of a selected column of rows in a table?

   (A) Add() (B) SUM() (C) AVG() (D) AVERAGE()

7. The function that returns the largest value of the selected column is

   (A) MAX() (B) LARGE()

   (C) HIGH() (D) MAXIMUM()

8. Which of the following is called the master table?

   (A) sqlite_master (B) sql_master

   (C) main_master (D) master_main

9. The most commonly used statement in SQL is

   (A) cursor (B) select (C) execute (D) commit

10. Which of the following clause avoid the duplicate?

    (A) Distinct (B) Remove (C) Where (D) GroupBy

## Part - II

1. Mention the users who uses the Database.
2. Which method is used to connect a database? Give an example.
3. What is the advantage of declaring a column as "INTEGER PRIMARY KEY"
4. Write the command to populate record in a table. Give an example.
5. Which method is used to fetch all rows from the database table?

## Part - III

1. What is SQLite?What is it advantage?
2. Mention the difference between fetchone() and fetchmany()
3. What is the use of Where Clause.Give a python statement Using the where clause.
4. Read the following details.Based on that write a python script to display department wise records

   database name :- organization.db

   Table name :- Employee

   Columns in the table :- Eno, EmpName, Esal, Dept
5. Read the following details.Based on that write a python script to display records in desending order of

   Eno

   database name :- organization.db

   Table name :- Employee

   Columns in the table :- Eno, EmpName, Esal, Dept

## Part - IV

1. Write in brief about SQLite and the steps used to use it.
2. Write the Python script to display all the records of the following table using fetchmany()

| Icode | ItemName | Rate |
|---|---|---|
| 1003 | Scanner | 10500 |
| 1004 | Speaker | 3000 |
| 1005 | Printer | 8000 |
| 1008 | Monitor | 15000 |

|  |  |  |
|---|---|---|
| 1010 | Mouse | 700 |

3. hat is the use of HAVING clause. Give an example python script
4. Write a Python script to create a table called ITEM with following specification.

   Add one record to the table.

   Name of the database :- ABC

   Name of the table :- Item

   Column name and specification :-

|  |  |  |
|---|---|---|
| Icode | :- | integer and act as primary key |
| Item Name | :- | Character with length 25 |
| Rate | :- | Integer |
| Record to be added | :- | 1008, Monitor,15000 |

5. Consider the following table Supplier and item .Write a python script for (i) to (ii)

| SUPPLIER |  |  |  |  |
|---|---|---|---|---|
| **Suppno** | **Name** | **City** | **Icode** | **SuppQty** |
| S001 | Prasad | Delhi | 1008 | 100 |
| S002 | Anu | Bangalore | 1010 | 200 |
| S003 | Shahid | Bangalore | 1008 | 175 |
| S004 | Akila | Hydrabad | 1005 | 195 |
| S005 | Girish | Hydrabad | 1003 | 25 |
| S006 | Shylaja | Chennai | 1008 | 180 |
| S007 | Lavanya | Mumbai | 1005 | 325 |

   i) Display Name, City and Itemname of suppliers who do not reside in Delhi.

   ii) Increment the SuppQty of Akila by 40

## References

1. *The Definitive Guide to SQLite by Michael Owens*
2. *Programming for Beginners: 2 Manuscripts: SQL & Python by Byron Francis*
3. *Tutorialspoint.com*