**POOL DE CONEXIONES A LA DB**

Un objeto que administra los objetos de conexión a la DB. Pueden existir más de un pool en una aplicación.

Cada pool “maneja” cierta cantidad de objetos de conexión. Cuando un usuario finaliza de usar un objeto, este se libera y se regresa al pool para ser usado por otro.

**MODIFICACIONES**

En la sección anterior se utilizaba la clase ConexionClase para manejar la conexión. Para implementar el pool es necesario modificar el código.

Para empezar, el objeto conexión y el objeto cursor se implementan en otra clase. También los métodos **get\_conexion** y **get\_cursor** se ven modificados.

La importación se realiza de la sig manera:

from psycopg2 import pool

Todo pool de conexiones debe manejar un número máximo y mínimo de objetos de conexión. Esto se resuelve con atributos de clase que se pasaran como argumento al momento de la creación del pool.

\_MIN\_CON = 1  
\_MAX\_CON = 5  
\_pool = None

@classmethod  
def get\_pool(cls):  
 if cls.\_pool is None:  
 try:  
 cls.\_pool = pool.SimpleConnectionPool(cls.\_MIN\_CON, cls.\_MAX\_CON, host=cls.\_HOST, user= cls.\_USERNAME, password= cls.\_PASSWORD, port= cls.\_DB\_PORT, database=cls.\_DATABASE)  
 log.debug(f'Creacion de pool exitosa {cls.\_pool}')  
 return cls.\_pool  
 except Exception as e:  
 log.error(f'Error al obtener el pool {e}')  
 else:  
 return cls.\_pool

El método get\_pool retorna el objeto de tipo pool o lo crea en caso de que no exista.

Ahora el método para obtener una conexión es el siguiente:

@classmethod  
def get\_connnection\_pool(cls):  
 conexion = cls.get\_pool().getconn()  
 log.debug(f'Objeto conexion creado correctamente {conexion}')  
 return conexion

Se guarda en la variable **conexión** una llamada al método get\_pool() y se utiliza el método propio de los objetos de tipo **pool** **.getconn()**

Para comprobar que se creen la cantidad de objetos de conexión, es necesario instanciar la clase ConexionClase e invocar al método **.get\_connection\_pool()** en cada una de estas instancias, siempre y cuando no supere el numero máximo de instancias permitidas por el pool.

**AGREGANDO METODOS**

@classmethod  
def liberar\_conexion(cls, conexion):  
 cls.get\_pool().putconn(conexion)  
 # el metodo .putconn() "pone" en el pool el objeto de conexion indicado por parametro  
 log.debug(f'conexion {conexion} regresada al pool.')  
  
@classmethod  
def cerrar\_conexiones(cls):  
 cls.get\_pool().closeall()

Los métodos liberar conexión se encargan de “liberar” un objeto de tipo conexión “poniendole” nuevamente en el pool de conexiones. Es necesario que se pase como argumento la instancia de tipo conexión a liberar.

Por ultimo, el método **cerrar\_conexiones** cierra todas las conexiones del pool.

**CREANDO EL CURSOR**

Lo que sigue es delegar la creación y administración del cursor a otra clase:

Esta clase, define los métodos \_\_init\_\_ para instanciar la clase y se espera utilizar con el contexto **with**, por tanto es necesario redefinir los métodos \_\_enter\_\_ y \_\_exit\_\_ que son a los que se recurren al momento de abrir y cerrar el recurso respectivamente.

class CursorPool:  
  
 def \_\_init\_\_(self):  
 self.\_conexion = None  
 self.\_cursor = None  
  
 def \_\_enter\_\_(self):  
 # crear la conexion y el cursor  
 log.debug(f'Iniciando el metodo with \_\_enter\_\_')  
 self.\_conexion = ConexionClase.get\_connnection\_pool()  
 self.\_cursor = self.\_conexion.cursor()  
 # esta ultima linea es la que hace que funcione el execute  
 return self.\_cursor  
  
 def \_\_exit\_\_(self, tipo\_excepcion, valor\_excepcion, detalle\_excepcion):  
 log.debug(f'ejecutando \_\_exit\_\_')  
  
 # se evalua si hay contenido en la excepcion, o sea, si ocurrio algun error  
 if valor\_excepcion:  
 self.\_conexion.rollback()  
 log.error(f'Error {tipo\_excepcion} {detalle\_excepcion}')  
 else:  
 self.\_conexion.commit()  
 log.info(f'commit realizado a la Database')  
 # realizar el commit y liberar la conexion

como init, los atributos conexión y cursor se inician como None.

Cuando se inicia el método \_\_enter\_\_() se cambia el valor de estos atributos, utilizando un llamado a la clase ConexionClase (modificada anteriormente) para conseguir los objetos de tipo connection. Por último se retorna el valor en el atributo **.\_cursor**

Esto significa que ahora, la instancia CursorPool() pasa a ser un objeto de clase <cursor> cuando antes era un objeto de clase <CursorPool>. Con esto, podemos recurrir al método .execute().

Con el método \_\_exit\_\_() llegamos a la parte final. Este método trae como argumentos el tipo de excepción, el valor y un detalle EN CASO DE QUE ESTA HALLA EXISTIDO. Si la excepción existe esto significa que ocurrió un error.   
Con esto podemos comprobar y decidir si realizamos un **rollback** o un **commit**

**TESTEANDO CLASE CURSORPOOL()**

Esta sería la forma de utilizar el contexto with con la clase CursorPool()

with CursorPool() as my\_cursor:  
 print(CursorPool())  
 print(my\_cursor)  
 my\_cursor.execute('SELECT \* FROM PERSONA')  
 regs = my\_cursor.fetchall()  
  
 for e in regs:  
 log.debug(e)

**ULTIMAS MODIFICACIONES EN LA CLASE PERSONA DAO**

Esta clase es la encargada de crear objetos de tipo persona con los datos de la DB e interactuar con la misma.

Es necesario modificar el código anterior para usar el contexto with con la clase CursorPool() como por ejemplo de la siguiente forma:

def insertar(cls, persona):  
 if isinstance(persona, Persona):  
 log.debug(f'persona a insertar {persona}')  
 try:  
 with CursorPool() as cursor:  
 cursor.execute(cls.\_INSERTAR, (persona.nombre, persona.apellido, persona.email))  
 reg\_modificados = cursor.rowcount  
 if reg\_modificados > 0:  
 log.debug(f'objeto insertado {persona}')  
 log.debug(f'modificacion exitosa, {reg\_modificados} agregados')  
 else:  
 log.warning('Por algun motivo no hubo modificaciones en la Base de Datos')  
  
 except Exception as e:  
 log.error(f'Error al insertar. Error: {e}')  
 log.error('Saliendo del sistema'.center(50, '-'))  
 sys.exit()  
 else:  
 log.error(f'Parametro introducido: {persona} no es válido para esta operacion')  
 sys.exit()

CABE RESALTAR QUE FUE NECESARIO INTRODUCIR LOS COMMIT() Y ROLLBACK() YA QUE EL CONTEXTO WITH SOBRE EL QUE ESTAMOS TRABAJANDO ES SOBRE UNA CLASE CursorPool() Y LA MISMA FUE DEFINIDA POR NOSOTROS. POR TANTO A DIFERENCIA DE USAR UN CONTEXTO WITH CON UN ARCHIVO DE CLASE CONEXIÓN O CURSOR, NECESITAMOS HACER ESTO DE FORMA MANUAL EN LA DECLARACION DE LOS METODOS \_\_ENTER\_\_ Y \_\_EXIT\_\_