Database Management Systems

Module 34: Application Design and Development/4: Python and PostgreSQL

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced the use of SQL from a programming language

Module Objectives

- To understand how to access PostgreSQL database from Python
- To understand Python Web Application with PostgresSQL

Module Outline

- Accessing PostgreSQL from Python
- Developing Web Application with Python

Working with PostgreSQL and Python

Python Modules for PostgreSQL

Following Python modules that can be used to work with a PostgreSQL database server:

- **psycopg2**
- pg8000
- py-postgresql
- PyGreSQL
- ocpgdb
- bpgsql
- SQLAlchemy (needs any of the above to be installed separately)

Source: *https://pynative.com/python-postgresql-tutorial/*

Package psycopg2

Advantages of psycopg2

- Most popular and stable module to work with PostgreSQL
- Used in most of the Python and Postgres frameworks
- An actively maintained package and supports Python 2.x and 3.x
- Thread-safe and designed for heavily multi-threaded applications.

Installing Psycopg2 using `pip` command

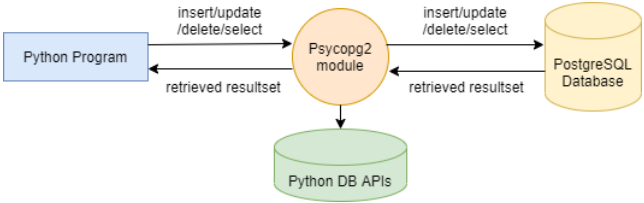
- The following `pip` command installs psycopg2 on different operating systems including Windows, MacOS, Linux, and Unix
  ```
  pip install psycopg2
  ```
- For installing specific version, the following command can be used
  ```
  pip install psycopg2=2.8.6
  ```

Source: *https://pynative.com/python-postgresql-tutorial/*

Steps to access PostgresSQL from Python

Steps to access PostgresSQL from Python using Psycopg

a) Create connection
b) Create cursor
c) Execute the query
d) Commit/rollback
e) Close the cursor
f) Close the connection



Source: *https://pynative.com/python-postgresql-tutorial/*

Python psycopg2 Module APIs: connection objects

- `psycopg2.connect(database="mydb", user="myuser", password="mypass" host="127.0.0.1", port="5432")`
  This API opens a connection to the PostgreSQL database. If database is opened successfully, it returns a connection object.
- `connection.close()`
  This method closes the database connection.

Important **psycopg2** module routines for managing cursor object:

- `connection.cursor()`
  This routine creates a cursor which will be used throughout the program.
- `cursor.close()`
  This method closes the cursor.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

Python psycopg2 Module APIs: insert, delete, update & stored procedures

- `cursor.execute(sql [, optional parameters])`
  This routine executes an SQL statement. The SQL statement may be parameterized (i.e., placeholders instead of SQL literals). The **psycopg2** module supports placeholder using %s sign. For example: `cursor.execute("insert into people values (%s, %s)", (who, age))`
- `cursor.executemany(sql, seq_of_parameters)`
  This routine executes an SQL command against all parameter sequences or mappings found in the sequence SQL.
- `cursor.callproc(procname[, parameters])`
  This routine executes a stored database procedure with the given name. The sequence of parameters must contain one entry for each argument that the procedure expects.
- `cursor.rowcount`
  This is a read-only attribute which returns the total number of database rows that have been modified, inserted, or deleted by the last `execute()`.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

Python psycopg2 Module APIs: select

- `cursor.fetchone()`
  This method fetches the next row of a query result set, returning a single sequence, or None when no more data is available.
- `cursor.fetchmany([size=cursor.arraysize])`
  This routine fetches the next set of rows of a query result, returning a list. An empty list is returned when no more rows are available. The method tries to fetch as many rows as indicated by the size parameter.
- `cursor.fetchall()`
  This routine fetches all (remaining) rows of a query result, returning a list. An empty list is returned when no rows are available.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

Python psycopg2 Module APIs: commit & rollback

- `connection.commit()`
  This method commits the current transaction. If you do not call this method, anything you did since the last call to `commit()` is not visible to other database connections.
- `connection.rollback()`
  This method rolls back any changes to the database since the last call to `commit()`.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

Connect to a PostgreSQL Database Server


```
import psycopg2
def connectDb(dbname, usrname, pwd, address, portnum):
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = dbname, user = usrname, \
            password = pwd, host = address, port = portnum)
        print ("Database connected successfully")
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        conn.close()            # close the connection
connectDb("mydb", "myuser", "mypass", "127.0.0.1", "5432")  # function call
```

Output:

```
Database connected successfully
```

`psycopg2.DatabaseError`: Exception raised for errors that are related to the PostgreSQL database.
We assume the following for all the programs in this module:

- Database Name: *mydb*
- Username: *myuser*
- Password: *mypass*
- Host Name: *localhost* or IP address *127.0.0.1*

Steps to execute SQL commands

1. Use the `psycopg2.connect()` method with the required arguments to connect PostgresSQL. It would return an Connection object if the connection established successfully.
2. Create a cursor object using the `cursor()` method of connection object.
3. The `execute()` methods run the SQL commands and return the result.
4. Use `cursor.fetchall()` or `fetchone()` or `fetchmany()` to read query result.
5. Use `commit()` to make the changes in database persistent, or use `rollback()` to revert the database changes.
6. Use `cursor.close()` and `connection.close()` method to close the cursor and PostgreSQL connection.

Source: *https://pynative.com/python-postgresql-tutorial/*

CREATE new PostgreSQL tables


```
import psycopg2
def createTable():
    conn = None
    try:
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432") # connect to the database
        cur = conn.cursor()  # create a new cursor
        cur.execute('''CREATE TABLE EMPLOYEE \
            (emp_num INT PRIMARY KEY     NOT NULL, \
            emp_name VARCHAR(40)    NOT NULL, \
            department VARCHAR(40)     NOT NULL)''') # execute the CREATE TABLE statement
        conn.commit()       # commit the changes to the database
        print ("Table created successfully")
        cur.close()         # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()    # close the connection
createTable()    #function call
```

Output (if table EMPLOYEE does not exist):

```
Table created successfully
```

Output (if table EMPLOYEE already exists):

```
relation "employee" already exists
```

Executing INSERT statement from Python


```
import psycopg2
def insertRecord(num, name, dept):
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432")
        cur = conn.cursor()              # create a new cursor
        # execute the INSERT statement
        cur.execute("INSERT INTO EMPLOYEE (emp_num, emp_name, department) \
            VALUES (%s, %s, %s)", (num, name, dept))
        conn.commit()                    # commit the changes to the database
        print ("Total number of rows inserted :", cur.rowcount);
        cur.close()                      # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()                 # close the connection
insertRecord(110, 'Bhaskar', 'HR')       #function call
```

Output:

```
Total number of rows inserted : 1
```

Output:

```
duplicate key value violates unique constraint "employee_pkey"
DETAIL:  Key (emp_num)=(110) already exists.
```

If a row already exists with emp_num = 110

Executing DELETE statement from Python


```
import psycopg2
def deleteRecord(num):
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432")
        cur = conn.cursor()     # create a new cursor
        # execute the DELETE statement
        cur.execute("DELETE FROM EMPLOYEE WHERE emp_num = %s", (num,))
        conn.commit()           # commit the changes to the database
        print ("Total number of rows deleted :", cur.rowcount)
        cur.close()             # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        conn.close()            # close the connection
deleteRecord(110)               #function call
```

Output:
```
Total number of rows deleted : 1
```

Output:
```
Total number of rows deleted : 0
```

If the row does not exist

Executing UPDATE statement from Python


```
import psycopg2
def updateRecord(num, dept):
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432")
        cur = conn.cursor()     # create a new cursor
        # execute the UPDATE statement
        cur.execute("UPDATE EMPLOYEE set department = %s where emp_num = \
            %s", (dept, num))
        conn.commit()           # commit the changes to the database
        print ("Total number of rows updated :", cur.rowcount)
        cur.close()             # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        conn.close()            # close the connection
updateRecord(110, "Finance")    #function call
```

Output:
```
Total number of rows updated : 1
```

Output:
```
Total number of rows updated : 0
```

If the row does not exist

Executing SELECT statement from Python


```
import psycopg2
def selectAll():
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432")
        cur = conn.cursor()     # create a new cursor
        # execute the SELECT statement
        cur.execute("SELECT emp_num, emp_name, department FROM EMPLOYEE")
        rows = cur.fetchall()   # fetches all rows of the query result set
        for row in rows:
            print (print ("Employee ID = ", row[0], ", NAME = ", \
                row[1], ", DEPARTMENT = ", row[2]))
        cur.close()             # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        conn.close()            # close the connection
selectAll()                     # function call
```

Output:

```
Employee ID = 110, NAME = Bhaskar, DEPARTMENT = HR
Employee ID = 111, NAME = Ishaan, DEPARTMENT = FINANCE
Employee ID = 112, NAME = Jairaj, DEPARTMENT = TECHNOLOGY
Employee ID = 113, NAME = Ananya, DEPARTMENT = TECHNOLOGY
```

Python Frameworks for PostgresSQL

Web and Internet Development using Python

Python offers several frameworks such as **bottle.py**, **Flask**, **CherryPy**, **Pyramid**, **Django** and **web2py** for web development.

- **Python offers many choices for web development**
  - Frameworks such as **Django** and **Pyramid**.
  - Micro-frameworks such as **Flask** and **Bottle**.
  - Advanced content management systems such as **Plone** and **django CMS**.
- **Python's standard library supports many internet protocols**
  - **HTML** and **XML**
  - **JSON**
  - **E-mail** processing
  - Support for **FTP**, **IMAP**, and other Internet protocols
  - Easy-to-use socket interface
- **The package Index has more libraries**
  - **Requests**, a powerful HTTP client library.
  - **Beautiful Soup**, an HTML parser that can handle all sorts of HTML.
  - **Feedparser** for parsing RSS/Atom feeds.
  - **Paramiko**, implementing the SSH2 protocol.
  - **Twisted Python**, a framework for asynchronous network programming.

Source: *https://www.python.org/about/apps/*

Flask Web Application Framework

- Flask is a lightweight **WSGI** (Web Server Gateway Interface) web application framework. It is designed to make *getting started* quick and easy, with the ability to scale up to complex applications.
- It began as a simple wrapper around **Werkzeug** (Werkzeug WSGI toolkit) and **Jinja** (Jinja template engine) and has since then become one of the most popular Python web application frameworks.
- Flask offers suggestions, but does not enforce any dependencies or project layouts. It is up to the developer to choose the tools and libraries they want to use.
- There are many extensions provided by the community that make adding new functionality easy.

Installing Flask using `pip` command

- `pip install -U Flask`

Source: *https://pypi.org/project/Flask/*

A Simple Example


```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return "Hello World"

if __name__ == '__main__':
    app.run()
```

- Importing flask module in the project is mandatory. Our WSGI application is an object of Flask class.
- Flask constructor takes the name of **current module** (`__name__`) as argument.

- The `route()` function of the Flask class is a decorator, which tells the application which URL should call the associated function.
  `app.route(rule, options)`
  - The **rule** parameter represents URL binding with the function.
  - The **options** is a list of parameters to be forwarded to the underlying Rule object.
- In the above example, '/' URL is bound with `hello_world()` function. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.
- Finally the `run()` method of Flask class runs the application on the local development server.

Source: *https://www.tutorialspoint.com/flask/flask_application.htm*

A Simple Example (2)

`app.run(host, port, debug, options)`

- `host`: Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to '0.0.0.0' to have server available externally
- `port`: Defaults to 5000
- `debug`: Defaults to false. If set to true, provides a debug information
- `options`: To be forwarded to underlying Werkzeug server.

Executing the code:

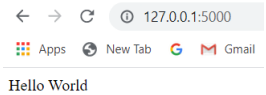
- `Python Hello.py`

Output:

A message in Python shell:

- `* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)`

Open the above URL (127.0.0.1:5000) in the browser



Python: Flask

- Consider the table **Candidate** (in PostgreSQL) as shown below:

```
                    Table "public.candidate"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 cno    | integer               |           | not null |
 name   | character varying(30) |           |          |
 email  | character varying(30) |           |          |
Indexes:
    "candidate_pkey" PRIMARY KEY, btree (cno)
```

- Code segment in Python:

```
from flask import Flask, \
    render_template, request
import psycopg2

app = Flask(
    __name__,
    template_folder='templates'
)
#functions to be added here for
#different actions
```

```
if __name__ == '__main__':
    # Run the Flask app
    app.run(
        host='127.0.0.1',
        debug=True,
        port=5000
    )
```

Python: Flask (2)

- Source code for `index.html`:

```
<!DOCTYPE html>
<html>
<head>
<title>Candidate Email Database</title>
</head>
<body>
    <h2>Candidate Email Database</h2>
    <a href="/add">Add Email</a><br><br>
    <a href="/viewall">View Email</a>
</body>
</html>
```

- Source code for rendering `index.html` and `add.html` pages:

```
@app.route("/")
def index():
    return render_template("index.html");

@app.route("/add")
def add():
    return render_template("add.html")
```

Python: Flask (3)

Python: Flask (4)

- Source code for `add.html` (in HTML):

```
<!DOCTYPE html>
<html>
<head>
    <title>Add Email</title>
</head>
<body>
    <h2>Email Information</h2>
    <form action = "/savedetails" method="post">
    <table>
        <tr><td>CNO</td><td><input type="text" name="cno" required></td></tr>
        <tr><td>Name</td><td><input type="text" name="name" required></td></tr>
        <tr><td>Email</td><td><input type="text" name="email" required></td></tr>
        <tr><td><input type="submit" value="Submit"></td></tr>
    </table>
    </form>
</body>
</html>
```

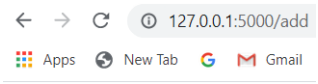
Python: Flask (5)

- savaDetails() function for add.html (in Python):

```
@app.route("/savedetails",methods = ["POST"])
def saveDetails():
    cno = request.form["cno"]
    name = request.form["name"]
    email = request.form["email"]
    conn = None
    try:
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432") # connect to the PostgreSQL database
        cur = conn.cursor()     # create a new cursor
        cur.execute("INSERT INTO Candidate (cno, name, email) \
            VALUES (%s, %s, %s)", (cno, name, email)) # execute the INSERT statement
        conn.commit() # commit the changes to the database
        cur.close() # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        render_template("fail.html")
    finally:
        if conn is not None:
            conn.close()    # close the connection
    return render_template("success.html")
```

Python: Flask (6)

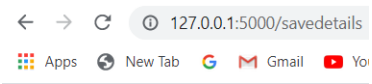
`http://127.0.0.1:5000/add`



Mobile Information

CNO
Name
Email
Submit

`http://127.0.0.1:5000/savedetails`



Data Successfully Added

Go Home

Python: Flask (7)

- `viewAll()` function for `viewall.html` (in Python):

```
@app.route("/viewall")
def viewAll():
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432")
        cur = conn.cursor()     # create a new cursor
        # execute the SELECT statement
        cur.execute("SELECT cno, name, email FROM Candidate")
        results = cur.fetchall()   # fetches all rows of the query result set
        cur.close()             # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        render_template("fail.html")
    finally:
        conn.close()            # close the connection
    return render_template("viewall.html",rows = results)
```

Python: Flask (8)

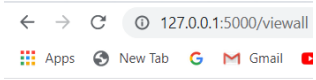
- Source code for `viewall.html` (in HTML):

```
<!DOCTYPE html>
<html>
<head>
    <title>Email List</title>
</head>
<body>

<h3>Email List</h3>
<table border=5>
    <tr>
        <th>CNO</td><th>Name</td><th>Email</td>
    </tr>
    {% for row in rows %}
        <tr>
            <td>{{row[0]}}</td> <td>{{row[1]}}</td> <td>{{row[2]}}</td>
        </tr>
    {% endfor %}
</table>
<br><br>
<a href="/">Go Home</a>
</body>
</html>
```

Python: Flask (9)

http://127.0.0.1:5000/viewall



Email List

CNO	Name	Mobile
101	Ishaan	ishaan@mymail.com
102	Jairaj	Jairaj@mymail.com
103	Ananya	ananya@mymail.com
104	Barkha	barkha@myemail.com
105	Piyush	piyush@mymail.com

Go Home

Module Summary

- Learnt how to access PostgreSQL database from Python
- Learnt to build Python Web Application with PostgresSQL and Flask

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".