



Manejo de Pool de Conexiones en Python con PostgreSQL

Introducción

En esta lección aprenderemos cómo optimizar el acceso a la base de datos PostgreSQL utilizando un **Pool de Conexiones**, gracias al módulo `psycopg_pool`.

Esto permite reducir el costo de crear conexiones individuales y mejora el rendimiento y la escalabilidad de tu aplicación.

Paso 1: Instalar el módulo `psycopg_pool`

Desde la versión 3, el módulo `psycopg` **ya no incluye el pool por defecto**, por lo que debemos instalarlo explícitamente con esta línea:

```
pip install "psycopg[binary,pool]"
```

¿Por qué se instala por separado?

La versión moderna `psycopg3` ha modularizado su funcionalidad.

Esto permite instalar solo lo que se necesita (por ejemplo, solo la parte binaria o solo el pool), lo que mejora la flexibilidad y reduce el tamaño del entorno.

✓ Al instalar con `[binary,pool]`, incluimos:

- La versión binaria optimizada (más rápida).
- El módulo `psycopg_pool` necesario para el manejo eficiente del pool de conexiones.

🔧 Paso 2: Eliminar la lógica de conexión anterior

📁 Archivo: `capa_datos_persona/conexion.py`

Antes, gestionábamos conexiones de forma manual, con código similar a este (ya no se usará):

```
_conexion = None
_cursor = None

@classmethod
def obtener_conexion(cls):
    ...
```

Todo eso lo reemplazaremos por un pool moderno más eficiente.

⚙️ Paso 3: Implementar el Pool de Conexiones

📁 Archivo trabajado: `capa_datos_persona/conexion.py`

🌟 Descripción

La clase `Conexion` ahora administrará un **pool de conexiones** utilizando `ConnectionPool` de `psycopg_pool`.

🔍 Código final trabajado:

```
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()

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


    with Conexion.obtener_conexion() as conexion:
        log.debug(f'Conexión abierta exitosamente: {conexion.info.dsn}')
    Conexion._pool.close() # cerrar el pool manualmente


```

Explicación paso a paso

- `ConnectionPool`: Objeto que mantiene un conjunto de conexiones abiertas y listas para usarse.
 - `min_size=1`: Al menos una conexión estará siempre disponible.
 - `max_size=5`: Límite máximo de conexiones simultáneas.
 - `timeout=5`: Tiempo máximo de espera para obtener una conexión.
 - `obtener_conexion()`: Obtiene una conexión del pool sin necesidad de configurarla manualmente.
 - El uso de `with` garantiza que la conexión se libere automáticamente una vez utilizada.
-

Conclusión

 ¡Ahora tu aplicación Python usa un pool de conexiones profesional y eficiente!
 Esto mejora el rendimiento, evita la saturación de recursos y sienta las bases para aplicaciones escalables.

 Recuerda que este cambio impacta positivamente en todos los métodos DAO (insertar, actualizar, eliminar, seleccionar), ya que ahora cada uno obtendrá su conexión desde un pool bien administrado.

✨ 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)