```
-- 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.
use filmStore;

 drop trigger clientes_BU_trigger;
 delimiter $$
          create trigger clientes_BU_trigger before update
    on clientes for each row
    begin
                    if (new.codigoPostal < 0) then
                              set new.codigoPostal = null;
        end if;
    end $$
 delimiter ;
-- b. Comprobar si se ha creado bien (SHOW CREATE TRIGGER clientes_BU_Trigger_CP;)
show create trigger clientes_BU_trigger;
-- 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.
select * from clientes;
-- 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
 drop trigger clientes_BU_trigger;
 delimiter $$
          create trigger clientes_BU_trigger before update
    on clientes for each row
    begin
                    if (new.codigoPostal < 0 or new.codigoPostal > 52999) then
                              set new.codigoPostal = null;
        end if;
    end $$
 delimiter ;
-- f. TEST: Actualizar el código postal (cp) a 58500 del cliente con dni 22.
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.
select * from clientes;
-- 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)
 drop trigger peliculas_BU_trigger;
 delimiter $$
          create trigger peliculas_BU_trigger before update
    on peliculas for each row
    begin
                    if (new.precio < 0 or new.precio > 30) then
                              set new.precio = old.precio;
        end if;
    end $$
 delimiter ;
```

```
-- i. TEST: Actualizar el precio a 40 del código de película 1
update peliculas set precio = 40 where codPelicula = 1;
-- j. Comprobar que el trigger se ha activado y el precio no se ha modificado.
select * from peliculas where codPelicula = 1;


-- 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(20) 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
 drop trigger compras_AT_trigger;
 delimiter $$
        create trigger compras_AT_trigger before update
   on compras for each row
   begin
                if old.dniCliente != new.dniCliente then
                        insert into log_updates values (current_user(), concat('cliente cambiado de ',
old.dniCLiente, ' a ', new.dniCliente));
                elseif (old.fechaCompra != new.fechaCompra) then
                        insert into log_updates values (current_user(), concat('fecha cambiada de ',
old.fechaCompra, ' a ', new.fechaCompra));
       end if;
   end $$
 delimiter ;
-- 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';


-- 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€.
-- 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?
-- 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
-- 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.
-- f. TEST: Insertar una nueva compra y comprobar que la tabla TOTAL_COMPRAS se actualiza.
-- g. TEST: Modificar el código de la película de la compra anterior y comprobar que la tabla TOTAL_COMPRAS se actualiza.
-- h. TEST: Eliminar la compra anterior y comprobar que la tabla TOTAL_COMPRAS se actualiza.

-- Resetea la base de datos filmStore, primero crea tablas, liego trigger y luego inserta
```
use filmStore;
describe clientes;
describe compras;
DROP TABLE TOTAL_COMPRAS;
create table TOTAL_COMPRAS(
        dniCliente char(9) primary key,
   cantidadInvertida double
);

-- TRIGGER ANTES DE INSERTAR
drop trigger compras_BI_Trigger;
delimiter $$
create trigger compras_BI_Trigger before insert
on compras for each row
begin
        declare dniClienteAux char(9) default (select dniCliente from TOTAL_COMPRAS where dniCliente = new.dniCliente);
   declare cantidadAux double default (select sum(precio) from compras join peliculas on compras.codPelicula = peliculas.codPelicula
                where compras.dniCliente = new.dniCliente);
        declare precioNuevo double default (select precio from peliculas where codPelicula = new.codPelicula);
   if dniClienteAux is null then
                insert into TOTAL_COMPRAS values(new.dniCliente, precioNuevo);
        else
        update TOTAL_COMPRAS set cantidadInvertida = (cantidadAux + precioNuevo) where dniCliente = dniClienteAux;
   end if;
end $$
delimiter ;

-- TRIGGER ANTES DE ACTUALIZAR (solo preparado para actualizar codigo y dando cliente , fecha compra y cod pelicula)
 drop trigger compras_BU_Trigger;
 delimiter $$
```

```
        create trigger compras_BU_Trigger before update
    on compras for each row
    begin
                declare precioViejo double default (select precio from peliculas where codPelicula =
old.codPelicula);
                declare precioNuevo double default (select precio from peliculas where codPelicula
= new.codPelicula);
                declare dniClienteAUx char(9) default (select dniCliente from compras where
                        codPelicula = old.codPelicula and dniCliente = old.dniCliente and
fechaCompra = old.fechaCompra);
                declare cantidadNueva double default ((select cantidadInvertida from
TOTAL_COMPRAS where dniCliente = dniClienteAux) - precioViejo + precioNuevo);
        update TOTAL_COMPRAS set cantidadInvertida = cantidadNueva where dniCliente =
dniClienteAux;
-- declare cantidadAux int default (select sum(precio) from compras join peliculas on
compras.codPelicula = peliculas.codPelicula
-- where compras.dniCliente = new.dniCliente) - old;
    end $$
 delimiter ;

-- TRIGGER BEFORE DELETE

 drop trigger compras_BD_Trigger;
 delimiter $$
        create trigger compras_BD_Trigger before delete
    on compras for each row
    begin
                declare cantidadRestar double default (select precio from peliculas where
peliculas.codPelicula = old.codPelicula);
                declare cantidadAux double default ((select cantidadInvertida from
TOTAL_COMPRAS where dniCliente =old.dniCliente) - cantidadRestar);
                update TOTAL_COMPRAS set cantidadInvertida = cantidadAux where dniCliente =
old.dniCliente;
    end $$
 delimiter ;

select * from TOTAL_COMPRAS;
-- f. TEST: Insertar una nueva compra y comprobar que la tabla TOTAL_COMPRAS se actualiza.
insert into compras values(11, '43535545C', '2012-08-07 00:00:00');
insert into compras values(11, '56987412X', '2012-08-07 00:00:00');
-- g. TEST: Modificar el código de la película de la compra anterior y comprobar que la tabla
TOTAL_COMPRAS se actualiza.
-- le voy a quitar una pelicula que cuesta 39.95 y le voy a poner una que cuesta '35.95'. El clienta
'43535545C' de 156 tendria que bajar a 152
update compras set codPelicula = 5 where codPelicula = 11 and dniCliente = '43535545C' and
fechaCompra = '2012-08-07 00:00:00';
-- h. TEST: Eliminar la compra anterior y comprobar que la tabla TOTAL_COMPRAS se actualiza.
-- Tendría que desaparecer una compra de 39.95 y pasar de 152  112.05
delete from compras where codPelicula = 11 and dniCliente = '43535545C' and fechaCompra =
'2013-08-06 00:00:00';
```