IIC2413

AYUDANTÍA 8

SP Y TRIGGERS

¿QUÉ VEREMOS?

Stored Procedures

Triggers

Ejercicios Jupyter Notebook

Stored Procedures

Son **funciones** definidas **mediante SQL** que se guardan en su DBMS y permiten ejecutar lógica compleja y repetitiva.

¿Por qué son útiles?

Porque permiten ejecutar **múltiples consultas** y hacer **control de flujo** con IFs o loops en **un solo proceso**.

(DBMS: Database Managment System. Ej: PostgreSQL)

Stored Procedures: Sintaxis

```
CREATE OR REPLACE FUNCTION  <nombre_funcion>(<argumentos>)
RETURNS
<tipo de dato> AS $$ --declaramos lo que retorna
DECLARE --declaramos variables a utilizar (en caso de ser necesario)
    <variable_1>
    <variable_2>
    ...
BEGIN --inicio de función
    <sentencias SQL>
END --finalización de función
$$ LANGUAGE plpgsql
```

Stored Procedures: Ejemplo 1

```
CREATE OR REPLACE FUNCTION  insertar_persona(rut varchar,
nombre varchar)
RETURNS  void AS $$
BEGIN
    INSERT INTO personas VALUES (rut, nombre);
END;
$$ LANGUAGE plpgsql;
```

En PostgreSQL, las funciones se invocan con SELECT porque devuelven un valor, incluso si ese valor es void. Entonces para llamar a insertar_persona:

```
SELECT insertar_persona('11.111.111-1', 'pepito');
```

Stored Procedures: Ejemplo 2

```
CREATE OR REPLACE FUNCTION get_nombre_por_id(int)
RETURNS varchar AS $$
DECLARE
    id ALIAS FOR $1; -- Aquí, $1 es el primer parámetro (tipo INT)
    resultado varchar;  -- Variable para almacenar el resultado
BEGIN
    SELECT nombre
    INTO resultado
    FROM personas
    WHERE personas.id = id;
    RETURN resultado;
END;
$$ LANGUAGE plpgsql;
```

Para llamar a la función:

```
SELECT get_nombre_por_id(3);
```

Stored Procedures: Ejemplo 3

```
CREATE OR REPLACE FUNCTION insertar_adultos()
RETURNS void AS $$
DECLARE
    p RECORD; --Guarda una fila completa con múltiples columnas
BEGIN
    FOR p IN SELECT * FROM personas LOOP
        IF p.edad >= 18 THEN
            INSERT INTO adultos(rut, nombre, edad)
            VALUES (p.rut, p.nombre, p.edad);
        END IF;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Para llamar a la función:

```
SELECT insertar_adultos();
```

Triggers

Son mecanismos que permiten automatizar tareas dentro de la base de datos. Se disparan (“triggerean”) automáticamente sin acción del usuario al realizar una determinada acción sobre una tabla.

¿Por qué son útiles?

Son útiles para siempre mantener la base de datos actualizada frente a cambios, evitar ciertos cambios en una tabla si no se cumple una condición y para sincronización de las tablas

Sintaxis

```
CREATE OR REPLACE FUNCTION nombre_funcion()
RETURNS TRIGGER AS $$
BEGIN
   -- Aquí va el código que se ejecuta con el trigger
   RETURN NEW; -- o RETURN OLD, según el tipo de trigger
END;
$$ LANGUAGE plpgsql;
```

NEW es para insert o update, OLD para update o delete

Sintaxis

```
CREATE TRIGGER nombre_del_trigger
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE } ON
nombre_tabla
FOR EACH ROW
EXECUTE FUNCTION nombre_funcion();
```

BEFORE hace que el trigger se ejecute antes de la acción que vamos a hacer, y AFTER luego.

El trigger se ejecuta dependiendo de si queremos eliminar, actualizar o insertar un registro.

Ejemplo 1

```
CREATE OR REPLACE FUNCTION notificar_insert()
RETURNS TRIGGER AS $$
BEGIN
    RAISE NOTICE 'Se insertó un producto: %', NEW.nombre;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_producto
AFTER INSERT ON productos
FOR EACH ROW
EXECUTE FUNCTION notificar_insert();
```

Ejemplo 2

```
CREATE TABLE cambios (
    id SERIAL,
    tabla TEXT,
    operacion TEXT,
    fecha TIMESTAMP DEFAULT now()
);
```

```
CREATE OR REPLACE FUNCTION log_update()
RETURNS TRIGGER AS $$
BEGIN
 INSERT INTO cambios(tabla, operacion) VALUES
(TG_TABLE_NAME, TG_OP);
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auditoria
AFTER UPDATE ON empleados
FOR EACH ROW
EXECUTE FUNCTION log_update();
```

Ahora cada vez que se actualice algo en empleados se va a guardar en la tabla cambios en que tabla se hizo el cambio y el tipo de operacion con la fecha

Ejemplo 3

```
CREATE OR REPLACE FUNCTION evitar_borrado_admin()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.rol = 'admin' THEN
        RAISE EXCEPTION 'No se puede borrar un administrador';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER no_borrar_admins
BEFORE DELETE ON usuarios
FOR EACH ROW
EXECUTE FUNCTION evitar_borrado_admin();
```

```
DELETE FROM usuarios
WHERE nombre = ‘Juan’ AND
rol = ‘admin’
```

no va a funcionar