Triggers Disparadores

Un disparador es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular.

Sintaxis de CREATE TRIGGER

```
CREATE TRIGGER nombre_disp momento_disp evento_disp ON nombre_tabla
FOR EACH ROW sentencia_disp;
```

`momento_disp` es el momento en que el disparador entra en acción. Puede ser BEFORE (antes) o AFTER (despues), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.

`evento_disp` indica la clase de sentencia que activa al disparador. Puede ser INSERT, UPDATE, o DELETE. Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia.

Por ejemplo, no se pueden tener dos disparadores BEFORE UPDATE. Pero sí es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE. sentencia_disp es la sentencia que se ejecuta cuando se activa el disparador. Si se desean ejecutar múltiples.

Las siguientes sentencias crean una DB con sus tablas y un disparador para sentencias INSERT dentro de la tabla.

```
CREATE DATABASE ejtrigger;
USE ejtrigger;
CREATE TABLE cliente( id SERIAL, nombre VARCHAR(255),
id_ultimo_pedido BIGINT );
CREATE TABLE ventas ( id SERIAL, id_articulo BIGINT, id_cliente
BIGINT, cantidad INT, precio DECIMAL(9,2) );

INSERT INTO cliente(nombre) VALUES ('Bob');
INSERT INTO cliente(nombre) VALUES ('Sally');
INSERT INTO cliente(nombre) VALUES ('Fred');
```

```
SELECT * FROM cliente;
```

Creamos el Trigger

```
CREATE TRIGGER nuevasventas AFTER INSERT ON ventas
    FOR EACH ROW
    UPDATE cliente SET id_ultimo_pedido = NEW.id
    WHERE id = NEW.id_cliente
;
```

Y procedemos a insertar datos en ventas ,

```
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio)
VALUES (1001, 3, 5, 19.95);
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio)
VALUES (1002, 2, 3, 14.95);
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio)
VALUES (1003, 1, 1, 29.95);
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio)
VALUES (1002, 3, 6, 29.95);
```

Consultamos tablas y observamos cambios,

```
SELECT * FROM ventas;

SELECT * FROM cliente;
```

Ejemplo Prevenir cambios con Triggers

```
DROP TABLE IF EXISTS ventas;

CREATE TABLE ventas ( id SERIAL, id_articulo BIGINT, id_cliente
BIGINT, cantidad INT, precio DECIMAL(9,2), bandera INT );

INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio,
bandera) VALUES (1001, 3, 5, 19.95, 0);
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio,
bandera) VALUES (2002, 2, 3, 14.95, 1);
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio,
bandera) VALUES (3003, 1, 1, 29.95, 0);

SELECT * FROM ventas;
```

Creamos Trigger,

```
DELIMITER //
CREATE TRIGGER actualizarventas BEFORE UPDATE ON ventas
 FOR EACH ROW
 BEGIN
   IF ( SELECT bandera FROM ventas WHERE id = NEW.id ) > 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: no se
      puedo actualizar el registro cantidad en ventas';
   END IF;
 END;
//
DELIMITER ;
```

El comando ‘DELIMITER’ permite que utilices el (;) para finalizar una sentencia sin necesariamente haber terminado la sentencia.

Si se desean ejecutar múltiples sentencias dentro de un disparador se deben colocarse entre BEGIN ... END.

SIGNAL funciona a partir de la versión 4.5 de mysql.

SQLSTATE '45000' es una excepción genérica controlada que nos permite establecer un mensaje de texto.

Ahora vamos a actualizar un registro en la tabla ventas para probar el disparador, se realiza en una transacción para que solo actualice datos si se cumple con la restricción anterior.

```
START TRANSACTION;
UPDATE ventas SET cantidad = cantidad + 9 WHERE id = 2;
COMMIT;

SELECT * FROM ventas;
```

```
START TRANSACTION;
UPDATE ventas SET cantidad = cantidad + 9 WHERE id = 3;
COMMIT;

SELECT * FROM ventas;
```

Registrar varias sentencias con un disparador

Primero vamos a realizar los siguientes cambios en el esquema ejtrigger.

```
USE ejtrigger;

DROP TABLE IF EXISTS ventas;

CREATE TABLE ventas ( id SERIAL, id_articulo BIGINT, id_cliente
BIGINT, cantidad INT, precio DECIMAL(9,2) );

CREATE TABLE registro ( id SERIAL, marca TIMESTAMP, evento
VARCHAR(255), nombreusuario VARCHAR(255),nombretabla VARCHAR(255),
id_tabla BIGINT);
```

Creamos un nuevo trigger,

```
DELIMITER //
CREATE TRIGGER marcaVenta AFTER INSERT ON ventas
    FOR EACH ROW
    BEGIN
        UPDATE cliente SET id_ultimo_pedido = NEW.id
            WHERE cliente.id = NEW.id_cliente;
        INSERT INTO registro (evento, nombreusuario, nombretabla,
            id_tabla)
        VALUES ('Inserto', 'el trigger', 'ventas', NEW.id);
    END
//
DELIMITER ;
```

Descripción: Al insertar datos en la tabla ventas, automáticamente se **actualiza** la tabla cliente en su campo id_ultimo_pedido, en el cual se agrega el id que se acabó de insertar en ventas, para un único cliente. Además al insertar datos en la tabla ventas se **inserta** una nueva tupla en la tabla registros.

Insertamos nuevo datos en la tabla ventas.

```
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio)
VALUES (1, 3, 5, 19.95);
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio)
VALUES (2, 2, 3, 14.95);
INSERT INTO ventas (id_articulo, id_cliente, cantidad, precio)
VALUES (3, 1, 1, 29.95);
```

Consultamos para observar cambios.

```
SELECT * FROM ventas;
SELECT * FROM cliente;
SELECT * FROM registro;
```

Para Borrar Triggers se usa,

```
DROP TRIGGER IF EXISTS nuevasventas;
```

Otro Ejemplo

Primera Parte….

```
CREATE DATABASE cuenta;
USE cuenta;
CREATE TABLE consigna (cod INT PRIMARY KEY AUTO_INCREMENT,
cantidad DECIMAL(15,2));

CREATE TABLE saldo (cod INT PRIMARY KEY AUTO_INCREMENT,
nuevo_saldo DECIMAL(15,2));


CREATE TRIGGER sum_saldo AFTER INSERT ON consigna
    FOR EACH ROW
       INSERT INTO saldo (nuevo_saldo) VALUES (NEW.cantidad);

INSERT INTO consigna (cantidad) VALUES (30000);
```

Segunda Parte…

Borramos Trigger anterior,

```
DROP TRIGGER sum_saldo;
```

Cambiamos trigger

```
CREATE TRIGGER sum_saldo AFTER INSERT ON consigna
    FOR EACH ROW
    UPDATE saldo SET nuevo_saldo = nuevo_saldo + NEW.cantidad
    WHERE cod = 1
;

INSERT INTO consigna (cantidad) VALUES (30000);

SELECT * FROM saldo;
```

Pero y si no existiera saldo anterior? Funciona?

Una posible solución seria.

```
DELIMITER //
CREATE TRIGGER sum_saldo AFTER INSERT ON consigna
    FOR EACH ROW
    BEGIN
        IF ( SELECT COUNT(nuevo_saldo) FROM saldo) > 0 THEN
            UPDATE saldo SET nuevo_saldo = nuevo_saldo +
            NEW.cantidad WHERE cod = 1;
        ELSE
            INSERT INTO saldo (nuevo_saldo)
            VALUES ((SELECT SUM(cantidad) FROM consigna));
        END IF;
    END
//
DELIMITER ;

INSERT INTO consigna (cantidad) VALUES (20000);

SELECT * FROM saldo;
```

OLD, NEW

Las palabras clave OLD y NEW permiten acceder a columnas en los registros afectados por un disparador. (OLD y NEW no son sensibles a mayúsculas).

En un disparador para INSERT, solamente puede utilizarse NEW.nom_col; ya que no hay una versión anterior del registro.

En un disparador para DELETE sólo puede emplearse OLD.nom_col, porque no hay un nuevo registro.

En un disparador para UPDATE se puede emplear OLD.nom_col para referirse a las columnas de un registro antes de que sea actualizado, y NEW.nom_col para referirse a las columnas del registro luego de actualizarlo.

Vamos a crear una nueva base de datos con sus respectivas tablas y datos.

```
CREATE DATABASE almacen;

USE almacen;

CREATE TABLE factura (
  id_factura INT NOT NULL,
  total_factura FLOAT NOT NULL,
  fecha DATE NOT NULL,
  PRIMARY KEY (id_factura)
);

INSERT INTO factura VALUES(1, 160000, '2013-11-02');

CREATE TABLE detalle_factura (
  id_detalle INT NOT NULL,
  id_factura INT NOT NULL,
  id_articulo INT NOT NULL,
  cantidad INT NOT NULL,
  precio FLOAT NOT NULL,
  total_detalle FLOAT NOT NULL,
  PRIMARY KEY (id_detalle,id_factura)
);
```

```
INSERT INTO detalle_factura VALUES(1, 1, 2, 3, 20000, 60000);
INSERT INTO detalle_factura VALUES(2, 1, 4, 2, 10000, 20000);
INSERT INTO detalle_factura VALUES(3, 1, 3, 4, 20000, 80000);

SHOW TABLES:

SELECT * FROM facture;
SELECT * FROM detalle_factura;
```

Lo que se pretende es actualizar en la tabla factura el campo total_factura con triggers, si se Inserta INSERT, Borra DELETE o actualice UPDATE la información en la tabla detalle_factura.

Para INSERT.

```
DELIMITER //
CREATE TRIGGER inserto BEFORE INSERT ON detalle_factura
    FOR EACH ROW
    BEGIN
        SET NEW.total_detalle = NEW.precio * NEW.cantidad;
        UPDATE factura
        SET total_factura = total_factura + NEW.total_detalle
        WHERE id_factura = NEW.id_factura;
    END
//
DELIMITER ;
```

```
INSERT INTO detalle_factura VALUES(4, 1, 8, 2, 5000, 10000);

SELECT * FROM factura;
```

Para DELETE.

```
DELIMITER //
CREATE TRIGGER borro AFTER DELETE ON detalle_factura
    FOR EACH ROW
    BEGIN
        UPDATE factura
        SET total_factura = total_factura - OLD.total_detalle
        WHERE id_factura = OLD.id_factura;
    END
//
DELIMITER ;


DELETE FROM detalle_factura WHERE id_detalle = 3 AND id_factura = 1;


SELECT * FROM factura;
```

Para UPDATE,

```
DELIMITER //
CREATE TRIGGER actualizo BEFORE UPDATE ON detalle_factura
    FOR EACH ROW
    BEGIN
        -- declaración de variable
        DECLARE v_variacion FLOAT;

        -- calculos
        SET NEW.total_detalle = NEW.precio * NEW.cantidad;
        SET v_variacion = NEW.total_detalle - OLD.total_detalle;

        -- actualizamos el total factura
        UPDATE factura
        SET total_factura = total_factura + v_variacion
        WHERE id_factura = NEW.id_factura;
    END
//
DELIMITER ;

UPDATE detalle_factura SET precio=7000, total_detalle= 14000
WHERE id_detalle = 2 AND id_factura = 1;

SELECT * FROM factura;
```

Ejercicio:

Supongamos que usted gestiona una base de datos de una empresa que distribuye una gran variedad de productos, por lo que el maestro de productos de esta BD es una gran tabla que contiene cientos de miles de registros. Para cada producto que cambia de precio debe realizarse un cálculo un tanto pesado, esto se realiza en un proceso nocturno todos los días.

Sabiendo que el programa nocturno procesa solo aquellos registros de la tabla **productos** cuyo campo **rectangular** contiene una “S” y que a su vez finaliza el cálculo actualiza el campo **rectangular** con una “N”.

Construye un disparador sobre la tabla **productos** para que cuando cambie el valor del campo **precio** marque el registro para su recalculo guardando una “S” en el campo **rectangular**.