Triggers y PL/pgSQL

Oscar Gutierrez Blanco
Lorena Lozano Plata
José Miguel Alonso
Maite Villalba

PL/pgSQL

- Lenguaje procedimental para ser utilizado en el servidor:
  - Mayor rendimiento
  - Soporte SQL
  - Portabilidad ⇒ Reusar código en el servidor Postgres
  - Crear funciones y disparadores (triggers)
  - Tipos creados por el usuario, funciones y operadores
  - Añadir estructuras de control al lenguaje SQL
  - Poder realizar computaciones complejas
  - Sencillo de utilizar
- Estructura: Lenguaje estructurado por bloques

```
[ <<etiqueta>>
   ]
[ DECLARE
declaraciones
   ]
BEGIN
estamentos
```

- https://www.postgresql.org/docs/current/plpgsql.html

PL/pgSQL - Declaraciones

- Declarar variables, filas y registros: `DECLARE`

  name [CONSTANT] *type* [COLLATE *collation_name*] [NOT NULL] [{ DEFAULT | := | = } *expression*];

- Cualquier tipo SQL: integer, varchar, char, etc.
  - Ejemplo: user_id integer;

- Inicialización/asignación de variables:
  - Identificador := expresión.
    - Ejemplo: user_id:=20;

- Ejemplo:

```
DECLARE
        cantidad INTEGER := 30;
```

PL/pgSQL – Esctructuras de Control

- `RETURN` expresión; -- Devuelve resultado desde una función.

- Condicionales:
  - `IF … THEN … ELSE END IF;`

- Bucles:
  - `LOOP ….. END LOOP;`
  - `EXIT` , para salir del bucle si cumple condición.
  - `WHILE` expresión `LOOP ….. END LOOP;`
  - `FOR` nombre IN expresión `LOOP … END LOOP` (solo Integer)
  - `FOR` registro | fila `IN` select_query `LOOP .. END LOOP` (bucles a través de los resultados de una consulta)

PL/pgSQL - Funciones

```
CREATE FUNCTION algunafuncion() RETURNS INTEGER AS '
DECLARE
   cantidad INTEGER := 30;
BEGIN
   RAISE NOTICE ''La cantidad aquí es %'',cantidad; --
   La cantidad aquí es 30
   cantidad := 50;
DECLARE
   cantidad INTEGER := 80;
BEGIN
   RAISE NOTICE ''La cantidad aquí es %'',cantidad; --
   La cantidad aquí es 80
END;
RAISE NOTICE ''La cantidad aquí es %'',cantidad; --
La cantidad aquí es 50
RETURN cantidad;
END;
' LANGUAGE 'plpgsql';
```

PL/pgSQL - Declaraciones

Triggers

- Es un procedimiento que se ejecuta cuando se realiza una operación sobre un objeto de la base de datos.
  - Por ejemplo, cuando se modifica una tabla mediante las instrucciones: `INSERT, UPDATE` o `DELETE`.
- Hay 2 tipos de triggers:
  - Nivel fila: `FOR EACH ROW`
    - El trigger se invoca por cada tupla de la tabla
  - Nivel sentencia: `FOR EACH STATEMENT`
    - Se invoca solo una vez por sentencia con independencia del número de tuplas de la tabla a modificar.
- Usos:
  - Para auditar las transacciones de una tabla almacenando los eventos a modo de log.
  - Para forzar restricciones check
  - Para poblar tablas

Triggers

```
CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE { FUNCTION | PROCEDURE } function_name ( arguments )

where event can be one of:

    INSERT
    UPDATE [ OF column_name [, ... ] ]
    DELETE
    TRUNCATE
```

Cuándo se dispara el trigger

Tabla sobre la que se crea

Timing

Si se dispara 1 vez/fila o 1 vez/SQL

Acción que dispara

- Más información:
  - https://www.postgresql.org/docs/15/sql-createtrigger.html

Triggers. Variables asociadas

$TG_name: Nombre del trigger en CREATE TRIGGER
$TG_relid: El ID del objeto de la tabla que causó la llamada al trigger.
$TG_table_name: Nombre de la tabla que causó la llamada al trigger.
$TG_when: evento BEFORE, AFTER, o INSTEAD OF del trigger
$TG_level: ROW or STATEMENT dependiendo del tipo de trigger.
$TG_op: INSERT, UPDATE, DELETE, o TRUNCATE.
$NEW: Variable tipo RECORD que almacena la nueva tupla de la tabla para operaciones INSERT/UPDATE en triggers de nivel ROW o vacío para DELETE.
$OLD: Variable tipo RECORD que almacena la antigua tupla de la tabla para operaciones UPDATE/DELETE en triggers de nivel ROW o vacío para INSERT.
$args: lista de argumentos de la función tal como se indiquen en el CREATE TRIGGER. Se puede llamar a los argumentos desde el cuerpo de la función como $1 ... $*n*.

Ejemplo Combinamos funciones y triggers

```
-- Se crea la tabla auditoria con los atributos dados en el enunciado
--CREATE TABLE auditoria (
--  nombretabla text,
--  fecha timestamp
--  ...etc.
--);

-- Se crea la función que se ejecutará

CREATE OR REPLACE FUNCTION fn_auditoria() RETURNS TRIGGER AS $fn_auditoria$
  DECLARE
  --  no declaro nada porque no me hace falta...de hecho DECLARE podría haberlo omitido en este caso
  BEGIN
  -- Se determina qué acción ha activado el trigger y se inserta un nuevo valor en la tabla dependiendo
  -- de dicha acción. Junto con la acción se escribe lo que solicita el enunciado
   IF TG_OP='INSERT' THEN
     INSERT INTO auditoria VALUES ('alta',..., current_timestamp);  -- Cuando hay una inserción
   ELSIF TG_OP='UPDATE' THEN
     INSERT INTO auditoria VALUES ('modificación',...,current_timestamp); -- Cuando hay una modificación
   ELSEIF TG_OP='DELETE' THEN
     INSERT INTO auditoria VALUES ('borrado',...,current_timestamp); -- Cuando hay un borrado
   END IF;
   RETURN NULL;
  END;
$fn_auditoria$ LANGUAGE plpgsql;



-- Se crea el trigger que se dispara cuando hay una inserción, modificación o borrado en cada tabla de la base de datos discos

CREATE TRIGGER tg_auditoria after INSERT or UPDATE or DELETE
  ON discos FOR EACH ROW
  EXECUTE PROCEDURE fn_auditoria();

-- Lo mismo para cada tabla
```

Triggers práctica 2

- Trigger AUDITORIA (ver ejemplo)
  1. Crear tabla para almacenar los datos de auditoria:
     - Con los atributos solicitados: nombre de tabla, tipo de evento, usuario y timestamp
     - id SERIAL PRIMARY KEY: Como PK ponemos un id automáticamente generado
  2. Crear la función que ejecutará el trigger: CREATE OR REPLACE FUNCTION
  3. Crear trigger para cada tabla.
- Trigger para INSERT de piloto_corre_GP
  1. Crear tabla para contabilizar el número total de puntos de cada piloto.
  2. Crear la función que ejecutará el trigger que sume los puntos conseguidos a los que ya tenía el piloto: CREATE OR REPLACE FUNCTION.
  3. Crear trigger que se ejecute después de insertar en la tabla piloto_corre_GP.