¿Qué utilidades tienen los Triggers?

Con los Triggers podemos implementar varios casos de uso que mantengan la integridad de la base de datos, como ***Validar información***, ***Calcular atributos derivados***, ***Seguimientos de movimientos en la base de datos***, etc.

Cuando surja una necesidad en donde veas que necesitas que se ejecute una acción implícitamente (sin que la ejecutes manualmente) sobre los registros de una tabla, entonces puedes considerar el uso de un Trigger.

Ejemplo de Trigger BEFORE en la sentencia UPDATE

A continuación veremos un Trigger que valida la edad de un cliente antes de una sentencia `UPDATE`. Si por casualidad el nuevo valor es negativo, entonces asignaremos `NULL` a este atributo.

```
DELIMITER //

CREATE TRIGGER cliente_BU_Trigger

BEFORE UPDATE ON cliente FOR EACH ROW

BEGIN

-- La edad es negativa?

IF NEW.edad<0 THEN

SET NEW.edad = NULL;

END IF;

END// DELIMITER ;
```

Este **Trigger** se ejecuta antes de haber insertado el registro, lo que nos da el poder de verificar primero si el nuevo valor de la edad esta en el rango apropiado, si no es así entonces asignaremos `NULL` a ese campo. **Grandes los Triggers!**

Ejemplo de Trigger AFTER en la sentencia UPDATE

Supongamos que tenemos una ***Tienda de accesorios para Gamers***. Para la actividad de nuestro negocio hemos creado un **sistema de facturación** muy sencillo, que registra las ventas realizadas dentro de una factura que contiene el detalle de las compras.

Nuestra tienda tiene 4 vendedores de turno, los cuales se encargan de registrar las compras de los clientes en el horario de funcionamiento.

Implementaremos un Trigger que guarde los cambios realizados sobre la tabla `DETALLE_FACTURA` de la base de datos realizados por los vendedores.

Veamos la solución:

```
DELIMITER //
```

```
CREATE TRIGGER detalle_factura_AU_Trigger

AFTER UPDATE ON detalle_factura FOR EACH ROW

BEGIN

INSERT INTO log_updates

(idusuario, descripcion)

VALUES (user( ),

CONCAT('Se modificó el registro ','(',

OLD.iddetalle,',', OLD.idfactura,',',OLD.idproducto,',',

OLD.precio,',', OLD.unidades,') por ',

'(', NEW.iddetalle,',', NEW.idfactura,',',NEW.idproducto,',',

NEW.precio,',', NEW.unidades,')'));



END//



DELIMITER ;
```

Con este registro de ***logs*** podremos saber si algún vendedor "ocioso" esta alterando las facturas, lo que lógicamente sería atentar contra las finanzas de nuestro negocio. Cada registro nos informa el usuario que modificó la tabla `DETALLE_FACTURA` y muestra una descripción sobre los cambios en cada columna.

Ejemplo de Trigger BEFORE en al sentencia INSERT

El siguiente ejemplo muestra como mantener la integridad de una base de datos con respecto a un atributo derivado.

Supón que tienes una ***Tienda de electrodomésticos*** y que has implementado un **sistema de facturación**. En la base de datos que soporta la información de tu negocio, existen varias tablas, pero nos vamos a centrar en la tabla `PEDIDO` y la tabla `TOTAL_VENTAS`.

`TOTAL_VENTAS` almacena las ventas totales que se le han hecho a cada cliente del negocio. Es decir, si el cliente **Armado Barreras** en una ocasión compró 1000 dolares, luego compró 1250 dolares y hace poco ha vuelto a comprar 2000 dolares, entonces el total vendido a este cliente es de 4250 dolares.

Pero supongamos que eliminamos el ultimo pedido hecho por este cliente, ¿que pasaría con el registro en `TOTAL_VENTAS`?,...**¡exacto!**, quedaría desactualizado.

Usaremos tres Triggers para solucionar esta situación. Para que cada vez que usemos un comando DML en la tabla `PEDIDO`, no tengamos que preocuparnos por actualizar manualmente `TOTAL_VENTAS`.

Veamos:

```
-- TRIGGER PARA INSERT

DELIMITER //

CREATE TRIGGER PEDIDO_BI_TRIGGER

BEFORE INSERT ON PEDIDO

FOR EACH ROW

BEGIN

DECLARE cantidad_filas INT;

SELECT COUNT(*)

INTO cantidad_filas

FROM TOTAL_VENTAS

WHERE idcliente=NEW.idcliente;

IF cantidad_filas > 0 THEN

UPDATE TOTAL_VENTAS

SET total=total+NEW.total

WHERE idcliente=NEW.idcliente;

ELSE

INSERT INTO TOTAL_VENTAS

(idcliente,total)

VALUES(NEW.idcliente,NEW.total);

END IF;

END//



-- TRIGGER PARA UPDATE

CREATE TRIGGER PEDIDO_BU_TRIGGER

BEFORE UPDATE ON PEDIDO

FOR EACH ROW

BEGIN

UPDATE TOTAL_VENTAS

SET total=total+(NEW.total-OLD.total)
```

```
WHERE idcliente=NEW.idcliente;

END//



-- TRIGGER PARA DELETE

CREATE TRIGGER PEDIDO_BD_TRIGGER

BEFORE DELETE ON PEDIDO

FOR EACH ROW

BEGIN

UPDATE TOTAL_VENTAS

SET total=total-OLD.total

WHERE idcliente=OLD.idcliente;

END//
```

Con todos ellos mantendremos el total de ventas de cada cliente actualizado dependiendo del evento realizado sobre un pedido.

Si insertamos un nuevo pedido generado por un cliente existente, entonces vamos rápidamente a la tabla `TOTAL_VENTAS` y actualizamos el total comprado por ese cliente con una sencilla suma acumulativa.

Ahora, si cambiamos el monto de un pedido, entonces vamos a `TOTAL_VENTAS` para descontar el monto anterior y adicionar el nuevo monto.

Y si eliminamos un pedido de un cliente simplemente descontamos del total acumulado el monto que con anterioridad habíamos acumulado. **¿Práctico verdad?**

Dada la siguiente base de datos: