FUNCIONES SQL

TRIGGER

En SQL, un trigger (disparador) es un objeto de base de datos que se ejecuta automáticamente en respuesta a ciertos eventos que ocurren en una tabla, como inserciones, actualizaciones o eliminaciones de datos.

Un trigger es un procedimiento almacenado que se asocia a una tabla específica y se activa cuando se produce un evento particular en esa tabla. Los eventos comunes que activan un trigger son:

- INSERT: Cuando se inserta una nueva fila en la tabla.
- UPDATE: Cuando se actualiza una fila existente en la tabla.
- DELETE: Cuando se elimina una fila de la tabla.

VENTAJAS

AUTOMATIZACIÓN

Ayuda a realizar actualizaciones a otras tablas o realizar alguna validación en una tabla.

INTEGRIDAD DE DATOS

Puede prevenir que se eliminen datos que estén referenciando a otras tablas

DESVENTAJAS

RECURSIVIDAD

Si un trigger modifica una tabla que a su vez activa otro trigger, podría entrar en un ciclo infinito.

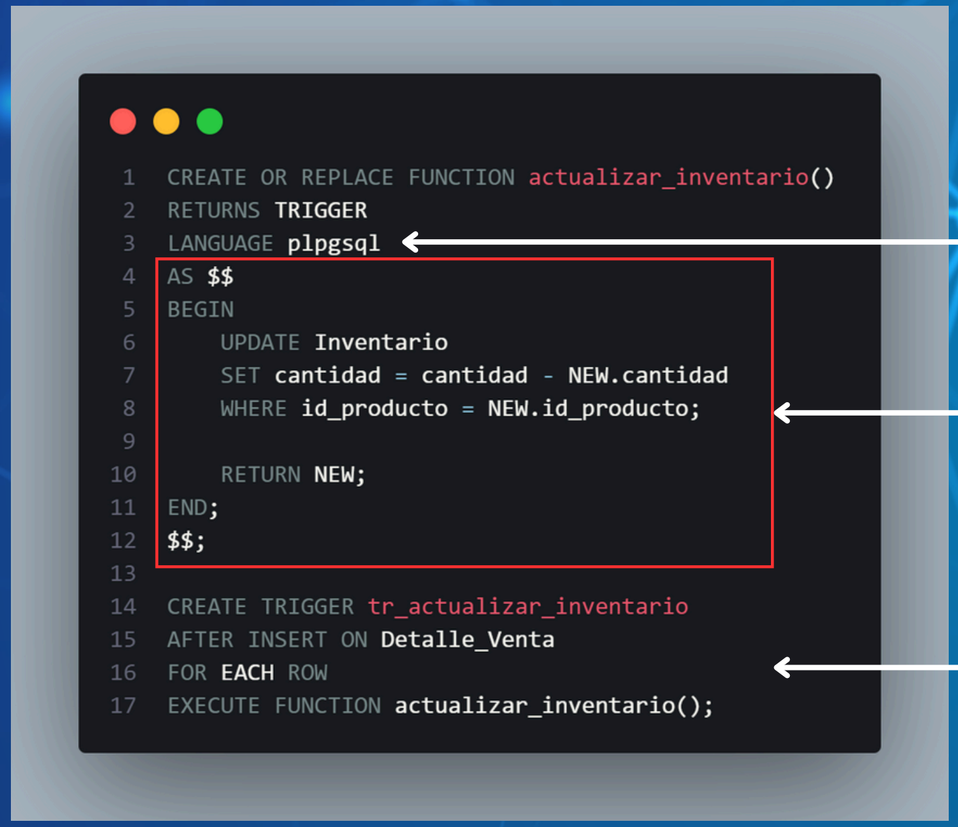
DESEMPEÑO

Los triggers pueden afectar negativamente el rendimiento si no se implementan correctamente, especialmente en bases de datos con un alto volumen de transacciones

ESTRUCTURA DE UN TRIGGER

```sql
CREATE TRIGGER trigger_name
    BEFORE INSERT ON tabla_ejemplo --Puede ser INSERT, UPADATE O DELETE
    FOR EACH ROW
    BEGIN
        -- Código a ejecutar antes o después de cada inserción
        -- Por ejemplo, validar datos o actualizar otra tabla
    END;
```

IMPLEMENTACIÓN



```sql
CREATE OR REPLACE FUNCTION actualizar_inventario()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Inventario
    SET cantidad = cantidad - NEW.cantidad
    WHERE id_producto = NEW.id_producto;

    RETURN NEW;
END;
$$;

CREATE TRIGGER tr_actualizar_inventario
AFTER INSERT ON Detalle_Venta
FOR EACH ROW
EXECUTE FUNCTION actualizar_inventario();
```

Procedural Language/PostgreSQL Structured Query Language.

La lógica de la función se define en esta parte. En PostgreSQL siempre se define una función antes del trigger.

El trigger se define después de la función, se especifica cuando se lanzara (AFTER O BEFORE) y que es lo que hará (llamado a la función actualizar_inventario()).

EJECUCIÓN

```
1  SELECT p.nombre, i.cantidad
2  FROM Inventario i
3  JOIN Productos p ON i.id_producto = p.id_producto
4  WHERE p.id_producto = 1;
```

Data Output | Messages | Notifications

	nombre character varying (100)	cantidad integer
1	Chocolate amargo 70%	42

1. Verificamos el inventario inicial.

Query | Query History

```
1  INSERT INTO Detalle_Venta (id_venta, id_producto, cantidad, precio_unitario)
2  VALUES (1, 1, 3, (SELECT precio FROM Productos WHERE id_producto = 1));
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 49 msec.
```

2. Ejecutamos un INSERT para activar el trigger

Query | Query History

```
1  INSERT INTO Detalle_Venta (id_venta, id_producto, cantidad, precio_unitario)
2  VALUES (1, 1, 1000, (SELECT precio FROM Productos WHERE id_producto = 1));
```

Data Output | Messages | Notifications

```
ERROR:  Stock insuficiente para el producto ID 1. Stock actual: 39, Cantidad solicitada: 1000
CONTEXT:  función PL/pgSQL validar_stock() en la línea 10 en RAISE

SQL state: P0001
```

ERROR: Creamos un trigger que valide el stock, si no se cumple la condición arrojara un mensaje de error.

Query | Query History

```
1  SELECT * FROM Inventario WHERE id_producto = 1;
```

Data Output | Messages | Notifications

	id_inventario [PK] integer	id_producto integer	cantidad integer	fecha_actualizacion timestamp without time zone
1	1	1	39	2025-07-08 15:21:27

3. Verificamos que se haga el descuento según los datos enviados

ERROR

```sql
CREATE OR REPLACE FUNCTION validar_stock()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_stock_actual INT;
BEGIN
    SELECT cantidad INTO v_stock_actual
    FROM Inventario
    WHERE id_producto = NEW.id_producto;

    IF v_stock_actual < NEW.cantidad THEN
        RAISE EXCEPTION 'Stock insuficiente para el producto ID %. Stock actual: %, Cantidad solicitada: %',
            NEW.id_producto, v_stock_actual, NEW.cantidad;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER tr_validar_stock
BEFORE INSERT ON Detalle_Venta
FOR EACH ROW
EXECUTE FUNCTION validar_stock();
```

ERROR: Creamos un trigger que valide el stock, si no se cumple la condición arrojara un mensaje de error.

```sql
SELECT p.nombre, i.cantidad
FROM Inventario i
JOIN Productos p ON i.id_producto = p.id_producto
WHERE p.id_producto = 1;
```

Data Output | Messages | Notifications

	nombre character varying (100)	cantidad integer
1	Chocolate amargo 70%	42

Query | Query History

```sql
INSERT INTO Detalle_Venta (id_venta, id_producto, cantidad, precio_unitario)
VALUES (1, 1, 1000, (SELECT precio FROM Productos WHERE id_producto = 1));
```

Data Output | Messages | Notifications

```
ERROR:  Stock insuficiente para el producto ID 1. Stock actual: 39, Cantidad solicitada: 1000
CONTEXT:  función PL/pgSQL validar_stock() en la línea 10 en RAISE

SQL state: P0001
```

valida el inventario actual con lo que se solicito antes de hacer el INSERT, como no cumple la condición del trigger no puede acceder al trigger que actualiza inventarios

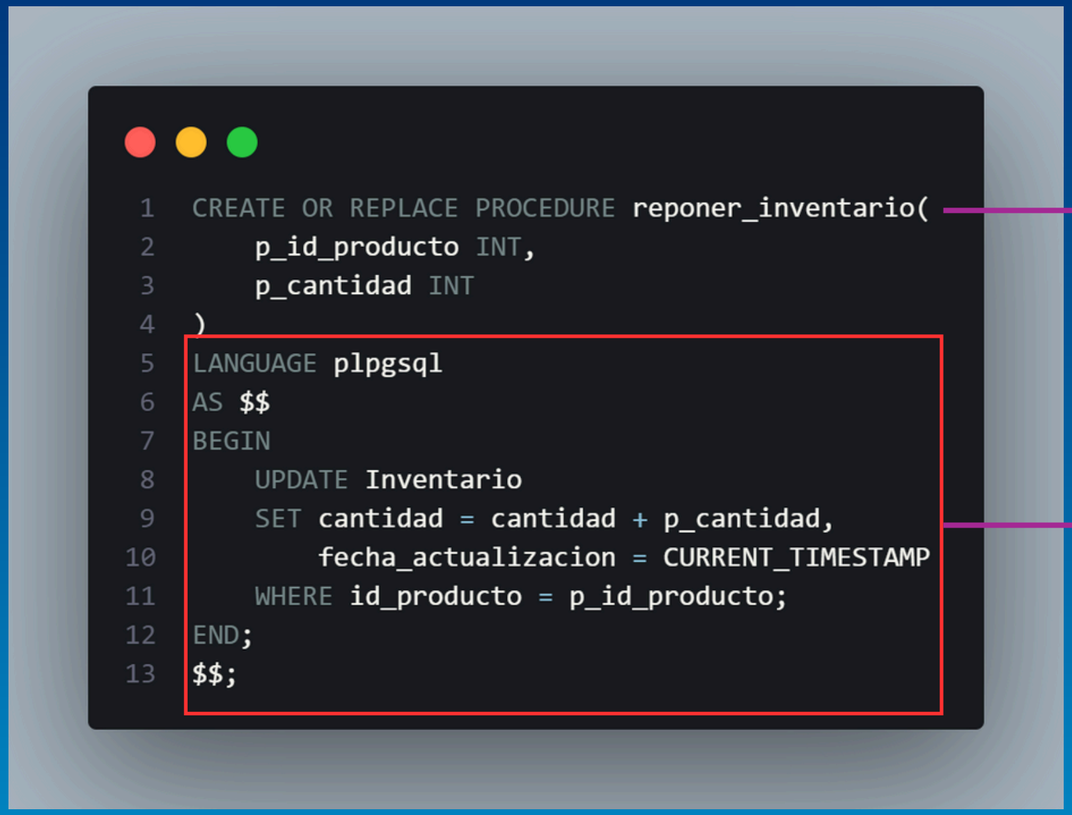
STORED PROCEDURES

Los stored procedures son un conjunto de instrucciones almacenadas que se ejecutan en la base de datos .

ESTRUCTURA DE LOS STORED PROCEDURES

```
CREATE OR REPLACE PROCEDURE procedure_name(
    -- declaracion de variables
)
language plpgsql
as $$
begin
-- Código a ejecutar
end;
$$
```

IMPLEMENTACIÓN



```sql
CREATE OR REPLACE PROCEDURE reponer_inventario(
    p_id_producto INT,
    p_cantidad INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Inventario
    SET cantidad = cantidad + p_cantidad,
        fecha_actualizacion = CURRENT_TIMESTAMP
    WHERE id_producto = p_id_producto;
END;
$$;
```

Se crea el procedure y se le asigna un nombre. También se declaran las variables a usar dentro de la lógica.

Contiene el código SQL o del lenguaje procedimental que define la lógica del procedimiento

EJECUCIÓN

Query | Query History

```
1  SELECT p.nombre, i.cantidad
2  FROM Inventario i
3  JOIN Productos p ON i.id_producto = p.id_producto
4  WHERE i.id_producto = 3;
```

Data Output | Messages | Notifications

	nombre character varying (100)	cantidad integer
1	Paleta de caramelo	116

1. Verificamos el inventario inicial del producto.

```
1  CALL reponer_inventario(3, 50);
```

Data Output | Messages | Notifications

```
CALL

Query returned successfully in 45 msec.
```

2. Hacemos el llamado a nuestra función pasando los argumentos necesarios (en este caso el id del producto y la cantidad a actualizar)

Query | Query History

```
1  SELECT p.nombre, i.cantidad
2  FROM Inventario i
3  JOIN Productos p ON i.id_producto = p.id_producto
4  WHERE i.id_producto = 3;
```

Data Output | Messages | Notifications

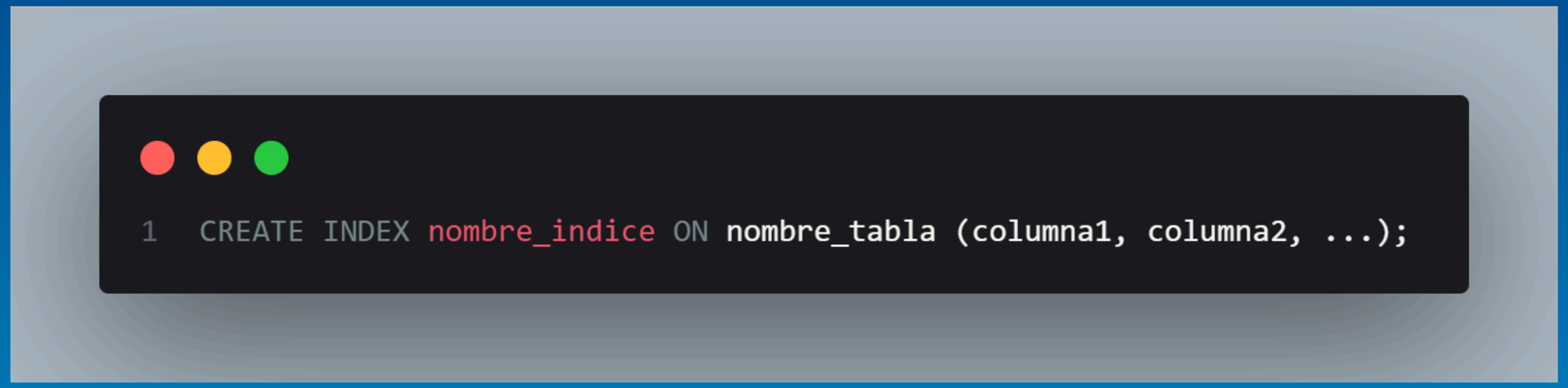
	nombre character varying (100)	cantidad integer
1	Paleta de caramelo	166

3. Solicitamos el inventario una vez mas y verificamos que se haya actualizado el inventario.

INDICES

Son una estructura de datos que mejora la velocidad de las consultas al permitir una recuperación de datos más rápida. Actúa como una tabla de búsqueda, permitiendo al motor de base de datos localizar filas específicas sin tener que examinar cada fila de la tabla.

ESTRUCTURA DE UN INDICE



IMPLEMENTACIÓN

```sql
CREATE INDEX idx_productos_nombre ON Productos(nombre);

```

```sql
SELECT
    tablename AS tabla,
    indexname AS índice
FROM
    pg_indexes
WHERE
    tablename IN ('productos', 'clientes', 'ventas', 'detalle_venta', 'inventario')
ORDER BY
    tablename;
```

Data Output | Messages | Graph Visualiser | Notifications

SQL

	tabla name	índice name
1	clientes	idx_clientes_nombre_apellido
2	clientes	clientes_pkey
3	detalle_venta	idx_detalle_venta_producto
4	detalle_venta	detalle_venta_pkey
5	inventario	inventario_pkey
6	productos	idx_productos_nombre
7	productos	productos_pkey
8	ventas	idx_ventas_fecha
9	ventas	ventas_pkey

FIN :D