

Disparadores

Índice

1.	Disparadores (<i>triggers</i>).....	3
2.	Sentenza CREATE TRIGGER.....	3
3.	Sentenzas SHOW TRIGGERS e SHOW CREATE TRIGGER.....	4
4.	Identificadores NEW e OLD en disparadores	5
5.	Utilización de disparadores.....	5
6.	Sentenza DROP TRIGGER	8

1. Disparadores (*triggers*)

Un disparador ou trigger é **un programa almacenado que está asociado a unha táboa**. O disparador está formado por un conxunto de sentenzas que son executadas cando sucede un determinado evento na táboa á que está asociado. Os eventos que activan os disparadores están relacionados con operacións de manipulación de datos. O seu uso axuda a manter a integridade dos datos facendo operacións como validar datos, calcular atributos derivados, levar rexistro das operacións que se fan sobre os datos, ... etc. MySQL permite utilizar triggers dende a versión 5.0.2.

2. Sentenza CREATE TRIGGER

A sentenza CREATE TRIGGER permite crear un disparador, dándolle un nome, establecendo cal é o evento disparador, a táboa á que está asociado, e escribindo as sentenzas que se van a executar cando suceda o evento. Sintaxe da sentenza:

```
CREATE [DEFINER = { usuario | CURRENT_USER }]
  TRIGGER nome_disparador
  momento_execución evento_disparador ON nome_táboa
  FOR EACH ROW corpo_disparador
```

- A cláusula DEFINER permite indicar o nome do usuario que vai a ser considerado o creador do disparador. O valor por defecto é CURRENT_USER que fai referencia ao usuario actual que está creando o disparador.
- *nome_disparador*: é o nome que se lle vai a dar ao disparador. Ten que ser único nunha base de datos, aínda que pode haber varios disparadores co mesmo nome pero en distintas bases de datos.
- *momento_execución*: indica o momento en que se activa ou executa o disparador. Pode tomar os valores:
 - BEFORE: O disparador actívase antes de que se execute a operación de manipulación de datos.
 - AFTER: O disparador actívase despois de que se execute a operación de manipulación de datos.
- *evento_disparador*: indica cal é a operación que activa o disparador. Os valores permitidos son:
 - INSERT: O disparador actívase cando se insire unha nova fila na táboa; por exemplo, porque se executou unha sentenza INSERT ou REPLACE.
 - UPDATE: O disparador actívase cando se modifica unha fila na táboa; por exemplo, porque se executou unha sentenza UPDATE.
 - DELETE: O disparador actívase cando se borra unha fila na táboa; por exemplo, porque se executou unha sentenza DELETE ou REPLACE.Por exemplo: BEFORE INSERT faría que o disparador se active, antes de executar unha sentenza INSERT sobre a táboa, e pode ser utilizado para verificar que os valores que se van a introducir son correctos.
- *nome_táboa*: especifica o nome da táboa á que está asociado o disparador. Non pode ser unha táboa temporal, nin unha vista.
- O texto **FOR EACH ROW** que vai antes do corpo do disparador fai referencia a que esas sentenzas vanse a executar cada vez que se insire, modifica, ou borra unha fila na táboa.

- *corpo_disparador*: pode ser unha sentenza, ou conxunto de sentenzas SQL en forma de bloque de programación empezando por BEGIN e rematando en END. O corpo do disparador pode incluír sentenzas de declaración de variables, de asignación de valores, de control de fluxo, de manipulación de datos, e a maioría das sentenzas da linguaxe SQL, e ademais, pode facer chamadas a rutinas almacenadas (procedementos e funcións definidas polo usuario). **Non pode haber dous disparadores asociados a unha táboa nos que coincida o momento de execución e o evento disparador. Para cada evento disparador (INSERT, UPDATE, DELETE) asociado a unha táboa pódense crear, como máximo, dous disparadores, un que se active antes (BEFORE) e outro que se active despois (AFTER). Por exemplo, non se poden crear dous disparadores BEFORE INSERT para a mesma táboa, pero pódese crear un disparador BEFORE INSERT e outro AFTER INSERT para a mesma táboa.**

Un disparador non se pode modificar unha vez creado, é dicir, non existe unha sentenza **ALTER TRIGGER**. No caso de que se quixera crear un novo disparador e xa existira outro disparador asociado á táboa que coincida no momento de execución e no evento disparador, é necesario borrar o disparador que xa está creado e crear o novo incluíndo no seu corpo todas as sentenzas do disparador que xa existía e as que corresponden ao novo.

Pódese establecer unha regra para os nomes dos disparadores que permita recordalos facilmente para non perder tempo cada vez que se quere facer referencia a eles para crealos ou boralos. Unha regra podería ser poñer primeiro o nome da táboa á que esta asociado, e despois as iniciais do momento de execución (B ou A) e do evento disparador (I, U, ou D). Por exemplo: O nome para un disparador asociado á táboa *artigos*, que se vai a executar antes (BEFORE) de inserir (INSERT) unha fila podería ser *artigosBI*.

3. Sentenzas SHOW TRIGGERS e SHOW CREATE TRIGGER

A información sobre os disparadores creados gárdase no dicionario de datos, igual que o resto de obxectos das bases de datos. En MySQL, a información sobre os disparadores pódese consultar en *information_schema.triggers* mediante unha sentenza SELECT. MySQL dispón, ademais, de dúas instrucións para consultar información sobre os disparadores.

- A sentenza SHOW TRIGGERS, permite ver os disparadores asociados a unha base de datos. Sintaxe da sentenza:

```
SHOW TRIGGERS [FROM nome_bd]
[LIKE 'patrón' | WHERE expresión];
```

Tendo en conta que non pode haber dous disparadores asociados a unha táboa nos que coincida o momento de execución e o evento disparador, é moi útil utilizar esta sentenza para ver os disparadores que hai creados antes de crear un novo.

- A sentenza SHOW CREATE TRIGGER, permite ver información de como foi creado o disparador, incluído o código SQL utilizado para a creación. Sintaxe:

```
SHOW CREATE TRIGGER nome_disparador
```

Exemplo:

```
show create trigger tendabd.detalle_vendasBI;
```

O resultado da sentenza anterior móstrase na zona *Result Grid* de Workbench.

Trigger	sql_mode	SQL Original Statement	character_set_client	collation_connection	Database Collation
detalle_vendasBI	NO_ENGINE_SUBSTITUTION	CREATE DEFINER='root'@'localhost' trigger ...	utf8	utf8_general_ci	latin1_spanish_ci

4. Identificadores NEW e OLD en disparadores

É obrigatorio utilizar os identificadores NEW ou OLD diante do nome da columna cando no corpo do disparador se teña que facer referencia a algunha columna da táboa á que está asociado. Exemplo: *NEW.nomeColumna* fai referencia ao novo valor que toma a columna despois de executar unha sentenza INSERT ou UPATE.

- Cando o evento disparador é unha operación INSERT, só se pode utilizar o identificador NEW para facer referencia aos valores das columnas da nova fila que se está a inserir.
- Cando o evento disparador é unha operación DELETE, só se pode utilizar o identificador OLD para facer referencia aos valores que tiñan as columnas da fila que se está a borrar.
- Cando o evento disparador é unha operación UPDATE, pódese utilizar o identificador NEW para facer referencia aos valores que toman as columnas despois de facer a modificación, e o identificador OLD para facer referencia aos valores que tiñan as columnas antes da modificación.

5. Utilización de disparadores

Unha vez creados os disparadores, quedan almacenados no servidor e actívanse de maneira automática cada vez que se executa a operación á que están asociados.

Alguns casos nos que poden ser de utilidade os disparadores:

- Validar os datos que se introducen nas táboas, verificando que cumpran as restricións impostas polo modelo, antes de que se execute unha sentenza INSERT sobre a táboa.

Exemplo: cando se fai unha venda, antes de gravar unha liña de detalle da venda cunha sentenza INSERT hai que comprobar as unidades que hai en stock para o artigo que se vai a vender.

```
use tendabd;
drop trigger if exists detalle_vendasBI;
delimiter //
create trigger detalle_vendasBI before insert on detalle_vendas
for each row
begin
declare vStockActual smallint;
set vStockActual = (select art_stock
                    from artigos
```

```

        where art_codigo = new.dev_artigo);
if new.dev_cantidad > vStockActual then
    signal sqlstate '45000' set message_text = 'Non hai stock suficiente';
end if;
end
//
delimiter ;

```

Antes de que se insira unha fila na táboa *detalle_vendas*, compróbase se a cantidade pedida é maior có número de unidades que hai no almacén. No caso de non ser suficiente, abórtase a inserción provocando un erro que se asocia a un código SQLSTATE, neste caso 45000, e móstrase unha mensaxe de erro. Pódese facer a proba executando unha sentenza INSERT como a seguinte:

```

insert into tendabd.detalle_vendas
values (1,3,'0713242',50,300.50,0);

```

Que provocaría a seguinte mensaxe:

```

333 11:06:55 insert into tendabd.detalle_vendas values (1,3,'07132... Error Code: 1644. Non hai stock suficiente 0.0100 sec

```

- Actualizar atributos derivados coa información recollida dunha operación de actualización (INSERT, UPDATE ou DELETE) nunha táboa.

Exemplo: cando se insire unha nova fila na táboa *detalle_vendas* hai que restar na columna *stock* da táboa de *artigos* o número de unidades que se venden, despois de comprobar o stock dese artigo.

```

use tendabd;
drop trigger if exists detalle_vendasAI;
delimiter //
create trigger detalle_vendasAI after insert on detalle_vendas
for each row
begin
update artigos
set art_stock = art_stock - new.dev_cantidad
where art_codigo = new.dev_artigo;
end
//
delimiter ;

```

Despois de gravar unha fila na táboa *detalle_vendas*, faise unha actualización da columna *art_stock* na táboa de *artigos*, restándolle o contido da columna *dev_cantidad* da táboa *detalle_vendas* no artigo que ten como código o valor almacenado na columna *dev_artigo*. Hai que ter en conta que xa está creado o disparador do exemplo anterior, polo que antes de inserir a fila na táboa *detalle_vendas*, xa se comprobou que o stock actual é suficiente para ese artigo. Pódese facer a proba executando unha sentenza INSERT como a seguinte:

```

insert into tendabd.detalle_vendas
values (1,3,'0713242',1,300.50,0);

```

- Crear táboas de auditoría que recollan os cambios que os usuarios fan nas táboas dunha base de datos. Deste xeito pódese saber quen fixo cambios nela e en que momento. Isto

pode ser de gran axuda para levar o rexistro de operacións que obriga a Lei Orgánica de Protección de Datos (LOPD) para certos datos sensibles. Para poder levar o rexistro das operacións que fan os usuarios en cada táboa hai que crear tres disparadores asociados a cada táboa, que se executen despois de facer cada operación de manipulación de datos (INSERT, UPDATE ou DELETE).

Exemplo: Levar un rexistro de todos os cambios que se fan na columna *clt_desconto* da táboa *clientes*. Cada vez que se fai un cambio no contido da columna, grávase unha fila na táboa *rexistro_cambios_desconto* co nome do usuario conectado, a data e hora en que se fai o cambio, valor da columna antes da modificación e o valor da columna despois da modificación.

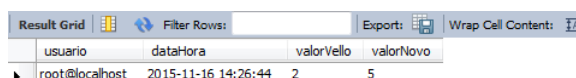
```
use tendabd;
-- creación da táboa de rexistro se non existe
create table if not exists rexistro_cambios_desconto
(
  usuario varchar(80),
  dataHora datetime default now(),
  valorVello tinyint unsigned,
  valorNovo tinyint unsigned
) engine MyISAM;

drop trigger if exists clientesAU;
delimiter //
-- creación do disparador
create trigger clientesAU after update on clientes
for each row
begin
  if new.clt_desconto <> old.clt_desconto then
    insert into rexistro_cambios_desconto (usuario, valorVello, valorNovo)
      values (user(), old.clt_desconto, new.clt_desconto);
  end if;
end
//
delimiter ;
```

O primeiro que fai o disparador é comprobar se foi modificado o contido da columna *clt_desconto*, e dicir, se o contido da columna antes de executar a sentenza UPDATE sobre a táboa *clientes*, é distinto do contido da columna despois de executar a sentenza UPDATE. Só no caso de que ese valores foran distintos, executase unha sentenza INSERT na táboa de rexistro. A data e hora do sistema gárdase na columna *dataHora* como valor por defecto. Pódese facer a proba executando unha sentenza UPDATE como a seguinte:

```
update clientes
  set clt_desconto = 5
  where clt_id = 1;

-- ver o contido da táboa de rexistro
select * from rexistro_cambios_desconto;
```



usuario	dataHora	valorVello	valorNovo
root@localhost	2015-11-16 14:26:44	2	5

- Controlar as restricións de integridade referencial en táboas non transacionais. Nas táboas transacionais (exemplo: *InnoDB*), o servidor é quen se encarga de comprobar que se verifican as restricións de integridade referencial, pero en táboas co motor non transacional (exemplo: *MyISAM*), o servidor non fai as comprobacións de restrición de integridade referencial.

Exemplo: As táboas da base de datos *traballadores* utilizan o motor de almacenamento *MyISAM* que é non transacional. Escribir un disparador que comprobe se existe o departamento no que traballa un empregado na táboa de departamentos antes de inserir os datos do empregado. No caso de non existir o departamento, abórtase a inserción e móstrase unha mensaxe co texto 'Non existe o departamento'.

```
use traballadores;
drop trigger if exists empregadoBI;
delimiter //
create trigger empregadoBI before insert on empregado
for each row
begin
if (select count(*) from departamento where depNumero = new.empDepartamento) = 0 then
    signal sqlstate '45000' set message_text = 'Non existe o departamento';
end if;
end
//
delimiter ;
```

O disparador busca o código do departamento do empregado na táboa *departamento*. De non existir (o número de filas que devolve a consulta é cero), devólvese un erro. O erro provoca que termine a execución do procedemento e móstrase a mensaxe coa información do erro. Pódese facer a proba executando unha sentenza INSERT como a seguinte:

```
insert into empregado (empNumero, empDepartamento, empExtension, empDataNacemento, empDataIngreso, empSalario, empComision, empFillos, empNome)
values (900, 850, 520, '1995-03-11', '2015-11-15', 900.00, 100.00, 1, 'SUAREZ, XULIO');
```

Ao non existir o departamento co código 900 na táboa departamento, abórtase a inserción e móstrase a mensaxe coa información do erro:

124 18:51:42 insert into empregado (empNumero, empDepartamen... Error Code: 1644. Non existe o departamento 0.000 sec

6. Senteza DROP TRIGGER

A sentenza DROP TRIGGER permite borrar un disparador. Sintaxe:

```
DROP TRIGGER [IF EXISTS] [nome_base_datos].nome_disparador
```

Cando se borra unha base de datos bórranse todos os disparadores asociados a ela.