




Guía: Obtención de conexiones desde un Pool en Python con `psycopg_pool`

Introducción

En esta lección vamos a implementar el método `obtener_conexion` utilizando la clase `ConnectionPool` del módulo `psycopg_pool`. Veremos cómo se comportan las conexiones cuando se reutilizan correctamente con `with` y cómo se agotan cuando no se liberan.

1. Implementación del Pool de Conexiones

 Archivo: `capa_datos_persona/conexion.py`

En esta clase creamos un pool de conexiones con un límite de 5 objetos activos a la vez:

```
from logger_base import log
from psycopg_pool import ConnectionPool

class Conexion:
    _pool = ConnectionPool(
        conninfo="host=127.0.0.1 port=5432 dbname=test_db user=postgres password=admin",
        min_size=1,
```

```

        max_size=5,
        timeout=5
    )

    @classmethod
    def obtener_conexion(cls):
        return cls._pool.connection()

```

🔍 Esta versión usa `cls._pool.connection()`, que **libera automáticamente la conexión** si se usa dentro de un bloque `with`.

2. Caso 1: Uso correcto con `with` ✅

Al usar `with`, la conexión se libera automáticamente después de usarse, por lo que el pool puede servir más de 5 peticiones sin error:

```

if __name__ == '__main__':
    Conexion._pool.check()
    log.debug('Pool de conexiones verificado correctamente.')

    # Caso con with, donde se reutilizan las conexiones
    for i in range(10): # Pide 10 conexiones, pero se liberan automáticamente
        with Conexion.obtener_conexion() as conexion:
            log.debug(f'Conexión #{i + 1} abierta y cerrada: id={id(conexion)}')

    Conexion._pool.close()
    log.debug('Pool de conexiones cerrado correctamente.')

```

✂ Aunque se soliciten 10 conexiones, se reutilizan correctamente gracias al contexto `with`.

3. Caso 2: Error si no se liberan ❌

Si usamos `getconn()` sin liberar con `putconn()`, se lanzará un error al superar el máximo:

```

@classmethod
def obtener_conexion(cls):
    return cls._pool.getconn() # Para probar el error, cambiar temporalmente esta línea

```

Y luego ejecutar:

```

if __name__ == '__main__':

```

```
Conexion._pool.check()
log.debug('Pool de conexiones verificado correctamente.')
conexiones = []
try:
    for i in range(6): # intenta obtener más de 5 sin liberar
        conexion = Conexion.obtener_conexion()
        conexiones.append(conexion)
        log.debug(f'Conexión #{i + 1} obtenida: id={id(conexion)}')
except Exception as e:
    log.error(f'Error al obtener conexión #{i + 1}: {e}')

# Liberar conexiones manualmente
for conexion in conexiones:
    Conexion._pool.putconn(conexion)

Conexion._pool.close()
log.debug('Pool de conexiones cerrado correctamente.')
```

🧠 Esto ayuda a comprobar que el límite del pool está funcionando correctamente.

✅ Conclusión

- Con `cls._pool.connection()` + `with`, las conexiones se **reutilizan automáticamente**.
 - Con `cls._pool.getconn()`, debes usar `putconn()` manualmente, y el **pool se agota si no liberas**.
 - Es muy útil para mejorar el rendimiento y manejo de recursos en aplicaciones que acceden frecuentemente a la base de datos.
-

🌟 Sigue adelante con tu aprendizaje 🚀, ¡el esfuerzo vale la pena!

¡Saludos! 🙌

Ing. Marcela Gamiño e Ing. Ubaldo Acosta

Fundadores de [GlobalMentoring.com.mx](https://www.globalmentoring.com.mx)