



Python PostgreSQL

**tutorialspoint**

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985-1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

This tutorial explains how to communicate with PostgreSQL database in detail, along with examples.

## Audience

---

This tutorial is designed for python programmers who would like to understand the psycopg2 modules in detail.

## Prerequisites

---

Before proceeding with this tutorial, you should have a good understanding of python programming language. It is also recommended to have basic understanding of the databases — PostgreSQL.

## Copyright & Disclaimer

---

© Copyright 2020 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

---

About the Tutorial .....	2
Audience.....	2
Prerequisites.....	2
Copyright & Disclaimer .....	2
Table of Contents .....	3
<b>1. Python PostgreSQL — Introduction.....</b>	<b>5</b>
<b>2. Python PostgreSQL — Database Connection.....</b>	<b>7</b>
Establishing connection using python .....	7
<b>3. Python PostgreSQL — Create Database .....</b>	<b>9</b>
Creating a database using python .....	10
<b>4. Python PostgreSQL - Create Table.....</b>	<b>11</b>
Creating a table using python.....	12
<b>5. Python PostgreSQL — Insert Data .....</b>	<b>14</b>
Inserting data using python.....	15
<b>6. Python PostgreSQL — Select Data.....</b>	<b>18</b>
Retrieving data using python.....	19
<b>7. Python PostgreSQL — Where Clause.....</b>	<b>22</b>
Where clause using python .....	23
<b>8. Python PostgreSQL — Order By .....</b>	<b>25</b>
ORDER BY clause using python.....	27
<b>9. Python PostgreSQL — Update Table .....</b>	<b>29</b>
Updating records using python .....	30
<b>10. Python PostgreSQL — Delete Data.....</b>	<b>33</b>
Deleting data using python .....	34
<b>11. Python PostgreSQL — Drop Table .....</b>	<b>37</b>
Removing an entire table using Python.....	38

**12. Python PostgreSQL – Limit .....40**  
    Limit clause using python ..... 41

**13. Python PostgreSQL — Join .....43**  
    Joins using python ..... 44

**14. Python PostgreSQL — Cursor Object.....46**

# 1. Python PostgreSQL — Introduction

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development phase and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

To communicate with PostgreSQL using Python you need to install `psycopg`, an adapter provided for python programming, the current version of this is **`psycopg2`**.

`psycopg2` was written with the aim of being very small and fast, and stable as a rock. It is available under PIP (package manager of python)

## Installing Psycopg2 using PIP

First of all, make sure python and PIP is installed in your system properly and, PIP is up-to-date.

To upgrade PIP, open command prompt and execute the following command:

```
C:\Users\Tutorialspoint>python -m pip install --upgrade pip
Collecting pip
  Using cached
  https://files.pythonhosted.org/packages/8d/07/f7d7ced2f97ca3098c16565efbe6b15fa
  fcba53e8d9bdb431e09140514b0/pip-19.2.2-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 19.0.3
  Uninstalling pip-19.0.3:
    Successfully uninstalled pip-19.0.3
Successfully installed pip-19.2.2
```

Then, open command prompt in admin mode and execute the **`pip install psycopg2-binary`** command as shown below:

```
C:\WINDOWS\system32>pip install psycopg2-binary
Collecting psycopg2-binary
  Using cached
  https://files.pythonhosted.org/packages/80/79/d0d13ce4c2f1addf4786f4a2ded802c2d
  f66ddf3c1b1a982ed8d4cb9fc6d/psycopg2_binary-2.8.3-cp37-cp37m-win32.whl
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.8.3
```

## Verification

To verify the installation, create a sample python script with the following line in it.

```
import mysql.connector
```

If the installation is successful, when you execute it, you should not get any errors:

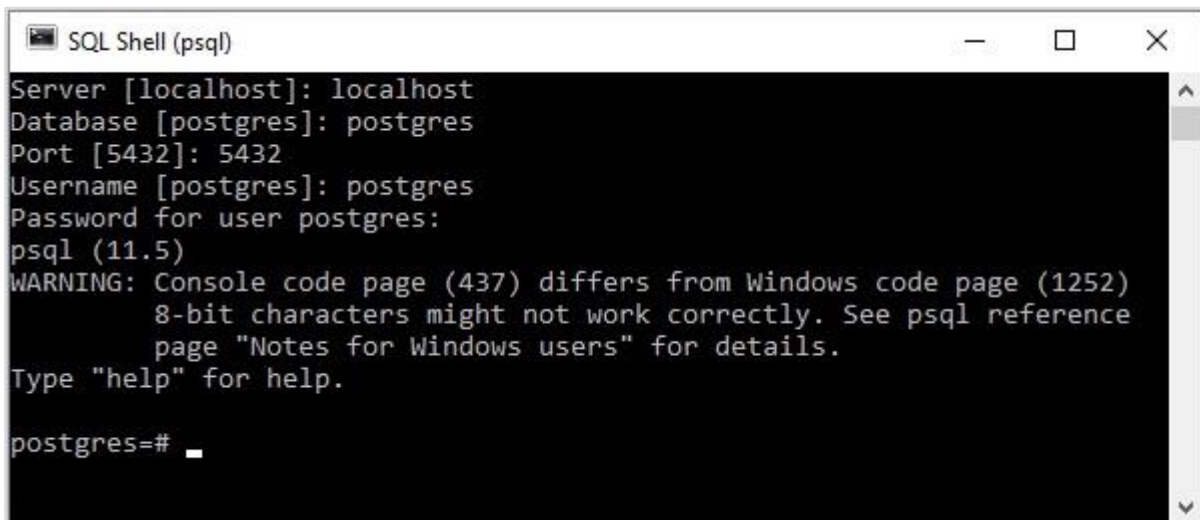
```
D:\Python_PostgreSQL>import psycopg2
```

```
D:\Python_PostgreSQL>
```

## 2. Python PostgreSQL — Database Connection

PostgreSQL provides its own shell to execute queries. To establish connection with the PostgreSQL database, make sure that you have installed it properly in your system. Open the PostgreSQL shell prompt and pass details like Server, Database, username, and password. If all the details you have given are appropriate, a connection is established with PostgreSQL database.

While passing the details you can go with the default server, database, port and, user name suggested by the shell.



```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Password for user postgres:
psql (11.5)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
Type "help" for help.
postgres=# _
```

### Establishing connection using python

The connection class of the **psycopg2** represents/handles an instance of a connection. You can create new connections using the **connect()** function. This accepts the basic connection parameters such as dbname, user, password, host, port and returns a connection object. Using this function, you can establish a connection with the PostgreSQL.

#### Example

The following Python code shows how to connect to an existing database. If the database does not exist, then it will be created and finally a database object will be returned. The name of the default database of PostgreSQL is *postgres*. Therefore, we are supplying it as the database name.

```
import psycopg2

#establishing the connection

conn = psycopg2.connect(database="postgres", user='postgres',
password='password', host='127.0.0.1', port= '5432')
```

```
#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Executing an MYSQL function using the execute() method
cursor.execute("select version()")

# Fetch a single row using fetchone() method.
data = cursor.fetchone()

print("Connection established to: ",data)

#Closing the connection
conn.close()

Connection established to: ('PostgreSQL 11.5, compiled by Visual C++ build
1914, 64-bit',)
```

## Output

```
Connection established to: ('PostgreSQL 11.5, compiled by Visual C++ build
1914, 64-bit',)
```



# 3. Python PostgreSQL — Create Database

You can create a database in PostgreSQL using the CREATE DATABASE statement. You can execute this statement in PostgreSQL shell prompt by specifying the name of the database to be created after the command.

## Syntax

Following is the syntax of the CREATE DATABASE statement.

```
CREATE DATABASE dbname;
```

## Example

Following statement creates a database named *testdb* in PostgreSQL.

```
postgres=# CREATE DATABASE testdb;  
CREATE DATABASE
```

You can list out the database in PostgreSQL using the \l command. If you verify the list of databases, you can find the newly created database as follows:

```
postgres=# \l  
  
                List of databases  
  Name          | Owner   | Encoding | Collate          | Ctype          |  
-----+-----+-----+-----+-----+  
 mydb          | postgres | UTF8     | English_United States.1252 | ..... |  
 postgres     | postgres | UTF8     | English_United States.1252 | ..... |  
 template0    | postgres | UTF8     | English_United States.1252 | ..... |  
 template1    | postgres | UTF8     | English_United States.1252 | ..... |  
 testdb      | postgres | UTF8     | English_United States.1252 | ..... |  
(5 rows)
```

You can also create a database in PostgreSQL from command prompt using the command *createdb*, a wrapper around the SQL statement CREATE DATABASE.

```
C:\Program Files\PostgreSQL\11\bin> createdb -h localhost -p 5432 -U postgres sampledb  
Password:
```

## Creating a database using python

---

The cursor class of psycopg2 provides various methods execute various PostgreSQL commands, fetch records and copy data. You can create a cursor object using the cursor() method of the Connection class.

The execute() method of this class accepts a PostgreSQL query as a parameter and executes it.

Therefore, to create a database in PostgreSQL, execute the CREATE DATABASE query using this method.

### Example

Following python example creates a database named mydb in PostgreSQL database.

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="postgres", user='postgres',
password='password', host='127.0.0.1', port= '5432')
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Preparing query to create a database
sql = '''CREATE database mydb''';

#Creating a database
cursor.execute(sql)
print("Database created successfully.....")

#Closing the connection
conn.close()
```

### Output

```
Database created successfully.....
```

# 4. Python PostgreSQL - Create Table

You can create a new table in a database in PostgreSQL using the CREATE TABLE statement. While executing this you need to specify the name of the table, column names and their data types.

## Syntax

Following is the syntax of the CREATE TABLE statement in PostgreSQL.

```
CREATE TABLE table_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    .....  
    columnN datatype,  
);
```

## Example

Following example creates a table with name CRICKETERS in PostgreSQL.

```
postgres=# CREATE TABLE CRICKETERS (  
    First_Name VARCHAR(255),  
    Last_Name VARCHAR(255),  
    Age INT,  
    Place_Of_Birth VARCHAR(255),  
    Country VARCHAR(255));  
  
CREATE TABLE  
postgres=#
```

You can get the list of tables in a database in PostgreSQL using the \dt command. After creating a table, if you can verify the list of tables you can observe the newly created table in it as follows:

```
postgres=# \dt  
  
List of relations  
  
Schema | Name | Type | Owner  
-----+-----+-----+-----  
public | cricketers | table | postgres
```

```
(1 row)
postgres=#
```

In the same way, you can get the description of the created table using \d as shown below:

```
postgres=# \d cricketers
                Table "public.cricketers"
   Column      |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----
 first_name    | character varying(255) |           |          |
 last_name     | character varying(255) |           |          |
 age          | integer                |           |          |
 place_of_birth | character varying(255) |           |          |
 country       | character varying(255) |           |          |

postgres=#
```

## Creating a table using python

To create a table using python you need to execute the CREATE TABLE statement using the execute() method of the Cursor of *pyscopg2*.

The following Python example creates a table with name employee.

```
import psycopg2

#Establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

#Creating table as per requirement
sql = '''CREATE TABLE EMPLOYEE(
        FIRST_NAME  CHAR(20) NOT NULL,
        LAST_NAME   CHAR(20),
        AGE INT,
```

```
        SEX CHAR(1),
        INCOME FLOAT)'''
cursor.execute(sql)
print("Table created successfully.....")

#Closing the connection
conn.close()
```

## Output

```
Table created successfully.....
```

# 5. Python PostgreSQL — Insert Data

You can insert record into an existing table in PostgreSQL using the **INSERT INTO** statement. While executing this, you need to specify the name of the table, and values for the columns in it.

## Syntax

Following is the recommended syntax of the INSERT statement:

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Where, column1, column2, column3,.. are the names of the columns of a table, and value1, value2, value3,... are the values you need to insert into the table.

## Example

Assume we have created a table with name CRICKETERS using the CREATE TABLE statement as shown below:

```
postgres=# CREATE TABLE CRICKETERS (
First_Name VARCHAR(255),
Last_Name VARCHAR(255),
Age INT,
Place_Of_Birth VARCHAR(255),
Country VARCHAR(255));
CREATE TABLE
postgres=#
```

Following PostgreSQL statement inserts a row in the above created table:

```
postgres=# insert into CRICKETERS (First_Name, Last_Name, Age, Place_Of_Birth,
Country) values('Shikhar', 'Dhawan', 33, 'Delhi', 'India');
INSERT 0 1
postgres=#
```

While inserting records using the *INSERT INTO* statement, if you skip any columns names Record will be inserted leaving empty spaces at columns which you have skipped.

```
postgres=# insert into CRICKETERS (First_Name, Last_Name, Country)
values('Jonathan', 'Trott', 'SouthAfrica');
INSERT 0 1
```

You can also insert records into a table without specifying the column names, if the order of values you pass is same as their respective column names in the table.

```
postgres=# insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale',
'Srilanka');
INSERT 0 1
postgres=# insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur',
'India');
INSERT 0 1
postgres=#
```

After inserting the records into a table you can verify its contents using the SELECT statement as shown below:

```
postgres=# SELECT * from CRICKETERS;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
 Shikhar   | Dhawan   | 33 | Delhi           | India
 Jonathan  | Trott    |    |                 | SouthAfrica
 Kumara    | Sangakkara | 41 | Matale         | Srilanka
 Virat     | Kohli    | 30 | Delhi           | India
 Rohit     | Sharma   | 32 | Nagpur         | India
(5 rows)
```

## Inserting data using python

The cursor class of psycopg2 provides a method with name execute() method. This method accepts the query as a parameter and executes it.

Therefore, to insert data into a table in PostgreSQL using python:

- Import **psycopg2** package.
- Create a connection object using the **connect()** method, by passing the user name, password, host (optional default: localhost) and, database (optional) as parameters to it.
- Turn off the auto-commit mode by setting false as value to the attribute **autocommit**.
- The **cursor()** method of the **Connection** class of the psycopg2 library returns a cursor object. Create a cursor object using this method.

- Then, execute the INSERT statement(s) by passing it/them as a parameter to the execute() method.

## Example

Following Python program creates a table with name EMPLOYEE in PostgreSQL database and inserts records into it using the execute() method:

```
import psycopg2

#Establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

# Preparing SQL queries to INSERT a record into the database.
cursor.execute('''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX,
      INCOME) VALUES ('Ramya', 'Rama priya', 27, 'F', 9000)''')

cursor.execute('''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX,
      INCOME) VALUES ('Vinay', 'Battacharya', 20, 'M', 6000)''')

cursor.execute('''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX,
      INCOME) VALUES ('Sharukh', 'Sheik', 25, 'M', 8300)''')

cursor.execute('''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX,
      INCOME) VALUES ('Sarmista', 'Sharma', 26, 'F', 10000)''')

cursor.execute('''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX,
      INCOME) VALUES ('Tripthi', 'Mishra', 24, 'F', 6000)''')

# Commit your changes in the database
conn.commit()

print("Records inserted.....")
```



```
# Closing the connection  
conn.close()
```

## Output

```
Records inserted.....
```

## 6. Python PostgreSQL — Select Data

You can retrieve the contents of an existing table in PostgreSQL using the SELECT statement. At this statement, you need to specify the name of the table and, it returns its contents in tabular format which is known as result set.

### Syntax

Following is the syntax of the SELECT statement in PostgreSQL:

```
SELECT column1, column2, columnN FROM table_name;
```

### Example

Assume we have created a table with name CRICKETERS using the following query:

```
postgres=# CREATE TABLE CRICKETERS ( First_Name VARCHAR(255), Last_Name  
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));  
CREATE TABLE  
postgres=#
```

And if we have inserted 5 records in to it using INSERT statements as:

```
postgres=# insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi',  
'India');  
INSERT 0 1  
postgres=# insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown',  
'SouthAfrica');  
INSERT 0 1  
postgres=# insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale',  
'Srilanka');  
INSERT 0 1  
postgres=# insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi',  
'India');  
INSERT 0 1  
postgres=# insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur',  
'India');  
INSERT 0 1
```

Following SELECT query retrieves the values of the columns FIRST\_NAME, LAST\_NAME and, COUNTRY from the CRICKETERS table.

```
postgres=# SELECT FIRST_NAME, LAST_NAME, COUNTRY FROM CRICKETERS;  
first_name | last_name | country
```

18

```

-----+-----+-----
Shikhar   | Dhawan   | India
Jonathan  | Trott    | SouthAfrica
Kumara    | Sangakkara | Srilanka
Virat     | Kohli    | India
Rohit     | Sharma   | India
(5 rows)

```

If you want to retrieve all the columns of each record you need to replace the names of the columns with "\*" as shown below:

```

postgres=# SELECT * FROM CRICKETERS;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
Shikhar    | Dhawan   | 33 | Delhi          | India
Jonathan   | Trott    | 38 | CapeTown       | SouthAfrica
Kumara     | Sangakkara | 41 | Matale         | Srilanka
Virat      | Kohli    | 30 | Delhi          | India
Rohit      | Sharma   | 32 | Nagpur         | India
(5 rows)

postgres=#

```

## Retrieving data using python

READ Operation on any database means to fetch some useful information from the database. You can fetch data from PostgreSQL using the fetch() method provided by the psycopg2.

The Cursor class provides three methods namely fetchall(), fetchmany() and fetchone() where,

- The fetchall() method retrieves all the rows in the result set of a query and returns them as list of tuples. (If we execute this after retrieving few rows, it returns the remaining ones).
- The fetchone() method fetches the next row in the result of a query and returns it as a tuple.
- The fetchmany() method is similar to the fetchone() but, it retrieves the next set of rows in the result set of a query, instead of a single row.

Note: A result set is an object that is returned when a cursor object is used to query a table.

## Example

The following Python program connects to a database named mydb of PostgreSQL and retrieves all the records from a table named EMPLOYEE.

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Retrieving data
cursor.execute(''SELECT * from EMPLOYEE'')

#Fetching 1st row from the table
result = cursor.fetchone();
print(result)

#Fetching 1st row from the table
result = cursor.fetchall();
print(result)

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

```
('Ramya', 'Rama priya', 27, 'F', 9000.0)
[('Vinay', 'Battacharya', 20, 'M', 6000.0),
('Sharukh', 'Sheik', 25, 'M', 8300.0),
('Sarmista', 'Sharma', 26, 'F', 10000.0),
```

```
('Tripthi', 'Mishra', 24, 'F', 6000.0)]
```

# 7. Python PostgreSQL — Where Clause

While performing SELECT, UPDATE or, DELETE operations, you can specify condition to filter the records using the WHERE clause. The operation will be performed on the records which satisfies the given condition.

## Syntax

Following is the syntax of the WHERE clause in PostgreSQL:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [search_condition]
```

You can specify a search\_condition using comparison or logical operators. like >, <, =, LIKE, NOT, etc. The following examples would make this concept clear.

## Example

Assume we have created a table with name CRICKETERS using the following query:

```
postgres=# CREATE TABLE CRICKETERS ( First_Name VARCHAR(255), Last_Name
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));
CREATE TABLE
postgres=#
```

And if we have inserted 5 records in to it using INSERT statements as:

```
postgres=# insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown',
'SouthAfrica');
INSERT 0 1
postgres=# insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale',
'Srilanka');
INSERT 0 1
postgres=# insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur',
'India');
INSERT 0 1
```

Following SELECT statement retrieves the records whose age is greater than 35:

```
postgres=# SELECT * FROM CRICKETERS WHERE AGE > 35;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
 Jonathan   | Trott     | 38 | CapeTown       | SouthAfrica
 Kumara     | Sangakkara | 41 | Matale         | Srilanka
(2 rows)

postgres=#
```

## Where clause using python

To fetch specific records from a table using the python program execute the *SELECT* statement with *WHERE* clause, by passing it as a parameter to the **execute()** method.

### Example

Following python example demonstrates the usage of WHERE command using python.

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

sql = '''CREATE TABLE EMPLOYEE(
        FIRST_NAME  CHAR(20) NOT NULL,
        LAST_NAME   CHAR(20),
        AGE INT,
        SEX CHAR(1),
        INCOME FLOAT)'''
```

```
cursor.execute(sql)

#Populating the table
insert_stmt = "INSERT INTO EMPLOYEE (FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
VALUES (%s, %s, %s, %s, %s)"
data = [('Krishna', 'Sharma', 19, 'M', 2000), ('Raj', 'Kandukuri', 20, 'M',
7000),
        ('Ramy', 'Ramapriya', 25, 'M', 5000), ('Mac', 'Mohan', 26, 'M', 2000)]
cursor.executemany(insert_stmt, data)

#Retrieving specific records using the where clause
cursor.execute("SELECT * from EMPLOYEE WHERE AGE <23")

print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

```
[('Krishna', 'Sharma', 19, 'M', 2000.0), ('Raj', 'Kandukuri', 20, 'M', 7000.0)]
```



# 8. Python PostgreSQL — Order By

Usually if you try to retrieve data from a table, you will get the records in the same order in which you have inserted them.

Using the **ORDER BY** clause, while retrieving the records of a table you can sort the resultant records in ascending or descending order based on the desired column.

## Syntax

Following is the syntax of the ORDER BY clause in PostgreSQL.

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

## Example

Assume we have created a table with name CRICKETERS using the following query:

```
postgres=# CREATE TABLE CRICKETERS ( First_Name VARCHAR(255), Last_Name
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));
CREATE TABLE
postgres=#
```

And if we have inserted 5 records in to it using INSERT statements as:

```
postgres=# insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown',
'SouthAfrica');
INSERT 0 1
postgres=# insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale',
'Srilanka');
INSERT 0 1
postgres=# insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur',
'India');
INSERT 0 1
```

Following SELECT statement retrieves the rows of the CRICKETERS table in the ascending order of their age:

```
postgres=# SELECT * FROM CRICKETERS ORDER BY AGE;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
Virat      | Kohli     | 30  | Delhi          | India
Rohit      | Sharma    | 32  | Nagpur         | India
Shikhar    | Dhawan    | 33  | Delhi          | India
Jonathan   | Trott     | 38  | CapeTown       | SouthAfrica
Kumara     | Sangakkara | 41  | Matale         | Srilanka
(5 rows)
```

You can use more than one column to sort the records of a table. Following SELECT statements sort the records of the CRICKETERS table based on the columns age and FIRST\_NAME.

```
postgres=# SELECT * FROM CRICKETERS ORDER BY AGE, FIRST_NAME;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
Virat      | Kohli     | 30  | Delhi          | India
Rohit      | Sharma    | 32  | Nagpur         | India
Shikhar    | Dhawan    | 33  | Delhi          | India
Jonathan   | Trott     | 38  | CapeTown       | SouthAfrica
Kumara     | Sangakkara | 41  | Matale         | Srilanka
(5 rows)
```

By default, the **ORDER BY** clause sorts the records of a table in ascending order. You can arrange the results in descending order using DESC as:

```
postgres=# SELECT * FROM CRICKETERS ORDER BY AGE DESC;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
Kumara     | Sangakkara | 41  | Matale         | Srilanka
Jonathan   | Trott     | 38  | CapeTown       | SouthAfrica
Shikhar    | Dhawan    | 33  | Delhi          | India
Rohit      | Sharma    | 32  | Nagpur         | India
Virat      | Kohli     | 30  | Delhi          | India
(5 rows)
```

## ORDER BY clause using python

To retrieve contents of a table in specific order, invoke the `execute()` method on the cursor object and, pass the `SELECT` statement along with `ORDER BY` clause, as a parameter to it.

### Example

In the following example, we are creating a table with name and Employee, populating it, and retrieving its records back in the (ascending) order of their age, using the `ORDER BY` clause.

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

#Creating a table
sql = '''CREATE TABLE EMPLOYEE(
            FIRST_NAME  CHAR(20) NOT NULL,
            LAST_NAME   CHAR(20),
            AGE INT, SEX CHAR(1),
            INCOME INT,
            CONTACT INT)'''

cursor.execute(sql)

#Populating the table
insert_stmt = "INSERT INTO EMPLOYEE (FIRST_NAME, LAST_NAME, AGE, SEX, INCOME,
CONTACT) VALUES (%s, %s, %s, %s, %s, %s)"
data = [('Krishna', 'Sharma', 26, 'M', 2000, 101), ('Raj', 'Kandukuri', 20,
'M', 7000, 102),
        ('Ramya', 'Ramapriya', 29, 'F', 5000, 103),('Mac', 'Mohan', 26, 'M',
2000, 104)]
```

```
cursor.executemany(insert_stmt, data)
conn.commit()

#Retrieving specific records using the ORDER BY clause
cursor.execute("SELECT * from EMPLOYEE ORDER BY AGE")

print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

```
[('Sharukh', 'Sheik', 25, 'M', 8300.0), ('Sarmista', 'Sharma', 26, 'F', 10000.0)]
```

# 9. Python PostgreSQL — Update Table

You can modify the contents of existing records of a table in PostgreSQL using the UPDATE statement. To update specific rows, you need to use the WHERE clause along with it.

## Syntax

Following is the syntax of the UPDATE statement in PostgreSQL:

```
UPDATE table_name
SET column1 = value1, column2 = value2..., columnN = valueN
WHERE [condition];
```

## Example

Assume we have created a table with name CRICKETERS using the following query:

```
postgres=# CREATE TABLE CRICKETERS ( First_Name VARCHAR(255), Last_Name
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));
CREATE TABLE
postgres=#
```

And if we have inserted 5 records in to it using INSERT statements as:

```
postgres=# insert into CRICKETERS values('Shikhar', 'Dhawan', 33, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values('Jonathan', 'Trott', 38, 'CapeTown',
'SouthAfrica');
INSERT 0 1
postgres=# insert into CRICKETERS values('Kumara', 'Sangakkara', 41, 'Matale',
'Srilanka');
INSERT 0 1
postgres=# insert into CRICKETERS values('Virat', 'Kohli', 30, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values('Rohit', 'Sharma', 32, 'Nagpur',
'India');
INSERT 0 1
```

Following statement modifies the age of the cricketer, whose first name is **Shikhar**:

```
postgres=# UPDATE CRICKETERS SET AGE = 45 WHERE FIRST_NAME = 'Shikhar' ;
```

```
UPDATE 1
postgres=#
```

If you retrieve the record whose FIRST\_NAME is Shikhar you observe that the age value has been changed to 45:

```
postgres=# SELECT * FROM CRICKETERS WHERE FIRST_NAME = 'Shikhar';
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
 Shikhar   | Dhawan   | 45 | Delhi          | India
(1 row)

postgres=#
```

If you haven't used the WHERE clause, values of all the records will be updated. Following UPDATE statement increases the age of all the records in the CRICKETERS table by 1:

```
postgres=# UPDATE CRICKETERS SET AGE = AGE+1;
UPDATE 5
```

If you retrieve the contents of the table using SELECT command, you can see the updated values as:

```
postgres=# SELECT * FROM CRICKETERS;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
 Jonathan   | Trott     | 39 | CapeTown       | SouthAfrica
 Kumara     | Sangakkara | 42 | Matale        | Srilanka
 Virat      | Kohli     | 31 | Delhi          | India
 Rohit      | Sharma    | 33 | Nagpur         | India
 Shikhar    | Dhawan    | 46 | Delhi          | India
(5 rows)
```

## Updating records using python

The cursor class of psycopg2 provides a method with name execute() method. This method accepts the query as a parameter and executes it.

Therefore, to insert data into a table in PostgreSQL using python:

- Import **psycopg2** package.

- Create a connection object using the **connect()** method, by passing the user name, password, host (optional default: localhost) and, database (optional) as parameters to it.
- Turn off the auto-commit mode by setting false as value to the attribute **autocommit**.
- The **cursor()** method of the **Connection** class of the psycopg2 library returns a cursor object. Create a cursor object using this method.
- Then, execute the UPDATE statement by passing it as a parameter to the execute() method.

## Example

Following Python code updates the contents of the *Employee* table and retrieves the results:

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Fetching all the rows before the update
print("Contents of the Employee table: ")
sql = '''SELECT * from EMPLOYEE'''
cursor.execute(sql)
print(cursor.fetchall())

#Updating the records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = 'M'"
cursor.execute(sql)
print("Table updated..... ")

#Fetching all the rows after the update
print("Contents of the Employee table after the update operation: ")
```

```
sql = '''SELECT * from EMPLOYEE'''
cursor.execute(sql)
print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

```
Contents of the Employee table:
[('Ramya', 'Rama priya', 27, 'F', 9000.0), ('Vinay', 'Battacharya', 20, 'M',
6000.0), ('Sharukh', 'Sheik', 25, 'M', 8300.0), ('Sarmista', 'Sharma', 26, 'F',
10000.0), ('Tripthi', 'Mishra', 24, 'F', 6000.0)]
Table updated.....
Contents of the Employee table after the update operation:
[('Ramya', 'Rama priya', 27, 'F', 9000.0), ('Sarmista', 'Sharma', 26, 'F',
10000.0), ('Tripthi', 'Mishra', 24, 'F', 6000.0), ('Vinay', 'Battacharya', 21,
'M', 6000.0), ('Sharukh', 'Sheik', 26, 'M', 8300.0)]
```



# 10. Python PostgreSQL — Delete Data

You can delete the records in an existing table using the **DELETE FROM** statement of PostgreSQL database. To remove specific records, you need to use WHERE clause along with it.

## Syntax

Following is the syntax of the DELETE query in PostgreSQL:

```
DELETE FROM table_name [WHERE Clause]
```

## Example

Assume we have created a table with name CRICKETERS using the following query:

```
postgres=# CREATE TABLE CRICKETERS ( First_Name VARCHAR(255), Last_Name
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));
CREATE TABLE
postgres=#
```

And if we have inserted 5 records in to it using INSERT statements as:

```
postgres=# insert into CRICKETERS values ('Shikhar', 'Dhawan', 33, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Jonathan', 'Trott', 38, 'CapeTown',
'SouthAfrica');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Kumara', 'Sangakkara', 41, 'Matale',
'Srilanka');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Virat', 'Kohli', 30, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Rohit', 'Sharma', 32, 'Nagpur',
'India');
INSERT 0 1
```

Following statement deletes the record of the cricketer whose last name is 'Sangakkara'.

```
postgres=# DELETE FROM CRICKETERS WHERE LAST_NAME = 'Sangakkara';
DELETE 1
```

If you retrieve the contents of the table using the SELECT statement, you can see only 4 records since we have deleted one.

```
postgres=# SELECT * FROM CRICKETERS;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
 Jonathan   | Trott     | 39  | CapeTown       | SouthAfrica
 Virat      | Kohli     | 31  | Delhi          | India
 Rohit      | Sharma    | 33  | Nagpur         | India
 Shikhar    | Dhawan    | 46  | Delhi          | India
(4 rows)
```

If you execute the DELETE FROM statement without the WHERE clause all the records from the specified table will be deleted.

```
postgres=# DELETE FROM CRICKETERS;
DELETE 4
```

Since you have deleted all the records, if you try to retrieve the contents of the CRICKETERS table, using SELECT statement you will get an empty result set as shown below:

```
postgres=# SELECT * FROM CRICKETERS;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
(0 rows)
```

## Deleting data using python

The cursor class of psycopg2 provides a method with name execute() method. This method accepts the query as a parameter and executes it.

Therefore, to insert data into a table in PostgreSQL using python:

- Import **psycopg2** package.
- Create a connection object using the **connect()** method, by passing the user name, password, host (optional default: localhost) and, database (optional) as parameters to it.
- Turn off the auto-commit mode by setting false as value to the attribute **autocommit**.
- The **cursor()** method of the **Connection** class of the psycopg2 library returns a cursor object. Create a cursor object using this method.

- Then, execute the DELETE statement by passing it as a parameter to the execute() method.

## Example

Following Python code deletes records of the EMPLOYEE table with age values greater than 25:

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Retrieving contents of the table
print("Contents of the table: ")
cursor.execute('''SELECT * from EMPLOYEE''')
print(cursor.fetchall())

#Deleting records
cursor.execute('''DELETE FROM EMPLOYEE WHERE AGE > 25''')

#Retrieving data after delete
print("Contents of the table after delete operation ")
cursor.execute("SELECT * from EMPLOYEE")
print(cursor.fetchall())

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

Contents of the table:

```
[('Ramya', 'Rama priya', 27, 'F', 9000.0), ('Sarmista', 'Sharma', 26, 'F', 10000.0), ('Tripthi', 'Mishra', 24, 'F', 6000.0), ('Vinay', 'Battacharya', 21, 'M', 6000.0), ('Sharukh', 'Sheik', 26, 'M', 8300.0)]
```

Contents of the table after delete operation:

```
[('Tripthi', 'Mishra', 24, 'F', 6000.0), ('Vinay', 'Battacharya', 21, 'M', 6000.0)]
```

# 11. Python PostgreSQL — Drop Table

You can drop a table from PostgreSQL database using the DROP TABLE statement.

## Syntax

Following is the syntax of the DROP TABLE statement in PostgreSQL:

```
DROP TABLE table_name;
```

## Example

Assume we have created two tables with name CRICKETERS and EMPLOYEES using the following queries:

```
postgres=# CREATE TABLE CRICKETERS (First_Name VARCHAR(255), Last_Name
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));
CREATE TABLE
postgres=#
postgres=# CREATE TABLE EMPLOYEE(FIRST_NAME CHAR(20) NOT NULL, LAST_NAME
CHAR(20), AGE INT, SEX CHAR(1), INCOME FLOAT);
CREATE TABLE
postgres=#
```

Now if you verify the list of tables using the “\dt” command, you can see the above created tables as:

```
postgres=# \dt;
                List of relations
 Schema |   Name   | Type  | Owner
-----+-----+-----+-----
 public | cricketers | table | postgres
 public | employee  | table | postgres
(2 rows)
postgres=#
```

Following statement deletes the table named Employee from the database:

```
postgres=# DROP table employee;
DROP TABLE
```

Since you have deleted the Employee table, if you retrieve the list of tables again, you can observe only one table in it.

```
postgres=# \dt;
          List of relations
 Schema | Name      | Type  | Owner
-----+-----+-----+-----
 public | cricketers | table | postgres
(1 row)

postgres=#
```

If you try to delete the Employee table again, since you have already deleted it, you will get an error saying "table does not exist" as shown below:

```
postgres=# DROP table employee;
ERROR:  table "employee" does not exist
postgres=#
```

To resolve this, you can use the IF EXISTS clause along with the DELTE statement. This removes the table if it exists else skips the DLETE operation.

```
postgres=# DROP table IF EXISTS employee;
NOTICE:  table "employee" does not exist, skipping
DROP TABLE
postgres=#
```

## Removing an entire table using Python

You can drop a table whenever you need to, using the DROP statement. But you need to be very careful while deleting any existing table because the data lost will not be recovered after deleting a table.

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
```

```
cursor = conn.cursor()

#Doping EMPLOYEE table if already exists
cursor.execute("DROP TABLE emp")
print("Table dropped... ")

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

```
#Table dropped...
```

# 12. Python PostgreSQL – Limit

While executing a PostgreSQL SELECT statement you can limit the number of records in its result using the LIMIT clause.

## Syntax

Following is the syntax of the LIMIT clause in PostgreSQL:

```
SELECT column1, column2, columnN
FROM table_name
LIMIT [no of rows]
```

## Example

Assume we have created a table with name CRICKETERS using the following query:

```
postgres=# CREATE TABLE CRICKETERS ( First_Name VARCHAR(255), Last_Name
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));
CREATE TABLE
postgres=#
```

And if we have inserted 5 records in to it using INSERT statements as:

```
postgres=# insert into CRICKETERS values ('Shikhar', 'Dhawan', 33, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Jonathan', 'Trott', 38, 'CapeTown',
'SouthAfrica');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Kumara', 'Sangakkara', 41, 'Matale',
'Srilanka');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Virat', 'Kohli', 30, 'Delhi',
'India');
INSERT 0 1
postgres=# insert into CRICKETERS values ('Rohit', 'Sharma', 32, 'Nagpur',
'India');
INSERT 0 1
```

Following statement retrieves the first 3 records of the Cricketers table using the LIMIT clause:



```
postgres=# SELECT * FROM CRICKETERS LIMIT 3;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
 Shikhar   | Dhawan    | 33 | Delhi          | India
 Jonathan  | Trott     | 38 | CapeTown       | SouthAfrica
 Kumara    | Sangakkara | 41 | Matale         | Srilanka
(3 rows)
```

If you want to get records starting from a particular record (offset) you can do so, using the OFFSET clause along with LIMIT.

```
postgres=# SELECT * FROM CRICKETERS LIMIT 3 OFFSET 2;
 first_name | last_name | age | place_of_birth | country
-----+-----+-----+-----+-----
 Kumara    | Sangakkara | 41 | Matale         | Srilanka
 Virat     | Kohli     | 30 | Delhi          | India
 Rohit     | Sharma    | 32 | Nagpur         | India
(3 rows)

postgres=#
```

## Limit clause using python

Following python example retrieves the contents of a table named EMPLOYEE, limiting the number of records in the result to 2:

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Retrieving single row
```

```
sql = '''SELECT * from EMPLOYEE LIMIT 2 OFFSET 2'''

#Executing the query
cursor.execute(sql)

#Fetching the data
result = cursor.fetchall();
print(result)

#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

```
[('Sharukh', 'Sheik', 25, 'M', 8300.0), ('Sarmista', 'Sharma', 26, 'F', 10000.0)]
```

# 13. Python PostgreSQL — Join

When you have divided the data in two tables you can fetch combined records from these two tables using Joins.

## Example

Assume we have created a table with name CRICKETERS and inserted 5 records into it as shown below:

```
postgres=# CREATE TABLE CRICKETERS (First_Name VARCHAR(255), Last_Name
VARCHAR(255), Age int, Place_Of_Birth VARCHAR(255), Country VARCHAR(255));
postgres=# insert into CRICKETERS values ('Shikhar', 'Dhawan', 33, 'Delhi',
'India');
postgres=# insert into CRICKETERS values ('Jonathan', 'Trott', 38, 'CapeTown',
'SouthAfrica');
postgres=# insert into CRICKETERS values ('Kumara', 'Sangakkara', 41, 'Matale',
'Srilanka');
postgres=# insert into CRICKETERS values ('Virat', 'Kohli', 30, 'Delhi',
'India');
postgres=# insert into CRICKETERS values ('Rohit', 'Sharma', 32, 'Nagpur',
'India');
```

And, if we have created another table with name OdiStats and inserted 5 records into it as:

```
postgres=# CREATE TABLE ODISTats (First_Name VARCHAR(255), Matches INT, Runs
INT, AVG FLOAT, Centuries INT, HalfCenturies INT);
postgres=# insert into OdiStats values ('Shikhar', 133, 5518, 44.5, 17, 27);
postgres=# insert into OdiStats values ('Jonathan', 68, 2819, 51.25, 4, 22);
postgres=# insert into OdiStats values ('Kumara', 404, 14234, 41.99, 25, 93);
postgres=# insert into OdiStats values ('Virat', 239, 11520, 60.31, 43, 54);
postgres=# insert into OdiStats values ('Rohit', 218, 8686, 48.53, 24, 42);
```

Following statement retrieves data combining the values in these two tables:

```
postgres=# SELECT
    Cricketers.First_Name, Cricketers.Last_Name, Cricketers.Country,
    OdiStats.matches, OdiStats.runs, OdiStats.centuries, OdiStats.halfcenturies
    from Cricketers INNER JOIN OdiStats ON Cricketers.First_Name = OdiStats.First_Name;
first_name | last_name | country | matches | runs | centuries | halfcenturies
-----+-----+-----+-----+-----+-----+-----
```

Shikhar	Dhawan	India	133	5518	17	27
Jonathan	Trott	SouthAfrica	68	2819	4	22
Kumara	Sangakkara	Srilanka	404	14234	25	93
Virat	Kohli	India	239	11520	43	54
Rohit	Sharma	India	218	8686	24	42

(5 rows)

postgres=#

## Joins using python

When you have divided the data in two tables you can fetch combined records from these two tables using Joins.

### Example

Following python program demonstrates the usage of the JOIN clause:

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()

#Retrieving single row
sql = '''SELECT * from EMP INNER JOIN CONTACT ON EMP.CONTACT = CONTACT.ID'''

#Executing the query
cursor.execute(sql)

#Fetching 1st row from the table
result = cursor.fetchall();

print(result)
```

```
#Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```

## Output

```
[('Ramya', 'Rama priya', 27, 'F', 9000.0, 101, 101, 'Krishna@mymail.com',
'Hyderabad'), ('Vinay', 'Battacharya', 20, 'M', 6000.0, 102, 102,
'Raja@mymail.com', 'Vishakhapatnam'), ('Sharukh', 'Sheik', 25, 'M', 8300.0,
103, 103, 'Krishna@mymail.com ', 'Pune'), ('Sarmista', 'Sharma', 26, 'F',
10000.0, 104, 104, 'Raja@mymail.com', 'Mumbai')]
```

# 14. Python PostgreSQL — Cursor Object

The `Cursor` class of the `psycopg` library provide methods to execute the PostgreSQL commands in the database using python code.

Using the methods of it you can execute SQL statements, fetch data from the result sets, call procedures.

You can create **Cursor** object using the `cursor()` method of the `Connection` object/class.

## Example

```
import psycopg2

#establishing the connection
conn = psycopg2.connect(database="mydb", user='postgres', password='password',
host='127.0.0.1', port= '5432')

#Setting auto commit false
conn.autocommit = True

#Creating a cursor object using the cursor() method
cursor = conn.cursor()
```

## Methods

Following are the various methods provided by the `Cursor` class/object.

Method	Description
<code>callproc()</code>	This method is used to call existing procedures PostgreSQL database.
<code>close()</code>	This method is used to close the current cursor object.
<code>executemany()</code>	This method accepts a list series of parameters list. Prepares an MySQL query and executes it with all the parameters.
<code>execute()</code>	This method accepts a MySQL query as a parameter and executes the given query.

fetchall()	This method retrieves all the rows in the result set of a query and returns them as list of tuples. (If we execute this after retrieving few rows it returns the remaining ones)
fetchone()	This method fetches the next row in the result of a query and returns it as a tuple.
fetchmany()	This method is similar to the fetchone() but, it retrieves the next set of rows in the result set of a query, instead of a single row.

## Properties

Following are the properties of the Cursor class:

Property	Description
description	This is a read only property which returns the list containing the description of columns in a result-set.
lastrowid	This is a read only property, if there are any auto-incremented columns in the table, this returns the value generated for that column in the last INSERT or, UPDATE operation.
rowcount	This returns the number of rows returned/updated in case of SELECT and UPDATE operations.
closed	This property specifies whether a cursor is closed or not, if so it returns true, else false.
connection	This returns a reference to the connection object using which this cursor was created.
name	This property returns the name of the cursor.
scrollable	This property specifies whether a particular cursor is scrollable.