PostgreSQL in Python.

### **What is Psycopg2?**

The Psycopg2 library uses the C programming language as a wrapper around the [libpq](https://www.postgresql.org/docs/current/libpq.html) PostgreSQL library to support the Python DB API 2.0 standards. The C implementation of Psycopg2 makes it incredibly quick and effective.

## **How to Install Psycopg2**

We must first install Psycopg2 in order to use it. We can install it via the terminal or command prompt using pip.

#installation

pip install psycopg2

pip3 install psycopg2

If we also decide to install the connector library in a virtual environment, you can do so using this code:

virtualenv env && source env/bin/activate

pip install psycopg2-binary

The Psycopg2 library and all of its dependencies will be installed into our Python virtual environment with this code snippet.

## **How to Query PostgreSQL using Python**

The first thing to do is to import the library (this is very important). We will make use of two Psycogp2 objects:

* ****Conection object****: The connection to a PostgreSQL database instance is managed by the connection object. It encapsulates a database session, created using the function connect().
* ****Cursor object****: The cursor object makes it possible for Python scripts to run PostgreSQL commands within a database session. The connection generates cursors, then the cursor() method ties them permanently to the connection. All commands are carried out within the framework of the connection-enclosed database session.

import psycopg2

conn = psycopg2.connect(database="db\_name",

host="db\_host",

user="db\_user",

password="db\_pass",

port="db\_port")

We have to specify those arguments in order to be able to connect to the database. Let's have a quick look into there arguments.

* ****database****: the name of the database we wish to access or connect to. Note that we can only connect to one database with one connection object.
* ****host****: this most likely refers to the database server's IP address or URL.
* ****user****: as the name implies, this refers to the name of the PostgreSQL user.
* ****password****: this is the password that matches the PostgreSQL user.
* ****port****: the PostgreSQL server's port number on localhost – it is usually 5432.

If our database credentials were entered correctly, we will receive a live database connection object that we can use to build a cursor object. We can go ahead and run any database queries and retrieve data with the aid of a cursor object.

cursor = conn.cursor()

Let's write a simple query:

cursor.execute("SELECT \* FROM DB\_table WHERE id = 1")

We apply the execute() function and supply a query string as its parameter. Then the database will be queried using the query that we entered.

After we have successfully achieved this, in order to be able to retrieve data from the database using Pyscopg2, we have to use any of these functions: fetchone() fetchall(), or fetchmany().

### **How to use**fetchone()**:**

After running the SQL query, this function will only return the first row. It is the simplest method of getting data out of a database.

#code

print(cursor.fetchone())

#output

(1, 'A-CLASS', '2018', 'Subcompact executive hatchback')

fetchone() example

The fetchone() function returns a single row in the form of a tuple, with the information arranged in the order specified by the query's supplied columns.

When constructing the query string, it's crucial to provide the column orders precisely in order to distinguish which data in the tuple belongs to which.

### **How to use**fetchall()**:**

The fetchall() function works the same way as fetchone() except that it returns not just one row but all the rows. So in case we want 20-200 rows or more, we make use of Psycopg2 fetchall().

#code

print(cursor.fetchall())

#output

[(1, 'A-CLASS', '2018', 'Subcompact executive hatchback'),

(2, 'C-CLASS', '2021', 'D-segment/compact executive sedan'),

(3, 'CLA', '2019', 'Subcompact executive fastback sedan'),

(4, 'CLS', '2018', 'E-segment/executive fastback sedan'),

(5, 'E-CLASS', '2017', 'E-segment/executive sedan'),

(6, 'EQE', '2022', 'All-electric E-segment fastback'),

(7, 'EQS', '2021', 'All-electric full-size luxury liftback'),

(8, 'S-CLASS', '2020', 'F-segment/full-size luxury sedan.'),

(9, 'G-CLASS', '2018', 'Mid-size luxury SUV, known as the G-Wagen'),

(10, 'GLE', '2019', 'Mid-size luxury crossover SUV')]

[...]

fetchall() example

### **How to use**fetchmany()**:**

The fetchmany() function allows us to get a number of records out of the database and gives us additional control over the precise number of rows we get.

#code

print(cursor.fetchmany(size=3))

#output

[(1, 'A-CLASS', '2018', 'Subcompact executive hatchback'),

(2, 'C-CLASS', '2021', 'D-segment/compact executive sedan'),

(3, 'CLA', '2019', 'Subcompact executive fastback sedan')]

fetchmany() example

Because we set the argument to 3, we only received three rows.

When we are done querying our database we need to close the connection with conn.close().