5. DISPARADORES (TRIGGERS)

Un disparador no es otra cosa que una acción definida en una tabla de nuestra base de datos y ejecutada automáticamente por una función programada para realizar ciertas acciones. Esta acción se activara, según la definamos, cuando realicemos uno de los siguientes eventos INSERT, un UPDATE o un DELETE en una tabla.

Un disparador se puede definir de las siguientes maneras:

- Para que ocurra ANTES de cualquier INSERT, UPDATE o DELETE.
- Para que ocurra DESPUES de cualquier INSERT, UPDATE o DELETE.
- Para que se ejecute una sola vez por comando SQL.
- Para que se ejecute por cada línea afectada por un comando SQL.

5.1. Composición de un trigger.

Los trigger en PostgresSQL se definen creando primero la función trigger y luego definiendo el trigger.

Un trigger consta de dos partes:

- **Función.-** La definición de la función asociada al trigger que es la que ejecuta la acción en respuesta al evento. Este tipo de función no recibe ni un tipo de parámetro.

  Estructura de una función de trigger.

  ```
  CREATE OR REPLACE FUNCTION nombre_función() RETURNS TRIGGER AS
  $aliasFunción$
    DECLARE
          Varibles ……..
    BEGIN
          Sentencias …….
    RETURN RETORNO;
    END;
  $aliasFunción$LANGUAGE PLPGSQL;
  ```

  EL tipo de retorno siempre será un trigger, y RETORNO puede tener un valor de null, new u old.

- **Trigger.-** La definición de trigger indicara: la tabla de la cual se esperarán los eventos y el evento al cual responderá el trigger.

  ```
  CREATE TRIGGER nombre_trigger { BEFORE | AFTER } { INSERT | UPDATE | DELETE [ OR ... ] } ON nombre_tabla [ FOR [ EACH ] { ROW | STATEMENT } ] EXECUTE PROCEDURE nombre_función ();
  ```

`CREATE TRIGGER nombre_trigger`	>Define la creación y el nombre que tendrá el trigger.
`{ BEFORE | AFTER }`	>Define si el trigger se dispara antes o después de realizado l evento.
`{ INSERT | UPDATE | DELETE [ OR ... ] }`	>Tipo de evento al que responderá el trigger.

ON nombre_tabla	>Nombre de la tabla de la cual se esperan los eventos.
[FOR [EACH] { ROW | STATEMENT }]	>Para cada fila o para cada sentencia.
EXECUTE PROCEDURE nombre_función ();	>Nombre de la función que se va a ejecutar.

5.2. Momentos de un trigger.

- **Before.-** Los triggers BEFORE se utilizan cuando queremos que la validación se lleve a cabo antes de realizar el evento. Por ejemplo, ejecutar una base de datos de un banco, en esta tenemos la tabla cuentas y la tabla operaciones.
  Si el usuario hace un retiro de su cuenta, se debe asegurar de que el usuario tenga suficientes fondos en su cuenta antes de realizar el retiro.
  El trigger BEFORE permitirá hacer esta validación antes de realizar la transacción si el saldo de la cuenta no es suficiente.
- **Alter.-** Los trigger AFTER se utilizan cuando queremos almacenar información de una tabla n otra.
  Estos se ejecutan después de que se han realizado los cambios en la base de datos. Para nuestro ejemplo del banco, después de una transacción exitosa, se debe actualizar la tabla cuentas, el trigger AFTER permitirá realizar esta operación.

5.3. Características y reglas a seguir.

- El procedimiento almacenado que se vaya a utilizar por el disparador debe de definirse e instalarse antes de definir el propio disparador.
- Un procedimiento que vaya a utilizar un disparador no puede tener argumentos y tiene que devolver el tipo “trigger”.
- El mismo procedimiento almacenado se puede utilizar por múltiples disparadores en diferentes tablas.
- Procedimientos almacenados utilizados por disparadores que se ejecutan una sola vez por comando SQL (statement-level) tienen que devolver siempre NULL.
- Procedimientos almacenados utilizados por disparadores que se ejecutan una vez por línea afectada por el comando SQL (row-level) pueden devolver una fila de tabla.
- Procedimientos almacenados utilizados por disparadores que se ejecutan una vez por fila afectada por el comando SQL (row-level) y ANTES de ejecutar el comando SQL que lo lanzo, pueden retornar NULL o devolver una fila de tabla RECORD (new u old).
- Procedimientos almacenados utilizados por disparadores que se ejecutan DESPUES de ejecutar el comando SQL que lo lanzo, ignoran el valor de retorno, así que pueden retornar NULL sin problemas.
- Si una tabla tiene más de un disparador definido para un mismo evento (INSERT, UPDATE y DELETE), estos se ejecutarán en orden alfabético por el nombre del disparador. En el caso de disparadores de tipo ANTES / row-level, la fila retornada por cada disparador, se convierte en la entrada del siguiente. Si alguno de ellos retorna NULL, la operación se anulada para la fila afectada.
- Procedimientos almacenados utilizados por disparadores pueden ejecutar sentencias SQL que a su vez pueden activar otros disparadores. Esto se conoce como disparadores en cascada. Tener en cuenta que no existe límite para el número de disparadores que se pueden llamar pero es responsabilidad del programador el evitar una recursión infinita de llamadas.

5.4. Variables triggers.

- **New.-** variable compuesta que almacena los nuevos valores de la tupla (fila), se utiliza para la **inserción** y **actualización** de nuevos datos.
- **Old.-** Variables compuesta que almacena los valores antiguos de la tupla, se utiliza para la **actualización** y **eliminación** de datos.

Acción SQL	OLD	NEW
Insert	Todos los campos toman valor null.	Valores que serán insertados cuando se completa la acción.
Update	Valores originales de la fila, antes de la actualización.	Nuevos valores que serán escritos cuando se complete la orden.
Delete	Valores, antes del borrado de la fila	Todos los campos toman el valor null.

- **Tg_name.-** tipo de dato **text,** variable que contiene el nombre del disparador que está usando la función actualmente.
- **Tg_when.-** tipo de dato **text,** una cadena de texto con el valor BEFORE o AFTER dependiendo de como el disparador que está usando la función actualmente ha sido definido.
- **Tg_level.-** tipo de dato **text,** una cadena de texto con el valor ROW o STATEMENT dependiendo de como el disparador que está usando la función actualmente ha sido definido.
- **Tg_op.-** tipo de dato **text,** una cadena de texto con el valor INSERT, UPDATE O DELETE dependiendo de la operación que ha activado el operador que está usando la función actualmente.
- **Tg_table_name.-** Tipo de dato name, el nombre de la tabla que ha activado el disparador que está usando la función actualmente.
- **Current_user.-** nos muestra el nombre del usuario que está actualmente conectado a la base de datos y que ejecuta las sentencias.
- **Current_date.-** nos muestra la fecha actual del sistema.
- **Current_time.-** nos muestra la hora actual del sistema.

5.5. Triggers de auditoria (Bitácoras).

- **Bitácora.-** Permite guardar las transacciones realizadas sobre una base de datos en específico, de tal manera que estas transacciones puedan ser auditadas y analizadas posteriormente. Para un buen control de la base de datos es necesario controlar la inserción, actualización y borrado de los datos, esto para asegurar una buena auditoria de la base de datos.

  Para realizar esta tarea es necesario crear un tabla donde se almacenara todas las modificaciones realizas en la base de dato, para tener una nomenclatura y por convención el nombre de esta tabla deber ser bitacora_nombreTabla, por ejemplo, create table bitacora_proveedor(nit_pro int, fecha timestamp, tipo_registro varchar(10), usuario varchar(10));

  Existen tres tipos de registros que podemos almacenar en una bitácora:

  - Insert.- cuando se realice alguna inserción a la tabla, se creara un registro en nuestra bitácora.

Ejemplo para la tabla proveedor.

Creacion de un trigger:

```
create trigger bit_proveedor_ai after insert on proveedor for each row execute
procedure insert_bitacora_proveedor();
```

Creación de la función **insert_bitacora_proveedor()**

```
create or replace function insert_Bitacora_proveedor() returns trigger as
$$
  begin
            insert into bitacora_proveedor values(new.nit_prov, now(), 'INSERT',
            current_user);
  return new;
  end;
$$language plpgsql;
```

- Update.- cuando se realice alguna actualización de los datos, se creara un registro en nuestra bitácora.

```
create table bitacora_cliente(numero serial,ci_cli int, fecha date,hora time,
tipo_registro varchar(10), usuario varchar(10));

create or replace function update_bitacora_cliente() returns trigger as
$$
  begin
            insert into bitacora_cliente(ci_cli,fecha,hora,tipo_registro,usuario)
values(new.ci_cli,current_date,current_time,Tg_op,current_user);
  return null;
  end;
$$language plpgsql;

create trigger bit_cliente_au after update on cliente for each row execute procedure
update_bitacora_cliente();

Update cliente set appaterno_cli='Garcia' Where ci_cli=3566487;
```

- Delete.- cuando se realice una eliminación de los datos, se creara un registro en nuestra bitácora.

5.6. Validación de valores.

```
create or replace function negativo_numcasa_prov() returns trigger as
$$
        declare
        tipo varchar(10):=tg_op;
  begin
            if new.dir_numcasa_prov <= 0 then
```

```
            raise exception 'El valor % no se puede ingresar porque es
            negativo',new.dir_numcasa_prov;
        else
            if upper(tipo) = 'INSERT' then
            raise notice 'El valor trigger % --> La fila se ingresó correctamente', tg_name;
          else
            raise notice 'El valor trigger % --> La fila se actualizo correctamente', tg_name;
            end if;
        end if;
    return new;
    end;
$$language plpgsql;

create trigger tri_validar_numcasa_prov after insert or update on proveedor for each row
execute procedure negativo_numcasa_prov();
Update proveedor set dir_numcasa_prov=11 Where nit_prov=7854;
```