Unidad 11 Programación de bases de datos II. Triggers y Eventos

1 TRIGGERS

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, un evento del tipo INSERT, DELETE o UPDATE.

Es decir, son objetos que se asocian a una tabla y se activan cuando ocurre un determinado evento en dicha tabla, por ejemplo, cuando agregamos un nuevo registro.

1.1 Gestión de disparadores

Las instrucciones para gestionar disparadores son CREATE TRIGGER, SHOW TRIGGER DROP TRIGGER.

Crear trigger

La sintaxis es:

```
CREATE TRIGGER [nombre_del_trigger] [momento] [evento]
ON [nombre_tabla] FOR EACH ROW
BEGIN
      [proceso]
END;
```

Donde:

- **[nombre_del_trigger] →** nombre del disparador, debe ser único.
- **[momento] →** determina cuando se ejecutara el disparador, puede ser **BEFORE** (antes) o **AFTER** (después) del **evento**.
- **[evento]** --> determina que proceso (sentencia) llamara al disparador, puede ser **INSERT** (insertar), **UPDATE** (actualizar) o **DELETE** (eliminar) datos. Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.
- **[nombre_tabla] →** será la tabla a la que se asigne el disparador (trigger).
- **[proceso] →** determina las instrucciones a seguir, bien podría ser una sola sentencia y no haría falta en ese caso usar la sentencia compuesta BEGIN... END, de lo contrario, es necesario.
- 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.

Las columnas de la tabla asociada con el disparador pueden referenciarse empleando los alias **OLD** y **NEW**. ***OLD.nombre_col*** hace referencia a una columna de una fila existente, antes de ser actualizada o borrada. ***NEW.nombre_col*** hace referencia a una columna en una nueva fila a punto de ser insertada o en una fila existente después de ser actualizada.

Las palabras clave OLD y NEW permiten acceder a columnas en los registros afectados por un disparador.

- En un disparador para INSERT, solamente puede utilizarse NEW.nom_col ya que no hay una versión anterior del registro.
- En un disparador para DELETE solo puede emplearse OLD.nom_col, porque no hay un nuevo registro.
- En un disparador para UPDATE se puede emplear OLD.nom_col para referirse a las columnas de un registro antes de que sea actualizada, y NEW.nom_col para referirse a las columnas del registro luego de actualizarlo.

- El uso de SET NEW.nom_col = valor necesita que se tenga el privilegio UPDATE sobre la columna. El uso de SET nombre_var = NEW.nombre_col necesita el privilegio SELECT sobre la columna.

Eliminación de triggers

Para eliminar el disparador, se emplea una sentencia DROP TRIGGER. La sintaxis es:

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

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*).

Consulta de triggers

Podemos obtener información de los triggers creados con SHOW TRIGGER.

```
SHOW TRIGGERS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

Este comando nos permite mostrar triggers de una base de datos filtrándolos con un patrón o cláusula WHERE.

Cuando creamos triggers se crea un nuevo registro en la tabla INFORMATION_SCHEMA llamada INFORMATION_SCHEMA.TRIGGERS que podemos visualizar con el siguiente comando:

```
SELECT trigger_name, action_statement FROM information_schema.triggers
```

Veamos un ejemplo practico del uso de los **triggers**, disponemos de una tabla donde almacenamos los datos de **artículos** y deseamos que se lleve un control de las operaciones que en ella se realizan. Para comenzar con algo sencillo, el siguiente **trigger** se ejecuta después de la creación de un nuevo registro:

```
CREATE TRIGGER trigger_log_articulos
AFTER
      INSERT ON articulos
FOR EACH ROW
      INSERT INTO log_articulos(fecha,usuario,proceso,id_articulo)
      VALUES (NOW(),CURRENT_USER(),'1',NEW.id_articulo);
```

Almacenará la fecha de la inserción, el nombre del usuario, el tipo de proceso y el id del registro agregado. veamos un ejemplo practico:

```
 /* Creamos la base de datos*/
      CREATE DATABASE triggers_db;
/* Seleccionamos la base de datos creada*/
      USE triggers_db;

/* Creamos la tabla donde se almacenan los artículos*/
CREATE TABLE articulos(
  id_articulo int not null auto_increment primary key,
    titulo varchar(200) not null,
    contenido Blob not null, -- máximo de 65.535 caracteres
    autor varchar(25) not null, -- podría ser el id de una tabla "usuarios"
    fecha_pub datetime not null, -- fecha de la publicacion
    estado bool not null default 1 -- 1 ó 0
    );

/*Creamos la tabla que almacena los proceso realizados sobre la tabla
"articulos"*/
CREATE TABLE log_articulos
 (
   id_log_art int not null auto_increment primary key,
```

```
   fecha datetime, -- fecha del proceso
   usuario varchar(40), -- usuario implicado
   proceso varchar(10), -- agregado, editado, eliminado
   articulo varchar(200) not null -- titulo del articulo
 );


/* Creamos el disparador*/
CREATE TRIGGER tgr_ins_logarticulos
 AFTER
  INSERT ON articulos
 FOR EACH ROW
   INSERT INTO log_articulos(fecha,usuario,proceso,articulo)
       VALUES (NOW(),CURRENT_USER(),'agregado',NEW.titulo);
/*Insertamos un dato en la tabla artículos*/

INSERT INTO articulos (titulo,contenido,autor,fecha_pub) VALUES
 ('ejemplo de triggers en MySQL','contenido..','autor_x',NOW());
```

En el ejemplo se ve, que una vez se inserta un registro en la tabla **articulos**, el **trigger** se ejecuta y almacena datos de la operación en la tabla **log_articulos**.

Si en una misma sentencia insertamos más de un registro, se dispararan tantos procesos como registros haya.

1.2 Uso de disparadores

Aunque su uso es muy variado y depende mucho del tipo de aplicación o base de datos con que trabajemos podemos hacer una clasificación más o menos general.

Control de sesiones

Por ejemplo tenemos una tabla donde un usuario de un banco ejecutas movimiento de dinero en sus cuentas y queremos ver cuales son sus movimientos acumulados en esa sesión. La tabla es

```
CREATE TABLE movimientos
(
      id_cuenta   INTEGER,
      fecha       DATETIME,
      cantidad    FLOAT
);
```

El trigger sería:

```
CREATE TRIGGER insertar_mov
BEFORE
INSERT ON movimientos FOR EACH ROW SET @sum=@sum+NEW.cantidad;
```

Veamos como al insertar uno o varios movimientos en una cuenta se acumula la cantidad de todos ellos en la variable de sesión @sum. Para utilizarlo establecemos el valor de la variable sum a cero, ejecutamos varias sentencias INSERT y vemos que valor presenta luego la variable:

```
SET @sum = 0;
INSERT INTO movimientos VALUES (137,'2012-10-12',14.98);
INSERT INTO movimientos VALUES (141,'2012-10-13',1937.50);
INSERT INTO movimientos VALUES (97,'2013-01-01',-100.00);
SELECT @sum AS "Total insertado";
```

Control de valores de entrada

Un uso posible de los disparadores es el control de valores insertados o actualizados en tablas.

Veamos un ejemplo donde se crea un disparador en la tabla movimientos para UPDATE que verifica los valores utilizados para actualizar cada columna, y modificar el valor para que se encuentre en un rango de 0 a 100. Esto debe hacerse en un disparador BEFORE porque los valores deben verificarse antes de emplearse para actualizar el registro.

```
DELIMITER $$
CREATE TRIGGER comprobacion_saldo BEFORE UPDATE ON movimientos
FOR EACH ROW
BEGIN
IF NEW.cantidad <0 THEN
      SET NEW.cantidad=0;
ELSEIF NEW.cantidad > 100 THEN
      SET NEW.cantidad=100;
END IF;
END; $$
```

En este caso cada vez que se actualice la tabla movimiento se controlará el valor del saldo para que siempre sea positivo y no mayor que 100.

Registro y auditoría

Cuando muchos usuarios acceden a las bases de datos puede ser que el registro de log no sea suficiente o simplemente dificulte la revisión de la actividad en el servidor en el sentido de saber quién ha hecho que operación y a qué hora. Usaremos un trigger para realizar dicha tarea. Podemos asignar un trigger a una tabla que se dispara después (AFTER) de una sentencia DELETE o UPDATE, que guarde los valores del registro, así como alguna otra información de utilidad en una tabla de log.

Veamos un caso práctico. Queremos saber quién y a qué hora modificó la tabla movimientos en la base de datos de un banco. Para ello creamos un trigger que registre dichas actualizaciones incluyendo los datos antiguos y los nuevos para cada registro modificado.

Lo primero es crear una tabla simple de auditoría:

```
CREATE TABLE auditoria_movimientos
(
      id_mov            INT NOT NULL AUTO_INCREMENT,
      id_cuenta_ant     INT,
      fecha_ant         DATETIME,
      cantidad_ant      FLOAT,
      id_cuenta_n       INT,
      fecha_n           DATETIME,
      cantidad_n        FLOAT,
      usuario           VARCHAR(40),
      fecha_mod         DATETIME,
      PRIMARY KEY(id_mov)
);
```

Y ahora crearemos un trigger para que vaya llenando los registros de esta tabla cada vez que alguién ejecute una actualización sobre la tabla:

```
CREATE TRIGGER trigger_auditoria_mov AFTER UPDATE ON movimientos
FOR EACH ROW
BEGIN
INSERT INTO auditoria_movimientos (id_cuenta_ant, fecha_ant, cantidad_ant,
```

```
id_cuenta_n,      fecha_n, cantidad_n, usuario, fecha_mod)
VALUES (OLD.id_cuenta, OLD.fecha, OLD.cantidad, NEW.id_cuenta, NEW.fecha,
NEW.cantidad, CURRENT_USER(), NOW() );
END;
```

Como se puede observar, el trigger creado anteriormente se activará con la ejecución de la actualización y agregará un nuevo registro a la tabla de auditoría cada vez que se actualice la tabla movimientos.

2 EVENTOS

En MySQL los eventos son tareas que se ejecutan de acuerdo a un horario. Por lo tanto, a veces nos referiremos a ellos como los eventos programados.

Se conocen como triggers temporales ya que conceptualmente son similares diferenciándose en el que el trigger se activa por un evento sobre la base de datos mientras que el evento según una marca de tiempo.

Un evento se identifica por su nombre y el esquema o base de datos al que se le asigna. Lleva a cabo una acción específica de acuerdo a un horario. Esta acción consiste en una o varias instrucciones SQL dentro de un bloque BEGIN/END.

Distinguiremos 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 para activarlo, OFF para desactivarlo u DISABLED si queremos imposibilitar la activación (ponerla a ON) en tiempo de ejecución.

Observando la salida del siguiente comando podemos comprobar que el programador de eventos está activo ya que se ejecuta como un hilo más del servidor.

```
mysql> SHOW PROCESSLIST \G
```

Para poder modificar el estado del planificador/scheduler podemos o bien modificar el fichero de configuración de MySQL y rearrancar la base de datos o bien modificar el valor de la variable *event_scheduler* ejecutando lo siguiente:

```
SET GLOBAL event_scheduler=ON;
```

2.1 GESTIÓN DE EVENTOS

Los comandos para la gestión de eventos son CREATE EVENT, ALTER EVENT, SHOW EVENT y DROP EVENT.

Creación de eventos

La sintaxis para crear eventos es

```
CREATE
[DEFINER = { user | CURRENT_USER }]
EVENT
[IF NOT EXISTS]
event_name
ON SCHEDULE schedule
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comment']
DO sql_statement;

schedule:
```

```
AT timestamp [+ INTERVAL interval] …
| EVERY interval
[STARTS timestamp [+ INTERVAL interval] …]
[ENDS timestamp [+ INTERVAL interval] …]

interval:
quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

Aquí se crea un evento con un nombre asociado a una base de datos o esquema determinado. Veamos las cláusulas:

- La cláusula ON SCHEDULE permite establecer cómo y cuándo se ejecutará el evento. Una sola vez, durante cierto tiempo o en una fecha, hora de inicio y fin determinadas.
- DEFINER especifica el usuario cuyos permisos se tendrán en cuenta en la ejecución del evento.
- event_body es el contenido del evento que se va a ejecutar.
- Las cláusulas COMPLETION permiten mantener el evento aunque haya expirado mientras que DISABLE permite crear el evento en estado inactivo.
- DISABLE ON SLAVE sirve para indicar que el evento se creó en el master de una replicación y que, por tanto, no se ejecutará en el esclavo.

Veamos un ejemplo de evento:

Suponer que tenemos la tabla cuentas de los clientes de un banco

```
CREATE TABLE cuentas
(
      id_cuenta   INT,
      id_cliente  INT,
      saldo       FLOAT,
      year        INT
      PRIMARY KEY(id_cuenta)
);
```

Creemos un evento para que dentro de 6 horas se borren todos los registros del año 2012, pero antes se guarden en una tabla de históricos.

```
DELIMITER $$
CREATE EVENT e_Borra2012
ON SCHEDULE AT now() + INTERVAL 6 HOUR
DO
BEGIN
INSERT INTO historico_cuentas SELECT * FROM cuentas WHERE year=2012;
DELETE FROM cuentas WHERE year = 2012;
END; $$
```

Veamos otro ejemplo: Creemos una tabla que contiene una columna con fechas

```
CREATE TABLE testEventTb
(
      date DATETIME NOT NULL,
      PRIMARY KEY (date)
);
```

Ahora creamos un evento que se ejecuta cada minuto, y en cada ejecución introduce una nueva entrada correspondiente al tiempo en que se ejecuta. Además vamos a borrar todas las entradas anteriores a la última hora.

```
DELIMITER                                                                    $$
CREATE EVENT e_test
ON SCHEDULE
EVERY 1 MINUTE
DO
BEGIN
INSERT INTO testEventTb (date) values (now());
DELETE FROM testEventTb where
date< DATE_SUB(now(),INTERVAL '1' HOUR);
END; $$
```

Modificación de eventos

Para modificar un evento usamos la orden ALTER EVENT con la siguiente sintaxis

```
ALTER
    [DEFINER = { user | CURRENT_USER }]
    EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    [DO event_body]
```

Consulta de eventos

Podemos ver los eventos en scheduling utilizando `SHOW EVENTS;`

Eliminar un evento

Para eliminar un evento usamos `DROP EVENT event_name;`