

Bases de datos

Unidade 7: Disparadores e eventos

1. A03. Disparadores e eventos

1.1 Introducción

1.1.1 Obxectivos

Os obxectivos desta actividade son:

- Crear disparadores (*triggers*).
- Planificar eventos.

1.1.2 Software

Utilizarase a plataforma WAMP (Windows-Apache-MySQL-PHP) WampServer 2.5 (última versión estable en outubro 2015), que inclúe MySQL Community Edition 5.6.17 como SXBDR (Sistema Xestor de Bases de Datos Relacional). As razóns de utilización deste software son que:

- É software libre, polo que o alumnado poderá descargalo de forma gratuíta e utilízalo legalmente na súa casa.
- É unha forma sinxela de facer a instalación do software necesario para desenvolver aplicacións web.



Páxina oficial de  WampServer: <http://www.wampserver.com>



Páxina oficial de  MySQL: <https://www.mysql.com/>

Utilizarase MySQL Workbench 6.3 como ferramenta cliente gráfica xa que é a recomendada por MySQL en outubro de 2015, aínda que tamén poderían utilizarse outras como phpMyAdmin, EMS MyManager, ou MySQL Query Browser.

O resultado das probas dos guións SQL desta actividade, ilustrarase normalmente cunha imaxe capturada da zona de manipulación de datos de Workbench. Para completar esa información, por exemplo, cando non é posible mostrar o resultado completo, mostráranse outras zonas de Workbench, como por exemplo a zona de saída (output) con información do estado da execución do guión e o número de filas que devolve.



En <https://www.mysql.com/products/workbench/> pode obterse información detallada sobre a ferramenta MySQL Workbench e descargar o software.



En <http://dev.mysql.com/doc/index-gui.html> pode descargarse o manual de MySQL Workbench.



O material anexo a esta actividade inclúe unha Guía básica de MySQL Workbench 6.3.

1.1.3 Bases de datos de traballo

As bases de datos *practicasl*, *traballadores* e *tendaBD*, utilizaranse nalgúns exemplos e tarefas desta actividade. Antes de empezar a probar os exemplos ou realizar as tarefas, hai que executar os scripts de creación no servidor e poñer en uso as bases de datos cando corresponda. Os scripts atópanse no cartafol anexo a esta actividade descrito no apartado '3.3 Material auxiliar'.

1.2 Actividade

1.2.1 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, ...

MySQL permite utilizar triggers dende a versión 5.0.2.

1.2.1.1 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, LOAD DATA, 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.

O nome dos disparadores debe seguir unha regra que permita recordalo facilmente para non perder tempo cada vez que se quere facer referencia a el para crealo ou boralo. 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*.

1.2.1.2 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];
```

No caso de non especificar o nome da base de datos (*nome_bd*), móstranse os disparadores asociados á base de datos activa. Pódense utilizar cláusulas LIKE ou WHERE para mostrar só os disparadores que cumpran unha condición, e non todos. Exemplo:

```
show triggers from tendabd where character_set_client = 'utf8';
```

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

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_client	collation_connection	Database Collation
dDetalle_vendasBI	INSERT	dDetalle_vendas	begin declare vStockActual smallint; set vStock...	BEFORE	2015-01-14 10:10:10	NO_ENGINE_SUBSTITUTION	root@localhost	utf8	utf8_general_ci	latin1_spanish_ci

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 sentença para ver os disparadores que hai creados antes de crear un novo.

- A sentença `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.dDetalle_vendasBI;`

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

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

1.2.1.3 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 sentença INSERT ou UPDATE.

- 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.

1.2.1.4 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 sentença INSERT sobre a táboa.

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

/*

u703exemplo_triggerBI.sql

NOME DISPARADOR: detalle_vendasBI

DATA CREACIÓN: 12/11/2015
 AUTOR: Grupo licenza 2015
 TAREFA A AUTOMATIZAR: - Comprobar o stock actual para o artigo que se vende cada vez que se insire unha fila na táboa detalle_venta
 EVENTO DISPARADOR: - INSERT
 MOMENTO DISPARADOR: - BEFORE
 RESULTADOS PRODUCIDOS: - Mensaxe no caso de non haber stock suficiente

```

*/
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



Tarefa 1. Crear disparadores para validar a entrada de datos.



Tarefa 2. Consultar información sobre os disparadores creados.

- 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.

```

/*
u703exemplo_triggerAI.sql

```

NOME DISPARADOR: detalle_vendasAI
 DATA CREACIÓN: 16/11/2015
 AUTOR: Grupo licenza 2015
 TAREFA A AUTOMATIZAR: - Actualizar o stock actual para o artigo que se vende cada vez que se insire unha fila na táboa detalle_venta
 EVENTO DISPARADOR: - INSERT

MOMENTO DISPARADOR: - AFTER
RESULTADOS PRODUCIDOS: - Columna de stock, na táboa de artigos, actualizada

```
*/  
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);
```



Tarefa 3. Crear disparadores para actualizar atributos derivados.

- 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 *registro_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.

```
/*  
u703exemplo_triggerAU.sql
```

NOME DISPARADOR: clientes_AU

DATA CREACIÓN: 16/11/2015

AUTOR: Grupo licenza 2015

TAREFA A AUTOMATIZAR: - Rexistra na táboa *registro_cambios_desconto*, todos os cambios que se fan na columna *clt_desconto* da táboa de *clientes*. Para cada cambio que se fai gárdase o nome do usuario que fai o cambio, data e hora na que se fai o cambio, valor da columna antes de facer a modificación, e valor da columna despois de facer a modificación.

EVENTO DISPARADOR: - UPDATE

MOMENTO DISPARADOR: - AFTER

RESULTADOS PRODUCIDOS: - Non mostra nada na pantalla. Cada vez que se fai unha

modificación do contido da columna `clt_desconto` da
táboa de clientes, insírese unha fila na táboa
`registro_cambios_desconto`

```
*/
use tendabd;
-- creación da táboa de rexistro se non existe
create table if not exists registro_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 registro_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, execútase 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 registro_cambios_desconto;
```

Result Grid			
Filter Rows:		Export:	Wrap Cell Content: IA
usuario	dataHora	valorVello	valorNovo
root@localhost	2015-11-16 14:26:44	2	5



Tarefa 4. Crear disparadores para levar rexistros de operacións.

- 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'.

```
/*
u703exemplo2_triggerBI.sql
*/
NOME DISPARADOR: empregadoBI
DATA CREACIÓN: 12/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR: - Comprobar que existe, na táboa departamento, o código do
                        departamento no que traballa o empregado cada vez que se
                        insire unha fila na táboa empregado.
EVENTO DISPARADOR:    - INSERT
MOMENTO DISPARADOR:    - BEFORE
RESULTADOS PRODUCIDOS: - Abortar a inserción no caso de non existir o código do
                        departamento no que traballa o empregado na táboa
                        departamento e mostrar a mensaxe '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



Tarefa 5. Crear disparadores para controlar as restricións referenciais en táboas non transacionais.

1.2.1.5 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.



Tarefa 6. Borrar disparadores.

1.2.2 Planificador de eventos

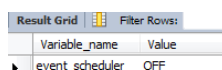
MySQL, a partir da versión 5.1, inclúe unha característica que se chama 'Planificador de eventos', que permite executar sentenzas SQL ou procedementos almacenados tendo en conta un calendario establecido, indicando o momento, ou o intervalo de tempo no que se executarán. Os eventos son especialmente útiles para realizar operacións de administración non presenciais como actualizacións periódicas de informes, vencemento de datos, análise de táboas, ou rotación de táboas de rexistro. Proporcionan unha gran potencia ao servidor utilizado conxuntamente cos procedementos almacenados e os disparadores.

A configuración dun servidor MySQL cando se fai unha instalación por defecto, non habilita o uso do planificador de eventos. Para habilitalo, hai que modificar o valor que ten a variable global *event_scheduler*, executando a sentenza:

```
set global event_scheduler = on;
```

No caso de que a variable tome o valor *off*, pódense crear eventos e non se mostra ningún erro, pero non funciona o calendario, e polo tanto non se van a executar. Pódese consultar o valor que ten a variable executando a sentenza:

```
show variables like 'event_scheduler';
```



Variable_name	Value
event_scheduler	OFF

1.2.2.1 Senteza CREATE EVENT

A sentenza CREATE EVENT permite crear e programar eventos. O evento creado queda asociado á base de datos que estea activa, ou ben á base de datos á que se fai referencia cando se utilizan nomes cualificados. O evento só se executará se o planificador de eventos está habilitado. Sintaxe:

```
CREATE [DEFINER = { usuario| CURRENT_USER }]
EVENT [IF NOT EXISTS] nome_evento
ON SCHEDULE calendario
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comentario']
DO corpo_evento;
```

- A cláusula DEFINER permite indicar o nome do usuario que vai ser considerado o creador do evento. Se non se especifica nada tómase CURRENT_USER que fai referencia ao usuario actual que está creando o evento.
- *nome_evento*: é o nome que vai ter o evento. Non é sensible a maiúsculas e minúsculas polo que *meuEvento* e *meuevento* serían iguais. Ten que ser único dentro do esquema dunha base de datos.
- A cláusula ON SCHEDULE determina cando ou con que frecuencia e durante canto tempo se executan as sentenzas que forman o corpo do evento.
- *calendario*: O calendario da programación ten a seguinte sintaxe:

```
AT dato_timestamp [+ INTERVAL intervalo] ...
| EVERY intervalo
  [STARTS dato_timestamp [+ INTERVAL intervalo] ...]
  [ENDS dato_timestamp [+ INTERVAL intervalo] ...]
```

No calendario pódense utilizar as opcións:

- AT para indicar o momento en que se van a executar as sentenzas contidas no corpo do evento.

- EVERY para indicar que as sentenzas teñen que executarse cada certo período de tempo. No caso de utilizar a opción EVERY, de forma opcional, tamén se pode indicar o momento en que empezan a executarse (STARTS) e o momento en que se deixan de executar (ENDS).
- *intervalo*: A definición do intervalo ten a seguinte sintaxe:
`cantidade {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE | WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE | DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}`
- A opción [ON COMPLETION [NOT] PRESERVE] permite indicar ao servidor se hai que borrar o evento unha vez que xa se executou. Se non se especifica nada, o comportamento normal é que un evento se borre de forma automática cando se deixa de executar, ven sexa porque só se executa unha vez (AT), ou porque acabou o tempo en que se tiña que executar (ENDS).
- A opción [ENABLE | DISABLE | DISABLE ON SLAVE] permite habilitar ou deshabilitar o evento no momento da creación. Pódese modificar máis tarde o seu estado coa sentenza ALTER EVENT.
- A opción [COMMENT 'comentario'] permite engadir un comentario na descrición do evento.
- *corpo_evento*: Pode ser unha sentenza, ou conxunto de sentenzas SQL en forma de bloque de programación empezando por BEGIN e rematando en END. No caso de estar formado por unha soa sentenza non sería necesario utilizar os delimitadores de bloque BEGIN e END. O corpo do evento pode incluír chamadas a rutinas almacenadas (procedementos e funcións definidas polo usuario).

1.2.2.2 Exemplos de creación de eventos

A creación dun evento require como mínimo:

- As palabras claves CREATE EVENT, e un nome para o evento (*'nome_evento'*) que ten que ser único dentro do esquema dunha base de datos.
- A cláusula ON SCHEDULE para establecer o calendario de execución.
- A cláusula DO na que se escribe o conxunto de sentenzas que van a executar.

Exemplo 1: Na táboa *concerto* da base de datos *practicasl* está almacenado o prezo dos concertos programados. Decídese incrementar os prezos dos concertos un 15%. Este incremento entrará en vigor dentro de 24 horas. Para este exemplo, pódese crear un evento que se execute unha soa vez dentro de 24 horas, tomando como referencia a data do sistema e se elimine despois automaticamente:

```
-- activar base de datos practicasl
use practicasl;
-- creación do evento
drop event if exists actualizaPrezo;
create event actualizaPrezo
on schedule at now() + interval 24 hour
do update concerto set prezo = prezo * 1.15;
```

Exemplo 2: Un caixeiro automático so permite retirar 1.000 € diarios por conta. O sistema irá sumando os reintegros feitos no día en cada conta, restándolle o seu importe ao límite dispoñible que está gardado na columna *limite_dia* na táboa de *contas*. Ás 00:00 de cada día habería que actualizar o límite a retirar e establecer os 1.000 € para o día que empeza. Para este exemplo, é útil crear un evento que se execute tódolos días ás 00:00.

```
-- activar base de datos practicasl
use practicasl;
```

```
-- creación da táboa contas se non existe
create table if not exists contas (
idConta char(20),
dataApertura date,
saldo decimal(12,2),
limiteDia decimal(10,2),
podeRetirar bit default 0
) engine = myisam;
-- creación do evento
drop event if exists actualizaLimite;
create event actualizaLimite
on schedule every 1 day starts '2015-01-01 00:00:00'
do update contas set limiteDia = 1000;
```

Este evento é recorrente, e executarase ata que se borre, porque non se utilizou a cláusula ENDS no momento da creación do evento. No caso de utilizar a cláusula ENDS, o evento deixa de executarse na data e hora sinalados e bórrase automaticamente.

1.2.2.3 Sentenzas SHOW EVENTS e SHOW CREATE EVENT

A información dos eventos creados gárdase no diccionario de datos, igual que o resto de obxectos das bases de datos. No caso de MySQL, a información sobre os disparadores pódese consultar en *mysql.event* e en *information_schema.events*. MySQL tamén dispón das sentenzas SHOW EVENTS e SHOW CREATE EVENTS.

SHOW EVENTS permite ver os eventos creados. A sintaxe é:

```
SHOW EVENTS [{FROM | IN} nome_bd]
[LIKE 'patrón' | WHERE expresión]
```

No caso de non especificar o nome da base de datos (*nome_bd*), móstranse os eventos asociados á base de datos activa. Pódense utilizar cláusulas LIKE ou WHERE para mostrar só os eventos que cumpran unha condición, e non todos. Exemplo:

```
show events from practicas1;
```

Db	Name	Definer	Time zone	Type	Execute at	Interval value	Interval field	Starts	Ends	Status	Original
practicas1	actualizaLimite	root@localhost	SYSTEM	RECURRING	NULL	1	DAY	2015-01-01 00:00:00	NULL	ENABLED	1
practicas1	actualizaPrezo	root@localhost	SYSTEM	ONE TIME	2015-11-18 17:40:21	NULL	NULL	NULL	NULL	ENABLED	1

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

```
SHOW CREATE EVENT nome_evento
```

Exemplo:

```
SHOW CREATE EVENT practicas1.actualizaLimite;
```

Event	sql_mode	time_zone	Create Event	character_set_client	collation_connection	Database Collation
actualizaLimite	NO_ENGINE_SUBSTITUTION	SYSTEM	CREATE DEFINER='root'@'localhost' EVENT ...	utf8	utf8_general_ci	utf8_unicode_ci

1.2.2.4 Senteza ALTER EVENT

Un evento só pode ser modificado polo usuario que o creou (DEFINER) ou por usuarios que teñan permisos sobre ese evento. O usuario que executa a orde ALTER pasará a ser considerado como o usuario que crea o evento (DEFINER).

A sentenza ALTER EVENT permite modificar un evento, sen necesidade de borralo e volvelo a crear. Sintaxe:

```
ALTER [DEFINER = { usuario | CURRENT_USER }]
```

```

EVENT nome_evento
[ON SCHEDULE calendario]
[ON COMPLETION [NOT] PRESERVE]
[RENAME TO novo_nome_evento]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comentario']
[DO corpo_evento]

```

A sintaxe das cláusulas `DEFINER`, `ON SCHEDULE`, `ON COMPLETION`, `ENABLE/DISABLE`, `COMMENT` e `DO`, é exactamente igual que na sentenza `CREATE EVENT`, pero só se pode executar para eventos que xa existan. A cláusula `RENAME` permite cambiar o nome a un evento.

Nunha sentenza `ALTER EVENT` pódense modificar unha ou máis cláusulas do evento, especificando só aquelas cláusulas nas que queremos facer cambios. As cláusulas que se omiten non se modificarán e permanecerán os valores que se lles deu no momento da creación. Exemplos:

- Para cambiar só o calendario do evento *actualizaLimite* do exemplo 1, hai que executar a seguinte sentenza:

```

alter event actualizaLimite
  on schedule every 1 day
  starts '2015-11-01 00:00:00'
  ends '2016-12-31 00:00:00';

```

Neste caso, faise un cambio que afecta ao calendario, logo só hai que incluír a cláusula `ON SCHEDULE`.

- É posible modificar o nome dun evento, ou movelo dunha base de datos a outra coa cláusula `RENAME TO`. Exemplo:

```

alter event practicas1.actualizaLimite
  rename to utilidades.actualizaLimite;

```

Neste exemplo, o evento *actualizaLimite* pásese da base de datos *practicas1* á base de datos *utilidades*.

- Pódese habilitar e deshabilitar un evento coa cláusula `ENABLE/DISABLE`. Exemplo:

```

alter event utilidades.actualizaLimite
  disable
  comment 'Deshabilitado por Julia Mendez o 12/12/2015';

```

Despois de executar a sentenza, o evento deixa de executarse pero permanece gardado no servidor e volverá a executarse cando se volva a habilitar. Ademais engade un comentario, empregando a cláusula `COMMENT`. Neste exemplo pódese ver que se poden modificar varias cláusulas na mesma sentenza `ALTER EVENT`.

1.2.2.5 Senteza DROP EVENT

A sentenza `DROP EVENT` permite borrar un evento. Sintaxe:

```

DROP EVENT [IF EXISTS] nome_evento

```

Cando se borra unha base de datos bórranse os eventos que estean asociados a ela.

1.3 Tarefas

As tarefas propostas son as seguintes:

- Tarefa 1. Crear disparadores para validar a entrada de datos.
- Tarefa 2. Consultar información sobre os disparadores creados.

- Tarefa 3. Crear disparadores para actualizar atributos derivados.
- Tarefa 4. Crear disparadores para levar rexistros de operacións.
- Tarefa 5. Crear disparadores para controlar as restricións referenciais en táboas non transacionais.
- Tarefa 6. Borrar disparadores.
- Tarefa 7. Planificar eventos.

1.3.1 Tarefa 1. Crear disparadores para validar a entrada de datos

A tarefa consiste en crear e probar un disparador na base de datos *practicasl* que valide o contido da columna *dni*, antes de inserir unha fila na táboa *empregado*. Se o dni é incorrecto porque a letra non é a que corresponde aos díxitos do dni, hai que abortar a inserción e mostrar a mensaxe 'DNI non válido'.

Solución

– Código de creación

```
/*
u703tarefa01.sql
*/

NOME DISPARADOR: practicas1.empregadoBI
DATA CREACIÓN: 16/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR: - Antes de inserir unha fila na táboa empregado validar o dni.
                        Para facer a validación utilízase a función utilidades.dni.
                        Extraese a letra da columna dni na variable vLetra, e o resto
                        de díxitos na variable vNumero. No caso que a letra calculada
                        coa función utilidades.dni(vNumero) non coincida co contido
                        da variable vLetra, abórtase a inserción e móstrase a unha
                        mensaxe de erro.

EVENTO DISPARADOR: - INSERT
MOMENTO DISPARADOR: - BEFORE
RESULTADOS PRODUCIDOS: - Se a letra do dni non é a que lle corresponde aborta a
                        inserción e mostra a mensaxe 'DNI non válido'

*/

drop trigger if exists practicas1.empregadoBI;
delimiter //
create trigger practicas1.empregadoBI before insert on practicas1.empregado
for each row
begin
    declare vLetra char(1);
    declare vNumero char(8);
    set vLetra = right(trim(new.dni),1);
    set vNumero = left(trim(new.dni),length(trim(new.dni))-1);
    if utilidades.letraDni(vNumero) != vLetra then
        signal sqlstate '45000' set message_text = 'DNI non válido';
    end if;
end
//
delimiter ;
```

– Proba de funcionamento

Pódese probar o funcionamento do disparador inserindo un empregado que teña un dni non válido, e logo facer a proba cun que teña o dni válido. A seguinte sentenza intenta

dar de alta un empregado cun dni non válido.

```
insert into practicas1.empleado
  (dni, nss, nome, dataNacemento, sexo, salario, codigoDepartamento)
values ('36578J','15845782','Diaz Lopez, Juan','1996/02/24','h',2500,2);
```

O resultado da execución da sentenza INSERT anterior en Workbench dá lugar á seguinte mensaxe de erro:

21 23:06:31 insert into practicas1.empleado (dni, nss, nome, dat... Error Code: 1644. DNI non válido 0.000 sec

1.3.2 Tarefa 2. Consultar información sobre os disparadores creados

A tarefa consiste en consultar os disparadores asociados á base de datos *practicas1*, e mostrar información do disparador *practicas1.empleadoBI* creado na *tarefa1*.

Solución

Pódese consultar información do disparador creado de dúas maneiras: executando unha consulta cunha sentenza SELECT na táboa *information_schema.triggers*, ou ben executando as sentenzas SHOW TRIGGERS, ou SHOW CREATE TRIGGER.

```
-- consulta na táboa information_schema.triggers de todos os triggers
select * from information_schema.triggers;
-- consulta na táboa information_schema.triggers dos triggers de practicas1
select * from information_schema.triggers
  where trigger_schema = tendabd;
-- consulta dos triggers de practicas1
show triggers from practicas1;
-- consulta da información do trigger practicas1.empleadoBI
show create trigger practicas1.empleadoBI;
```

Unha parte da información mostrada en Workbench despois de executar a última das sentenzas anteriores é:

Trigger	sql_mode	SQL Original Statement	character_set_client	collation_connect
empleadoBI	NO_ENGINE_SUBSTITUTION	CREATE DEFINER='root'@'localhost' trigger ...	utf8	utf8_general_ci

1.3.3 Tarefa 3. Crear disparadores para actualizar atributos derivados

A tarefa consiste en crear e probar o funcionamento dos disparadores necesarios na base de datos *tendabd* para poder manter actualizado o valor das columnas *clt_vendas* e *clt_ultima_venta* na táboa *clientes*. A columna *clt_vendas* garda información do número de vendas que se lle fixeron ao cliente, e a columna *clt_ultima_venta* garda información da data na que se lle fixo a última venda.

Solución

- Código de creación

Hai que crear tres disparadores para a táboa vendas, para as operacións AFTER INSERT, AFTER UPDATE, e AFTER DELETE.

Antes hai que executar a sentenza SHOW TRIGGERS para saber se xa hai algún disparador para esas operacións:

```
show triggers from tendabd;
```

Despois de confirmar que non hai disparadores asociados a esas operacións, escríbese o

guión de sentenzas para crear os tres disparadores:

```
/*
u703tarefa03.sql
*/

NOME DISPARADOR: tendabd.vendasAI,
                 tendabd.vendasAU,
                 tendabd.vendasAD

DATA CREACIÓN: 19/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR: - Manter actualizada a columna clt_vendas da clientes, que
                        contén información do número de vendas que se lle fixeron
                        ao cliente
                        * Cando se insire unha nova venda hai que sumarlle 1 ao
                        contido da columna correspondente ao cliente ao que se lle
                        fai a venda.
                        * Cando se borra unha venda hai que restarlle 1 ao contido
                        da columna correspondente ao cliente ao que se lle fai a
                        venda.
                        * Cando se cambia o cliente dunah fila da táboa de vendas, hai
                        que restarlle 1 á columna correspondente ao cliente ao que lle
                        correspondía a venda antes de facer o cambio, e sumarlle 1 á
                        columna correspondente ao cliente ao se lle asignou a venda,
                        despois de facer o cambio.
                        - Manter actualizada a columna clt_ultima_venta, que contén
                        información da data na que se lle fixo a última venda
                        ao cliente.

EVENTO DISPARADOR: - INSERT, UPDATE, DELETE
MOMENTO DISPARADOR: - AFTER
RESULTADOS PRODUCIDOS: - Columnas clt_vendas e clt_ultima_venta actualizadas
*/

-- disparador que actualiza as columnas clt_vendas e clt_ultima_venta despois de
-- inserir unha fila na táboa de vendas
drop trigger if exists tendabd.vendasAI;
delimiter //
create trigger tendabd.vendasAI after insert on tendabd.vendas
for each row
begin
update clientes
  set clt_vendas = ifnull(clt_vendas,0)+1,
      clt_ultima_venta = date(new.ven_data)
  where clt_id = new.ven_cliente;
end
//
delimiter;

-- disparador que actualiza as columnas clt_vendas e clt_ultima_venta despois de
-- modificar unha fila na táboa de vendas
drop trigger if exists tendabd.vendasAU;
delimiter //
create trigger tendabd.vendasAU after update on tendabd.vendas
for each row
begin
-- actualización da columna clt_vendas
if old.ven_cliente != new.ven_cliente then
  update clientes
    set clt_vendas = clt_vendas-1
    where clt_id = old.ven_cliente;
  update clientes
    set clt_vendas = clt_vendas+1
```



```

        where clt_id = new.ven_cliente;
end if;
-- actualización da columna clt_ultima_venta
if date(new.ven_data) > (select clt_ultima_venta
                        from clientes
                        where clt_id = new.ven_cliente)
then
    update clientes
        set clt_ultima_venta = date(new.ven_data)
        where clt_id = new.ven_cliente;
end if;
end
//
delimiter;
-- disparador que actualiza as columnas clt_vendas e clt_ultima_venta despois de
-- borrar unha fila na táboa de vendas
drop trigger if exists tendabd.vendasAD;
delimiter //
create trigger tendabd.vendasAD after delete on tendabd.vendas
for each row
begin
-- actualización da columna clt_vendas
    update clientes
        set clt_vendas = clt_vendas-1
        where clt_id = old.ven_cliente;
-- actualización da columna clt_ultima_venta
if date(old.ven_data) = (select clt_ultima_venta
                        from clientes
                        where clt_id = old.ven_cliente)
    and (select count(*)
        from vendas
        where ven_cliente=old.ven_cliente
        and date(ven_data) = date(old.ven_data)) = 0
then
    update clientes
        set clt_ultima_venta = (select max(date(ven_data)) from vendas
                                where ven_cliente=old.ven_cliente
                                and date(ven_data) != date(old.ven_data))
        where clt_id = old.ven_cliente;
end if;
end
//
delimiter ;

```

- Proba de funcionamento

Pódese probar o funcionamento dos disparadores inserindo, modificando e borrando unha fila na táboa de vendas:

```

-- comprobación para a operación de inserción na táboa de vendas:
select clt_id, clt_cif, clt_apellidos, clt_nome, clt_vendas, clt_ultima_venta
from tendabd.clientes;

```

Result Grid						
Filter Rows:						
Edit:     Export/Import:						
	dt_id	dt_cif	dt_apellidos	dt_nome	dt_vendas	dt_ultima_venta
▶	1	14013338J	Portela Carracedo	Francisco	2	2015-06-10
	2	32727807A	Barreira Vila	Juan	1	2015-05-27
	3	18533827J	Armas Tellado	Jose	3	2015-06-17
	4	31410232Y	Rodríguez Piñeiro	Brian	0	NULL

```

insert into vendas (ven_tenda, ven_empleado, ven_cliente, ven_data)
values (1,1,1,now());
select clt_id, clt_cif, clt_apellidos, clt_nome, clt_vendas, clt_ultima_venta

```

```
from tendabd.clientes;
```

dt_id	dt_cif	dt_apellidos	dt_nome	dt_vendas	dt_ultima_venta
1	14013338J	Portela Carracedo	Francisco	3	2015-11-20
2	32727807A	Barreira Vila	Juan	1	2015-05-27
3	18533827J	Armas Tellado	Jose	3	2015-06-17
4	31410232Y	Rodríguez Piñeiro	Brian	0	NULL

```
-- comprobación para a operación de modificación na táboa de vendas:
-- a venda anterior cámbiase para o cliente 2
update vendas
  set ven_cliente = 2
where ven_id = 151;
select clt_id, clt_cif, clt_apellidos, clt_nome, clt_vendas, clt_ultima_venta
from tendabd.clientes;
```

dt_id	dt_cif	dt_apellidos	dt_nome	dt_vendas	dt_ultima_venta
1	14013338J	Portela Carracedo	Francisco	2	2015-06-10
2	32727807A	Barreira Vila	Juan	2	2015-11-20
3	18533827J	Armas Tellado	Jose	3	2015-06-10
4	31410232Y	Rodríguez Piñeiro	Brian	0	2015-06-10

```
-- comprobación para a operación de modificación na táboa de vendas:
-- bórrase a venda anterior (ven_id=151)
delete from vendas
where ven_id = 151;
select clt_id, clt_cif, clt_apellidos, clt_nome, clt_vendas, clt_ultima_venta
from tendabd.clientes;
```

dt_id	dt_cif	dt_apellidos	dt_nome	dt_vendas	dt_ultima_venta
1	14013338J	Portela Carracedo	Francisco	2	2015-06-10
2	32727807A	Barreira Vila	Juan	1	2015-05-27
3	18533827J	Armas Tellado	Jose	3	2015-06-10
4	31410232Y	Rodríguez Piñeiro	Brian	0	2015-06-10

1.3.4 Tarefa 4. Crear disparadores para levar rexistros de operacións

A tarefa consiste en crear e probar os disparadores necesarios para levar o rexistro de todas as operacións que modifiquen (*insert*, *update* e *delete*) os datos almacenados nas táboas que hai no seu esquema (*centro*, *departamento*, *empregado*). Para iso débese crear unha táboa na base de datos *traballadores* para o rexistro de todas esas operacións. O código para crear a táboa de rexistro é:

```
/*
Creación dunha táboa para levar un rexistro de todas as operacións
que se realicen sobre as táboas da base de datos de traballadores. Cada
operación de manipulación de datos (insert, update, delete) rexistrarase
nesta táboa de forma automática, creando os disparadores necesarios.
*/
create table if not exists traballadores.rexistroOperacions
(
  idOperacion integer unsigned not null auto_increment,
  usuario char(100),          # usuario que fai oa modificación
  dataHora datetime,         # data e hora na que se fai a modificación
  taboa char(50),            # táboa na que se fai a modificación
  operacion char(6),         # operación de modificación: INSERT, UPDATE, DELETE
);
```

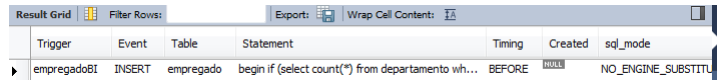
```
primary key (idOperacion)
)engine = myisam;
```

Solución

Hai que crear tres disparadores por cada táboa para as operacións AFTER INSERT, AFTER UPDATE, e AFTER DELETE. En total nove disparadores.

Antes hai que executar a sentenza SHOW TRIGGERS para saber se xa hai algún disparador para esas operacións:

```
show triggers from traballadores;
```



Trigger	Event	Table	Statement	Timing	Created	sql_mode
empregadoBI	INSERT	empleado	begin if (select count(*) from departamento wh...	BEFORE	2015-11-16 11:16:16	NO_ENGINE_SUBSTITUTION

Despois de confirmar que non hai disparadores asociados a esas operacións, escríbense os guións de sentenzas para crear os disparadores.

- Código de creación do disparador asociado á operación AFTER DELETE da táboa *departamento*

```
/*
u703tarefa4.sql

NOME DISPARADOR: traballadores.departamentoAD
DATA CREACIÓN: 16/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR: - Inserir unha fila na táboa rexistroOperacions cada vez que
                        se borra unha fila na táboa de departamento.

EVENTO DISPARADOR: - DELETE
MOMENTO DISPARADOR: - AFTER
RESULTADOS PRODUCIDOS: - Non mostra nada na pantalla. Cada vez que se borra un
                        departamento insírese unha liña na táboa rexistroOperacions

*/
```

```
delimiter //
create trigger traballadores.departamentoAD after delete on departamento
for each row
begin
insert into traballadores.rexistroOperacions (usuario, dataHora, taboa, operacion)
values (user(), now(), 'departamento', 'delete');
end
//
delimiter ;
```

- Código de creación do disparador asociado á operación AFTER INSERT da táboa *departamento*

```
delimiter //
create trigger traballadores.departamentoAI after insert on departamento
for each row
begin
insert into traballadores.rexistroOperacions (usuario, dataHora, taboa, operacion)
values (user(), now(), 'departamento', 'insert');
end
//
delimiter ;
```

- Código de creación do disparador asociado á operación AFTER UPDATE da táboa *departamento*

```
delimiter //
```

```

create trigger traballadores.departamentoAU after update on departamento
for each row
begin
insert into traballadores.rexistroOperacions (usuario, dataHora, taboa, operacion)
values (user(),now(),'departamento','update');
end
//
delimiter ;

```

O resto dos disparadores para as táboas *empregado* e *centro* terían un código similar ao anterior da táboa *departamento*.

– Proba de funcionamento

As probas serían todas moi parecidas sen máis que cambiar o nome da táboa e a operación a realizar sobre ela. Mostrarase como comprobar o funcionamento do disparador *traballadores.departamentoAD*, executando unha sentenza DELETE sobre a táboa *departamento*, e consultando a táboa *rexistroOperacions*.

```

delete from traballadores.departamento
where depNumero = 100;
select * from traballadores.rexistroOperacions;

```

idOperacion	usuario	dataHora	taboa	operacion
1	root@localhost	2015-11-17 21:29:00	departamento	delete

1.3.5 Tarefa 5. Crear disparadores para controlar as restricións referenciais en táboas non transacionais

A tarefa consiste en simular o comportamento de borrado en cascada asociado á restrición de integridade referencial. Para iso, créase e próbase un disparador na base de datos *traballadores*, que faga que cada vez que se borre unha fila na táboa *departamento*, se borren as filas da táboa *empregado* correspondentes aos empregados que traballan nese departamento (o valor da columna *empDepartamento* coincide co *depNumero* do departamento borrado).

Solución

– Código de creación

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 AFTER INSERT para a mesma táboa, pero pódese crear un disparador BEFORE INSERT e outro AFTER INSERT para a mesma táboa. Polo tanto, débese de executar a sentenza SHOW TRIGGERS para saber se xa hai algún disparador para esa operación.

```
show triggers from traballadores;
```

Trigger	Event	Table	Statement	Timing	Created	sql_mode
departamentoAI	INSERT	departamento	begin insert into traballadores.rexistroOperacio...	AFTER	NO_ENGINE_SU	NO_ENGINE_SU
departamentoAU	UPDATE	departamento	begin insert into traballadores.rexistroOperacio...	AFTER	NO_ENGINE_SU	NO_ENGINE_SU
departamentoAD	DELETE	departamento	begin insert into traballadores.rexistroOperacio...	AFTER	NO_ENGINE_SU	NO_ENGINE_SU
empregadoBI	INSERT	empregado	begin if (select count(*) from departamento wh...	BEFORE	NO_ENGINE_SU	NO_ENGINE_SU

Na saída que mostra a sentenza SHOW TRIGGERS pódese ver que para a táboa *departamento* xa existe un disparador asociado á operación AFTER DELETE. Neste caso hai

que borrar o disparador e crealo de novo, incluíndo no corpo do disparador todas as sentenzas do disparador que xa existía e as que corresponden ao novo.

```
/*
u703tarefa05.sql

NOME DISPARADOR: traballadores.departamentoAD
DATA CREACIÓN: 19/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:    - Rexistrar na táboa rexistroOperacions a operación de
                           borrado na táboa de departamento (da tarefa 4)
                           - Borrado en cascada: Cando se borre unha fila da táboa
                             departamento bórranse todas as filas da táboa empregado
                             que teñan o número dese departamento na columna
                             empDepartamento

EVENTO DISPARADOR:        - DELETE
MOMENTO DISPARADOR:        - BEFORE
RESULTADOS PRODUCIDOS:    - Cada vez que se borra un departamento bórranse todos os
                             empregados asignados a el e insírese unha fila na táboa
                             rexistroOperacions

*/

drop trigger if exists traballadores.departamentoAD;
delimiter //
create trigger traballadores.departamentoAD after delete on departamento
for each row
begin
-- sentenzas que ten o disparador que existe
insert into traballadores.rexistroOperacions (usuario, dataHora, taboa, operacion)
    values (user(),now(),'departamento','delete');
-- sentenzas correspondentes as novas accións do disparador
delete from empregado
    where empDepartamento = old.depNumero;
end
//
delimiter ;

- Proba de funcionamento
-- Probas de borrado
-- Antes de borrar o departamento 110, compróbase o número de empregados que existen
select count(*) from traballadores.empregado; # devolve 39 filas
-- Tamén se pode comprobar cantos empregados hai do departamento 110
select count(*) from traballadores.empregado
    where depNumero = 110; # devolve 3 filas
delete from traballadores.departamento
    where depNumero = 110;
-- Despois de borrar o departamento 110, compróbase o número de empregados que existen
select * from traballadores.empregado; # devolve 36 filas
```

1.3.6 Tarefa 6. Borrar disparadores

A tarefa consiste en borrar o disparador *practicass1.empregadoBI* creado na tarefa 1.

Solución

```
- Código de creación
drop trigger if exists practicas1.empregadoBI;
```

1.3.7 Tarefa 7. Planificar eventos

A tarefa consiste en crear eventos atendendo a varios supostos:

- Tarefa 7.1. Crear un evento na base de datos *traballadores* que execute cada hora o procedemento almacenado *sp_actualizar_depEmpregados*. O evento empeza a executarse dentro de 12 horas, e non remata de executarse ata que se borre o evento, ou se deshabilite.
- Tarefa 7.2. Crear un evento na base de datos *tendabd* que faga o peche anual das vendas o día 1 de xaneiro de 2016 ás 00:00. Para facer o peche hai que:
 - Copiar a información das *ventas* que xa foron facturadas (*ven_factura* distinto de null) na táboa *hventas* que recolle a información histórica das vendas, e despois borrar da táboa *vendas* as filas copiadas.
 - Facer o mesmo coas filas da táboa *detalle_vendas* correspondentes as vendas borradas, na táboa *hdetalle_vendas*.

Solución

■ Tarefa 7.1

```
-- habilitar o planificador de eventos
set global event_scheduler = on;
-- crear o evento
delimiter //
create event traballadores.actualizar_depEmpregados
  on schedule every 1 hour
  starts now() + interval 12 hour
do
begin
  call sp_actualizar_depEmpregados();
end
//
delimiter ;
```

■ Tarefa 7.2

```
-- habilitar o planificador de eventos
set global event_scheduler = on;
-- crear o evento
drop event if exists tendabd.peche_anual_vendas;
delimiter //
create event tendabd.peche_anual_vendas
on schedule at '2016-01-01 00:00:00'
do
begin
  /*Inserción: faise a inserción das filas que hai que copiar nos táboas coa
  información histórica*/
  insert into hventas (hven_id,hven_tenda,hven_empregado,hven_cliente,hven_data)
    select ven_id,ven_tenda,ven_empregado,ven_cliente,ven_data
    from vendas
    where ven_factura is not null;

  insert into hdetalle_vendas(hdet_venta, hdet_numero, hdet_artigo, hdet_cantidad,
hdet_importe)
    select dev_venta, dev_numero, dev_artigo, dev_cantidad,
      (dev_prezo_unitario*dev_cantidad)*dev_desconto/10 # Cálculo de hdet_importe
    from detalle_vendas
    where dev_venta in (select ven_id from vendas where ven_factura is not null);
  /*Borrado: bórranse primeiro as liñas de detalle das vendas porque hai que utilizar
```

```
información das vendas que hai que borrar na subconsulta*/  
delete from detalle_vendas  
    where dev_venta in (select ven_id from vendas where ven_factura is not null);  
delete from vendas  
    where ven_factura is not null;  
end  
//  
delimiter ;
```