**Trigger (Disparador)**

**Mecanismo de la base de datos** que se ejecuta **automáticamente** cuando ocurre un evento específico sobre una tabla o vista: por ejemplo, un INSERT, UPDATE o DELETE.

Sirve para **automatizar tareas** dentro de la base de datos. Algunos usos comunes:

* Mantener integridad[[1]](#footnote-1) entre tablas.
* Actualizar valores derivados.
* Validar datos antes de insertar.
* Sincronizar con otras bases (como tu caso: PostgreSQL ↔ ChromaDB).
* Guardar un historial o log de cambios.

Se usa cuando **cierta lógica se debe ejecutar siempre que ocurra algo**. Por ejemplo: "Cada vez que se inserta una nueva pregunta validada, actualizar los embeddings."

**NO se puede invocar desde Python**, pero **sí se activa** cuando hacés un INSERT, UPDATE o DELETE desde Python.

**NO se recomienda usar triggers:**

* La lógica es muy compleja o costosa → puede hacer lentas las operaciones.
* Se necesita control más explícito desde la aplicación en python.
* Se dificulta entender el flujo (porque todo pasa “automáticamente”).
* Si se necesita depurar o testear paso a paso: los triggers son más difíciles de rastrear.

|  |  |  |  |
| --- | --- | --- | --- |
| Concepto | ¿Qué es? | ¿Se ejecuta automáticamente? | ¿Se puede llamar desde Python? |
| Función | Bloque que devuelve un valor | No | Sí |
| Procedimiento | Bloque que ejecuta acciones (sin devolver valor) | No | Sí |
| Trigger | Se ejecuta al ocurrir un evento en la tabla | Sí | No directamente (pero se activa al modificar la tabla desde Python) |

Un trigger **llama a una función** internamente. El trigger es como un sensor (que detecta algo), y la **función** que llama es la que hace el trabajo.

**Estructura básica de un trigger en PostgreSQL**

-- ***Paso 1: crear una función que hará el trabajo***

CREATE OR REPLACE FUNCTION mi\_funcion\_trigger()

RETURNS TRIGGER AS $$

BEGIN

-- lógica aquí

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

***-- Paso 2: crear el trigger que usa esa función***

CREATE TRIGGER nombre\_del\_trigger

AFTER INSERT ON nombre\_de\_la\_tabla -- que se activa después de insertar en

FOR EACH ROW – por cada fila insertada

EXECUTE FUNCTION mi\_funcion\_trigger();

* **RETURNS TRIGGER (no significa que devuelve un trigger)**

Debe ser así porque **es una función para *ser llamada* por un trigger (no para otra cosa)**, y debe **devolver la nueva o vieja fila**, dependiendo del tipo de trigger.

**Es una convención obligatoria**: todas las funciones usadas por triggers deben tener RETURNS TRIGGER.

* **NEW**

**Variable especial** **que existe solo dentro de funciones para triggers**, y representa la **fila que se está insertando o actualizando**.

* En un INSERT: NEW contiene **los datos que se van a insertar**.
* En un UPDATE: NEW contiene **los nuevos datos**, y OLD contiene **los datos anteriores**.
* En un DELETE: solo tenés OLD.

|  |  |  |
| --- | --- | --- |
| Palabra clave | ¿Qué es? | Cuándo aparece |
| RETURNS TRIGGER | Tipo de función usada por un trigger | Siempre en funciones disparadas por triggers |
| NEW | Fila nueva que se está insertando o actualizando | INSERT o UPDATE |
| OLD | Fila anterior antes del cambio | UPDATE o DELETE |

**Ejemplo sencillo con dos tablas: guardar en log cada pregunta insertada**

***--Suponer estas dos tablas:***

CREATE TABLE preguntas (

id SERIAL PRIMARY KEY,

texto TEXT NOT NULL

);

CREATE TABLE log\_preguntas (

id SERIAL PRIMARY KEY,

texto TEXT,

fecha TIMESTAMP DEFAULT now()

);

***--Función que guarda en log (la “ function trigger”: la que se usa solo para el trigger)***

CREATE OR REPLACE FUNCTION log\_pregunta\_insertada()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO log\_preguntas (texto)

VALUES (NEW.texto); -- accedés al valor que se está insertando

RETURN NEW; -- devolvés la fila insertada para que el trigger la procese

END;

$$ LANGUAGE plpgsql;

* NEW.texto : valor que alguien acaba de enviar al hacer INSERT INTO preguntas (texto).
* RETURN NEW: le decís a PostgreSQL “seguí con la operación usando esta fila”.

***--Trigger que se activa después de insertar en tabla preguntas:***

CREATE TRIGGER trigger\_log\_preguntas

AFTER INSERT ON preguntas

FOR EACH ROW

EXECUTE FUNCTION log\_pregunta\_insertada();

**¿Qué pasa ahora?**

Cada vez que hacés: (ya sea en postgresql o desde código de afuera, como en python)

INSERT INTO preguntas (texto) VALUES ('¿Qué es LangChain?');

Se dispara automáticamente el trigger y también se insertará en log\_preguntas.

Ejemplo desde Python:

import psycopg

def insertar\_pregunta(texto\_pregunta):

try:

# Conexión a la base de datos (ajustá los valores a tu entorno)

with psycopg.connect(

"dbname=tu\_db user=tu\_usuario password=tu\_password host=localhost port=5432"

) as conn:

# Abrimos un cursor dentro del contexto

with conn.cursor() as cur:

# Ejecutamos el INSERT

cur.execute( "INSERT INTO preguntas (texto) VALUES (%s)", (texto\_pregunta,))

print("✅ Pregunta insertada correctamente.")

# El commit se hace automáticamente al salir del with

except Exception as e:

print("❌ Ocurrió un error:", e)

# El rollback es automático si usás 'with'

NOTA: **en psycopg también necesitás confirmar con commit(). Generalmente se hace implícitamente con with. Caso contrario la inserción y cualquier trigger asociado se revierten.**

¿Cómo usarla? (en Python probar con:) insertar\_pregunta("¿Qué es LangChain?")

Esto:

* Inserta la pregunta.
* Dispara cualquier trigger que tengas en AFTER INSERT.
* Hace commit automáticamente si todo sale bien.
* Hace rollback automáticamente si ocurre algún error.

**¿Qué pasa si hay un error en el trigger?**

* El trigger lanza una excepción.
* El error sube al try en Python.
  + (sí, esto es así porque los triggers en PostgreSQL forman parte de la misma transacción que la operación que los dispara. En este caso, el INSERT. Esto significa que se hacen juntas y de manera atómica:
    - si falla el insert, el triggers no se ejecuta y se deshace todo lo que se agregó parcialmente hasta ese momento con dicha instrucción
    - si se hace el insert pero falla el trigger : se deshace todo lo que se agregó parcialmente hasta ese momento con el insert y el trigger.)
* Se imprime el error y **no se guarda nada en la base de datos**.
* ¡Todo limpio y sin efectos colaterales!

**¿Los triggers son atómicos?**

**Sí.** Los triggers en PostgreSQL **forman parte de la misma transacción que la operación que los dispara**.

Cuando ocurre un error dentro de una operación (como un INSERT, UPDATE, DELETE) que está en una transacción implícita, y se lanza una excepción (por ejemplo, falla un trigger), **PostgreSQL revierte toda la transacción automáticamente**.

Eso significa que si hacés un INSERT y el trigger falla (por ejemplo, por un error en la función que llama), **todo se revierte**: el INSERT, el trigger y cualquier otra acción relacionada.

En resumen: **los triggers son transaccionales y atómicos por naturaleza**. Si fallan, se revierte todo.

* **PostgreSQL no permite hacer COMMIT o ROLLBACK dentro de funciones o triggers.** Lo maneja automáticamente. el COMMIT y/o ROLLBACK es automático.
* **los triggers no necesitan que vos escribas BEGIN o COMMIT**, porque ya forman parte del BEGIN implícito del INSERT o UPDATE.

Ejemplo:

-- insertás una pregunta

INSERT INTO preguntas (texto) VALUES ('¿Qué es Mistral?');

-- el trigger intenta insertar en log\_preguntas, pero falla por una constraint

-- resultado: se cancela también el INSERT original

Esto es bueno porque asegura consistencia. No queda la base en un estado “a medias”.

**Resumen claro:**

|  |  |  |
| --- | --- | --- |
| Situación | ¿Necesitás escribir ROLLBACK? | ¿Qué pasa si hay error? |
| Usás INSERT, UPDATE, etc. directamente (sin BEGIN) | ❌ No | PostgreSQL hace rollback automático |
| Usás BEGIN ... COMMIT explícitamente | ❌ No (aunque *podrías* hacer rollback manual si manejás errores) | PostgreSQL revierte automáticamente si no capturás el error |
| Dentro de funciones o triggers (plpgsql) | ❌ No podés ni deberías usar ROLLBACK | PostgreSQL revierte todo si la función o trigger falla |

**Dentro de funciones o triggers no podés usar COMMIT ni ROLLBACK manualmente. PostgreSQL no lo permite por diseño, porque ellos están dentro del contexto de una transacción ya abierta.**

**En resumen: PostgreSQL siempre mantiene consistencia y revierte automáticamente si hay errores en triggers. No necesitás (ni podés) usar ROLLBACK dentro de funciones o triggers. Lo que sí podés hacer, desde Python, por ejemplo, es capturar el error y decidir qué hacer después.**

**¿Cuándo conviene usar procedimientos con control de transacciones?**

Cuando:

* Necesitás **hacer varias operaciones distintas en orden**, con lógica de control (ej. "si esto falla, hacer otra cosa").
* Querés **manejar errores de forma personalizada**.
* El proceso **involucra varios pasos independientes** (ej. insertar, consultar, actualizar registros en distintas tablas).

1. datos sean correctos, completos y consistentes entre diferentes tablas relacionadas [↑](#footnote-ref-1)