U.T. VII.- Disparadores (triggers) en MySQL.

1.- Introducción.

- Un disparador, desencadenador o trigger es un objeto de BD dotado de un identificador que se asocia a una tabla y se activa cuando ocurre un evento de actualización en ella.
    - Es decir, un disparador es un bloque de código que se ejecuta cuando se lanza una sentencia INSERT, DELETE o UPDATE sobre la tabla a la que está asociado.
- Al crear un disparador debe especificarse si su ejecución deberá dener lugar antes o después de la ejecución de la sentencia de actualización que lo activó.
- Para crear o eliminar un disparador se emplean las sentencias CREATE TRIGGER y DROP TRIGGER.
- No puede haber 2 disparadores con el mismo identificador en la misma BD.
- En la actualidad, los disparadores no son activados por acciones llevadas a cabo en cascada por las restricciones de claves ajenas (ON UPDATE | DELETE CASCADE).

2.- Sintaxis de **CREATE TRIGGER.**

CREATE TRIGGER nom_disp BEFORE | AFTER INSERT | UPDATE | DELETE

ON nombre_tabla FOR EACH ROW sentencia;

- **sentencia** puede ser simple o compuesta.
    - En este último caso habrá de ir encerrada en un bloque begin ... end con las mismas características que en el caso de los procedimientos almacenados.

- **nombre_tabla** es el identificador de una tabla permanente existente en la BD activa.
- La cláusula BEFORE | AFTER indica si el disparador se ejecutará antes o después de la sentencia que lo activa.
- La cláusula INSERT | UPDATE | DELETE indica la sentencia que va a activar al disparador.
    - Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores antes de ser insertados.

3.- Sintaxis de DROP TRIGGER

```
DROP TRIGGER nombre_disp
```

- Esta sentencia elimina el disparador identificado como **nombre_disp**

4.- Limitaciones.

- Limitaciones en los tipos de disparadores que pueden crearse:
    - No pueden exisitir dos disparadores para una misma tabla que sean activados en el mismo momento y por el mismo evento.
        - Por ejemplo, no se pueden definir dos BEFORE INSERT o dos AFTER UPDATE en una misma tabla.
            - Lo anterior puede evitarse definiendo un disparador que ejecute múltiples sentencias.
    - Sí es posible tener 2 disparadores BEFORE UPDATE y AFTER INSERT o BEFORE UPDATE y AFTER UPDATE.

- Limitaciones sobre el contenido del bloque de sentencias que el disparador ejecutará al activarse:
    - Un disparador no puede invocar procedimientos almacenados utilizando la sentencia CALL.

- Un disparador no puede utilizar sentencias que inicien o finalicen una transacción.
  - Tales como START TRANSACTION, COMMIT, o ROLLBACK.
- Un disparador no puede incluir sentencias de actualización de datos que afecten a la tabla a la que está asociado.

5.- Palabras clave OLD y NEW.

- Las palabras clave old y new permiten acceder, dentro de un disparador, a columnas de los registros afectados por la sentencia que lo activó.
- En el caso de un disparador INSERT, new.nombre_columna hará referencia a una columna del registro que se va a insertar.
- En el caso de un disparador DELETE, old.nombre_columna hará referencia a una columna del registro que se va a aeliminar.
- En el caso de un disparador UPDATE podrán utilizarse ambas:
  - old.nombre_columna hará referencia al contenido de una columna del registro que se va a actualizar antes de la modificación.
  - new.nombre_columna hará referencia al contenido de una columna del resgistro que se va a actualizar después de la modificación.
- Una columna precedida por new es de lectura/escritura.
- Una columna precedida por old es de sólo lectura.

Ejemplo.

```
create
create table tabla1 (
        cod int primary key,
        nombre varchar(10)
);
create table tabla2 (
        cod int primary key,
        nombre varchar(10)
);
```

```
create table tabla3 {
        nomantiguo varchar(10) primary key,
        nomnuevo varchar(10)
);
```

a) Disparador que, al insertar una fila en tabla1 se inserta una fila en tabla2 con el doble valor de código y el nombre en mayúsculas.

```
create trigger ejemplo1a after insert
        on tabla1 for each row
        insert tabla2 values(2*new.cod,upper(new.nombre));

insert tabla1 values(1,'pepe');
insert tabla1 values(2,'juan');
insert tabla1 values(3,'luis');
```

b) Disparador que, al borrar una fila de tabla1, borre la fila de tabla2 cuyo valor de código sea el doble de la del registro borrado.

```
create trigger ejemplo1b after delete on tabla1 for each row
        delete from tabla2 where cod = 2*old.cod;

 delete from tabla1 where cod=2;
```

c) Disparador que, al modificar un nombre en tabla1, almacene en una fila de tabla3 el nombre antiguo y el nuevo.

```
create trigger ejemplo1c after update on tabla1 for each row
        linsert tabla3 values(old.nombre,new.nombre);

update tabla1 set nombre='josé' where nombre='pepe'
```

6.- Gestión de errores.

- MySQL gestiona los errores ocurridos durante la ejecución de disparadores de esta manera:
    - Si lo que falla es un disparador BEFORE, no se ejecuta la sentencia que lo activó
    - Un disparador AFTER sólo se ejecuta si la sentencia que lo activó se ejecuta con éxito.
    - Un error durante la ejecución de un disparador BEFORE o AFTER deriva en la falla de toda la sentencia que provocó su activación.

Ejercicio.

Considerando creadas las tablas del ejemplo 1 sin disparadores asociados.

a) Disparador que pase a mayúsculas todos los nombres de tabla1 antes de insertarlos.

b) Disparador que, al borrar una fila de tabla1, borre la fila de tabla2 cuyo valor de código sea el doble de la del registro borrado.

c) Disparador que evite que se modifique un nombre asignándole otro ya existente en tabla1.