Capítulo 20. Disparadores (triggers)

Tabla de contenidos

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:

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

Este capítulo describe la sintaxis para crear y eliminar disparadores, y muestra algunos ejemplos de cómo utilizarlos. Las restricciones en el uso de disparadores se tratan en Apéndice H, *Restricciones en características de MySQL*.

En Sección 19.3, “Registro binario de procedimientos almacenados y disparadores” se describe la forma en que se realiza el registro binario (binary logging) para los disparadores.

20.1. 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 (despues), 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. Consulte Sección 19.2.7, “Sentencia compuesta BEGIN ... END”.

Note: Antes de MySQL 5.0.10, los disparadores no podían contener referencias directas a tablas por su nombre. A partir de MySQL 5.0.10, se pueden escribir disparadores como el llamado `testref`, que se muestra en este ejemplo:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END
|

DELIMITER ;

INSERT INTO test3 (a3) VALUES
  (NULL), (NULL), (NULL), (NULL), (NULL),
  (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
  (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

Si en la tabla `test1` se insertan los siguientes valores:

```
mysql> INSERT INTO test1 VALUES
    -> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

Entonces los datos en las 4 tablas quedarán así:

```
mysql> SELECT * FROM test1;
+------+
| a1   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+------+
| a2   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
```

```
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+----+
| a3 |
+----+
|  2 |
|  5 |
|  6 |
|  9 |
| 10 |
+----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM test4;
+----+------+
| a4 | b4   |
+----+------+
|  1 |    3 |
|  2 |    0 |
|  3 |    1 |
|  4 |    2 |
|  5 |    0 |
|  6 |    0 |
|  7 |    1 |
|  8 |    1 |
|  9 |    0 |
| 10 |    0 |
+----+------+
10 rows in set (0.00 sec)
```

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 luego de que fue actualizada.

El uso de `SET NEW.nombre_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.

Nota: Actualmente, los disparadores no son activados por acciones llevadas a cabo en cascada por las restricciones de claves extranjeras. Esta limitación se subsanará tan pronto como sea posible.

La sentencia `CREATE TRIGGER` necesita el privilegio `SUPER`. Esto se agregó en MySQL 5.0.2.

20.2. 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.

La sentencia `DROP TRIGGER` necesita que se posea el privilegio `SUPER`, que se introdujo en MySQL 5.0.2.

20.3. 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 despues 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`. La sintaxis de las mismas se describe en Sección 20.1, “Sintaxis de `CREATE TRIGGER`” y Sección 20.2, “Sintaxis de `DROP TRIGGER`”.

Este es un ejemplo sencillo que asocia un disparador con una tabla para cuando reciba sentencias `INSERT`. Actúa como un acumulador que suma los valores insertados en una de las columnas de la tabla.

La siguiente sentencia crea la tabla y un disparador asociado a ella:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> 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 aqui es `AFTER`.
- La plabra 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.

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+----------------------+
| Total amount inserted |
+----------------------+
```

```
| 1852.48               |
+-----------------------+
```

En este caso, el valor de `@sum` luego de haber ejecutado la sentencia `INSERT` es `14.98 + 1937.50 - 100`, o `1852.48`.

Para eliminar el disparador, se emplea una sentencia `DROP TRIGGER`. El nombre del disparador debe incluir el nombre de la tabla:

```
mysql> DROP TRIGGER account.ins_sum;
```

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`. (Más adelante en esta sección puede verse un ejemplo).

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`.

Las palabras clave `OLD` y `NEW` permiten acceder a columnas en los registros afectados por un disparador. (`OLD` y `NEW` no son sensibles a mayúsculas). 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` sólo 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 actualizado, y `NEW.`*`nom_col`* para referirse a las columnas del registro luego de actualizarlo.

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.

`OLD` y `NEW` son extensiones de MySQL para los disparadores.

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. El siguiente ejemplo ilustra estos aspectos. En él se crea un disparador para `UPDATE`, que verifica los valores utilizados para actualizar cada columna, y modifica 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:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END;//
mysql> delimiter ;
```

Podría parecer más fácil definir una rutina almacenada e invocarla desde el disparador utilizando una simple sentencia `CALL`. Esto sería ventajoso también si se deseara invocar la misma rutina desde distintos disparadores. Sin embargo, una limitación de los disparadores es que no pueden utilizar `CALL`. Se debe escribir la sentencia compuesta en cada `CREATE TRIGGER` donde se la desee emplear.

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.