**Conectando py con postgresql**

Para esto se crea un nuevo proyecto en pycharm, es muy importante que cuente con el entorno virtual correspondiente, para asi incluir librerías externas.

El comando a ejecutar es

pip install psycopg2

para utilizar mysql: pip install mysql-connector

**creando la primera conexión**

luego de un: pip install psycopg2 se importa:

import psycopg2

conn = psycopg2.connect(user='postgres', password='admin', host="localhost", port="5432", database="test\_db\_python")

con el objeto psycopg2 se guarda en una variable la llamada al método **.connect()** con los parámetros necesarios para la conexión a la DB creada anteriormente.

Para comenzar a hacer consultas es necesario crear un “cursor”.

cursor = conn.cursor()

con este objeto cursor, ahora podemos utilizar el método **.execute()**

sql = 'SELECT \* from persona order by id\_persona asc'  
cursor.execute(sql)

por ultimo podemos guardar estos resultados en una variable e imprimirlos:

registros = cursor.fetchall()  
for e in registros:  
 print(e)

este método fetchall() nos devuelve un objeto de clase “lista” que dentro contiene una tupla.

Luego de utilizar la conexión y el cursor, es necesario cerrarles.

cursor.close()  
conn.close()

**uso de with**

al igual que con el manejo de archivos, puede recurrirse al bloque **with** para anidar las acciones que se realicen con una conexión a bases de datos o con un cursor. A pesar de esto, es necesario cerrar manualmente la conexión, por lo tanto, pueden englobarse ambos bloques dentro de un try-finally.

try:  
 conn = psycopg2.connect(user='postgres', password='admin', host="localhost", port="5432", database="test\_db\_python")  
  
 with conn:  
 with conn.cursor() as cursor:  
 sql = 'SELECT \* FROM persona ORDER BY id\_persona ASC'  
 cursor.execute(sql)  
 registros = cursor.fetchall()  
  
 for e in registros:  
 print(e)  
finally:  
 cursor.close()  
 conn.close()

**operador fetchone()**

el método fetchone funciona igual que fetchall() solo que retorna una tupla con los datos de un solo registro.

**Placeholders en sentencias sql**

Para evitar inyección sql, pueden utilzarse placeholders como en PHP.

El carácter que se usa es %s y los argumentos se envían en la llamada al método execute como segundo parámetro en forma de tupla.

En caso de que el placeholder sea un único valor, la forma de convertirlo en tupla sería encerrarlo entre () y colocar una , luego del valor:

sql = 'SELECT nombre FROM persona WHERE id\_persona = %s'  
placeholder\_value = 2  
cursor.execute(sql, (placeholder\_value, ))  
registros = cursor.fetchone()

de esta forma se rescata el valor nombre del registro con id\_persona = 2 (que al ser retornado de fetchone, es en realidad una tupla)

**SENTENCIAS**

Con respecto a las sentencias no se detalla demasiado ya que son como en mysql

Se remarca el uso del operador IN para determinar ciertos valores, por ejemplo:

"SELECT \* FROM persona WHERE id\_persona IN (1,3)"

Esta sentencia devolvería todas las columnas de la tabla persona en los id\_persona 1 y 3

Importante recordar la sintaxis de update:

"UPDATE persona set nombre=%s, apellido=%s, email=%s WHERE id\_persona = %s"

%s -> son los placeholders donde deberían ir los valores

**REALIZANDO CAMBIOS EN LA DB**

Cuando se realizan cambios en la db, es necesario invocar a la variable conexión y ejecutar el método **.commit()**

**INSERTANDO VARIOS REGISTROS**

Cuando se utiliza el bloque with—el .commit() está implicito.

Si durante la ejecución de este bloque ocurre algun error con alguno de los registros, se ejecuta un **rollback**. Este concepto consiste en “dar marcha atrás” a la inserción y omitir los cambios.

Para insertar varios registros, se utilizan placeholders, pero al momento de pasar la tupla con los valores del placeholder se da como argumento **una tupla de tuplas.**

Es muy importante que los datos sean de este tipo y no listas.

Con este código podríamos insertar varios registros:

sql\_varios = "INSERT INTO persona (nombre, apellido, email) VALUES (%s, %s, %s)"  
  
valores = (('mariano', 'gonzalez', 'mariangon@mail.com'),  
 ('miriam', 'williams', 'mirw@hotmail.com'),  
 ('gino', 'pereyra', 'ginop@yahoo.com')  
 )  
cursor.executemany(sql\_varios, valores)  
  
print(f'registros insertados {cursor.rowcount}')

la variable **valores** es una tupla de tuplas, por tanto la secuencia es correcta. La principal diferencia con códigos anteriores es que se utiliza el método **executemany()**

**ACTUALIZANDO VARIOS REGISTROS**

Para actualizar varios, se procede igual que para insertar solo que hay que modificar la consulta SQL:

sql\_varios = "UPDATE persona set nombre =%s , apellido = %s, email =%s WHERE id\_persona = %s "  
  
valores = (('ubaldo ', 'acosta', 'ing\_ubaldo@mail.com', 6),  
 ('ricardo', 'mollo', 'aplanador@hotmail.com', 7),  
 ('alberto', 'spinacca', 'luifa@yahoo.com', 15)  
 )  
  
cursor.executemany(sql\_varios, valores)  
  
print(f'registros actualizados {cursor.rowcount}')

Igual que en el caso anterior, pueden verse los registros afectados, como el ->affected\_rows en php con **.rowcount**

**ELIMINANDO VARIOS**

Para eliminar varios registros puedo utilizar este método, que recibe una tupla de tuplas, ya que el valor del placeholder es una tupla como tal:

sql\_varios = "DELETE FROM persona WHERE id\_persona IN %s "  
  
#utilizando IN  
entrada= input('ingrese ids detallados para eliminar')  
tupla = (tuple(entrada.split(',')),)  
#print(tupla)  
cursor.execute(sql\_varios, tupla)  
  
print(f'registros eliminados {cursor.rowcount}')

o esta forma:

#utilizando where id\_persona =  
#mysql = "DELETE FROM persona WHERE id\_persona = %s"  
#val = (  
# (5,),  
# (14,),  
# (222,)  
#)  
#cursor.executemany(mysql, val)  
#print(f'registros eliminados {cursor.rowcount}')

En donde cambio la estructura de la sentencia SQL, en este caso ejecutando varias en lugar de una.