**Procedimientos**

Un **procedimiento almacenado** en PostgreSQL:

* Es un **bloque de código SQL** que puede **ejecutar acciones:**
  + como **insertar, actualizar, eliminar, crear o modificar estructuras de BDD.**
* Se debe usar cuando se quiera ejecutar una serie de instrucciones que modifiquen datos o el estado de la base de datos.
* **No están pensados para devolver valores (en ese caso usar funciones)** ni para usarse en un SELECT.
  + Se usa para llamarlos la instrucción CALL que no devuelve resultados

**Ejemplos de uso:**

Automatizar tareas repetitivas (ej: cerrar cuentas inactivas).

Agrupar operaciones (ej: transferir dinero entre cuentas: debitar y acreditar en una transacción).

Hacer tareas administrativas (ej: archivar registros viejos).

|  |  |  |
| --- | --- | --- |
| Característica | Función | Procedimiento |
| ¿Devuelve valor? | Sí | No |
| Se usa en consultas | Sí (SELECT mi\_función()) | No |
| Se invoca con | SELECT | CALL |
| Orientado a | Computar y devolver valor | Ejecutar acciones |

**Estructura básica de un procedimiento**

CREATE PROCEDURE nombre\_procedimiento(parámetro tipo\_de\_dato)

LANGUAGE plpgsql

AS $$

BEGIN

-- bloque de instrucciones

END;

$$;

**¿Cómo se invoca un procedimiento?**

CALL nombre\_del\_procedimiento(param1, param2);

***Ejemplo simple de PROCEDIMIENTO***

CREATE PROCEDURE aumentar\_sueldo(id\_empleado INT, incremento NUMERIC)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE empleados

SET sueldo = sueldo + incremento

WHERE id = id\_empleado;

END;

$$;

***Invocación en PostgreSQL:***

CALL aumentar\_sueldo(5, 1000);

* **PostgreSQL** hace el commit automáticamente **al final** si no hubo errores (deja los cambios efectuados efectivamente en la base de datos).
* Si ocurre un error y **no lo manejás**, **se revierte todo** igual (es como un rollback automático).

Pero si **vos controlás la transacción desde afuera (por ejemplo, desde Python)**, entonces vos decidís cuándo hacer commit o rollback

***¿SE PUEDE INVOCAR UN PROCEDIMIENTO CON PSYCOPG[BINARY]?***

**Con psycopg[binary] se puede invocar procedimientos**.

***TRANSACCIONES EN PYTHON***

* *Siempre cerrá la transacción con commit() o rollback() en Python.*
* *Con psycopg, se tiene el* ***control total*** *de la transacción desde el código Python.*

*-- procedimeinto que inserta un cliente para pruebas, no recibe parametros*

*CREATE OR REPLACE PROCEDURE insertar\_cliente\_tests()*

*LANGUAGE plpgsql*

*AS $$*

*BEGIN*

*INSERT INTO customer (store\_id, first\_name, last\_name, address\_id, active)*

*VALUES (2, 'PRUEBA', 'SIN\_PARAM', 1, 1);*

*END;*

*$$;*

*#se controla transacción con commit y rollback*

*from psycopg import connect*

*try:*

*with connect("dbname=pagila user=postgres password=0909casajardinpaz0707", autocommit=False) as conn:*

*with conn.cursor() as cur:*

*cur.execute("CALL insertar\_cliente\_tests()")  # Puede fallar*

*conn.commit()  # Si todo salió bien ,  para grabar cambios*

*print("Todo salió bien")*

*except Exception as e:*

*conn.rollback()   # Cancela todos los cambios si hubo error . Si no querés grabarlos (porque falló algo o estás probando)*

*print("Ocurrió un error:", e)*

|  |  |
| --- | --- |
| Instrucción Python | ¿Qué hace? |
| conn.commit() | Le dice a PostgreSQL: "guardá todo lo que hice hasta ahora" |
| conn.rollback() | Le dice: "¡Ups! deshacé todo lo que hice en esta transacción" |
| conn.cursor() | Abre un cursor (una especie de sesión para ejecutar SQL) |
| cur.execute(...) | Ejecuta una consulta o procedimiento SQL |
| except psycopg.Error | Atrapa cualquier error que venga de PostgreSQL |

*-- procedimiento que inserta un cliente para pruebas,recibiendo parametros*

*CREATE OR REPLACE PROCEDURE insertar\_cliente\_param(*

*nombre TEXT,*

*apellido TEXT,*

*email TEXT*

*)*

*LANGUAGE plpgsql*

*AS $$*

*BEGIN*

*INSERT INTO customer (store\_id, first\_name, last\_name, email, address\_id, active)*

*VALUES (1, nombre, apellido, email, 1, 1);*

*END;*

*$$;*

*#se controla transacción con commit y rollback desde python*

*import psycopg*

*# Conectamos (sin autocommit para controlar la transacción)*

*conn = psycopg.connect("dbname=pagila user=postgres password=tu\_contra", autocommit=False)*

*cur = conn.cursor()*

*# Usás %s como placeholder para los parámetros.*

*try:*

*# Llamada al procedimiento con parámetros*

*cur.execute("CALL insertar\_cliente\_param(%s, %s,%s)", ("LOU", "Gomez","lif@life.com"))*

*#El segundo argumento de execute es una tupla con los valores.*

*conn.commit() # usamos commit() solo si todo sale bien. Si hay un error, usamos rollback().*

*print("Cliente insertado correctamente.")*

*except Exception as e:*

*conn.rollback()*

*print("Ocurrió un error:", e)*

*finally:*

*cur.close()*

*conn.close()*

***¿Qué pasa en Python si hay errores como tipo incorrecto, NOT NULL, etc?***

* *Disparan un rollback.*

***Caso 1: Tipo de dato incorrecto****:*

*INSERT INTO usuarios (edad) VALUES ('no es un número');*

*Error → Python lo detecta → entra en except → se hace rollback.*

***Caso 2: Violación de NOT NULL****:*

*INSERT INTO usuarios (nombre) VALUES (NULL);*

*Error → rollback.*

***Caso 3: Clave duplicada (violación de PRIMARY KEY)****:*

*INSERT INTO usuarios (id) VALUES (1); -- y ya existía el ID 1*

*Boom → rollback.*

*Entonces:* ***cualquier error de PostgreSQL automáticamente "rompe" la transacción****, y hasta que no hagas un rollback, no podés seguir usando la conexión.*

*Cuando ocurre un error:*

1. *PostgreSQL* ***detiene*** *la ejecución y dice “¡ERROR!”.*
2. *psycopg recibe ese error y lo transforma en una* ***excepción de Python*** *(psycopg.Error o alguna subclase).*
3. *Tu código entra al except, donde* ***vos decidís*** *qué hacer: rollback, loguear el error, mostrar algo al usuario, etc.*