Unidad 5 – Disparadores (triggers)

Disparadores

A partir de MySQL 5.0.2 se incorporó el soporte básico para disparadores (triggers).
Un disparador es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular.
Por ejemplo, las siguientes sentencias crean una tabla y un disparador para sentencias INSERT dentro de la tabla. El disparador suma los valores insertados en una de las columnas de la tabla:

```
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
CREATE TRIGGER ins_sum BEFORE INSERT ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
```

Sintaxis de CREATE TRIGGER

```
CREATE TRIGGER nombre_disp momento_disp evento_disp
ON nombre_tabla FOR EACH ROW sentencia_disp
```

Un disparador es un objeto con nombre en una base de datos que se asocia con una tabla, y se activa cuando ocurre un evento en particular para esa tabla.
El disparador queda asociado a la tabla **nombre_tabla**. Esta debe ser una tabla permanente, no puede ser una tabla TEMPORARY ni una vista.
momento_disp es el momento en que el disparador entra en acción. Puede ser BEFORE (antes) o AFTER (después), para indicar que el disparador se ejecute antes o después que la sentencia que lo activa.
evento_disp indica la clase de sentencia que activa al disparador. Puede ser INSERT, UPDATE, o DELETE. Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.
No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia. Por ejemplo, no se pueden tener dos disparadores BEFORE UPDATE. Pero sí es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE.
sentencia_disp es la sentencia que se ejecuta cuando se activa el disparador. Si se desean ejecutar múltiples sentencias, deben colocarse entre BEGIN ... END, el constructor de sentencias compuestas. Esto además posibilita emplear las mismas sentencias permitidas en rutinas almacenadas.

Sintaxis de DROP TRIGGER

```
DROP TRIGGER [nombre_esquema.]nombre_disp
```

Elimina un disparador. El nombre de esquema es opcional. Si el esquema se omite, el disparador se elimina en el esquema actual.
Anteriormente a la versión 5.0.10 de MySQL, se requería el nombre de tabla en lugar del nombre de esquema. (nom_tabla.nom_disp).

Nota: cuando se actualice desde una versión anterior de MySQL 5 a MySQL 5.0.10 o superior, se deben eliminar todos los disparadores antes de actualizar y volver a crearlos después, o DROP TRIGGER no funcionará luego de la actualización.

Utilización de disparadores

El soporte para disparadores se incluyó a partir de MySQL 5.0.2. Actualmente, el soporte para disparadores es básico, por lo tanto hay ciertas limitaciones en lo que puede hacerse con ellos. Esta sección trata sobre el uso de los disparadores y las limitaciones vigentes.
Un disparador es un objeto de base de datos con nombre que se asocia a una tabla, y se activa cuando ocurre un evento en particular para la tabla. Algunos usos para los disparadores es verificar valores a ser insertados o llevar a cabo cálculos sobre valores involucrados en una actualización.
Un disparador se asocia con una tabla y se define para que se active al ocurrir una sentencia INSERT, DELETE, o UPDATE sobre dicha tabla. Puede también establecerse que se active antes o después de la sentencia en cuestión. Por ejemplo, se puede tener un disparador que se active antes de que un registro sea borrado, o después de que sea actualizado.
Para crear o eliminar un disparador, se emplean las sentencias CREATE TRIGGER y DROP TRIGGER.

Ejemplo:

```
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
CREATE TRIGGER ins_sum BEFORE INSERT ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
```

La sentencia CREATE TRIGGER crea un disparador llamado ins_sum que se asocia con la tabla account. También se incluyen cláusulas que especifican el momento de activación, el evento activador, y qué hacer luego de la activación:

- La palabra clave BEFORE indica el momento de acción del disparador. En este caso, el disparador debería activarse antes de que cada registro se inserte en la tabla. La otra palabra clave posible aquí es AFTER.
- La palabra clave INSERT indica el evento que activará al disparador. En el ejemplo, la sentencia INSERT causará la activación. También pueden crearse disparadores para sentencias DELETE y UPDATE.
- Las sentencia siguiente, FOR EACH ROW, define lo que se ejecutará cada vez que el disparador se active, lo cual ocurre una vez por cada fila afectada por la sentencia activadora. En el ejemplo, la sentencia activada es un sencillo SET que acumula los valores insertados en la columna amount. La sentencia se refiere a la columna como NEW.amount, lo que significa “el valor de la columna amount que será insertado en el nuevo registro.”

Para utilizar el disparador, se debe establecer el valor de la variable acumulador a cero, ejecutar una sentencia INSERT, y ver qué valor presenta luego la variable.

Debido a que un disparador está asociado con una tabla en particular, no se pueden tener múltiples disparadores con el mismo nombre dentro de una tabla. También se debería tener en cuenta que el espacio de nombres de los disparadores puede cambiar en el futuro de un nivel de tabla a un nivel de base de datos, es decir, los nombres de disparadores ya no sólo deberían ser únicos para cada tabla sino para toda la base de datos. Para una mejor compatibilidad con desarrollos futuros, se debe intentar emplear nombres de disparadores que no se repitan dentro de la base de datos.

Adicionalmente al requisito de nombres únicos de disparador en cada tabla, hay otras limitaciones en los tipos de disparadores que pueden crearse. En particular, no se pueden tener 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. Es improbable que esta sea una gran limitación, porque es posible definir un disparador que ejecute múltiples sentencias empleando el constructor de sentencias compuestas BEGIN ... END luego de FOR EACH ROW.

También hay limitaciones sobre lo que puede aparecer dentro de la sentencia que el disparador ejecutará al activarse:

- El disparador no puede referirse a tablas directamente por su nombre, incluyendo la misma tabla a la que está asociado. Sin embargo, se pueden emplear las palabras clave OLD y NEW.
  OLD se refiere a un registro existente que va a borrarse o que va a actualizarse antes de que esto ocurra.
  NEW se refiere a un registro nuevo que se insertará o a un registro modificado luego de que ocurre la modificación.
- El disparador no puede invocar procedimientos almacenados utilizando la sentencia CALL. (Esto significa, por ejemplo, que no se puede utilizar un procedimiento almacenado para eludir la prohibición de referirse a tablas por su nombre).
- El disparador no puede utilizar sentencias que inicien o finalicen una transacción, tal como START TRANSACTION, COMMIT, o ROLLBACK.

OLD y NEW

Los triggers tienen dos palabras clave, OLD y NEW que se refieren a los valores que tienen las columnas antes y después de la modificación.
Los INSERT permiten NEW, los DELETE solo OLD y los UPDATE ambas.
OLD y NEW no son sensibles a mayúsculas.

CREATE TRIGGER <nombre>
{BEFORE l AFTER }
{INSERT l UPDATE l DELETE}
ON
<nombre_tabla>
FOR EACH ROW
BEGIN
<sentencias SQL
END;>

Una columna precedida por OLD es de sólo lectura. Es posible hacer referencia a ella pero no modificarla.
Una columna precedida por NEW puede ser referenciada si se tiene el privilegio SELECT sobre ella. En un disparador BEFORE, también es posible cambiar su valor con SET NEW.nombre_col = valor si se tiene el privilegio de UPDATE sobre ella. Esto significa que un disparador puede usarse para modificar los valores antes que se inserten en un nuevo registro o se empleen para actualizar uno existente.
En un disparador BEFORE, el valor de NEW para una columna AUTO_INCREMENT es 0, no el número secuencial que se generará en forma automática cuando el registro sea realmente insertado.

Empleando el constructor BEGIN ... END, se puede definir un disparador que ejecute sentencias múltiples. Dentro del bloque BEGIN, también pueden utilizarse otras sintaxis permitidas en rutinas almacenadas, tales como condicionales y bucles. Como sucede con las rutinas almacenadas, cuando se crea un disparador que ejecuta sentencias múltiples, se hace necesario redefinir el delimitador de sentencias si se ingresará el disparador a través del programa mysql, de forma que se pueda utilizar el caracter ';' dentro de la definición del disparador.

Los TRIGGERs se almacenan en una tabla específica de la base information_schema, y también en archivos .TRG en la base donde está la tabla a la cual el TRIGGER pertenece.

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 operación en el correspondiente registro.
- Un disparador AFTER se ejecuta solamente si el disparador BEFORE (de existir) y la operación se ejecutaron exitosamente.
- Un error durante la ejecución de un disparador BEFORE o AFTER deriva en la falla de toda la sentencia que provocó la invocación del disparador.

- En tablas transaccionales, la falla de un disparador (y por lo tanto de toda la sentencia) debería causar la cancelación (rollback) de todos los cambios realizados por esa sentencia. En tablas no transaccionales, cualquier cambio realizado antes del error no se ve afectado.

Ejemplo:
Se disponen de las siguientes tablas:
- PERSONA: pk_persona, apellido, nombre, barrio_pk
- AUDITORIA: pk_auditoria, apellido_ant, apellido_nuevo,nombre_ant,nombre_nuevo

Trigger de inserción
Triggers que inserta un registro en la tabla auditoria, cuando se inserta una nueva persona, en donde los valores anteriores son null.

```
DROP TRIGGER IF EXISTS insertar_auditoria_persona;
CREATE TRIGGER insertar_auditoria_persona AFTER INSERT ON personas
FOR EACH ROW
INSERT INTO auditoria(apellido_ant,apellido_nuevo,nombre_ant,nombre_nuevo)
VALUES (null, NEW.apellido, null, NEW.nombre)
```

Trigger de modificación
Triggers que inserta un registro en la tabla auditoria, cuando se modifica una nueva persona, almacenando el valor anterior y posterior a la modificación de cada campo.

```
DROP TRIGGER IF EXISTS modificar_auditoria_persona;
CREATE TRIGGER modificar_auditoria_persona BEFORE UPDATE ON personas
FOR EACH ROW
INSERT INTO auditoria(apellido_ant,apellido_nuevo,nombre_ant,nombre_nuevo)
VALUES (OLD.apellido, NEW.apellido, OLD.nombre, NEW.nombre)
```

Trigger de eliminación
Triggers que inserta un registro en la tabla auditoria, cuando se elimina una nueva persona, en donde los valores nuevos son null.

```
DROP TRIGGER IF EXISTS eliminar_auditoria_persona;
CREATE TRIGGER eliminar_auditoria_persona AFTER DELETE ON personas
FOR EACH ROW
INSERT INTO auditoria(apellido_ant,apellido_nuevo,nombre_ant,nombre_nuevo)
VALUES (OLD.apellido, null, OLD.nombre, null)
```

Caso práctico:

Crear un cursor que se encargue de insertar en la tabla “resultados” los clientes cargados en la tabla “clientes”, cuyos campos son “cliente” y “resultado”.
En el campo “cliente” ingrese la concatenación del apellido y nombre y en el campo “resultado” ingrese la leyenda “menor a 3” o “mayor igual a 3”, dependiendo si el barrio_pk es menor o mayor/igual a 3.

```
delimiter //
CREATE PROCEDURE prueba()
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE a,b CHAR(45);
DECLARE c INT;

DECLARE cur1 CURSOR FOR SELECT apellido, nombre, barrio_pk FROM clientes;
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

OPEN cur1;

REPEAT
FETCH cur1 INTO a, b, c;

IF NOT done THEN
        IF c < 3 THEN
                INSERT INTO resultados (cliente,estado)
                VALUES (concat(a,' ',b),concat(c,' menor a 3'));
        ELSE
                INSERT INTO resultados (cliente,estado)
                VALUES (concat(a,' ',b),concat(c,' mayor igual a 3'));
        END IF;
END IF;
UNTIL done END REPEAT;

CLOSE cur1;

END
//
```