Triggers MYSQL

Esta es el la estructura de como se declara un trigger:

CREATE TRIGGER [Nombre_de_Trigger]
[Momento] [Evento] ON [Nombre_de_Tabla]
ON / FOR EACH ROW
[Sentencia_SQL]

DROP TRIGGER Nombre_Tabla.nombre_trigger;

Donde:

- **Nombre_de_Trigger:** Es el nombre que identificara al trigger
- **Momento:** Sera cuando se ejecute *(BEFORE, AFTER)*
- **Evento:** Proceso que desencadena el trigger **(INSERT, UPDATE, DELETE)**
- **Sentencia_SQL:** SQL que se ejecutara al realizar la acción anterior

Ejemplos de triggers
Este ejecutaría algo justo antes de actualizar cada registro

create trigger trigger_ejemplo before update on mi_tabla on **each** row ...

Eso significa que cada vez que hagamos un update y justo antes de tocar cada registro afectado se ejecutará la sentencia que haya.

create trigger trigger_ejemplo after insert on mi_tabla on **each** row ...

Eso significa que cada vez que hagamos un insert y justo después de la inserción se ejecutará la sentencia que haya.

create trigger trigger_ejemplo before delete on mi_tabla on **each** row ...

Eso significa que cada vez que borremos un registro en la tabla y justo antes de hacerlo se ejecutará ...

Es decir, en el momento de creación de la base de datos, creamos esos triggers y el resto ya será automático.

La sentencia
En los enunciados anteriores está omitida la sentencia para poder hablar con más claridad de ella aquí. La sentencia puede ser una única instrucción o un bloque de instrucciones. En el caso de los bloques comienzan por BEGIN y acaban por END.

```
create trigger trigger_ejemplo before delete on mi_tabla
on each row
begin
  insert into otra_tabla set nombre=mi_tabla.nombre;
```

```
  delete from otra_tabla_dos where nombre=mi_tabla.nombre;
end
delimiter ;
```

Es decir, en este caso cada vez que borramos un registro en mi_tabla se inserta un registro en otra_tabla y ahí guardamos un (o varios) campos que van a ser borrados. Además aprovechamos a borrar de otra_tabla_dos los registros que deben ser borrados para mantener consistencia de datos.

En el caso de los bloques de instrucciones es necesario cambiar el delimitador que indica el final de cada instrucción. El motivo es que cuando el analizador sintactico de sql encuentre un ; no ejecute el código sino que llegue hasta el final del bloque de instrucciones y sea entonces cuando lo ejecute como si fuera un bloque. En el ejemplo de abajo se establece el símbolo # como delimitador (delimiter) para indicar el final del bloque de instrucciones. Podríamos utilizar cualquier otro símbolo o secuencia de símbolos ($, $$, $#$, $!$,….)

```
delimiter #
create trigger trigger_ejemplo before delete on mi_tabla on each row
begin
  insert into otra_tabla set nombre=mi_tabla.nombre;
  delete from otra_tabla_dos where nombre=mi_tabla.nombre;
end
delimiter #
```

Además disponemos de los prefijos OLD y NEW que se utilizan para acceder al dato "viejo" y el dato "nuevo" tras el cambio.

```
create trigger trigger_ejemplo after update on mi_tabla
on each row
  insert into otra_tabla set CodigoAnterior=old.codigo,
codigoNuevo=NEW.codigo, PrecioAnterior=OLD.precio,
PrecioNuevo=New.precio;
```

En este caso como solo es una sentencia no hemos necesitado poner el begin ni el end ni los delimitadores. Este trigger lo que hace es que cada vez que se actualiza un registro en la tabla mi_tabla inserta otro en la tabla otra_tabla con los siguientes datos.
* En el campo CodigoAnterior se mete old.codigo que es valor del campo codigo ANTES DEL UPDATE.
* en el campo CodigoNuevo se mete NEW.codigo que es el valor del campo DESPUES DEL UPDATE.
* en el campo PrecioAnterior se mete OLD.Precio que es el precio que tenía ANTES DEL UPDATE.
* En el campo PrecioNuevo se mete NEW.Precio que es el precio que tiene DESPUES DEL UPDATE.

Yo creía que no, pero por lo visto si se puede llamar a procedimientos

almacenados en un trigger. Quizá es una cuestión de versiones.

Funciones nativas Mysql

También se pueden llamar funciones nativas de mysql como en el caso siguiente

```
create trigger trigger_ejemplo before update on mi_tabla
on each row
  set NEW.FechaActualizacion=now(),
NEW.UsuarioActualizador=CURRENT_USER()
```

Este trigger lo que hace es ante cualquier actualización de una fila en la tabla mi_tabla actualizar dos campos adicionales de esa tabla llamados FechaActualizacion y UsuarioActualizador. En ellos guarda la fecha del sistema y el usuario que actualizó. De esta forma podemos consultar posteriormente quien y cuando modificó un determinado registro. En mysql el usuario que ejecuta el trigger siempre es root por lo que no es muy útil current_user que en un trigger siempre devuelve root, pero si se puede actualizar el nombre del procedimiento o rutina o aplicación que lo modifica.

Un ejemplo de trigger seria guardar un log de cambios de datos de un usuario.

Primero crearemos la tabla `user`

```
CREATE TABLE `user`(
        `id`      int not null auto_increment,
        `name`  varchar(100),
        `email` varchar(50),
        PRIMARY KEY(id)
) ENGINE = InnoDB;
```

Insertaremos un par de registros

```
INSERT INTO `user` (`name`, `email`) VALUES
('Cesar', 'cesar@craftyman.net'),
('Maria', 'maria@gmail.com'),
('Jose', 'jose@hotmail.com'),
('Albert','albertpr@yahoo.com');
```

Esta sera la tabla de logs que guardara los datos históricos del usuario

```
CREATE TABLE log_user (
        `id`          int not null auto_increment,
        `name`        varchar(100),
        `email`       varchar(50),
        `id_user`     int not null,
        `user`        varchar(40),
        `date_update` datetime,
        primary key(id)
) ENGINE = InnoDB;
```

y finalmente un trigger que se disparará cada vez que alguien modifique un dato de la tabla clientes y lo guardará en una tabla junto al nombre del usuario y la fecha.

Ahora crearemos nuestro trigger llamado *"trigger_log_users"* que se ejecutara justo despues de actualizar la tabla `user` y guardara los datos anteriores de `user` en la tabla `log_user`

```
CREATE TRIGGER trigger_log_users AFTER UPDATE ON `user`
   FOR EACH ROW
   INSERT INTO log_user(`name`, `email`, `id_user`, `user`,
`date_update` )
   VALUES (OLD.`name`, OLD.`email`, OLD.`id`, CURRENT_USER(),
NOW() );
```

CREATE SCHEMA BANCO;

USE BANCO;

- Creamos la tabla de CUENTAS con una clave primaria única.

```
DROP TABLE IF EXISTS CUENTA;

CREATE TABLE CUENTA (

       CUENTA CHAR(20) NOT NULL default '',

       TITULAR  CHAR(20) NOT NULL,

       TITULAR2 CHAR(20) NOT NULL,

       IMPORTE_INICIAL DOUBLE(10,2) DEFAULT 0,

       IMPORTE_ACTUAL DOUBLE(10,2) DEFAULT 0,

       PRIMARY KEY (CUENTA) )

   ENGINE=InnoDB;
```

- Creamos la tabla de APUNTES donde se reflejará todo moviento de efectivo realizado sobre una cuenta. Los ingresos tendrán un valor >0 y las retiradas de efectivo serán valor<0.

```
DROP TABLE IF EXISTS APUNTES;

       CREATE TABLE APUNTES (

       ASIENTO INT(8) DEFAULT AUTOINCREMENT,

       LINEA SMALLINT(5) DEFAULT 0,

       FECHA DATE DEFAULT `2020-01-01',

       CONCEPTO VARCHAR(40) default '',

       CUENTA CHAR(20) default '',

       IMPORTE DOUBLE(10,2) DEFAULT 0,

       CONSTRAINT PK_APUNTES PRIMARY KEY (ASIENTO,FECHA),

       CONSTRAINT FK_CUENTA FOREING KEY(CUENTA) REFERENCES CUENTAS(CUENTA))

   ENGINE=InnoDB;
```

Implementar mediante triggers toda la operativa de realización de movimientos en las cuentas.