Informe Teórico-Técnico sobre construcción y funcionamiento del módulo spgpy.

Título: Optimización de la conexión de la Base de Datos PostgreSQL con Python3. Modulo spgpy.

Autor: Jorge Felix Martinez Pazos

Brigada 1302, Facultad 1, Universidad de las Ciencias Informáticas, Carretera San Antonio de los Baños, km 2 ½, La Lisa, La Habana, Cuba

Septiembre de 2022

Resumen: PostgreSQL es un gestor que trabaja con bases de datos relacionales y que está orientado a objetos. Se trata de un programa de código abierto u open source, es decir, no está bajo el control de ninguna compañía particular, sino que cuenta con una comunidad de desarrolladores que trabajan en mejorar el programa de forma desinteresada. Hemos desarrollado un módulo de Python spgpy, haciendo uso de otras librerías como pscopg2 para facilitar la conexión del lenguaje con la base de datos PostgreSQL y usar las funciones principales del mismo.

Palabras clave: PostgreSQL; módulo; Python; spgpy; pscopg2;

Abstract: PostgreSQL is a manager that works with relational databases and is object-oriented. It is an open source program, that is, it is not under the control of any particular company, but has a community of developers who work on improving the program selflessly. We have developed a Python spgpy module, using other libraries such as pscopg2 to facilitate the connection of the language with the PostgreSQL database and use the main functions of it.

Keywords: PostgreSQL; module; Python; spgpy; pscopg2;

Introducción

El objetivo de este proyecto es, crear un modulo de python que permita la reutilización a gran escala de código y así evitar que los desarrolladores que usen esta tecnología tengan que reescribir el código por ellos mismos una y otra vez. El modulo spgpy permite instanciar un objeto de tipo Coneccion y usar los métodos ya definidos en cualquier proyecto que se esté desarrollando usando Python como lenguaje de programación y PostgreSQL como base de datos.

Problemática: La reutilización de código es uno de los aspectos que puede ser más beneficioso en un proyecto de desarrollo de software, ya que por un lado permite ahorrar costes y por otro ahorra quebraderos de cabeza, ya que gran parte de los problemas que se pudieron producir en su desarrollo ya han sido resueltos. La librería psycopg2 con el cual se trabaja para la conexión de la base de datos PostgreSQL con el lenguaje

Python tiene numerosas funciones, algunos de las cuales se pueden optimizar aún mas, de tal manera que se potencie la reutilización de código.

Problema de investigación: ¿Como contribuir a la optimización de la libraría psycopg2?

Tareas de Investigación:

1-Conocer a fondo la librería psycopg2

2-Estudiar y profundizar en las prácticas de programación profesional en Python PEP8.

3-Investigar como desplegar nuestro modulo en Python Package Index o PyPI.

Objetivo General: Crear un modulo que permita la reutilización de código en el entorno del trabajo de Python con base de datos PostgreSQL siguiendo practicas de programación profesionales como el uso de PEP8 en la escritura del código. Proporcionar una visión de como trabajar con el modulo spgpy, especificando cada una de las funcionalidades y estructura del mismo.

Resultados Esperados: Como culminación de este proyecto se espera la correcta creación de un módulo o librería de Python desarrolladas con prácticas profesionales de programación para optimizar las conexiones con base de datos en función principalmente a la reutilización de código y desplegarlo en Python Package Index o PyPI para su uso por terceros. Además se espera el incremento de conocimiento en practicas de programación avanzadas como son Python Package Index o PyPI y la guía de codificación PEP8.

Desarrollo

PostgreSQL es una de los Sistema Gestores de Base de Datos (SGBD) más usados hoy en día, debido a su estructura de open source. Muchos de los desarrolladores que trabajan con Python como lenguaje y PostgreSQL como SGBD tiene que reescribir constantemente los algoritmos para iniciar la conexión de lenguaje con la base de datos o para usar alguna de las funcionalidades principales de PostgreSQL como: SELECT, UPDATE, DELETE or DROP. Las anteriormente nombradas son sintaxis simples las cuales hemos recopilado en nuestro modulo spgpy.

El primer paso durante el desarrollo de nuestro software es la creación del entorno virtual y la instalación de todas las herramientas y requisitos necesarios.

Stage #1: Module Docstrig

```
"""A Python Module for PostgreSQL Using psycopg2 librari

spgpy is a small python module for use the main function of
PostgreSQL Database faster.
This is a Beta Version of spgpy.

.. _PostgreSQL: https://www.postgresql.org/
.. _Python: https://www.python.org/

@autor: Wise_George
feedback => gmail: binaryteck@gmail.com

"""
```

En esta sección se plasman a grandes rasgos las características y funcionalidades del módulo.

Stage #2: Comands

```
#Commands:
    #=> init_conection(self,host: str, database: str, user: str, password: str, port: int = 5432):
    #=> init_cursor(self):
    #=> close_conection_cursor(self):
    #=> select_one_from_table(self, table: str):
    #=> select_all_from_table(self, table: str):
    #=> delete_table(self, table: str):
    #=> delete_from_table(self, table: str, where_algoritm: str):
    #=> update_table(self, table: str, set_algoritm: str, where_algoritm: str):

```

Métodos Principales del modulo, parámetros necesarios y su tipo de datos.

Stage #3: Imports

```
25    import psycopg2
26    from datetime import datetime as dt
```

Módulos necesarios para el correcto funcionamiento de spgpy

Stage #4: Decorators

```
30    #Decorators
31    def Time_Lapse(function):
32        def Wrapper(*args, **kwargs):
33            initial_Time = dt.now()
34            function(*args, **kwargs)
35            final_Time = dt.now()
36            time_lapse = (final_Time - initial_Time).microseconds
37            print("RunTime: {} Microseconds\n".format(time_lapse))
38
39        return Wrapper
40
41    def Raise_Exception(function):
42        def Wrapper(*args, **kwargs):
43            try:
44                function(*args, **kwargs)
45            except Exception as error:
46                print("Exception Caught: {}".format(error))
47
48        return Wrapper
```

- Time_Lapse muestra en pantalla el Runtime (microsegundos) de cualquier método.

- Raise_Exception maneja las excepciones de manera global en cualquier método.

Stage #5: Clase Connection

```python
class Connection(object):
    """
    Connection Class to use with psycopg2.
    Execute Main Functions of Simple postgreSQL Faster.
    Create a Connection object also start a new connection and a new cursor.
    """

    def __init__(self,host: str, database: str, user: str, password: str, port: int = 5432) -> None:

        #Conection Settings
        self.host = host
        self.database = database
        self.user = user
        self.password = password
        self.port = port
        self.connection = self.init_conection(self.host, self.database, self.user, self.password, self.port)
        self.cursor = self.init_cursor()
```

Definición de la Clase Connection, su constructor recibe como parámetros: host, database, user, password y port (default value)

Se inician como atributos de la clase además de los parámetros self.connection y self.cursor, atributos que permitirán la interacción con la base de datos.

Stage #5: Funciones Principales (Iniciar Conexión & Iniciar Cursor)

- ***init_conection(self, ags):*** toma como parámetros los datos de la base de datos a la cual se quiere conectar, inicia la conexión con la base de datos y devuelve un objeto tipo conexión usando el modulo pscopg2.
- ***Init_cursor(self):*** no toma parámetros, crea un objeto tipo cursor usando el objeto tipo conexión obtenido de **init_conection()** y lo devuelve.
- Estas dos funciones son iniciadas en el constructor de la clase y asignadas a un atributo cada una respectivamente, por tanto, tenemos como atributo de la clase desde iniciarla **self.conection** and **self.cursor**, para poder trabajar con sus funciones eventualmente.
    - Posibles excepciones son tratadas manualmente en ambos métodos sin hacer uso del decorador @Raise_Exception.

```python
    def init_conection(self,host: str, database: str, user: str, password: str, port: int = 5432):
        """
        Initializing Connection:
            Args: (host, database, user, password, port)
            Return: connection object
        """
        try:
            connect = psycopg2.connect(
            host = host,
            database = database,
            user = user,
            password = password,
            port = port
        )
            print(" 🔌 Connected Succesfully to {} Database 🔌 ".format(database))
        except Exception as error:
            print("Error Connecting to {} Database".format(database))
        return connect


    def init_cursor(self):
        """
        Initializing Cursor:
            Args: (None)
            Return: cursor object
        """
        try:
            my_cursor = self.connection.cursor()
            print(" 🔌 Cursor Created Successfully 🔌 ")
            return my_cursor
        except Exception as error:
            print("Error Initializing Cursor")
```

Stage #6: Métodos de Acceso y Consultas a las Base de Datos.

- Close_conection_cursor(self): Como Buena practica de programación al terminar el trabajo con una base de datos hay que cerrar la conexión, este método nos facilita esa operación. Fig 1.7
- Select_one_from_table(self,table): Ejecuta la consulta SELECT en PostgreSQL y devuelve una tuple con el primer elemento de dicha consulta. Recibe como parámetro la tabla o la vista que se desea consultar Fig 1.8
- Select_all_from_table(self,table): Ejecuta la consulta SELECT en PostgreSQL y devuelve una lista con todos los elementos de dicha consulta. Recibe como parámetro la tabla o la vista que se desea consultar Fig 1.8

- Delete_table(self,table): Elimina una tabla recibida por parámetro de la base de datos. Fig 1.9
- Delete_from_table(self,table,where_algoritm): Elimina de una table recibida por parametro todas las instancias donde se cumpla el where_algoritm recibido también por paramero. Fig 1.9
    - Ejemplo: delete_from_table('user','nombre = Jorge')
- Update_table(self, table, set_algorimt, where_algoritm): Actualiza una tabla recibida por parámetro, con los valores entrados en el parámetro set_algorimt en las intancias donde se cumpla el where_algorimt. Fig 2.1
    - Ejemplo: update_table('user','edad = 23','nombre = Jorge')

Figura 1.7:

```
106     @Raise_Exception
107     def close_conection_cursor(self):
108         """Close cursor and connection:
109                 Args: None
110                 Return: None
111             **Always execute close_conection_cursor
112               At the end of your code**
113         """
114         if self.cursor is not None:
115             self.cursor.close()
116             print(" Cursor Closed Successfully ")
117         if self.connection is not None:
118             self.connection.close()
119             print(" Connection Closed Successfully \n")
```

Figura1.8

```
121     @Raise_Exception
122     def select_one_from_table(self, table: str):
123         """
124         Select one from table:
125         Also Select one from view:
126             Args: table
127             Return: __result(tuple)
128         """
129 
130         self.cursor.execute("SELECT * FROM {}".format(table))
131         __result = self.cursor.fetchone()
132         print("\n==>{} table: ".format(table))
133         print(__result)
134         print("")
135         return __result
136 
137     @Raise_Exception
138     def select_all_from_table(self, table: str):
139         """
140         Select all from table:
141         Also Select all from view:
142             Args: table
143             Return: __result(list(tuples))
144         """
145 
146         self.cursor.execute("SELECT * FROM {}".format(table))
147         __result = self.cursor.fetchall()
148         print("\n==>{} table: ".format(table))
149         for i in range (len(__result)):
150             print("{}=>>{}".format(i+1, __result[i]))
151         print("\n")
152         return __result
```

Figura 1.9:

```python
    @Raise_Exception
    def delete_table(self, table: str) -> None:
        """
        Delete Table:
            Args: table
            Return: None
        """
        self.cursor.execute("DROP TABLE {}".format(table))
        print("Table {} Deleted Successfully".format(table))
        self.connection.commit()

    @Raise_Exception
    def delete_from_table(self, table: str, where_algoritm: str):
        """
        Delete from Table:
            Args: table(str), where_algoritm(str)
            Return: None
        """

        self.cursor.execute("SELECT * FROM {} WHERE {}".format(table, where_algoritm))
        __result = self.cursor.fetchall()
        if len(__result) < 1:
            print("⁉ Error Trying to Execute:")
            print("DELETE FROM {} WHERE {}".format(table, where_algoritm))
            print("Instance not Found\n")

        else:
            self.cursor.execute("DELETE FROM {} WHERE {}".format(table, where_algoritm))
            self.connection.commit()
            print("⚠ Instance deleted successfully from {} WHERE {}\n".format(table, where_algoritm))
```

Figura 2.0:

```python
    @Raise_Exception
    def update_table(self, table: str, set_algoritm: str, where_algoritm: str):
        """
        Update Table:
            Args: table(str),set_algoritm(str), where_algoritm(str)
            Return: None
        """

        self.cursor.execute("SELECT * FROM {} WHERE {}".format(table, where_algoritm))
        __result = self.cursor.fetchall()
        if len(__result) < 1:
            print("⁉ Error Trying to Execute:")
            print("DELETE FROM {} WHERE {}".format(table, where_algoritm))
            print("Instance not Found\n")

        else:
            self.cursor.execute("UPDATE {} SET {} WHERE {}".format(table,set_algoritm, where_algoritm))
            self.connection.commit()
            print("⚠️ Instance updated successfully")
            print("UPDATE {} SET {} WHERE {}\n".format(table,set_algoritm, where_algoritm))
```

Como usar el modulo spgpy

- Se importa la clase Connection del módulo spgpy.
- Se instancia un objeto tipo Connection dándole los parámetros de la base de datos a conectar.
- Usar las funciones del modulo sobre el objeto creado.
- Al finalizar el uso de spgpy siempre ejecutar el método close_connection_cursor().

INPUT() and OUTPUT()

```python
# How Use spgpy

from spgpy import Connection

my_connection = Connection('127.0.0.1','Teziutlan Exercises','postgres','Jorgito16*')
my_connection.select_all_from_table('usuarios_gmail')
my_connection.select_one_from_table('usuarios_icloud')
my_connection.delete_from_table('transaction', 'id_transaction = 5')
my_connection.update_table('users', 'edad = 15', 'id = 6')
my_connection.close_conection_cursor()

```

```
PROBLEMS  OUTPUT  TERMINAL  JUPYTER  SQL CONSOLE: MESSAGES  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas caracterís

PS D:\Software Development\PythonLearning\Curso-Python> & C:/Users/Wise_Geor
ython/10. Python_Data_Bases/Modulos_PostgreSQL/Testing.py"
 Connected Succesfully to Teziutlan Exercises Database
 Cursor Created Successfully

==>usuarios_gmail table:
1=>>(1, 'George', 'Martinez', 22, 'jorgito16040@gmail.com')
2=>>(4, 'Eduardo', 'Gonsalez', 22, 'eduardjs@gmail.com')
3=>>(6, 'Alejandro', 'Robaina', 15, 'ale@gmail.com')
4=>>(5, 'Edey', 'Gomez', 19, 'edey@gmail.com')
5=>>(10, 'Jessica', 'Hernandez', 19, 'jess@gmail.com')
6=>>(11, 'Jennifer', 'Romero', 23, 'jenn@gmail.com')
7=>>(13, 'Karla', 'Puig', 21, 'darlap@gmail.com')
8=>>(15, 'David', 'Bueno', 27, 'davidb@gmail.com')
9=>>(16, 'Laura', 'Bueno', 30, 'laurabueno@gmail.com')
10=>>(17, 'Arlet', 'Acevedo', 29, 'arleta@gmail.com')
11=>>(18, 'Estefany', 'Martinez', 19, 'stef@gmail.com')
12=>>(19, 'Loraine', 'Martinez', 28, 'lora@gmail.com')
13=>>(20, 'Aliana', 'Martinez', 27, 'aliana@gmail.com')
14=>>(21, 'Claudia', 'Martinez', 30, 'claudia@gmail.com')
15=>>(24, 'Jose', 'Pupo', 22, 'pupojose@gmail.com')
16=>>(25, 'Angel', 'Enrique', 26, 'angel@gmail.com')
17=>>(26, 'Henrry', 'Enrique', 27, 'henry@gmail.com')
18=>>(27, 'Lazaro', 'Gonzales', 30, 'lazaro@gmail.com')
19=>>(29, 'Daniel', 'Fernandez', 22, 'daniel@gmail.com')
20=>>(30, 'Alberto', 'Ganzalez', 47, 'alcf@gmail.com')
21=>>(31, 'Fernanda', 'Fernandez', 19, 'fernandafernandes@gmail.com')
22=>>(45, 'Fernando', 'Rodriguez', 22, 'alain@gmail.com')
23=>>(43, 'Jorge', 'Gonzalez', 22, 'damian@gmail.com')


RunTime: 19131 Microseconds
```

```
==>usuarios_icloud primera entrada:
(2, 'William ', 'Gomez', 22, 'willwolf@icloud.com')

⁉ Error Trying to Execute:
DELETE FROM transaction WHERE id_transaction = 2
Instance not Found

⚠Instance updated successfully
UPDATE users SET edad = 19 WHERE id = 5

🔌Cursor Closed Successfully🔌
🔌Connection Closed Successfully🔌
```

Referencias bibliográficas

https://peps.python.org/pep-0008/

https://docs.python.org/es/3/tutorial/

https://www.psycopg.org/docs/

https://www.postgresql.org/docs/

Conclusiones

Al terminar el proyecto se ha obtenido un software completamente funcional con altos estándares de desarrollo, hemos demostrado en este informe el proceso completo para su utilización, además de que el código del módulo se encuentra comentado, legible y todas sus funciones con sus respectivos docstrings para facilitar la interpretación del mismo, presentamos problemas en desplegar el módulo en Python Package Index PyPI para su instalación mediante el comando: pip install spgpy, el módulo spgpy está en su versión beta y se continuara trabajando en nuevas características y funcionalidades. Descargue spgpy desde Google Drive: https://drive.google.com/file/d/1D4H9zS_y0ZTISFXKjdgCj0nlr1Y1ok09/view?usp=sharing