Módulo:	**Bases de Datos**
UT06:	**Programación de BD**
Actividad de desarrollo:	**Trigger**

1. Objetivo general

Aprender a construir triggers en MySQL.

2. Metodología y material.

Máquina virtual con el sistema operativo Ubuntu y MySQL, Workbench y/o phpMyAdmin instalado. Consultar apuntes de la asignatura en el campus.

IMPORTANTE: En esta práctica es necesario tener la base de datos FilmStore cargada

3. Descripción:

Usando el software MySQL Workbench crear scripts que realice los siquientes puntos:

1. TRIGGER BEFORE EN LA SENTENCIA UPDATE

a. Utilizando la BD FilmStore genera un trigger que valide el código postal de un cliente antes de una sentencia UPDATE. Si por casualidad el nuevo valor es negativo, entonces asignaremos NULL a este atributo.

```
1   Delimiter //
2 • create trigger clientes_BU_trigger_CP before
3   update on clientes for each row
4       begin
5       if (new.codigoPostal < 0)then
6       set new.codigoPostal = null;
7       END IF ;
8       end //
9       delimiter ;
```

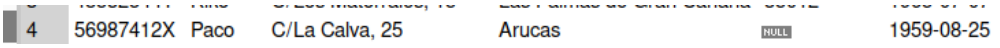
b. Comprobar si se ha creado bien (SHOW CREATE TRIGGER clientes_BU_Trigger_CP;)

#	Trigger	sql_mode	SQL Original Statement	character_set_clier	collation_con
1	clientes_BU_trigger_CP	ONLY_FULL_GROUP_BY,STRICT…	CREATE DEFINER=`root`@`localh… if (new.codigoPostal < 0)then set new.codigoPostal = null; END IF ; end	utf8mb4	utf8mb4_0900

c. TEST: Actualizar el código postal (cp) a -1 del cliente con dni 56987412X.

```
Update clientes set codigoPostal = -1 where dniCliente="56987412X";
```

d. Comprobar que el trigger se ha activado y aparece NULL en el CP del cliente con dni 56987412X.



4	56987412X	Paco	C/La Calva, 25	Arucas	NULL	1959-08-25

e. Modificar el trigger clientes_BU_Trigger_CP para que asigne NULL al CP si el valor introducido no está entre 0 y 52.999

```
1  •     Drop trigger clientes_BU_trigger_CP;
2     Delimiter //
3  •  create trigger clientes_BU_trigger_CP before
4     update on clientes for each row
5         begin
6         if (new.codigoPostal < 0 || new.codigoPostal > 52.999)then
7         set new.codigoPostal = null;
8         END IF ;
9         end //
10        delimiter ;
```

f. TEST: Actualizar el código postal (cp) a 58500 del cliente con dni 22. (Como no está este dni se lo cambio a manuel

```
1   update clientes set codigoPostal=58500 where nombre="Manuel";
```

g. Comprobar que el trigger se ha activado y aparece NULL en el CP del cliente con dni 56987412X.

1	43535545C	Manuel	NULL	Telde	NULL	1985-10-01

h. Crear un trigger para que cuando se modifique el precio de una película a un valor menor que 0 o mayor que 30 lo vuelva a dejar con el valor antiguo. (Como si no lo hubiese modificado)

```
delimiter //
create trigger peliculas_BU_trigger_Pre before
update on peliculas for each row
    begin
    if (new.precio <0 and new.precio>30)then
    set new.precio = old.precio;
    END IF ;
    end //
    delimiter ;
```

i. **TEST: Actualizar el precio a 40 del código de película 1**

```
2   delimiter //
3   create trigger peliculas_BU_trigger_Pre before
4   update on peliculas for each row
5       begin
6       if (new.precio <0 || new.precio>30)then
7       set new.precio = old.precio;
8       END IF ;
9       end //
10      delimiter ;
11
```

j. **Comprobar que el trigger se ha activado y el precio no se ha modificado.**

1	1	El Irlandés	2020-04-22	1		10	30.05

Nose si ese era el precio original, pero me falló a la primera modifiqué el trigger y ya me funcionó bien

2. TRIGGER AFTER EN LA SENTENCIA UPDATE

a. Para la actividad del negocio se ha creado un sistema de facturación sencillo, que registra las compras realizadas dentro de una tabla que contiene el detalle de las compras.

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

c. Crear una tabla: log_updates(idUser, descripción). *NOTA: idUser es el usuario que está ejecutando en ese momento la sentencia update.*

```
create table log_updates (idUser varchar(50) primary key, descripcion
varchar(90));
```

d. Crear el Trigger compras_AU_Trigger. Implementa un trigger que guarde los cambios realizados por los vendedores sobre la tabla COMPRAS en una nueva tabla que llamaremos

```
2    delimiter //
3 •  create trigger compras_AT_trigger_upd AFTER
4    update on compras for each row
5        begin
6        if (old.dniCliente != new.dniCliente)then
7        insert into log_updates values ((current_user()), (concat(' se ha cambiado el
8        ,new.dniCliente)));
9        ELSEif (old.fechaCompra != new.fechaCompra)then
10       insert into log_updates values ((current_user()), (concat(' se ha cambiado la
11       ,new.fechaCompra)));
12       END IF ;
13       end //
14       delimiter ;
15 •    create table log_updates (idUser varchar(50) primary key, descripcion
```

Con este registro de logs sabemos si algún vendedor está alterando las facturas. Cada registro informa el usuario que modificó la tabla COMPRAS y muestra una descripción sobre los cambios en cada columna.

e. **TEST: Modificar cualquier dato de la tabla compras y verificar si se agrega un registro con los cambios en la tabla log_updates.**

```
update compras set fechaCompra = "2022-01-01 00:00:00" where fechaCompra="2023-01-01 00:00:00";
```

Result Grid | Filter Rows: | Edit: | Export/Impo

#	idUser	descripcion
1	root@localhost	se ha cambiado la fecha de compra a 2022-01-01 00:00:00
*	NULL	NULL

No se puede cambiar ni el codPelicula ni el DNI ya que el primero es una clave primaria y el segundo es una clave foránea y no nos permite cambiarlo.

3. TRIGGER BEFORE en la sentencia INSERT

a. **A continuación, conocerás como mantener la integridad de una base de datos con respecto a un atributo derivado.**

b. **Crea una tabla llamada: TOTAL_COMPRAS en ella se almacenarán las ventas totales que se han hecho a cada cliente del negocio, es decir, la cantidad invertida por cada cliente.**

EJ: Si el cliente Pedro Sánchez en una ocasión compró 10€, luego compró 12€ y hace poco ha vuelto a comprar 10€, entonces el total vendido a este cliente es de 32€.

```
create table total_compras (dniCliente varchar(9) primary key, precioGastado double);
insert into total_compras (select compras.dniCliente,SUM(peliculas.precio) from compras
inner join peliculas on peliculas.codPelicula=compras.codPelicula group by dniCliente);
```

c. **Vamos a suponer ahora que eliminamos la última compra realizada por este cliente, ¿qué pasaría con el registro en TOTAL_COMPRAS? *¿quedaría desactualizado no?***

Si, quedaría desactualizado

d. **Para solucionar esta situación vamos a crear tres Triggers. Para que cada vez que use un comando DML en la tabla COMPRAS, no tenga que preocuparse por actualizar manualmente TOTAL_COMPRAS**

i. **Crear los triggers compras_BI_Trigger, compras_BU_Trigger y compras_BD_Trigger**

```
delimiter //
create trigger peliculas_BI_trigger before
insert on compras for each row
begin
declare a double default (select precio from peliculas where codPelicula = new.codPelicula);

if Exists (select dniCLiente from total_compras where dniCliente=new.dniCLiente) then
update total_compras set precioGastado= (precioGastado + a) where DNICliente = new.dniCliente;
else
delete from total_compras;

insert into total_compras (select compras.dniCliente,SUM(peliculas.precio) from compras
inner join peliculas on peliculas.codPelicula=compras.codPelicula group by dniCliente);

END IF;

end //
delimiter ;
```

```
drop trigger peliculas_BD_trigger;
delimiter //
create trigger peliculas_BD_trigger before
delete on compras for each row
begin
declare a double default (select precio from peliculas where codPelicula = old.codPelicula);

update total_compras set precioGastado= (precioGastado - a) where DNICliente = old.dniCliente;

end //
delimiter ;
```

```
drop trigger peliculas_BU_trigger;
delimiter //
create trigger peliculas_BU_trigger before
update on compras for each row
begin
declare befo double default (select precio from peliculas where codPelicula = old.codPelicula);
declare afte double default (select precio from peliculas where codPelicula = new.codPelicula);
if (afte !=befo and old.dniCliente=new.dniCliente) then

update total_compras set precioGastado= (precioGastado + (afte-befo)) where DNICliente = old.dniCliente;

else

update total_compras set precioGastado= (precioGastado - befo) where DNICliente = old.dniCliente;

update total_compras set precioGastado= (precioGastado + afte) where DNICliente = new.dniCliente;

END IF ;


end //
delimiter ;
```

e. **TEST: Eliminar la base de datos. Ahora crear la base de datos, *pero antes de insertar ningún dato en ella crear la 3 triggers* del punto anterior. Comprobar que la tabla TOTAL_COMPRAS se ha rellenado.**

```
drop database filmstore;
```

	dniCliente	precioGastado
*	NULL	NULL

dniCliente	precioGastado
43535545C	116.4
45525540A	140.64999999999998
48532544T	135.44
78535540Q	226.95
NULL	NULL

f. **TEST: Insertar una nueva compra y comprobar que la tabla TOTAL_COMPRAS se actualiza.**

```
insert into compras values (10,'43535545C','2013-08-06 01:00:00');
```

dniCliente	precioGastado
43535545C	156.95
45525540A	140.64999999999998
48532544T	135.44
78535540Q	226.95
NULL	NULL

g. **TEST: Modificar el código de la película de la compra anterior y comprobar que la tabla TOTAL_COMPRAS se actualiza.**

```
update compras set codPelicula = 3 where codPelicula =10 and dniCliente= '43535545C';
```

dniCliente	precioGastado
43535545C	137.39
45525540A	140.64999999999998
48532544T	144.14
78535540Q	226.95
NULL	NULL

h. **TEST: Eliminar la compra anterior y comprobar que la tabla TOTAL_COMPRAS se actualiza.**

```
delete from compras where codPelicula=3 and dniCLiente='43535545C';
```

dniCliente	precioGastado
43535545C	116.39999999999999
45525540A	140.64999999999998
48532544T	144.14
78535540Q	226.95
NULL	NULL