Triggers

Jesús García

Triggers

Jesús García

08/06/2020

Índice

Introducción.

En la unidad anterior vimos cómo definir funciones mediante PL/pgSQL y cómo podíamos llamarlas mediante el uso de consultas SELECT. En PostgreSQL, y en la mayoría de los sistemas gestores de bases de datos relacionales, podemos hacer que estas funciones se llamen automáticamente cuando se produzca un evento. Este tipo de funciones recibe el nombre de *triggers* o disparadores.

Así pues, los disparadores son muy similares a las funciones, pero con la particularidad de estar asociadas a tablas, de forma que se ejecutan automáticamente cuando se realiza una operación de manipulación de datos (inserción, actualización o borrado) sobre dicha tabla.

El uso de *triggers* nos permite implementar restricciones que no podemos expresar con el modelo relacional. Por ejemplo, en una base de datos sobre equipos de fútbol, podemos limitar, mediante un *trigger* el número de jugadores que puede tener un equipo. También podemos utilizar los *triggers* para comprobar que los valores que se quieren guardar en la base de datos son válidos.

Sintaxis básica

Para crear un *trigger* en primer lugar hay que definir la función que se llamará cuando se produzca un evento. Esta función se caracteriza por:

- No tener parámetros.
- Devolver una variable de tipo *trigger*.
- Entre la cláusula BEGIN y END escribiremos RETURN NEW; como se puede ver en el siguiente ejemplo. Más adelante veremos otras opciones y su significado.

```
CREATE OR REPLACE FUNCTION nombre_funcion() RETURNS trigger AS $$
DECLARE
    -- Declaración de variables
BEGIN
    -- Sentencias SQL y PL/pgSQL
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

sintaxis_basica

Una vez creada la función hay que crear el *trigger* que es el elemento encargado de asociar la función con el evento que hará que se ejecute. La sintaxis que permite crear un *trigger* es la siguiente:

```
CREATE TRIGGER nombre_trigger { BEFORE | AFTER } { evento }
    ON tabla
    FOR EACH ROW
    EXECUTE PROCEDURE nombre_funcion ()
```

Donde:

- *nombre_trigger* es el nombre que le queremos asignar al *trigger*.
- *evento* puede tomar los siguientes valores: INSERT, UPDATE o DELETE. Esto hará que el *trigger* se dispare cuando se realice una operación de inserción, actualización o borrado respectivamente. También es posible combinar diferentes eventos mediante la cláusula OR. Si por ejemplo indicamos como envento INSERT OR UPDATE el *trigger* se disparará cuando se realice una operación de inserción o actualización.
- Las cláusulas *BEFORE* o *AFTER* sirven para indicar si queremos que la función se llame antes o después de realizar la operación.
- *tabla* es el nombre de la tabla en la que se creará el *trigger*. El *trigger* se ejecutará cuando se produzca el evento correspondiente sobre la tabla indicada.
- *FOR EACH ROW* indica que la función deberá llamarse una vez por cada fila que se quiera actualizar, insertar o borrar.
- *nombre_funcion* es el nombre de la función que queremos que se llame cuando se produzca el evento.

Para comprender mejor el funcionamiento de los *triggers* haremos un ejemplo partiendo de una tabla *empleado*:

```
CREATE TABLE empleado (
    nombre VARCHAR(20),
    salario INT
);
```

El objetivo es crear un *trigger* que se lance cada vez que se inserte una fila en la tabla empleado. El *trigger* lo único que hará será mostrar un mensaje por pantalla indicando que se ha introducido una fila en la tabla.

En primer lugar, crearemos la función:

```
CREATE OR REPLACE FUNCTION muestra_mensaje() RETURNS trigger AS $$
    DECLARE
    BEGIN
           RAISE NOTICE 'Se ha insertado una fila';
           RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

Y a continuación el *trigger*:

```
CREATE TRIGGER tg_insercion_empleado AFTER INSERT ON empleado
FOR EACH ROW EXECUTE PROCEDURE muestra_mensaje();
```

En el *trigger tg_insercion_empleado* estamos indicando que tras cada inserción sobre la tabla *empleado* se deberá llamar a la función *muestra_mensaje*.

Si realizamos una inserción sobre la tabla empleado, veremos que efectivamente se muestra el mensaje por pantalla.

```
INSERT INTO empleado VALUES ('Pepito', 2000);
NOTICE: Se ha insertado una fila
```

Si insertamos más de una fila con una misma cláusula INSERT veremos que la función se llama una vez por cada fila insertada:

```
INSERT INTO empleado VALUES ('Juanito', 2400),
                            ('Luisito', 1900);
NOTICE: Se ha insertado una fila
NOTICE: Se ha insertado una fila
```

Si en algún momento queremos borrar un *trigger* lo podemos hacer de la siguiente forma:

```
DROP TRIGGER nombre_trigger ON tabla;
```

Ejercicio 1

Crea un *trigger* que muestre por pantalla el número de filas de la tabla empleado tras insertar o borrar una fila de la tabla.

Ejercicio 2

Crea un *trigger* que muestre por pantalla el número de filas de la tabla empleado antes de insertar o borrar una fila de la tabla.

Uso de variables

Cuando utilizamos *triggers* se crean, automáticamente, una serie de variables especiales:

- NEW: variable de tipo RECORD. El contenido de la variable depende del tipo de evento que haya disparado el *trigger*. En el caso que se trate de una inserción la variable contiene los datos de la fila que se va a insertar. En el caso de que sea una actualización contiene los nuevos datos con los que se va a actualizar la fila.
- OLD: variable de tipo RECORD. El contenido de la variable depende del tipo de evento que haya disparado el *trigger*. En el caso de que sea una actualización contiene los datos antiguos de la fila que se va actualizar. En el caso de que sea un borrado contiene los datos de la fila que se va a borrar.
- TG_OP: variable de tipo TEXT. Contiene una cadena de texto cuyo valor puede ser INSERT, UPDATE o DELETE y nos indica cuál ha sido el tipo de evento que ha disparado el *trigger*.
- TG_TABLE_NAME: variable que contiene el nombre de la tabla que provocó la invocación del *trigger*.

Estas variables podemos utilizarlas dentro del bloque BEGIN END de nuestra función.

En el siguiente ejemplo se muestra una función que sirve para validar que los datos que se quieren insertar en la tabla empleado son válidos. Para ello se hace uso de la variable NEW. En el caso de que se detecte que el nombre o salario son inválidos se lanzará una excepción y la función no finalizará su ejecución.

```
CREATE OR REPLACE FUNCTION disp_empleado() RETURNS trigger AS $$
    DECLARE
    BEGIN
          IF NEW.nombre IS NULL THEN
                RAISE EXCEPTION 'El nombre no puede ser nulo';
          END IF;
          IF NEW.salario IS NULL THEN
                RAISE EXCEPTION 'El salario no puede ser nulo';
          END IF;
          IF NEW.salario < 0 THEN
                RAISE EXCEPTION 'El salario no puede ser inferior a 0';
          END IF;
          RETURN NEW;
    END
$$ LANGUAGE PLPGSQL;
```

El *trigger* lo crearemos de la siguiente forma:

```
CREATE TRIGGER disp_empleado BEFORE INSERT OR UPDATE ON empleado FOR EACH
ROW EXECUTE PROCEDURE disp_empleado();
```

Ejercicio 3

Crea un *trigger* de forma que al insertar o actualizar una fila en la tabla empleado, si el nuevo salario es inferior a 700€, éste se fijará en 700€. Es decir, no puede guardarse ningún salario por debajo de los 700€.

Ejercicio 4

Partiendo del caso práctico 1 del tema anterior crea un *trigger* de forma que cada vez que se inserte, actualice o borre una venta se actualice el importe total del vendedor correspondiente.

Ejercicio 5

Partiendo del caso práctico 2 del tema anterior crea un *trigger* de forma que solo se podrá insertar una fila en la tabla *jugador_partida* si la partida tiene menos de dos jugadores y si el jugador que se quiere insertar no juega ya a la partida.

Bibliografía

Group, T. P. (2020). *PostgreSQL 12.2 Documentation.*

PostgreSQL Tutorial. (4 de Abril de 2020). Obtenido de https://www.postgresqltutorial.com/

Atribuciones

- Portada de *Berti Weber* descargada de *www.pexels.com*
- Iconos diseñados por *Eucalyp* descargados de *www.flaticon.es*