Database Connection

In this section of the tutorial, we will discuss the steps to connect the python application to the database.

There are the following steps to connect a python application to our database.

1. Import mysql.connector module
2. Create the connection object.
3. Create the cursor object
4. Execute the query

Creating the connection

To create a connection between the MySQL database and the python application, the connect() method of mysql.connector module is used.

Pass the database details like HostName, username, and the database password in the method call. The method returns the connection object.

The syntax to use the connect() is given below.

1. Connection-Object= mysql.connector.connect(host = <host-name> , user = <username> , passwd = <password> )

Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google")
6. #printing the connection object
7. **print**(myconn)

**Output:**

<mysql.connector.connection.MySQLConnection object at 0x7fb142edd780>

Here, we must notice that we can specify the database name in the connect() method if we want to connect to a specific database.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google", database = "mydb")
6. #printing the connection object
7. **print**(myconn)

**Output:**

<mysql.connector.connection.MySQLConnection object at 0x7ff64aa3d7b8>

Creating a cursor object

The cursor object can be defined as an abstraction specified in the Python DB-API 2.0. It facilitates us to have multiple separate working environments through the same connection to the database. We can create the cursor object by calling the 'cursor' function of the connection object. The cursor object is an important aspect of executing queries to the databases.

The syntax to create the cursor object is given below.

1. <my\_cur>  = conn.cursor()

Example

1. **import** mysql.connector
2. #Create the connection object
3. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google", database = "mydb")
5. #printing the connection object
6. **print**(myconn)
8. #creating the cursor object
9. cur = myconn.cursor()
11. **print**(cur)

**Output:**

<mysql.connector.connection.MySQLConnection object at 0x7faa17a15748>

MySQLCursor: (Nothing executed yet)

Creating new databases

In this section of the tutorial, we will create the new database PythonDB.

Getting the list of existing databases

We can get the list of all the databases by using the following MySQL query.

1. >  show databases;

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. dbs = cur.execute("show databases")
11. **except**:
12. myconn.rollback()
13. **for** x **in** cur:
14. **print**(x)
15. myconn.close()

**Output:**

('EmployeeDB',)

('Test',)

('TestDB',)

('information\_schema',)

('javatpoint',)

('javatpoint1',)

('mydb',)

('mysql',)

('performance\_schema',)

('testDB',)

Creating the new database

The new database can be created by using the following SQL query.

1. >  create database <database-name>

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #creating a new database
11. cur.execute("create database PythonDB2")
13. #getting the list of all the databases which will now include the new database PythonDB
14. dbs = cur.execute("show databases")
16. **except**:
17. myconn.rollback()
19. **for** x **in** cur:
20. **print**(x)
22. myconn.close()

**Output:**

('EmployeeDB',)

('PythonDB',)

('Test',)

('TestDB',)

('anshika',)

('information\_schema',)

('javatpoint',)

('javatpoint1',)

('mydb',)

('mydb1',)

('mysql',)

('performance\_schema',)

('testDB',)

Creating the table

In this section of the tutorial, we will create the new table Employee. We have to mention the database name while establishing the connection object.

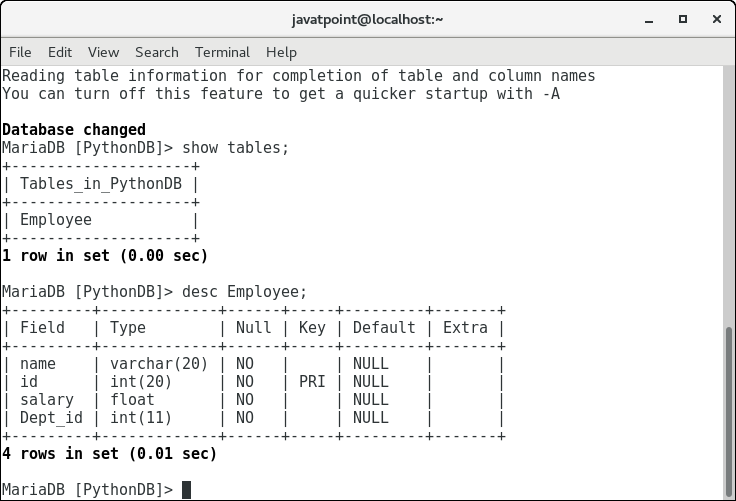
We can create the new table by using the CREATE TABLE statement of SQL. In our database PythonDB, the table Employee will have the four columns, i.e., name, id, salary, and department\_id initially.

The following query is used to create the new table Employee.

1. >  create table Employee (name varchar(20) **not** null, id int primary key, salary float **not** null, Dept\_Id int **not** null)

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Creating a table with name Employee having four columns i.e., name, id, salary, and department id
11. dbs = cur.execute("create table Employee(name varchar(20) not null, id int(20) not null primary key, salary float not null, Dept\_id int not null)")
12. **except**:
13. myconn.rollback()
15. myconn.close()



Now, we may check that the table Employee is present in the database.

Alter Table

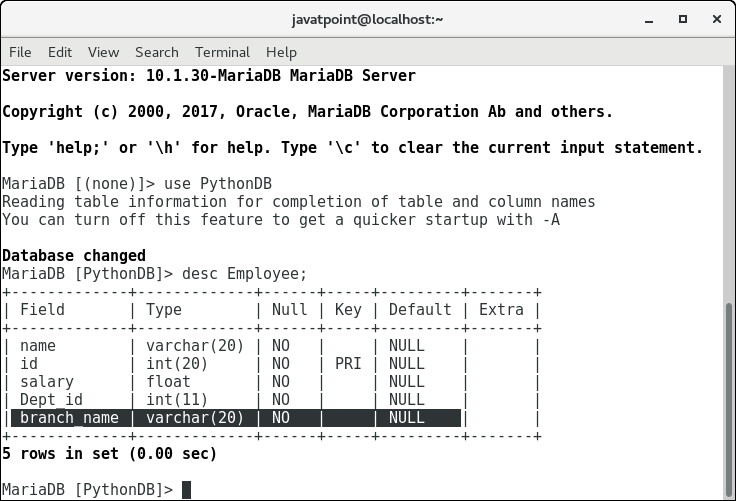
Sometimes, we may forget to create some columns, or we may need to update the table schema. The alter statement used to alter the table schema if required. Here, we will add the column branch\_name to the table Employee. The following SQL query is used for this purpose.

1. alter table Employee add branch\_name varchar(20) **not** null

Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #adding a column branch name to the table Employee
11. cur.execute("alter table Employee add branch\_name varchar(20) not null")
12. **except**:
13. myconn.rollback()
15. myconn.close()



Insert Operation

Adding a record to the table

The **INSERT INTO** statement is used to add a record to the table. In python, we can mention the format specifier (%s) in place of values.

We provide the actual values in the form of tuple in the execute() method of the cursor.

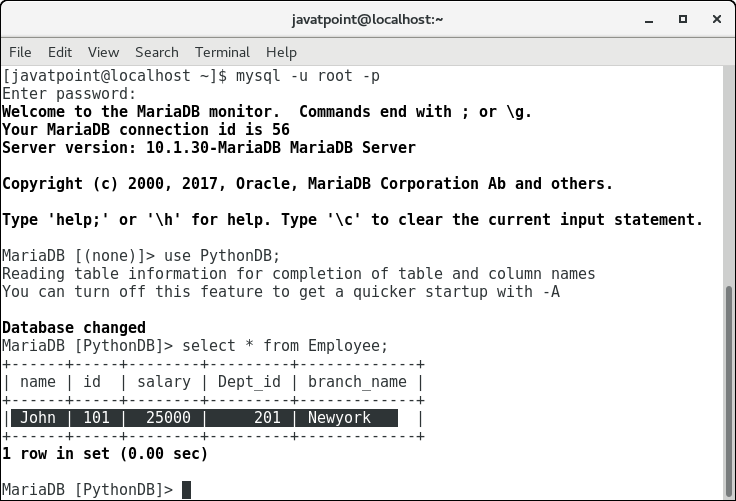
Consider the following example.

Example

1. **import** mysql.connector
2. #Create the connection object
3. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
4. #creating the cursor object
5. cur = myconn.cursor()
6. sql = "insert into Employee(name, id, salary, dept\_id, branch\_name) values (%s, %s, %s, %s, %s)"
8. #The row values are provided in the form of tuple
9. val = ("John", 110, 25000.00, 201, "Newyork")
11. **try**:
12. #inserting the values into the table
13. cur.execute(sql,val)
15. #commit the transaction
16. myconn.commit()
18. **except**:
19. myconn.rollback()
21. **print**(cur.rowcount,"record inserted!")
22. myconn.close()

**Output:**

1 record inserted!



Insert multiple rows

We can also insert multiple rows at once using the python script. The multiple rows are mentioned as the list of various tuples.

Each element of the list is treated as one particular row, whereas each element of the tuple is treated as one particular column value (attribute).

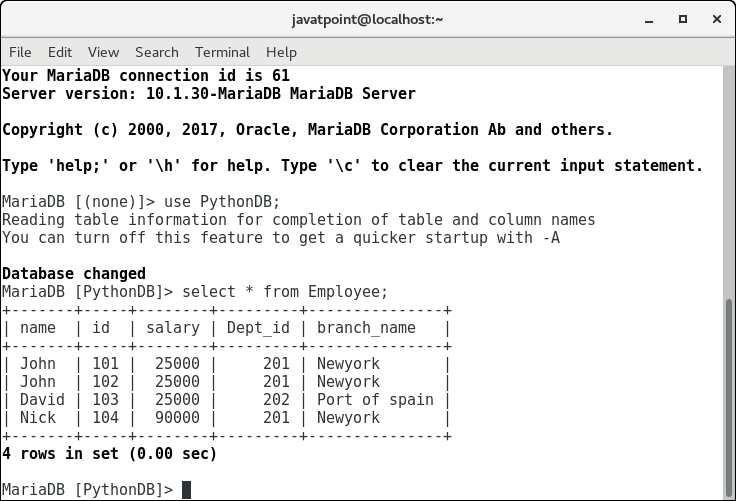
Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
8. sql = "insert into Employee(name, id, salary, dept\_id, branch\_name) values (%s, %s, %s, %s, %s)"
9. val = [("John", 102, 25000.00, 201, "Newyork"),("David",103,25000.00,202,"Port of spain"),("Nick",104,90000.00,201,"Newyork")]
11. **try**:
12. #inserting the values into the table
13. cur.executemany(sql,val)
15. #commit the transaction
16. myconn.commit()
17. **print**(cur.rowcount,"records inserted!")
19. **except**:
20. myconn.rollback()
22. myconn.close()

**Output:**

3 records inserted!



Row ID

In SQL, a particular row is represented by an insertion id which is known as row id. We can get the last inserted row id by using the attribute lastrowid of the cursor object.

Consider the following example.

Example

1. **import** mysql.connector
2. #Create the connection object
3. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
4. #creating the cursor object
5. cur = myconn.cursor()
7. sql = "insert into Employee(name, id, salary, dept\_id, branch\_name) values (%s, %s, %s, %s, %s)"
9. val = ("Mike",105,28000,202,"Guyana")
11. **try**:
12. #inserting the values into the table
13. cur.execute(sql,val)
15. #commit the transaction
16. myconn.commit()
18. #getting rowid
19. **print**(cur.rowcount,"record inserted! id:",cur.lastrowid)
21. **except**:
22. myconn.rollback()
24. myconn.close()

**Output:**

1 record inserted! Id: 0

Read Operation

The SELECT statement is used to read the values from the databases. We can restrict the output of a select query by using various clause in SQL like where, limit, etc.

Python provides the fetchall() method returns the data stored inside the table in the form of rows. We can iterate the result to get the individual rows.

In this section of the tutorial, we will extract the data from the database by using the python script. We will also format the output to print it on the console.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Reading the Employee data
11. cur.execute("select \* from Employee")
13. #fetching the rows from the cursor object
14. result = cur.fetchall()
15. #printing the result
17. **for** x **in** result:
18. **print**(x);
19. **except**:
20. myconn.rollback()
22. myconn.close()

**Output:**

('John', 101, 25000.0, 201, 'Newyork')

('John', 102, 25000.0, 201, 'Newyork')

('David', 103, 25000.0, 202, 'Port of spain')

('Nick', 104, 90000.0, 201, 'Newyork')

('Mike', 105, 28000.0, 202, 'Guyana')

Reading specific columns

We can read the specific columns by mentioning their names instead of using star (\*).

In the following example, we will read the name, id, and salary from the Employee table and print it on the console.

Example

1. **import** mysql.connector
2. #Create the connection object
3. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
4. #creating the cursor object
5. cur = myconn.cursor()
6. **try**:
7. #Reading the Employee data
8. cur.execute("select name, id, salary from Employee")
10. #fetching the rows from the cursor object
11. result = cur.fetchall()
12. #printing the result
13. **for** x **in** result:
14. **print**(x);
15. **except**:
16. myconn.rollback()
17. myconn.close()

**Output:**

('John', 101, 25000.0)

('John', 102, 25000.0)

('David', 103, 25000.0)

('Nick', 104, 90000.0)

('Mike', 105, 28000.0)

The fetchone() method

The fetchone() method is used to fetch only one row from the table. The fetchone() method returns the next row of the result-set.

Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Reading the Employee data
11. cur.execute("select name, id, salary from Employee")
13. #fetching the first row from the cursor object
14. result = cur.fetchone()
16. #printing the result
17. **print**(result)
19. **except**:
20. myconn.rollback()
22. myconn.close()

**Output:**

('John', 101, 25000.0)

Formatting the result

We can format the result by iterating over the result produced by the fetchall() or fetchone() method of cursor object since the result exists as the tuple object which is not readable.

Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
11. #Reading the Employee data
12. cur.execute("select name, id, salary from Employee")
14. #fetching the rows from the cursor object
15. result = cur.fetchall()
17. **print**("Name    id    Salary");
18. **for** row **in** result:
19. **print**("%s    %d    %d"%(row[0],row[1],row[2]))
20. **except**:
21. myconn.rollback()
23. myconn.close()

**Output:**

Name id Salary

John 101 25000

John 102 25000

David 103 25000

Nick 104 90000

Mike 105 28000

Using where clause

We can restrict the result produced by the select statement by using the where clause. This will extract only those columns which satisfy the where condition.

Consider the following example.

Example: printing the names that start with j

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Reading the Employee data
11. cur.execute("select name, id, salary from Employee where name like 'J%'")
13. #fetching the rows from the cursor object
14. result = cur.fetchall()
16. **print**("Name    id    Salary");
18. **for** row **in** result:
19. **print**("%s    %d    %d"%(row[0],row[1],row[2]))
20. **except**:
21. myconn.rollback()
23. myconn.close()

**Output:**

Name id Salary

John 101 25000

John 102 25000

Example: printing the names with id = 101, 102, and 103

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Reading the Employee data
11. cur.execute("select name, id, salary from Employee where id in (101,102,103)")
13. #fetching the rows from the cursor object
14. result = cur.fetchall()
16. **print**("Name    id    Salary");
18. **for** row **in** result:
19. **print**("%s    %d    %d"%(row[0],row[1],row[2]))
20. **except**:
21. myconn.rollback()
23. myconn.close()

**Output:**

Name id Salary

John 101 25000

John 102 25000

David 103 2500

Ordering the result

The ORDER BY clause is used to order the result. Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Reading the Employee data
11. cur.execute("select name, id, salary from Employee order by name")
13. #fetching the rows from the cursor object
14. result = cur.fetchall()
16. **print**("Name    id    Salary");
18. **for** row **in** result:
19. **print**("%s    %d    %d"%(row[0],row[1],row[2]))
20. **except**:
21. myconn.rollback()
23. myconn.close()

**Output:**

Name id Salary

David 103 25000

John 101 25000

John 102 25000

Mike 105 28000

Nick 104 90000

Order by DESC

This orders the result in the decreasing order of a particular column.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Reading the Employee data
11. cur.execute("select name, id, salary from Employee order by name desc")
13. #fetching the rows from the cursor object
14. result = cur.fetchall()
16. #printing the result
17. **print**("Name    id    Salary");
18. **for** row **in** result:
19. **print**("%s    %d    %d"%(row[0],row[1],row[2]))
21. **except**:
22. myconn.rollback()
24. myconn.close()

**Output:**

Name id Salary

Nick 104 90000

Mike 105 28000

John 101 25000

John 102 25000

David 103 25000

Update Operation

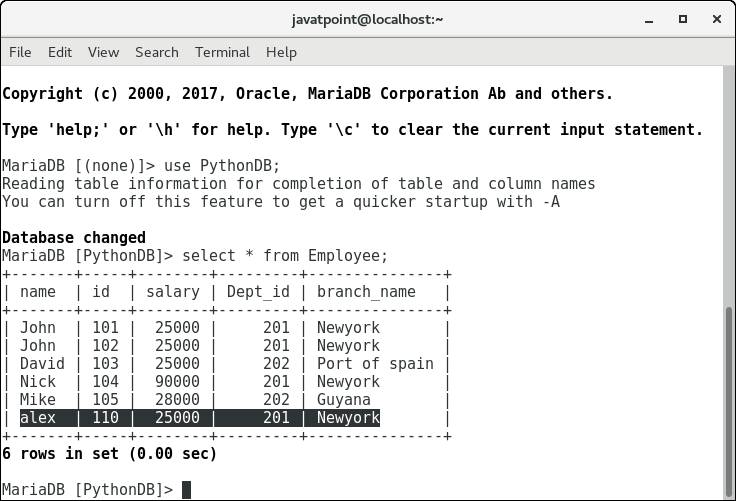
The UPDATE-SET statement is used to update any column inside the table. The following SQL query is used to update a column.

1. >  update Employee set name = 'alex' where id = 110

Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #updating the name of the employee whose id is 110
11. cur.execute("update Employee set name = 'alex' where id = 110")
12. myconn.commit()
13. **except**:
15. myconn.rollback()
17. myconn.close()



Delete Operation

The DELETE FROM statement is used to delete a specific record from the table. Here, we must impose a condition using WHERE clause otherwise all the records from the table will be removed.

The following SQL query is used to delete the employee detail whose id is 110 from the table.

1. >  delete **from** Employee where id = 110

Consider the following example.

Example

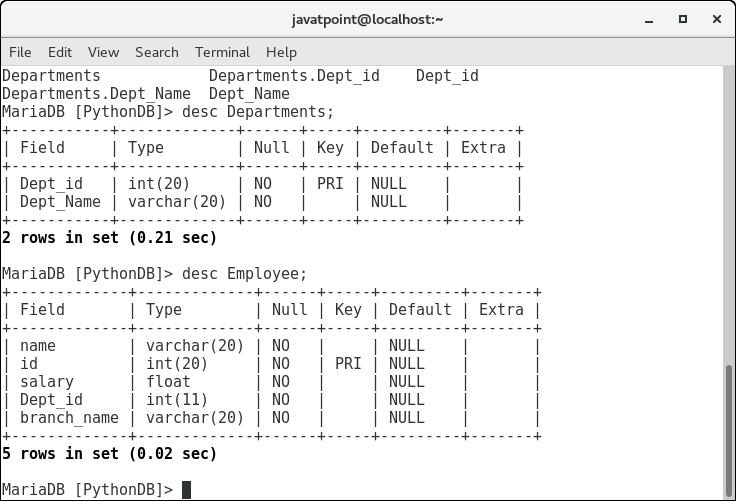
1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #Deleting the employee details whose id is 110
11. cur.execute("delete from Employee where id = 110")
12. myconn.commit()
13. **except**:
15. myconn.rollback()
17. myconn.close()

Join Operation

We can combine the columns from two or more tables by using some common column among them by using the join statement.

We have only one table in our database, let's create one more table Departments with two columns department\_id and department\_name.

1. create table Departments (Dept\_id int(20) primary key **not** null, Dept\_Name varchar(20) **not** null);

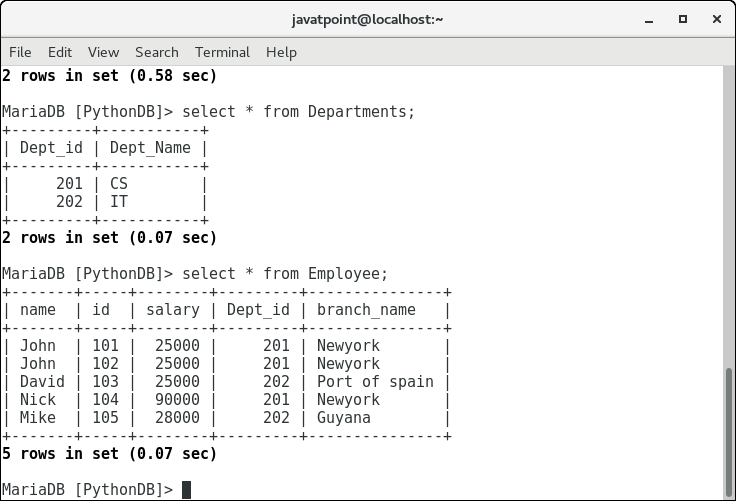


As we have created a new table Departments as shown in the above image. However, we haven't yet inserted any value inside it.

Let's insert some Departments ids and departments names so that we can map this to our Employee table.

1. insert into Departments values (201, "CS");
2. insert into Departments values (202, "IT");

Let's look at the values inserted in each of the tables. Consider the following image.



Now, let's create a python script that joins the two tables on the common column, i.e., dept\_id.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #joining the two tables on departments\_id
11. cur.execute("select Employee.id, Employee.name, Employee.salary, Departments.Dept\_id, Departments.Dept\_Name from Departments join Employee on Departments.Dept\_id = Employee.Dept\_id")
12. **print**("ID    Name    Salary    Dept\_Id    Dept\_Name")
13. **for** row **in** cur:
14. **print**("%d    %s    %d    %d    %s"%(row[0], row[1],row[2],row[3],row[4]))
16. **except**:
17. myconn.rollback()
19. myconn.close()

**Output:**

ID Name Salary Dept\_Id Dept\_Name

101 John 25000 201 CS

102 John 25000 201 CS

103 David 25000 202 IT

104 Nick 90000 201 CS

105 Mike 28000 202 IT

Right Join

Right join shows all the columns of the right-hand side table as we have two tables in the database PythonDB, i.e., Departments and Employee. We do not have any Employee in the table who is not working for any department (Employee for which department id is null). However, to understand the concept of right join let's create the one.

Execute the following query on the MySQL server.

1. insert into Employee(name, id, salary, branch\_name) values ("Alex",108,29900,"Mumbai");

This will insert an employee Alex who doesn't work for any department (department id is null).

Now, we have an employee in the Employee table whose department id is not present in the Departments table. Let's perform the right join on the two tables now.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #joining the two tables on departments\_id
11. result = cur.execute("select Employee.id, Employee.name, Employee.salary, Departments.Dept\_id, Departments.Dept\_Name from Departments right join Employee on Departments.Dept\_id = Employee.Dept\_id")
13. **print**("ID    Name    Salary    Dept\_Id    Dept\_Name")
15. **for** row **in** cur:
16. **print**(row[0],"    ", row[1],"    ",row[2],"    ",row[3],"    ",row[4])


20. **except**:
21. myconn.rollback()
23. myconn.close()

**Output:**

ID Name Salary Dept\_Id Dept\_Name

101 John 25000.0 201 CS

102 John 25000.0 201 CS

103 David 25000.0 202 IT

104 Nick 90000.0 201 CS

105 Mike 28000.0 202 IT

108 Alex 29900.0 None None

Left Join

The left join covers all the data from the left-hand side table. It has just opposite effect to the right join. Consider the following example.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. #joining the two tables on departments\_id
11. result = cur.execute("select Employee.id, Employee.name, Employee.salary, Departments.Dept\_id, Departments.Dept\_Name from Departments left join Employee on Departments.Dept\_id = Employee.Dept\_id")
12. **print**("ID    Name    Salary    Dept\_Id    Dept\_Name")
13. **for** row **in** cur:
14. **print**(row[0],"    ", row[1],"    ",row[2],"    ",row[3],"    ",row[4])


18. **except**:
19. myconn.rollback()
21. myconn.close()

**Output:**

ID Name Salary Dept\_Id Dept\_Name

101 John 25000.0 201 CS

102 John 25000.0 201 CS

103 David 25000.0 202 IT

104 Nick 90000.0 201 CS

105 Mike 28000.0 202 IT

Performing Transactions

Transactions ensure the data consistency of the database. We have to make sure that more than one applications must not modify the records while performing the database operations. The transactions have the following properties.

1. **Atomicity**  
   Either the transaction completes, or nothing happens. If a transaction contains 4 queries then all these queries must be executed, or none of them must be executed.
2. **Consistency**  
   The database must be consistent before the transaction starts and the database must also be consistent after the transaction is completed.
3. **Isolation**  
   Intermediate results of a transaction are not visible outside the current transaction.
4. **Durability**  
   Once a transaction was committed, the effects are persistent, even after a system failure.

Python commit() method

Python provides the commit() method which ensures the changes made to

the database consistently take place.

The syntax to use the commit() method is given below.

1. conn.commit() #conn is the connection object

All the operations that modify the records of the database do not take place until the commit() is called.

Python rollback() method

The rollback() method is used to revert the changes that are done to the database. This method is useful in the sense that, if some error occurs during the database operations, we can rollback that transaction to maintain the database consistency.

The syntax to use the rollback() is given below.

1. Conn.rollback()

Closing the connection

We need to close the database connection once we have done all the operations regarding the database. Python provides the close() method. The syntax to use the close() method is given below.

1. conn.close()

In the following example, we are deleting all the employees who are working for the CS department.

Example

1. **import** mysql.connector
3. #Create the connection object
4. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
6. #creating the cursor object
7. cur = myconn.cursor()
9. **try**:
10. cur.execute("delete from Employee where Dept\_id = 201")
11. myconn.commit()
12. **print**("Deleted !")
13. **except**:
14. **print**("Can't delete !")
15. myconn.rollback()
17. myconn.close()

**Output:**

Deleted !