Procedimientos, Funciones y Triggers

Bases de Datos 1 (CC3088) - 2026

Procedimientos, Funciones y Triggers

Semestre 01, 2026

Código repetido en SQL

Varias partes de la aplicación necesitan calcular el total de un pedido.

Cada una repite la misma lógica SQL.

Si la lógica cambia, hay que encontrar y modificar cada lugar.

El mismo problema

Al insertar un pedido: calcular el total con descuento.

Al generar una factura: calcular el total con impuestos.

Al hacer un reporte: calcular el total por cliente.

La fórmula es la misma. Está duplicada en tres lugares.

Solución

Escribir la lógica una sola vez, directamente en la base de datos.

Cualquier cliente que se conecte puede usarla.

Si cambia, se cambia en un solo lugar.

Eso es lo que ofrecen las funciones y los procedimientos almacenados.

Funciones

Una función es un bloque de código con nombre que vive en la base de datos.

Recibe parámetros, ejecuta lógica y devuelve un valor.

Se puede usar dentro de consultas SQL como cualquier función nativa.

Sintaxis básica

```
CREATE OR REPLACE FUNCTION nombre_funcion(parametro tipo)
RETURNS tipo_retorno AS $$
DECLARE
    variable tipo;
BEGIN
    -- lógica
    RETURN valor;
END;
$$ LANGUAGE plpgsql;
```

- `CREATE OR REPLACE` → crea o actualiza si ya existe
- `$$` → delimitador del cuerpo de la función
- `DECLARE` → variables locales (opcional)
- `LANGUAGE plpgsql` → PL/pgSQL, el lenguaje procedural de PostgreSQL

Función simple: calcular IVA

```
CREATE OR REPLACE FUNCTION calcular_iva(precio NUMERIC)
RETURNS NUMERIC AS $$
BEGIN
    RETURN precio * 0.12;
END;
$$ LANGUAGE plpgsql;
```

Uso dentro de una consulta:

```
SELECT nombre, precio, calcular_iva(precio) AS iva
FROM productos;
```

Función con lógica condicional

```
CREATE OR REPLACE FUNCTION aplicar_descuento(precio NUMERIC, cantidad INT)
RETURNS NUMERIC AS $$
BEGIN
    IF cantidad >= 10 THEN
        RETURN precio * 0.90;  -- 10% de descuento
    ELSIF cantidad >= 5 THEN
        RETURN precio * 0.95;  -- 5% de descuento
    ELSE
        RETURN precio;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT nombre, precio,
       aplicar_descuento(precio, 8) AS precio_final
FROM productos;
```

Función que consulta la base de datos

```
CREATE OR REPLACE FUNCTION total_pedido(p_pedido_id INT)
RETURNS NUMERIC AS $$
DECLARE
    v_total NUMERIC;
BEGIN
    SELECT SUM(cantidad * precio_unitario)
    INTO v_total
    FROM pedido_detalle
    WHERE pedido_id = p_pedido_id;

    RETURN COALESCE(v_total, 0);
END;
$$ LANGUAGE plpgsql;
```

```
SELECT id, total_pedido(id) AS total
FROM pedidos
WHERE cliente_id = 5;
```

Funciones que devuelven tablas

```
CREATE OR REPLACE FUNCTION pedidos_por_cliente(p_cliente_id INT)
RETURNS TABLE(pedido_id INT, fecha DATE, total NUMERIC) AS $$
BEGIN
    RETURN QUERY
    SELECT p.id, p.fecha, total_pedido(p.id)
    FROM pedidos p
    WHERE p.cliente_id = p_cliente_id
    ORDER BY p.fecha DESC;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT * FROM pedidos_por_cliente(3);
```

PL/pgSQL — elementos básicos

PL/pgSQL es el lenguaje procedural nativo de PostgreSQL.

Combina SQL con estructuras de control: variables, condicionales, ciclos.

Variables

```
DECLARE
    v_nombre    TEXT;
    v_precio    NUMERIC := 0;
    v_contador  INT     := 0;
    v_fecha     DATE    := CURRENT_DATE;
```

- Se declaran en el bloque `DECLARE`.
- Se inicializan con `:=`.
- Por convención, se prefijan con `v_` para distinguirlas de columnas.

Asignación

```
-- Asignación directa
v_precio := 150.00;

-- Asignación desde una consulta
SELECT precio INTO v_precio
FROM productos
WHERE id = p_id;
```

Condicionales

```
IF condicion THEN
    -- ...
ELSIF otra_condicion THEN
    -- ...
ELSE
    -- ...
```

```
END IF;
```

Ciclos

```
-- Ciclo con contador
FOR i IN 1..10 LOOP
    -- ...
END LOOP;

-- Ciclo sobre resultados de una consulta
FOR registro IN SELECT * FROM productos LOOP
    RAISE NOTICE 'Producto: %', registro.nombre;
END LOOP;

-- Ciclo WHILE
WHILE v_contador < 10 LOOP
    v_contador := v_contador + 1;
END LOOP;
```

Manejo de errores

```
BEGIN
    -- operaciones
EXCEPTION
    WHEN unique_violation THEN
        RAISE NOTICE 'Ya existe un registro con ese valor único';
    WHEN foreign_key_violation THEN
        RAISE EXCEPTION 'Referencia inválida: %', SQLERRM;
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Error inesperado: %', SQLERRM;
END;
```

Procedimientos almacenados

Un procedimiento es similar a una función pero no devuelve un valor.

Se usa para operaciones que modifican datos: insertar, actualizar, eliminar.

A diferencia de las funciones, puede controlar transacciones con `COMMIT` y `ROLLBACK`.

Diferencias clave

Función:

- Devuelve un valor
- Se usa dentro de un `SELECT` o expresión
- No puede usar `COMMIT` / `ROLLBACK`

Procedimiento:

- No devuelve un valor
- Se llama con `CALL`
- Puede controlar transacciones con `COMMIT` / `ROLLBACK`

Sintaxis básica

```
CREATE OR REPLACE PROCEDURE nombre_procedimiento(parametro tipo)
LANGUAGE plpgsql AS $$
BEGIN
    -- lógica
END;
$$;
```

```
CALL nombre_procedimiento(argumento);
```

Ejemplo: registrar un pedido completo

```
CREATE OR REPLACE PROCEDURE registrar_pedido(
    p_cliente_id  INT,
    p_producto_id INT,
    p_cantidad    INT
)
LANGUAGE plpgsql AS $$
DECLARE
    v_precio        NUMERIC;
    v_nuevo_pedido  INT;
BEGIN
    -- Obtener el precio del producto
    SELECT precio INTO v_precio
    FROM productos
    WHERE id = p_producto_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Producto % no existe', p_producto_id;
    END IF;

    -- Crear el pedido
    INSERT INTO pedidos (cliente_id, fecha, estado)
    VALUES (p_cliente_id, CURRENT_DATE, 'pendiente')
    RETURNING id INTO v_nuevo_pedido;

    -- Agregar el detalle
    INSERT INTO pedido_detalle (pedido_id, producto_id, cantidad, precio_unitario)
    VALUES (v_nuevo_pedido, p_producto_id, p_cantidad, v_precio);

    -- Actualizar stock
    UPDATE productos
    SET stock = stock - p_cantidad
    WHERE id = p_producto_id;

    RAISE NOTICE 'Pedido % registrado correctamente', v_nuevo_pedido;
END;
$$;
```

```
CALL registrar_pedido(3, 12, 5);
```

Ventaja de los procedimientos

Si cualquier operación falla, todo se revierte automáticamente.

La lógica de negocio vive en la base de datos, no en la aplicación.

Cualquier cliente que se conecte ejecuta exactamente la misma lógica.

Triggers

Un trigger es un procedimiento que se ejecuta automáticamente cuando ocurre un evento en una tabla.

No se llama explícitamente. Se dispara solo.

Eventos que activan un trigger

- `INSERT` — al insertar una fila
- `UPDATE` — al modificar una fila
- `DELETE` — al eliminar una fila
- `TRUNCATE` — al vaciar la tabla completa

Cuándo se ejecuta

- `BEFORE` — antes de la operación; puede modificar o cancelar lo que va a ocurrir
- `AFTER` — después de la operación; útil para auditoría y notificaciones
- `INSTEAD OF` — en lugar de la operación; solo en vistas

Nivel de ejecución

- `FOR EACH ROW` — se ejecuta una vez por cada fila afectada

- `FOR EACH STATEMENT` — se ejecuta una vez por sentencia, sin importar cuántas filas cambiaron

Variables especiales en triggers de fila

- `NEW` — disponible en INSERT y UPDATE; contiene los valores nuevos de la fila
- `OLD` — disponible en UPDATE y DELETE; contiene los valores anteriores de la fila
- `TG_OP` — indica la operación que disparó el trigger: `'INSERT'`, `'UPDATE'` o `'DELETE'`
- `TG_TABLE_NAME` — nombre de la tabla que disparó el trigger

Crear un trigger

Un trigger requiere dos pasos:

1. Crear una función de trigger que devuelve `TRIGGER`.
2. Crear el trigger que vincula el evento con la función.

Paso 1: la función de trigger

```
CREATE OR REPLACE FUNCTION nombre_funcion_trigger()
RETURNS TRIGGER AS $$
BEGIN
    -- lógica usando NEW y OLD
    RETURN NEW;  -- para INSERT y UPDATE
    -- RETURN OLD;  -- para DELETE
    -- RETURN NULL;  -- cancela la operación (solo en BEFORE)
END;
$$ LANGUAGE plpgsql;
```

Paso 2: el trigger

```
CREATE TRIGGER nombre_trigger
    BEFORE | AFTER  INSERT | UPDATE | DELETE
    ON nombre_tabla
    FOR EACH ROW | STATEMENT
    EXECUTE FUNCTION nombre_funcion_trigger();
```

Ejemplo 1: auditoría automática

Registrar automáticamente quién modificó qué y cuándo.

```
-- Tabla de auditoría
CREATE TABLE auditoria_productos (
    id          SERIAL PRIMARY KEY,
    producto_id INT,
    operacion   TEXT,
    precio_ant  NUMERIC,
    precio_nuevo NUMERIC,
    modificado_en TIMESTAMP DEFAULT NOW()
);
```

```
-- Función del trigger
CREATE OR REPLACE FUNCTION auditar_productos()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        INSERT INTO auditoria_productos
            (producto_id, operacion, precio_ant, precio_nuevo)
        VALUES
            (OLD.id, 'UPDATE', OLD.precio, NEW.precio);
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO auditoria_productos
            (producto_id, operacion, precio_ant, precio_nuevo)
        VALUES
            (OLD.id, 'DELETE', OLD.precio, NULL);
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Crear el trigger
CREATE TRIGGER trg_auditoria_productos
    AFTER UPDATE OR DELETE
    ON productos
    FOR EACH ROW
    EXECUTE FUNCTION auditar_productos();
```

Ahora cada UPDATE o DELETE en productos queda registrado automáticamente.

Ejemplo 2: validación de datos

Rechazar automáticamente precios negativos o stock insuficiente.

```
CREATE OR REPLACE FUNCTION validar_precio()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.precio <= 0 THEN
        RAISE EXCEPTION 'El precio debe ser mayor a cero. Recibido: %', NEW.precio;
    END IF;

    IF NEW.stock < 0 THEN
        RAISE EXCEPTION 'El stock no puede ser negativo. Recibido: %', NEW.stock;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_validar_precio
    BEFORE INSERT OR UPDATE
```

```
    ON productos
    FOR EACH ROW
    EXECUTE FUNCTION validar_precio();
```

Cualquier intento de insertar o actualizar con precio ≤ 0 es rechazado automáticamente, desde cualquier cliente.

Ejemplo 3: actualización automática de totales

Recalcular el total de un pedido cada vez que cambia su detalle.

```
CREATE OR REPLACE FUNCTION actualizar_total_pedido()
RETURNS TRIGGER AS $$
DECLARE
    v_pedido_id INT;
BEGIN
    -- Determinar qué pedido fue afectado
    IF TG_OP = 'DELETE' THEN
        v_pedido_id := OLD.pedido_id;
    ELSE
        v_pedido_id := NEW.pedido_id;
    END IF;

    -- Recalcular y guardar el total
    UPDATE pedidos
    SET total = (
        SELECT COALESCE(SUM(cantidad * precio_unitario), 0)
        FROM pedido_detalle
        WHERE pedido_id = v_pedido_id
    )
    WHERE id = v_pedido_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_total_pedido
    AFTER INSERT OR UPDATE OR DELETE
    ON pedido_detalle
    FOR EACH ROW
    EXECUTE FUNCTION actualizar_total_pedido();
```

Gestión de funciones y triggers

Ver funciones existentes

```
SELECT routine_name, routine_type
FROM information_schema.routines
WHERE routine_schema = 'public'
ORDER BY routine_type, routine_name;
```

Ver triggers existentes

```
SELECT trigger_name, event_manipulation, event_object_table, action_timing
FROM information_schema.triggers
WHERE trigger_schema = 'public'
ORDER BY event_object_table;
```

Eliminar funciones y triggers

```
DROP FUNCTION calcular_iva(NUMERIC);
DROP PROCEDURE registrar_pedido(INT, INT, INT);

-- Primero el trigger, luego la función
DROP TRIGGER trg_auditoria_productos ON productos;
DROP FUNCTION auditar_productos();
```

Deshabilitar un trigger temporalmente

```
-- Deshabilitar (útil durante migraciones de datos)
ALTER TABLE productos DISABLE TRIGGER trg_validar_precio;

-- Volver a habilitar
ALTER TABLE productos ENABLE TRIGGER trg_validar_precio;

-- Deshabilitar todos los triggers de una tabla
ALTER TABLE productos DISABLE TRIGGER ALL;
```