1. Disparadores

1.1. Disparadores (*triggers*)

Un disparador o *trigger* es un programa almacenado que está asociado a una tabla. El disparador está formado por un conjunto de sentencias que son ejecutadas cuando sucede un determinado evento en la tabla a la que está asociado. Los eventos que activan los disparadores están relacionados con operaciones de manipulación de datos. Su uso ayuda a mantener la integridad de los datos haciendo operaciones como validar datos, calcular atributos derivados, llevar registro de las operaciones que se hacen sobre los datos, etc.

1.1.1. Sentencia CREATE TRIGGER

La sentencia CREATE TRIGGER permite crear un disparador, dándole un nombre, estableciendo cuál es el evento disparador, la tabla a la que está asociado, y escribiendo las sentencias a ejecutar cuando suceda el evento. La sintaxis de la sentencia es:

```
CREATE
    [DEFINER = user]
    TRIGGER [IF NOT EXISTS] trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

- La cláusula DEFINER permite indicar el nombre del usuario que va a ser considerado el creador del disparador. El valor por defecto es CURRENT_USER, que hace referencia al usuario actual que está creando el disparador.
- *trigger_name*: es el nombre a darle al disparador. Tiene que ser único en la base de datos.
- *trigger_time*: indica el momento en que se activa o ejecuta el disparador. Puede tomar los valores:
  - BEFORE: El disparador se activa antes de que se ejecute la operación de manipulación de datos.
  - AFTER: El disparador se activa después de que se ejecute la operación de manipulación de datos.
- *trigger_event*: indica cuál es la operación que activa el disparador. Los valores permitidos son:

- INSERT: El disparador se activa cuando se inserta una nueva fila en la tabla.
- UPDATE: El disparador se activa cuando se modifica una fila en la tabla.
- DELETE: El disparador se activa cuando se borra una fila en la tabla.

Por ejemplo: BEFORE INSERT haría que el disparador se active, antes de ejecutar una sentencia INSERT sobre la tabla, y puede ser utilizado para verificar que los valores a introducir son correctos.

- *tbl_name*: especifica el nombre de la tabla a la que está asociado el disparador. No puede ser una tabla temporal, ni una vista.
- El texto FOR EACH ROW que va antes del cuerpo del disparador hace referencia a que esas sentencias se van a ejecutar por cada fila de la tabla afectada por el evento definido anteriormente.
- *trigger_order*: sirve para establecer un orden de ejecución de aquellos disparadores que tienen el mismo evento y tiempo de ejecución. Por ejemplo, podemos tener dos disparadores BEFORE UPDATE para una misma tabla. Por defecto, estos disparadores se ejecutarían en el orden en el que fueron creados. Si queremos definir otro orden de ejecución, podemos utilizar:
  - FOLLOWS: el nuevo disparador se activa después del otro disparador especificado.
  - PRECEDES: el nuevo disparador se activa antes del disparador especificado.
- *trigger_body*: es el conjunto de sentencias que conforman este programa almacenado. Si hay más de una sentencia, irán dentro de un bloque BEGIN … END.

Un disparador no se puede modificar una vez creado, es decir, no existe una sentencia ALTER TRIGGER. En el caso de que se quisiera crear un nuevo disparador y ya existiese otro disparador asociado a la tabla que coincida en el momento de ejecución y en el evento del disparador, es necesario borrar el disparador que ya está creado y crear el nuevo incluyendo en su cuerpo todas las sentencias del disparador que ya existía y las que corresponden al nuevo.

1.1.2. Sentencias SHOW TRIGGERS y SHOW CREATE TRIGGER

La información sobre los disparadores creados se guarda en el Diccionario de Datos, al igual que el resto de objetos de la bases de datos. En MySQL, la información sobre los disparadores se puede consultar en *information_schema.triggers* mediante una sentencia SELECT:

```
select * from information_schema.triggers;
```

MySQL dispone, además, de dos instrucciones para consultar información sobre los disparadores.

- La sentencia **SHOW TRIGGERS**, permite ver los disparadores asociados a una base de datos. La sintaxis es:

```
SHOW TRIGGERS
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

En el caso de no especificar el nombre de la base de datos (*db_name*), se mostrarán los disparadores asociados a la base de datos activa. Se pueden utilizar cláusulas LIKE o WHERE para mostrar solo los disparadores que cumplan una condición, y no todos. Por ejemplo:

```
show triggers from tenda where character_set_client = 'utf8';
```

- La sentencia **SHOW CREATE TRIGGER** permite ver información de cómo fue creado el disparador, incluido el código SQL utilizado para la creación. La sintaxis es:

```
SHOW CREATE TRIGGER trigger_name
```

Simplemente tenemos que indicar el nombre del disparador en el campo *trigger_name*.

1.1.3. Identificadores NEW y OLD en disparadores

Es obligatorio utilizar los identificadores NEW o OLD delante del nombre de la columna cuando en el cuerpo del disparador se tenga que hacer referencia a alguna columna de la tabla a la que está asociado. Por ejemplo: *NEW.nombreColumna* hace referencia al nuevo valor que toma la columna después de ejecutar una sentencia INSERT o UPDATE.

- Cuando el evento disparador es una operación INSERT, sólo puede utilizarse el identificador NEW para hacer referencia a los valores de las columnas de la nueva fila que se está insertando.
- Cuando el evento disparador es una operación DELETE, sólo puede utilizarse el identificador OLD para hacer referencia a los valores que tenían las columnas de la fila que se está borrando.
- Cuando el evento disparador es una operación UPDATE, se puede utilizar el identificador NEW para hacer referencia a los valores que toman las columnas después de hacer la modificación, y el identificador OLD para hacer referencia a los valores que tenían las columnas antes de la modificación.

1.1.4. Uso de disparadores

Una vez creados los disparadores, quedan almacenados en el servidor y se activan de manera automática cada vez que se ejecuta la operación a la que están asociados.

Algunos casos en los que pueden ser de utilidad los disparadores:

- Validar los datos que se introducen en las tablas, verificando que cumplan las restricciones impuestas por el modelo, antes de que se ejecute una sentencia INSERT sobre la tabla.

  Ejemplo: cuando se hace una venta, antes de guardar una línea de detalle de la venta con una sentencia INSERT hay que comprobar las unidades que hay en stock para el artículo a vender.

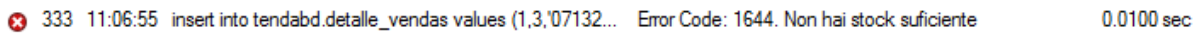
```
use tenda;
drop trigger if exists detalle_vendasBI;
delimiter //
```

```
create trigger detalle_vendasBI before insert on detalle_vendas
for each row
begin
declare vStockActual smallint;
set vStockActual = (select art_stock
                    from artigos
                    where art_codigo = new.dev_artigo);
if new. dev_cantidade > vStockActual then
  signal sqlstate '45000' set message_text = 'No hay stock suficiente';
end if;
end
//
delimiter ;
```

Antes de que se inserte una fila en la tabla *detalle_vendas*, se comprueba si la cantidad pedida es mayor que el número de unidades que hay en el almacén. En el caso de no ser suficiente, se abortará la inserción provocando un error que se asocia a un código SQLSTATE, en este caso 45000, y se mostrará un mensaje de error. Se puede hacer la prueba ejecutando una sentencia INSERT como la siguiente:

```
insert into tenda.detalle_vendas  values (1,3,'0713242',50,300.50,0);
```

Que provocaría un error con el mensaje “No hay stock suficiente”:



- Actualizar atributos derivados con la información recogida de una operación de actualización (INSERT, UPDATE o DELETE) en una tabla.

  Ejemplo: cuando se inserta una nueva fila en la tabla *detalle_vendas* hay que restar en la columna *stock* de la tabla de *artigos* el número de unidades que se venden, después de comprobar el stock de ese artículo.

```
use tenda;
drop trigger if exists detalle_vendasAI;
delimiter //
create trigger detalle_vendasAI after insert on detalle_vendas
for each row
begin
update artigos
set art_stock = art_stock - new.dev_cantidade
where art_codigo = new.dev_artigo;
end
//
delimiter ;
```

Después de insertar una fila en la tabla *detalle_vendas*, se hace una actualización de la columna *art_stock* en la tabla de *artigos*, restándole el contenido de la columna *dev_cantidade* de la tabla *detalle_vendas* en el artículo que tiene como código el valor almacenado en la columna *dev_artigo*. Hay que tener en cuenta que ya está creado el disparador del ejemplo anterior, por lo que antes de insertar la fila en la tabla *detalle_vendas*, ya se comprobó que el stock actual es suficiente para ese artículo. Se puede hacer la prueba ejecutando una sentencia INSERT como la siguiente:

```
insert into tenda.detalle_vendas  values (1,3,'0713242',1,300.50,0);
```

Si comprobamos la columna *art_stock* del artículo con código 0713242, veremos que hay una unidad menos en stock tras la inserción.

- Crear tablas de auditoría que recojan los cambios que los usuarios hacen en las tablas de una base de datos. De este modo se puede saber quién hizo cambios en ella y en qué momento. Ello puede ser de gran ayuda para llevar el registro de operaciones que obliga la Ley Orgánica de Protección de Datos (LOPD) para ciertos datos sensibles. Para poder llevar el registro de las operaciones que harán los usuarios en cada tabla hay que crear tres disparadores asociados a cada tabla, que se ejecuten después de hacer cada operación de manipulación de datos (INSERT, UPDATE o DELETE).

  Ejemplo: Llevar un registro de todos los cambios que se hagan en la columna *clt_desconto* de la tabla *clientes*. Cada vez que se hace un cambio en el contenido de la columna, se guarda una fila en la tabla *rexistro_cambios_desconto* con el nombre del usuario conectado, la fecha y hora en que se hace el cambio, valor de la columna antes de la modificación y el valor de la columna después de la modificación.

```
use tendabd;

-- creación de la tabla de registro si no existe
create table if not exists rexistro_cambios_desconto  (
   usuario varchar(80),
   dataHora datetime default now(),
   valorVello tinyint unsigned,
   valorNovo tinyint unsigned
);

drop trigger if exists clientesAU;
delimiter //
-- creación del disparador
create trigger clientesAU after update on clientes
for each row
begin
if new.clt_desconto != old.clt_desconto then
```

```
  insert into rexistro_cambios_desconto (usuario, valorVello, valorNovo)
values (user(), old.clt_desconto, new.clt_desconto);
end if;
end
//
delimiter ;
```

Lo primero que hace el disparador es comprobar si fue modificado el contenido de la columna *clt_desconto*, es decir, si el contenido de la columna antes de ejecutar la sentencia UPDATE sobre la tabla *clientes* es distinto del contenido de la columna después de ejecutar la sentencia UDPDATE. Sólo en el caso de que los valores fuesen distintos, se ejecuta una sentencia INSERT en la tabla de registro. La fecha y hora del sistema se guarda en la columna *dataHora* como valor por defecto. Se puede hacer la prueba ejecutando una sentencia UPDATE como la siguiente:

```
update clientes
  set clt_desconto = 5
  where clt_id = 1;
-- ver el contenido de la tabla de registro
select * from rexistro_cambios_desconto;
```

1.1.5. Sentencia DROP TRIGGER

La sentencia DROP TRIGGER permite borrar un disparador. La sintaxis es:

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

Simplemente debemos indicar el nombre del disparador a borrar. Cuando se borra una base de datos se borran todos los disparadores asociados a ella.

1.2. Documentación oficial

En el siguiente enlace podemos encontrar la documentación oficial de todas las sentencias vistas en este documento:

https://dev.mysql.com/doc/refman/8.0/en/sql-data-definition-statements.html