TEMA 13: DISPARADORES Y EVENTOS.

1. Disparadores o *triggers*.

Un disparador es un tipo especial de rutina almacenada asociada a una tabla que se ejecuta o dispara automáticamente cuando ocurre una inserción (INSERT), borrado (DELETE) o modificación (UPDATE) sobre una tabla. Los disparadores se pueden emplear para diferentes propósitos, entre los que se encuentran:

- Implementar restricciones complejas de seguridad o de integridad.
- Impedir transacciones erróneas.
- Generar automáticamente valores derivados.
- Auditar actualizaciones, incluso enviando alertas.

La orden para crear un disparador es CREATE TRIGGER y requiere de la siguiente sintaxis:

```
CREATE TRIGGER nombre_disparador momento_disparo evento_disparo
ON nombre_tabla FOR EACH ROW
[orden_disparador]
sentencia_disparador
```

Como se puede fácilmente deducir, después de la palabra TRIGGER hay que escribir el nombre que se desea asignar al disparador. Después se debe indicar obligatoriamente el momento del disparo, que puede ser BEFORE (antes) o AFTER (después). A continuación se debe indicar el evento del disparo, que puede ser INSERT, UPDATE o DELETE. Luego se debe poner la palabra ON y el nombre de una tabla. Con toda esta información estaremos indicando si el disparador se tiene que ejecutar antes o después de realizar una inserción, borrado o modificación sobre una tabla.

Después se escribirá FOR EACH ROW, lo que significa que la sentencia del disparador se ejecutará por cada fila que se inserte, borre o actualice sobre la tabla especificada tras la palabra ON.

Debe tenerse en cuenta que puede haber varios disparadores que correspondan al mismo momento y evento de disparo sobre la misma tabla, por ejemplo, dos disparadores para ejecutarse antes de insertar en una determinada tabla. En ese caso, los dos disparadores se activarían en el mismo orden en el que fueron creados. Si se desea especificar el orden de ejecución de varios disparadores creados para el mismo momento y evento de disparo, se puede usar la sintaxis:

```
{follows | precedes} nombre_disparador_existente
```

Si se especifica *follows nombre_disparador_existente*, se está indicando que el disparador que se está creando se activará después del disparador existente cuyo nombre se indica después de la palabra *follows*. Por el contrario, si se especifica *precedes nombre_disparador_existente*, se está indicando que el disparador que se está creando se activará antes del disparador existente cuyo nombre se indica después de la palabra *precedes*.

Al final, se indicará la sentencia del disparador, esto es, la sentencia que queremos que se ejecute cuando se active el disparador. Si se desean ejecutar varias sentencias, deben colocarse entre las palabras BEGIN y END, que permiten construir sentencias complejas.

A las columnas de la tabla asociada al disparador nos podemos referir con los alias OLD y NEW. Con OLD.nombre_atributo nos referimos al valor de un atributo de una fila existente antes de ser borrada o modificada. Con NEW.nombre_atributo nos referimos al valor de un atributo en una nueva fila que va a ser insertada o después de ser modificada una fila existente.

Para trabajar con disparadores vamos a crear dos tablas similares a *Emple* y *Depart*. Comenzaremos creando una tabla llamada *Dep* conteniendo por cada departamento su número, nombre y el número de empleados que tiene:

```
create table Dep
(numdep int primary key,
nomdep varchar(30) not null,
numemple int not null default 0);
```

Creamos la tabla *Emp*, conteniendo por cada empleado, su número, nombre, salario y número del departamento en el que trabaja, que es una clave ajena a la tabla *Dep*:

```
create table Emp
(numemp int primary key,
nomemp varchar(40) not null,
salemp float not null,
numdep int not null,
foreign key(numdep) references Dep(numdep) on update cascade);
```

Queremos que el atributo *numemple* de la tabla *Dep* se mantenga siempre actualizado, de manera que refleje de manera fiel el número de empleados de cada departamento. He asignado a este atributo el valor por defecto cero para que cuando se cree un departamento, si no se indica valor en la correspondiente sentencia INSERT, se le asigne un 0 a este atributo, como debe ser. Para mantener actualizado este atributo hemos de considerar las siguientes situaciones:

- Cada vez que se añada un nuevo empleado a la tabla *Emp*, se debe incrementar en una unidad el número de empleados (atributo *numemple*) para la fila correspondiente al departamento del empleado en la tabla *Dep*. Por ello, deberemos crear un disparador BEFORE o AFTER INSERT sobre la tabla *Emp*. En la sentencia del disparador deberemos indicar que se incremente el valor del atributo *numemple* de la tabla *Dep* en 1 para la fila cuyo número de departamento (*numdep*) sea el del departamento del nuevo empleado que se acaba de añadir.

```
create trigger NuevoEmpleado
after insert on Emp for each row
update Dep set numemple=numemple+1 where numdep=NEW.numdep;
```

Para probar que este disparador funciona correctamente vamos a insertar primero dos departamentos en la tabla Dep y luego vamos a añadir un empleado al departamento número 1.

```
insert into Dep (numdep,nomdep)
values (1, 'Compras'), (2, 'Ventas');

insert into Emp values (1, 'Jose Gil', 1250.45, 1);
```

Para comprobar si el disparador funciona correctamente, veamos el contenido de la tabla *Dep*.

```
mysql> select * from Dep;
+--------+---------+----------+
| numdep | nomdep  | numemple |
+--------+---------+----------+
|      1 | Compras |        1 |
|      2 | Ventas  |        0 |
+--------+---------+----------+
2 rows in set (0.00 sec)
```

Podemos observar como el número de empleados del departamento número 1 es 1, lo que quiere decir que el disparador se ha ejecutado y ha incrementado el número de empleados de la tabla *Dep* de cero a uno.

- Cada vez que se elimine a un empleado de la tabla *Emp*, se debe decrementar en una unidad el número de empleados (atributo *numemple*) para la fila correspondiente al departamento del empleado en la tabla *Dep*. Por ello, deberemos crear un disparador BEFORE o AFTER DELETE sobre la tabla *Emp*. En la sentencia del disparador deberemos indicar que se decremente el valor del atributo *numemple* de la tabla *Dep* en 1 para la fila cuyo número de departamento (*numdep*) sea el del departamento del empleado que se acaba de borrar.

```
create trigger BajaEmpleado
after delete on Emp for each row
update Dep set numemple=numemple-1 where numdep=OLD.numdep;
```

Para probar si el disparador funciona correctamente eliminemos al empleado 1 creado con anterioridad, con lo que el número de empleados de su departamento (el número 1) deberá volver a tomar valor cero.

```
delete from Emp where numemp=1;

mysql> select * from Dep;
+--------+---------+----------+
| numdep | nomdep  | numemple |
+--------+---------+----------+
|      1 | Compras |        0 |
|      2 | Ventas  |        0 |
+--------+---------+----------+
2 rows in set (0.00 sec)
```

- Para terminar, cada vez que se cambie en la tabla *Emp* el número del departamento en el que trabaja un empleado, se debe decrementar en una unidad el número de empleados (atributo *numemple*) para la fila correspondiente al departamento en el que trabajaba el empleado en la tabla *Dep* e incrementar en una unidad el número de empleados (atributo *numemple*) para la fila correspondiente al nuevo departamento en el que trabaja el empleado. Por ello, deberemos crear un disparador BEFORE o AFTER UPDATE sobre la tabla *Emp*. Deberemos efectuar las dos actualizaciones solo en el caso de que el número de departamento después de efectuar la modificación (NEW) sea diferente del número de departamento antes de llevar a cabo el cambio (OLD). Deberemos incluir varias sentencias en el disparador, las cuales deberán ir, por tanto, entre las palabras BEGIN y END. En este caso, para poder delimitar las sentencias con el símbolo punto y coma (;) dentro del disparador, deberemos cambiar el delimitador de la sentencia que crea el disparador (en este caso se ha empleado el delimitador //). Por un lado, deberemos indicar que se decremente el valor del atributo *numemple* de la tabla *Dep* en 1 para la fila cuyo número de departamento (*numdep*) sea el del antiguo departamento del empleado. Por otro lado, deberemos indicar que se incremente el valor del atributo *numemple* de la tabla *Dep* en 1 para la fila cuyo número de departamento (*numdep*) sea el del nuevo departamento del empleado.

```
delimiter //
create trigger CambioDep
after update on Emp
for each row
begin
/*Si se cambia al empleado de n° de dpto, resto 1 al n° de empleados del
dpto. viejo y sumo 1 al n° de empleados del dpto. nuevo*/
if NEW.numdep != OLD.numdep then
  update Dep set numemple = numemple - 1 where numdep = OLD.numdep;
  update Dep set numemple = numemple + 1 where numdep = NEW.numdep;
end if;
end;//
```

Para probar el funcionamiento de este disparador añadamos dos empleados a la tabla *Emp*, los cuales trabajen en el departamento número 1.

```
insert into Emp values (1, 'Ana Gil', 1350, 1);
insert into Emp values (2, 'Luisa Gómez', 2156.34, 1);
```

Como podemos ver a continuación, el contenido de la tabla *Dep* en cuanto al número de empleados del departamento número 1 es correcto, ya que le acabamos de asignar dos empleados.

```
mysql> select * from Dep;
+--------+---------+----------+
| numdep | nomdep  | numemple |
+--------+---------+----------+
|      1 | Compras |        2 |
|      2 | Ventas  |        0 |
+--------+---------+----------+
2 rows in set (0.00 sec)
```

Ahora vamos a cambiar el número de departamento en el que trabaja la empleada número 1, asignándole el departamento número 2:

```
update Emp set numdep=2 where numemp=1;
```

Tras esta modificación el número de empleados del departamento 1 debe ser 1 y el número de empleados del departamento número 2 debe ser 1 también. Comprobémoslo:

```
mysql> select * from Dep;
+--------+---------+----------+
| numdep | nomdep  | numemple |
+--------+---------+----------+
|      1 | Compras |        1 |
|      2 | Ventas  |        1 |
+--------+---------+----------+
2 rows in set (0.00 sec)
```

Para eliminar un disparador se emplea la orden DROP TRIGGER, cuya sintaxis es la siguiente:

```
DROP TRIGGER [IF EXISTS] nombre_disparador
```

Por ejemplo, para borrar el disparador *BajaEmpleado* creado anteriormente escribiremos:

```
mysql> drop trigger BajaEmpleado;
Query OK, 0 rows affected (0.05 sec)
```

No existe ninguna sentencia SQL tipo ALTER TRIGGER para modificar un disparador. En caso de que se desee modificar un disparador, será necesario eliminarlo con DROP TRIGGER y volverlo a crear con CREATE TRIGGER.

Los disparadores se almacenan en la tabla TRIGGERS de la base de datos *information_schema*. Además de poder consultar información sobre los disparadores creados

en esta tabla, se puede hacer uso de la instrucción SHOW TRIGGERS, que presenta el formato:

```
SHOW TRIGGERS [[FROM | IN] nombre_BD];
```

Así, por ejemplo, para visualizar los disparadores existentes en la base de datos *Empresa*, escribiremos:

```
show triggers from Empresa;
```

2. Eventos.

Los eventos son tareas que se ejecutan en un momento determinado (día y hora) que se especifica al crear el evento. Por este motivo, a veces nos referimos a ellos como eventos programados.

Un evento se identifica por un nombre y el esquema o base de datos al que se asigna. Lleva a cabo la acción o acciones especificadas de acuerdo con un horario. Si son varias las acciones, se deben especificar entre BEGIN y END.

Podemos hablar de dos tipos de eventos: los que se programan para una única ocasión y los que ocurren periódicamente cada cierto tiempo.

La variable global *event_scheduler* determina si el programador de eventos está habilitado y en ejecución en el servidor. Esta variable puede tomar los valores ON (activado), OFF (desactivado) y DISABLED, en cuyo caso no se puede habilitar la activación en tiempo de ejecución.

Cuando el programador de eventos se detiene (variable global *event_scheduler* está a OFF), puede ser iniciado estableciendo la variable *event_scheduler* a ON. Podemos ver si está activo o no de dos maneras:

- Consultando la variable del sistema event_scheduler:

```
mysql> show variables like 'event%';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | ON    |
+-----------------+-------+
1 row in set, 1 warning (0.04 sec)
```

- Observando la salida del comando siguiente, ya que si está activo, el programador de eventos se ejecuta como un hilo más del servidor:

```
mysql> show processlist;
+----+-----------------+-----------------+---------+---------+--------+-----------------------+------------------+
| Id | User            | Host            | db      | Command | Time   | State                 | Info             |
+----+-----------------+-----------------+---------+---------+--------+-----------------------+------------------+
|  4 | event_scheduler | localhost       | NULL    | Daemon  | 285871 | Waiting on empty queue | NULL             |
|  9 | root            | localhost:51181 | pedidos | Query   |      0 | starting              | show processlist |
+----+-----------------+-----------------+---------+---------+--------+-----------------------+------------------+
2 rows in set (0.00 sec)
```

Vemos que el programador de eventos está activo. Si en algún momento estuviese inactivo (OFF), si *event_scheduler* no toma el valor DISABLED, podemos activarlo con el siguiente comando:

```
mysql> set GLOBAL event_scheduler = ON;
Query OK, 0 rows affected (0.04 sec)
```

Para la creación de eventos se usa la orden CREATE EVENT, cuya sintaxis se muestra a continuación:

```
CREATE EVENT [IF NOT EXISTS] nombre_evento ON SCHEDULE momento
[ENABLE|DISABLE] [COMMENT 'comentarios']
DO cuerpo_evento;

momento:
AT momento [+INTERVAL intervalo] | EVERY intervalo
[STARTS momento [+INTERVAL intervalo]]
[ENDS momento [+INTERVAL intervalo]]

intervalo:
número [ YEAR | QUARTER | MONTH | WEEK | DAY |HOUR | MINUTE | SECOND]
```

Como se puede observar, después de CREATE EVENT es necesario escribir el nombre del evento y, a continuación, indicar después de ON SCHEDULE AT el momento en el que se desea que este se ejecute. Este momento debe ser un día y hora especificado en el formato 'AAAA-MM-DD HH:MM:SS' o CURRENT_TIMESTAMP, que indica ahora mismo. También se puede especificar un momento consistente en sumar un intervalo de tiempo a otro momento, o bien, que se ejecute cada cierto intervalo de tiempo (EVERY intervalo). En este último caso, se debe especificar cuándo debe comenzar a ejecutarse el evento (después de STARTS) y se puede indicar opcionalmente además cuándo debe finalizar su ejecución. En caso de que se desee especificar un intervalo de tiempo, se indicará un número y a continuación la palabra YEAR (año), QUARTER (trimestre)….o SECOND (segundo). Así, si se indica EVERY 1 WEEK STARTS '2021-01-01 00:00:00' quiere decir que se debe ejecutar a las 0 horas del 1 de enero de 2021 y periódicamente cada semana.

Se puede crear un evento, pero dejarlo inactivo escribiendo la palabra DISABLE. Si se especifica ENABLE en vez de DISABLE, se indicará que se desea que esté activo. Además, recordemos que para que los eventos puedan funcionar, la variable *event_scheduler* debe toma el valor ON.

Por último, después de la palabra DO se debe indicar obligatoriamente la sentencia o sentencias que deseamos que se ejecuten como parte del evento. Si se trata de varias sentencias, deberán especificarse entre BEGIN y END.

Vamos a crear un evento por medio del cual se calcule en el instante de crearlo y cada semana la antigüedad (número de años en la empresa) de los empleados de la tabla *Emple*. Para ello, vamos a crear una tabla *Emple2* copia de *Emple*, pero le vamos a añadir un campo entero llamado *antigüedad*, al que vamos a asignar un 0 como valor por defecto:

```
create table emple2 as select * from emple;

alter table emple2
add antigüedad int default 0;
```

Veamos ahora el contenido de la tabla *Emple2*:

```
mysql> select * from emple2;
+--------+----------+------------+------+------------+---------+----------+---------+------------+
| emp_no | apellido | oficio     | dir  | fecha_alt  | salario | comision | dept_no | antigüedad |
+--------+----------+------------+------+------------+---------+----------+---------+------------+
|   7369 | SÁNCHEZ  | EMPLEADO   | 7900 | 2016-12-12 |     600 |        0 |      20 |          0 |
|   7499 | ARROYO   | VENDEDOR   | 7698 | 2013-02-20 |    1200 |      240 |      30 |          0 |
|   7521 | SALA     | VENDEDOR   | 7698 | 2014-02-22 |     960 |      390 |      30 |          0 |
|   7566 | JIMÉNEZ  | DIRECTOR   | 7839 | 2014-02-04 |    2300 |        0 |      20 |          0 |
|   7654 | MARTÍN   | VENDEDOR   | 7698 | 2014-09-29 |     965 |     1000 |      30 |          0 |
|   7698 | NEGRO    | DIRECTOR   | 7839 | 2014-05-01 |    2200 |        0 |      30 |          0 |
|   7738 | CEREZO   | DIRECTOR   | 7839 | 2014-09-06 |    2210 |        0 |      10 |          0 |
|   7788 | GIL      | ANALISTA   | 7566 | 2017-04-23 |    2350 |        0 |      20 |          0 |
|   7839 | REY      | PRESIDENTE | NULL | 2014-11-17 |    3900 |        0 |      10 |          0 |
|   7844 | TOVAR    | VENDEDOR   | 7698 | 2014-09-08 |    1100 |        0 |      30 |          0 |
|   7876 | ALONSO   | EMPLEADO   | 7788 | 2017-08-09 |     860 |        0 |      20 |          0 |
|   7900 | JIMENO   | EMPLEADO   | 7698 | 2014-12-03 |     725 |        0 |      30 |          0 |
+--------+----------+------------+------+------------+---------+----------+---------+------------+
12 rows in set (0.00 sec)
```

Pues bien, vamos a crear el evento con el nombre *CalcularAntigüedad*. La antigüedad se calcula como el número de años transcurridos entre la fecha del día de hoy (fecha que nos devuelve la función *sysdate()*) y la fecha de alta del empleado en la empresa (atributo *fecha_alt*). Para llevar a cabo este cálculo, MySQL nos proporciona la función *timestampdiff (unidad_tiempo, fecha1, fecha2)* que recibe como parámetro una unidad de tiempo y dos fechas tal que fecha2 es superior o igual a fecha1. Esta función nos devuelve la diferencia entre esas dos fechas en la unidad de tiempo indicada como primer parámetro. La unidad de tiempo puede ser *year*, *month*, *week*, *day*, *hour*, *minute*, *second*… Pues bien, como lo que a nosotros nos interesa es el número de años entre las dos fechas, pondremos *year* como unidad de medida. El evento nos quedará como sigue:

```
create event CalcularAntigüedad
on schedule every 1 week starts current_timestamp
do
update emple2
set antigüedad=timestampdiff(year, fecha_alt, sysdate());
```

Si ahora consultamos el contenido de la tabla *Emple2*, veremos como el atributo antigüedad toma el valor correcto:

```
mysql> select * from emple2;
+--------+----------+------------+------+------------+---------+----------+---------+------------+
| emp_no | apellido | oficio     | dir  | fecha_alt  | salario | comision | dept_no | antigüedad |
+--------+----------+------------+------+------------+---------+----------+---------+------------+
|   7369 | SÁNCHEZ  | EMPLEADO   | 7900 | 2016-12-12 |     600 |        0 |      20 |          3 |
|   7499 | ARROYO   | VENDEDOR   | 7698 | 2013-02-20 |    1200 |      240 |      30 |          7 |
|   7521 | SALA     | VENDEDOR   | 7698 | 2014-02-22 |     960 |      390 |      30 |          6 |
|   7566 | JIMÉNEZ  | DIRECTOR   | 7839 | 2014-02-04 |    2300 |        0 |      20 |          6 |
|   7654 | MARTÍN   | VENDEDOR   | 7698 | 2014-09-29 |     965 |     1000 |      30 |          6 |
|   7698 | NEGRO    | DIRECTOR   | 7839 | 2014-05-01 |    2200 |        0 |      30 |          6 |
|   7738 | CEREZO   | DIRECTOR   | 7839 | 2014-09-06 |    2210 |        0 |      10 |          6 |
|   7788 | GIL      | ANALISTA   | 7566 | 2017-04-23 |    2350 |        0 |      20 |          3 |
|   7839 | REY      | PRESIDENTE | NULL | 2014-11-17 |    3900 |        0 |      10 |          6 |
|   7844 | TOVAR    | VENDEDOR   | 7698 | 2014-09-08 |    1100 |        0 |      30 |          6 |
|   7876 | ALONSO   | EMPLEADO   | 7788 | 2017-08-09 |     860 |        0 |      20 |          3 |
|   7900 | JIMENO   | EMPLEADO   | 7698 | 2014-12-03 |     725 |        0 |      30 |          6 |
+--------+----------+------------+------+------------+---------+----------+---------+------------+
12 rows in set (0.00 sec)
```

Los eventos se pueden modificar haciendo uso de la sentencia ALTER EVENT, cuya sintaxis es la siguiente:

```
ALTER EVENT nombre_evento
[ON SCHEDULE momento]
[RENAME TO nuevo_nombre_evento]
[ENABLE|DISABLE]
[COMMENT 'comentarios']
[DO cuerpo_evento];
```

Como se puede observar, se pueden modificar todas las características del evento, incluso su cuerpo y además se le puede cambiar de nombre.

Por su parte, para eliminar un evento se usará el comando DROP EVENT con la siguiente sintaxis:

```
DROP EVENT [IF EXISTS] nombre_evento
```

Se puede hacer uso del comando SHOW EVENTS para visualizar los eventos de una determinada base de datos, cuya sintaxis es la misma que la del comando SHOW TRIGGERS:

```
SHOW EVENTS [[FROM | IN] nombre_BD];
```